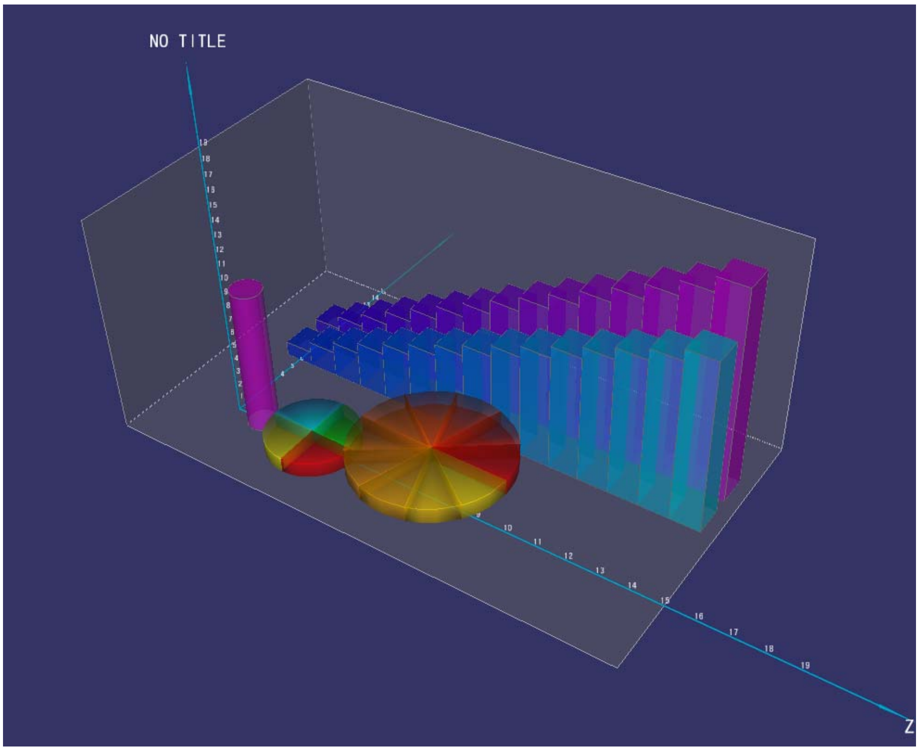
DDRL: Dynamic Data Representation Language



Columbia University

COMS W4115 Programming Languages and Translators

Spring 2007

Prof. Stephen Edwards
TA: Abhilash Itharaju

Team Members:
Zhiyang Cao zc2109@columbia.edu
Yitao Wang yw2226@columbia.edu
Alex Ling Lee al2537@columbia.edu
Yan Zhang yz2197@columbia.edu
Kyung Kwan Kim kk2367@columbia.edu

Part I Introduction

1 Motivation

Charts are commonly used to show the trend of sets of data. Although some tools such as Excel and Matlab are capable of generating charts by using script language, the results are usually static and hard to show the changing of data. Our objective is to make a simple but powerful language that represents data by using animated charts.

2 Description

The DDRL language is designed as a C-like language that represents formed data dynamically. It uses more flexible data type, expression and control structure, but is much easier and simpler than C language. Since the DDRL focuses on generating the graphics shapes and control sequence for dynamic presentation of the data, only basic flow controls and simple operators is borrowed from C language.

Our language consists of three parts: basic flow control and expression, graphics defines and control, animation sequence define and control, we will specify all these following.

The powerful back-end is an important feature of DDRL. Written in OpenGL, the back-end is capable of generating 3D graphic objects and controlling the animation of the shapes.

Part II Analysis of Examples and Tutorial

1. Example 1

```
num main()
{
    num number, i, j;
    number = 20;
    j=0;
    array num a;
    for(i=0; i < number ;i=i+1){
        if(j=0){
            a[i] = i*2;
            j=1;
        }else{
            j=0;
13                  }
14                  output a[i];
15              }
16      }
```

Line 3-6 is the declaration of the main program. Our declaration can be made in anywhere of the program. In here, we have only declared num and array of num. The size of array will be variable as vector in C++ and we set it as 10 in the for loop(line 7-15). The syntax of for loop is similar to C beside that we only support “i=i+1” instead of “i++”. Line8-13 is the if-else statement, it is also a C-like function, the operators of this statement are”=”, “!=”, “>”, “<”, “>=”, “<=”. At the end, we have an output statement(line 14) to printout the array of num “a” to screen.

2. Example 2

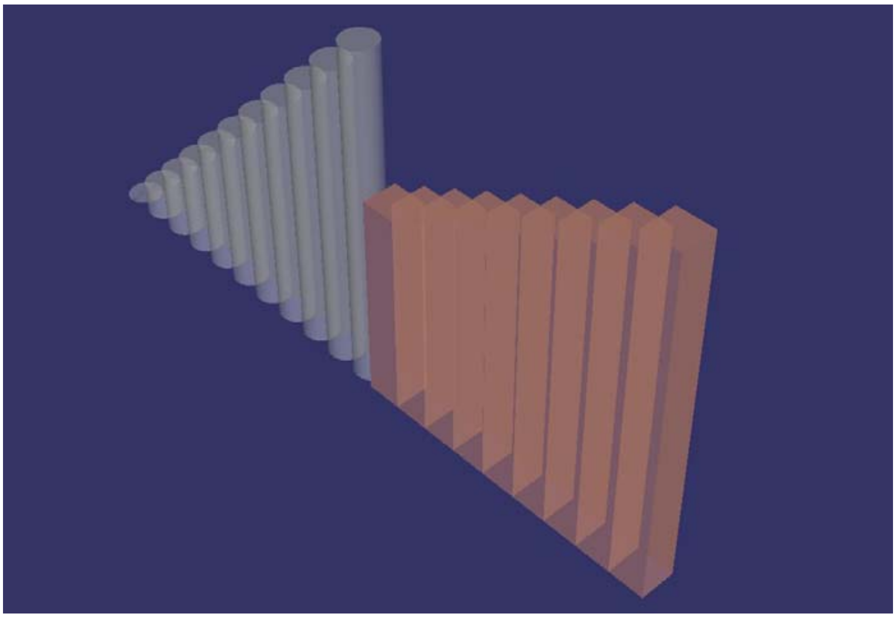
```
1    displaymodel columnchart(num size)
2    {
3         array num data;               //define the data array
4         num r, g, b, a;               //the color for our selection
5         r = 255;
6         g = 155;
7         b = 155;
8         a = 155;
9         $<display
10             num i;
11             for(i=0;i<size;i=i+1){        //loop the array for all the data
12                  if( i>size/2){
13                       color r,g,b,a;       //use our defined color
14                       box            //draw the box, height as value
15                            vertex i*2,0.0,0.0;     //start at current time of this model
16                            <$,#>:
17                            2.0 , 2.0, data[i]*2;   //duration is default time for Box
18                       endbox;
19                  } else {
20                       color 255 255 255 100;
21                       cylinder
22                            vertex i*2, 0.0, 0.0;
23                            <$,#>:
24                            2.0, data[i]*2;
25                       endcylinder
26                  }
27             }
28        $display>
29   }
```

The code above is a graphics define function, this function define the models like box and cylinder that will used when the data is showing up. Line 3-8 is the declaration of this function, same as the declaration in main program, the declaration of variables can be in everywhere and thus we are using static scoping in our program. Line 9-28 is the display model definition, we have defined 2 models, one is box(line14-18) and one is cylinder(line21-25). In our language, we can also have model “slice” and “plane”. For the box model(line14-18), the first token must be “vertex”(line15). It will be followed by 3 variables which imply the position of the model at the x, y, z axis on the graph. In here, we use “i*2” as the input argument for the x variable. It means that x variable will change according to the value of i in the control flow.

Line 16 is the time-tag statement; it controls the time slice of the animation of box model. For easy define the start time and duration of animation, the language provides two wildcard for time tag statement: ‘$’ and ‘#’. ‘$’: the time that last animation ends. It can only used for starttime in time tag statement. ‘#’: the duration of the graphics animation. The default length is 1 second. Other number should also fit in this field. Line 17 includes 3 numbers which define the size of box. The last line of this model definition is endbox(line18), this line just simple means that the definition of box finished. Similar to box definition, cylinder definition has nearly the same syntax, but at line 24, it includes only 2 variables, the first one is the radius of the cylinder and the next one is height of the cylinder. There will be other more models that we can use, and the syntax has stated in our LRM.

```
32   num main()
33   {
34          num number;
35          number = 20;
36          displaymodel columnchart    x(20);
37          num i;
38          for(i=0; i<number; i=i+1){
39                 x.data[i] = i*2;
40                 output x.data[i];
41          }
42          <4,10>: show x;
43   }
```

In the main program, we need to declare the graphics define function before we use it (at line 36). In line 38-41, the data is fed into the array “data”. Line 42 is the control graphics statement, “<4, 10>” is the time-tag statement that stated above. “show x” means that the displaymodel x ‘s state is “show”. There are other states like rotation, scaling, and unshow which is also stated in the LRM.



Part III LRM

1. Lexical Conventions

1.1 Comments

Every line of comments should begin with “//” and end at the end of the line.

1.2 Identifiers

An identifier must start with a lower case letter, and only include lower case letter and a number. A number, if contained by an identifier, can only be placed at the end of it.

1.3 Keywords

The following words are reserved by the system as keywords and should not be used as identifiers. Note that all key words in DDRL starts with capital letters

Basic:

if else while for foreach

return break continue

num dataset string array

struct newstruct

main

Graphics Define:

color vertex pointsize linewidth

point endpointline endline trianglemesh endtranglemesh

box endbox cylinder endcylinder slice endslice

plane endplane axis endaxis axismarker endaxismarker

text endtext displaymodel display

Time tag:

loop clear

Graphics Control:

show unshow position rotation scaling

1.4 Types and Constants

3 types of data type are supported: num, string.

num: 32 bit float point data type;

array: a sequence of data objects;

string: a sequence of characters, cannot contain “ ” ” and “\n”;

dataset: A buffered data manipulator, used to manipulate data from a file.

displaymodel: A buffered data manipulator, used to define the display module.

Accordingly, available constants include:num constants, string constants.

Note that programmers can use the attributes and member function of dataset type to manipulate the content of a file. It can retrieve the column size, row size, column name, row name and the data.

Example:

```
dataset mydata = OpenFile(“./example.txt”);
num size = mydata.columnsize;
string rowname = mydata.rowname[1];
array num data[size] = mydata.row[1];
```

2. Expressions

DDRL expressions are list by precedence form high to low:

2.1.1 Primary expressions

Primary expressions include identifiers, constants, function calls, contained within group left to right.

2.1.2 Identifier

Identifier can be num, string, dataset, displaymodel;

2.1.3 Constant

Constants in DDRL include num constant, and string constant;

2.1.4 *(expression)*

The parenthesis is used to construct the precedence;

2.1.5 *Primary-expression <expression-list>*

Such expression is used to clarify the point on time axis;

2.1.6 *Primary-expression (expression-list)*

This is used to describe function call, which is primary expression. As a part of function call, parenthesis is required. Since DDRL doesn't support pointer type, all arguments are passed by value. Recursively function calls are supported.

2.2 Operators

2.2.1 Unary Operator

-expression

The type of the operand here must be num. Operator – doesn't change the type of the expression, it only gives the negative value of the original operand. It associates from right to left.

2.3 Additive Operators

expression +expression,

expression – expression

2.4 Multiplicative Operators

*expression * expression*

expression / expression

2.5 Exponential Operator

expression ^ expression

2.6 Relational Operators

expression < expression

expression > expression

expression <= expression

expression >= expression

2.7 Equality Operators

expression == expression

expression != expression

2.8 Logical Operators

expression && expression

expression || expression

2.9 Assignment operator

identifier = expression

2.10 Other Operators

expression , expression

Comma is used to separate 2 or more expressions;

expression;

“;” is used to terminate a statement;

{expression;}

“{“,”}” are used to group expressions.

(expression)

“(“,”)” are used to hold function arguments.

[expression]

“[“,”]” are used to hold index of array members.

$

“$” is used to identify current time.

#

“#” is used to identify time elapsed.

3. Delearation

3.1 Type specifiers

The type specifiers are:

Type-specifier:

num

string

dataset

displaymodel

newstruct

id

3.2 Declarators

declarator-list:

declarator

declarator declarator-list

declarator:

Type identifier

declarator declarator-list:

Type identifier

3.3 Declaration of struct type

struct identifier { typedecllist}

To standardize declaration, DDRL only support one way of declaring struct.

3.4 Declaration of array type

array type-specifier identifier[size]

3.5 Display Model Dclaration

DisplayModel identifier (parameter-list)

This will be specified in §8

3.6 Function Definition

func definition:

typespecifier identifier(parameter-list)

{

vardeclaration-list

basic-statement-list

graphics-control-statement-list

}

parameter-list:

typespecifier identifier

typespecifier identifier parameter-list

vardeclaration-list:

vardeclaration

vardeclaration vardeclaration-list

basic-statement-list:

basic-statement

basic-statement statement-list

graphic-control-statement-list

graphic-control-statement

graphic-control-statement graphic-control-statement-list

Note that functions should be defined before declared.

4. Basic Statements

Every statement should be ended with a semicolon. Statement is executed in sequence.

4.1 Expression statement

Most statements are expression statements, which have the form *expression ;*

Usually expression statements are assignments or function calls.

4.2 Compound statement

compoundstatement:

{ statementlist}

basic-statementlist:

basic-statement

basic-statement basic-statementlist

4.3 Conditional statement

If (expression) statement

If (expression) statement Else statement

4.4 While statement

While (expression) statement

4.5 For statement

Fo*r (expression; expression; expression) statement*

4.6 Foreach statement

Foreach (identifier In identifier) statement

4.7 Switch statement

Switch (expression) statement

4.8 Break statement

Break;

4.9 Continue statement

Continue;

4.10 Return statement

Return (expression);

5. Graphics Define Statement

5.1 Vertex Statement

Specify a 3D coordinate for a vertex, which is used to define a vertex in a 3D model. It has the form

Vertex-statement:

Vex expression, expression, expression;

Normally, the expression would be identifier, constant and other simple expression; There expressions are the 3D coordinate of the vertex in form of *x, y, z*;

5.2 Compound Graphics Statement

The compound graphics statement consists of two parts: one is the basic statement; another is the vertex-statement. It has following form.

Compound-graphics-statement:

Basic-statement

Vertex-statement-list

Basic-statement Compound-graphics-statement

Vertex-statement-list Compound-graphic-statement

Vertex-statement-list:

Vertex-statement

Vertex-statement vertex-statement-list

5.3 Color Statement

Specify a color for the following 3D model; by default, the system color is black. When a color statement specifies a kind of color for system, it will remain this color for following any 3D model unless another color statement specifies a different color.

The Statement

Color-statement:

Color expression, Time-tag-statement,expression, expression, expression;

Four expressions are the color exponents of *red, green, blue, alpha*; all values of color component should in the range of 0 - 255.

5.4 Point Size Statement

Define the size of the point displayed in 3D. It has the form

Point-size-statement:

Pointsize expression;

The point size should be an integer which at least is 1.

5.5 Line Width Statement

This statement defines the width of the line displayed in 3D. It has the form

Line-width-statement:

Linewidth expression;

The line width should be an integer which at least is 1.

5.6 Point Statement

Point statement defines a serial of point that in the 3D. This statement should also define at least one point by using *vertex-statement*.

Point-statement:

Point Compound-graphics-statement Time-tag-statement EndPoint;

5.7 Line Statement

Line Statement defines a single line connected all the vertexes specified in the statement. This statement should define at least two points by vertex-statement. And the line will connect the point in order.

Line-statement:

Line Compound-graphics-statement Time-tag-statement EndLine;

5.8 Polygon statement

This statement defines the polygon in 3D by specifying the vertex of the polygon. The polygon should have at least 3 vertexes and all the vertexes should convex, otherwise the polygon will not be showed.

Polygon-statement:

Polygon Compound-graphics-statement Time-tag-statement EndPolygon;

5.9 Box statement

Box statement defines a box in 3D by specifying the *width, length* and *height*.

The box normally is placed at the point defined. And the width is the value on *x-axis*, length is the value on *z-axis*, height will be the value on *y-axis*;

Box-statement:

Box Compound-graphics-statement Time-tag-statement expression, expression, expression; EndBox;

5.10 Cylinder statement

Cylinder statement defines a cylinder in 3D by specifying the *radius and height*. The cylinder will be placed at the point defined. And the height will be the value on *y-axis*.

Cylinder-statement:

Cylinder Compound-graphics-statement Time-tag-statement expression, expression; EndCylinder;

5.11 Slice statement

Slice statement will defines parts of the pie model in 3D. It should define the radius, height, startAngle and angle of the slice. And the height is the value on y-axis. And the startAngle on the positive direction of x-axis is 0.

Slice-statement:

Slice Compound-graphics-statement Time-tag-statement expression, expression, expression; EndSlice;

The four expressions are radius, height, startAngle and Angle in order.

5.12 Plane statement

This statement defines a plane in 3D by specifying its leftmost vertex and rightmost vertex.

Plane-statement:

Plane Compound-graphics-statement Time-tag-statement EndPlane;

5.13 Axis statement

This defines and displays the axis in 3D. Only three axes X, Y, Z can be defined.

Axis-statement:

Axis X expression, expression, expression; EndAxis;

Axis Y expression, expression, expression; EndAxis;

Axis Z expression, expression, expression; EndAxis;

The three expressions are the displayed *title* of the axis, the *length* of the axis and the *interval* of the axis for marker.

The title is string type;

The length is integer type;

The interval is integer type;

5.14 AxisMarker statement

AxisMarker statement defines the title for the interval marker on the axis.

AxisMarker-statement:

AxisMarker Marker-statement-list EndAxisMarker;

Marker-statement-list:

Marker-statement

Marker-statement Marker-statement-list

Marker-statement:

expression, expression;

The two expressions are the index of the interval marker and the title of this marker. The index should be integer and the title should be string.

5.15 text statement

The text statement defines the text displayed in both 3D and 2D. The text should be a string. It has the form

Text-statement:

Text expression vertex-statement vertex-statement

If the vertex-statement third expression equals to zero, then the text will be showed in 2D, and the last two expression are the starting point and the end point of the moving word.

5.16 graphics define statement

For following specification, we state this statement which is one of the most important parts in our language. The graphics define statement is used to define the 3D elementary models that will be displayed.

Graphics-define-statement:

Color-statement

Point-size-statement

Line-width-statement

Point-statement

Line-statement

Polygon-statement

Box-statement

Cylinder-statement

Slice-statement

Plane-statement

Axis-statement

AxisMarker-statement

Text-statement

Graphics-define-statement-list:

Graphics-define-statement

Graphics-define-statement   Graphics-define-statement-list

6. Time Tag Statement

The time tag statement controls the time slice of the animation of special graphics statement. The time tag statement specifies the start time and the duration of the animation. The time tag statement can only control (precedence of) the graphics statement, like graphics define statement and graphics control statement.

6.1 Time tag statement

Time-tag-statement:

<*starttime, duration*>*:*

<*starttime, duration*>*:Loop*

<*starttime, duration*>*:Clear*

The *starttime* controls the start time of the following graphics animation. The *duration* is the length of this animation.

Keyword *Loop* means redisplay the following animation after it finishes.

Keyword *Clear* means clear the following graphics elements after it finishes.

The *starttime* and *duration* are integer type, and should be positive.

Time tag statement only effects the graphics statement on the same line. And graphics statement includes *Graphics-define-statement* and *Graphics-control-statement.*

We will specify the *Graphics-control-statement* at §9.

6.2 time wildcard

For easy define the start time and duration of animation, the language provides two wildcard for time tag statement: '$' and '#'.

'$': the time that last animation ends. It can only used for starttime in time tag statement.

'#': the duration of the graphics statement animation. The *Graphics-define-statement* default length is 1 second. And duration of the *Graphics-control-statement* is the length of its controlled Display Model. Looped animation will only count once for its duration.

Example: <$, #>:*Loop*

The scope of the wildcard is only in the current display model.

7. Display Model Definition

In our language, the major display model is Display Model. The model is defines by *type-decl-list* and *Graphics-define-statement.* The *type-decl-list* defines the data member of the Model. This should be at the top of the model. The *Graphics-define-statement* will define the graphics and animation sequence of the model.

The Display Model is the basic element of program. It can only be controlled by graphics control statement.

```
Display-model-definition:
            DisplayModel identifier( parameter-list )
            {
                type-decl-list
                $<DISPLAY
                    Display-model-def
                $DISPLAY>
            }
```

```
parameter-list:
            typespecifier identifier
            typespecifier identifier parameter-list
```

```
Display-model-def:
        Basic-statement
        Graphics-define-statement
        Basic-statement Display-model-def
        Graphics-define-statement Display-model-def
```

8. Graphics Control Statement

The graphics control statement control the display or animation of the display model. It can only control the display model. Control includes show the model; unshow the model, position of the model, rotation of the model and scaling of the model.

8.1 show model statement

This statement will show the display model on the screen according to its position setting. By default the model position will be at the (0, 0, 0);

Show-statement:

Show DisplayModel;

8.2 Unshow model statement

This statement will hide the display model on the screen.

Unshow-statement:

Unshow DisplayModel;

8.3 Position model statement

Set the position of the model. x,y,z are the position.

Position-statement:

Position DisplayModel x,y,z;

8.4 Rotation model statement

Set the rotation of the model. x,y,z are the rotation angle.

Rotation-statement:

Rotation DisplayModel x,y,z;

8.5 Scaling model statement

Set the scaling of the model.

Scaling-statement:

Scaling DisplayModel num;

8.6 Graphics Control Statement

All statements above compose the graphics control statement.

Graphics-control-statement:

Show-statement

Unshow-statement

Position-statement

Rotation-statement

Scaling-statement

9. Scope rules

DDRL supports the declaration of global variables.

For local variables which are declared inside specific functions, they belong only to those functions and models.

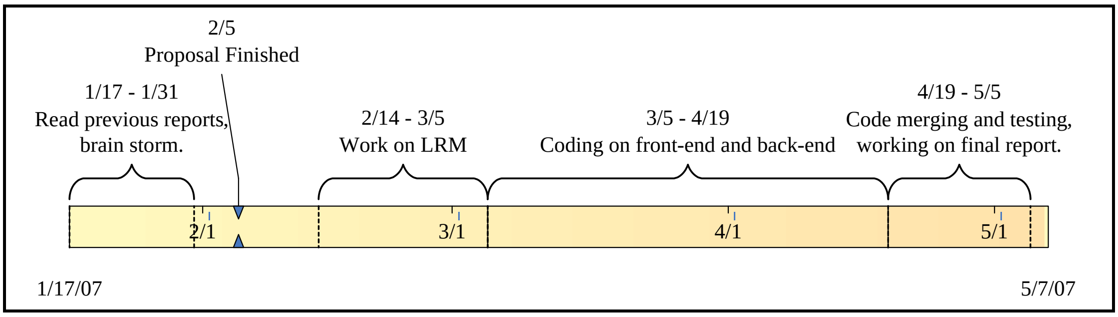
In Compound statement, the local variables are only effective between ‘{‘ and ‘}’;

Part IV Project Plan

1 Team Responsibilities

The project is done by module and everyone participates in some parts of it. The front-end and back-end are proceed at the same time before code merge.

2 Time line



3 Responsibility of team members

Team Member	Responsibility
Zhiyang Cao	Architecture, grammar, Graphics Generation, Data factory and symbol table. Control flow.
Yitao Wang	Backend(Data Type), function, display model function, arithmetic and logic operation.
Yan Zhang	Parser, Lexer, Walker and Testing(Data Type), control flow,
Alexandre Ling Lee	Parser, Walker and Testing(Graphic control), display model function
KK	Testing and document

LRM, Interface design, test, Final Report is completed by all team members.

4 Project Log

Jan 17	Reading previous reports and discuss
Jan 31	Brain storming and decide language features
Feb 5	Proposal Finished
Feb 14	Draft of LRM finished
March 5	Language Reference Manual finished
March 6	Started coding on front-end and back-end
April 2	Basic data types are finished
April 10	Back-end function (graphic control) finished
Mid April	Code merge started
Mid April- First week of May	Testing and error recovery
May 5	Final report draft finished
May 8	Final Presentation

5 Software Environment

Different from other projects, DDRL project is written in C++. The lexer and parser are both written in Antlr, and translated to C++ code afterward. To ensure lexer and parser work well before back-end work , some early test cases were written in java.

JDK version: 1.4.2
Operating System: Windows XP

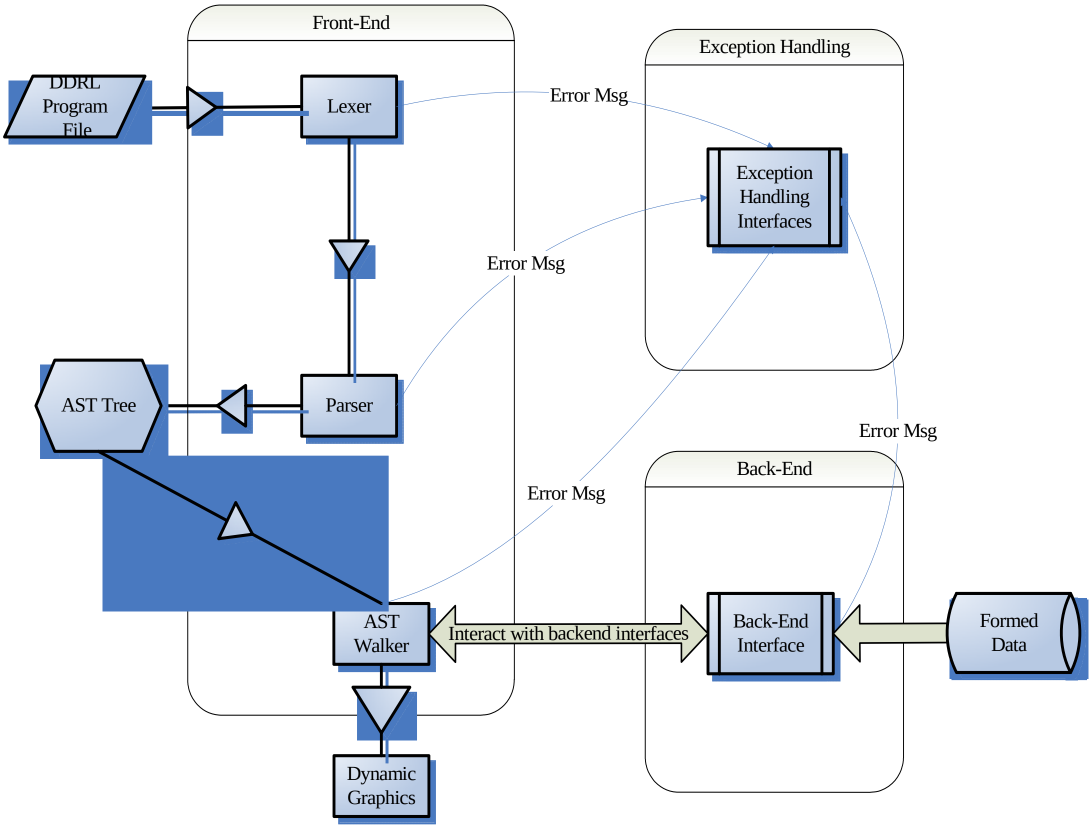
SVN: Our team used SVN to control version in the process of developing. The newest version will be uploaded onto the SVN server after changes are made.

There are 106 versions of our source code by 5/8/2007

Part V Architecture Design

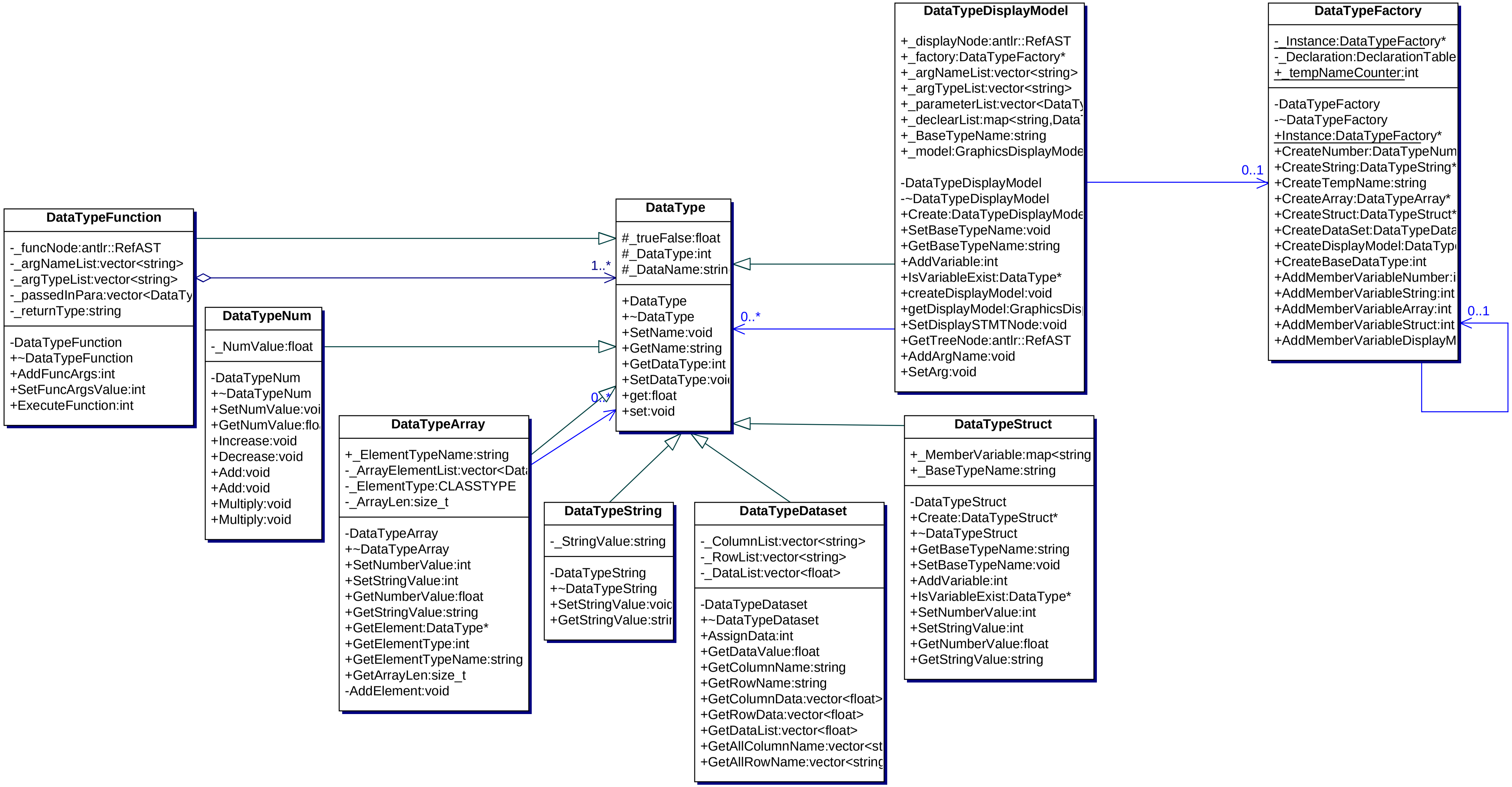
1 Program Architecture

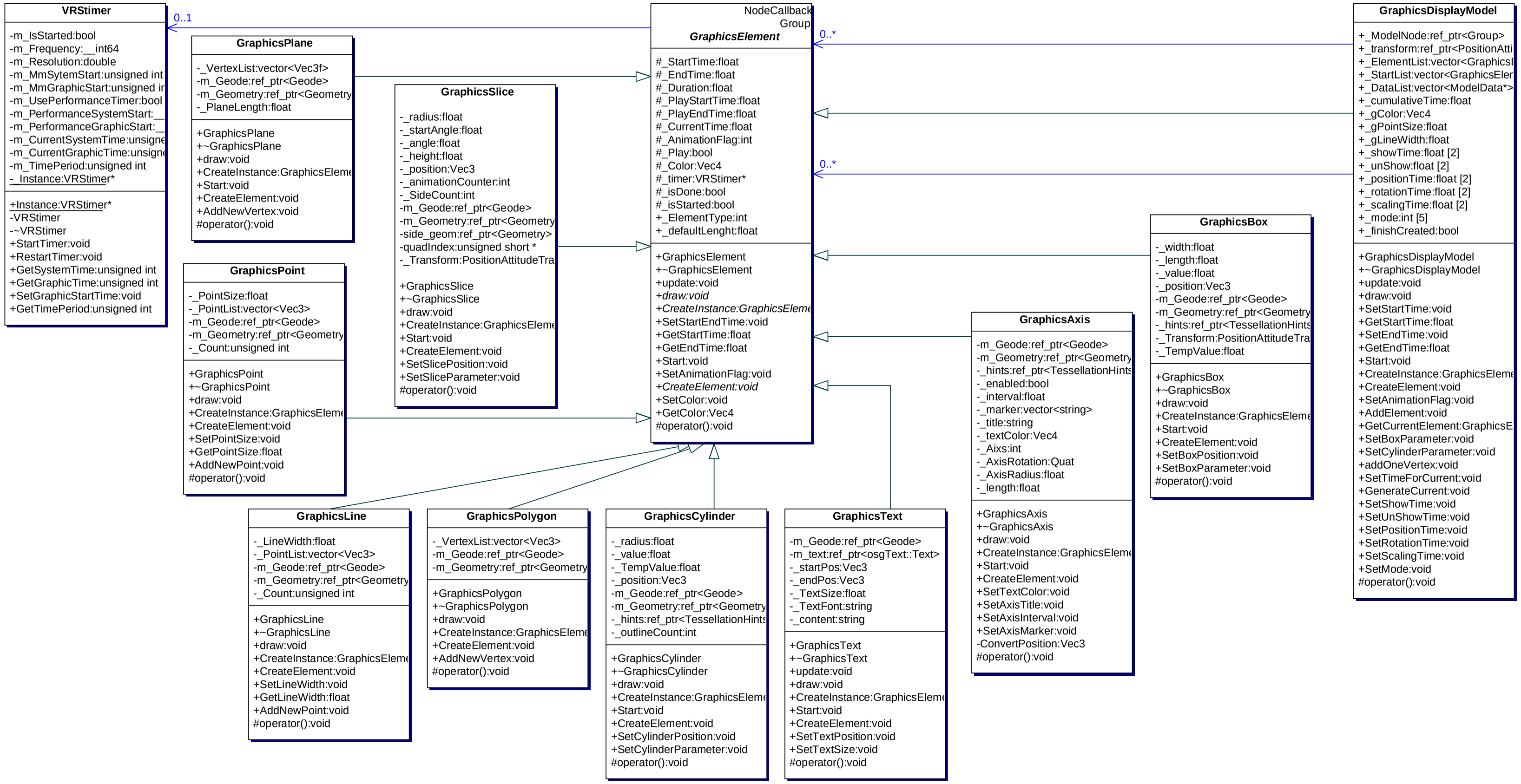
DDRL architecture consists of 2 parts: front-end and back-end. Front-end read in DDRL program, creates a lexer, parser and walker object. In the process, an AST tree is generated by parser. The walker walks the tree and calls the back-end interface functions. The backend will import data from a data table and generate animated charts.



2 Data type design

DDRL provides complex and powerful data structure for users. Basic data type in DDRL includes num (which combines float and int) and string. Advance data types which are used to contain other data types include struct, array, and displaymodel. DDRL struct is the container of other DDRL data types. It can contain num, string and array as its member. DDRL array is quite different from C array. The element in a DDRL array can be num, string, struct, displaymodel, function, and any data type defined in out language. To achieve our objective, all data type classes (DataTypeNum, DataTypeString, DataTypeStruct, DataTypeArray) are derived from a base class called DataType, and is manipulated by a factory class called DataFactory. The DataFactory class also manipulates the declaration table and symbol table. The relationship of classes is shown in the following chart.


DataTypeFunction
-_funcNode:antlr::RefAST
-_argNameList:vector<string>
-_argTypeList:vector<string>
-_passedInPara:vector<DataTy
-_returnType:string
-DataTypeFunction
+~DataTypeFunction
+AddFuncArgs:int
+SetFuncArgsValue:int
+ExecuteFunction:int
DataTypeNum
-_NumValue:float
-DataTypeNum
+~DataTypeNum
+SetNumValue:voi
+GetNumValue:flo
+Increase:void
+Decrease:void
+Add:void
+Add:void
+Multiply:void
+Multiply:void
DataTypeArray
+_ElementTypeName:string
-_ArrayElementList:vector<Data
-_ElementType:CLASSTYPE
-_ArrayLen:size_t
-DataTypeArray
+~DataTypeArray
+SetNumberValue:int
+SetStringValue:int
+GetNumberValue:float
+GetStringValue:string
+GetElement:DataType*
+GetElementType:int
+GetElementTypeName:string
+GetArrayLen:size_t
-AddElement:void
DataTypeString
-_StringValue:string
-DataTypeString
+~DataTypeString
+SetStringValue:voic
+GetStringValue:strir
DataTypeDataset
-_ColumnList:vector<string>
-_RowList:vector<string>
-_DataList:vector<float>
-DataTypeDataset
+~DataTypeDataset
+AssignData:int
+GetDataValue:float
+GetColumnName:string
+GetRowName:string
+GetColumnData:vector<float>
+GetRowData:vector<float>
+GetDataList:vector<float>
+GetAllColumnName:vector<st
+GetAllRowName:vector<string
DataType
#_trueFalse:float
#_DataType:int
#_DataName:strin
+DataType
+~DataType
+SetName:void
+GetName:string
+GetDataType:int
+SetDataType:voi
+get:float
+set:void
1..*
0..*
0..*
DataTypeDisplayModel
+_displayNode:antlr::RefAST
+_factory:DataTypeFactory*
+_argNameList:vector<string>
+_argTypeList:vector<string>
+_parameterList:vector<DataTy
+_declearList:map<string,Data
+_BaseTypeName:string
+_model:GraphicsDisplayMode
-DataTypeDisplayModel
-~DataTypeDisplayModel
+Create:DataTypeDisplayMode
+SetBaseTypeName:void
+GetBaseTypeName:string
+AddVariable:int
+IsVariableExist:DataType*
+createDisplayModel:void
+getDisplayModel:GraphicsDis
+SetDisplaySTMTNode:void
+GetTreeNode:antlr::RefAST
+AddArgName:void
+SetArg:void
DataTypeStruct
+_MemberVariable:map<string
+_BaseTypeName:string
-DataTypeStruct
+Create:DataTypeStruct*
+~DataTypeStruct
+GetBaseTypeName:string
+SetBaseTypeName:void
+AddVariable:int
+IsVariableExist:DataType*
+SetNumberValue:int
+SetStringValue:int
+GetNumberValue:float
+GetStringValue:string
0..1
DataTypeFactory
-_Instance:DataTypeFactory*
-_Declaration:DeclarationTable
+_tempNameCounter:int
-DataTypeFactory
-~DataTypeFactory
+Instance:DataTypeFactory*
+CreateNumber:DataTypeNum
+CreateString:DataTypeString*
+CreateTempName:string
+CreateArray:DataTypeArray*
+CreateStruct:DataTypeStruct*
+CreateDataSet:DataTypeData
+CreateDisplayModel:DataTyp
+CreateBaseDataType:int
+AddMemberVariableNumber:i
+AddMemberVariableString:int
+AddMemberVariableArray:int
+AddMemberVariableStruct:int
+AddMemberVariableDisplayM
0..1


VRStimer
-m_IsStarted:bool
-m_Frequency:__int64
-m_Resolution:double
-m_MmSytemStart:unsigned int
-m_MmGraphicStart:unsigned ir
-m_UsePerformanceTimer:bool
-m_PerformanceSystemStart:__
-m_PerformanceGraphicStart:__
-m_CurrentSystemTime:unsigne
-m_CurrentGraphicTime:unsigne
-m_TimePeriod:unsigned int
-_Instance:VRStimer*
+Instance:VRStimer*
-VRStimer
-~VRStimer
+StartTimer:void
+RestartTimer:void
+GetSystemTime:unsigned int
+GetGraphicTime:unsigned int
+SetGraphicStartTime:void
+GetTimePeriod:unsigned int
0..1
GraphicsPlane
-_VertexList:vector<Vec3f>
-m_Geode:ref_ptr<Geode>
-m_Geometry:ref_ptr<Geometry
-_PlaneLength:float
+GraphicsPlane
+~GraphicsPlane
+draw:void
+CreateInstance:GraphicsEleme
+Start:void
+CreateElement:void
+AddNewVertex:void
#operator():void
GraphicsSlice
-_radius:float
-_startAngle:float
-_angle:float
-_height:float
-_position:Vec3
-_animationCounter:int
-_SideCount:int
-m_Geode:ref_ptr<Geode>
-m_Geometry:ref_ptr<Geometry
-side_geom:ref_ptr<Geometry>
-quadIndex:unsigned short *
-_Transform:PositionAttitudeTra
+GraphicsSlice
+~GraphicsSlice
+draw:void
+CreateInstance:GraphicsEleme
+Start:void
+CreateElement:void
+SetSlicePosition:void
+SetSliceParameter:void
#operator():void
GraphicsPoint
-_PointSize:float
-_PointList:vector<Vec3>
-m_Geode:ref_ptr<Geode>
-m_Geometry:ref_ptr<Geometry
-_Count:unsigned int
+GraphicsPoint
+~GraphicsPoint
+draw:void
+CreateInstance:GraphicsEleme
+CreateElement:void
+SetPointSize:void
+GetPointSize:float
+AddNewPoint:void
#operator():void
NodeCallback
Group
GraphicsElement
#_StartTime:float
#_EndTime:float
#_Duration:float
#_PlayStartTime:float
#_PlayEndTime:float
#_CurrentTime:float
#_AnimationFlag:int
#_Play:bool
#_Color:Vec4
#_timer:VRStimer*
#_isDone:bool
#_isStarted:bool
+_ElementType:int
+_defaultLenght:float
+GraphicsElement
+~GraphicsElement
+update:void
+draw:void
+CreateInstance:GraphicsEleme
+SetStartEndTime:void
+GetStartTime:float
+GetEndTime:float
+Start:void
+SetAnimationFlag:void
+CreateElement:void
+SetColor:void
+GetColor:Vec4
#operator():void
0..*
0..*
GraphicsDisplayModel
+_ModelNode:ref_ptr<Group>
+_transform:ref_ptr<PositionAtti
+_ElementList:vector<GraphicsE
+_StartList:vector<GraphicsEle
+_DataList:vector<ModelData*>
+_cumulativeTime:float
+_gColor:Vec4
+_gPointSize:float
+_gLineWidth:float
+_showTime:float [2]
+_unShow:float [2]
+_positionTime:float [2]
+_rotationTime:float [2]
+_scalingTime:float [2]
+_mode:int [5]
+_finishCreated:bool
+GraphicsDisplayModel
+~GraphicsDisplayModel
+update:void
+draw:void
+SetStartTime:void
+GetStartTime:float
+SetEndTime:void
+GetEndTime:float
+Start:void
+CreateInstance:GraphicsEleme
+CreateElement:void
+SetAnimationFlag:void
+AddElement:void
+GetCurrentElement:GraphicsE
+SetBoxParameter:void
+SetCylinderParameter:void
+addOneVertex:void
+SetTimeForCurrent:void
+GenerateCurrent:void
+SetShowTime:void
+SetUnShowTime:void
+SetPositionTime:void
+SetRotationTime:void
+SetScalingTime:void
+SetMode:void
#operator():void
GraphicsBox
-_width:float
-_length:float
-_value:float
-_position:Vec3
-m_Geode:ref_ptr<Geode>
-m_Geometry:ref_ptr<Geometry
-_hints:ref_ptr<TessellationHints
-_Transform:PositionAttitudeTra
-_TempValue:float
+GraphicsBox
+~GraphicsBox
+draw:void
+CreateInstance:GraphicsEleme
+Start:void
+CreateElement:void
+SetBoxPosition:void
+SetBoxParameter:void
#operator():void
GraphicsAxis
-m_Geode:ref_ptr<Geode>
-m_Geometry:ref_ptr<Geometry
-_hints:ref_ptr<TessellationHints
-_enabled:bool
-_interval:float
-_marker:vector<string>
-_title:string
-_textColor:Vec4
-_Aixs:int
-_AxisRotation:Quat
-_AxisRadius:float
-_length:float
+GraphicsAxis
+~GraphicsAxis
+draw:void
+CreateInstance:GraphicsEleme
+Start:void
+CreateElement:void
+SetTextColor:void
+SetAxisTitle:void
+SetAxisInterval:void
+SetAxisMarker:void
-ConvertPosition:Vec3
#operator():void
GraphicsLine
-_LineWidth:float
-_PointList:vector<Vec3>
-m_Geode:ref_ptr<Geode>
-m_Geometry:ref_ptr<Geometry
-_Count:unsigned int
+GraphicsLine
+~GraphicsLine
+draw:void
+CreateInstance:GraphicsEleme
+CreateElement:void
+SetLineWidth:void
+GetLineWidth:float
+AddNewPoint:void
#operator():void
GraphicsPolygon
-_VertexList:vector<Vec3>
-m_Geode:ref_ptr<Geode>
-m_Geometry:ref_ptr<Geometry
+GraphicsPolygon
+~GraphicsPolygon
+draw:void
+CreateInstance:GraphicsEleme
+CreateElement:void
+AddNewVertex:void
#operator():void
GraphicsCylinder
-_radius:float
-_value:float
-_TempValue:float
-_position:Vec3
-m_Geode:ref_ptr<Geode>
-m_Geometry:ref_ptr<Geometry
-_hints:ref_ptr<TessellationHints
-_outlineCount:int
+GraphicsCylinder
+~GraphicsCylinder
+draw:void
+CreateInstance:GraphicsEleme
+Start:void
+CreateElement:void
+SetCylinderPosition:void
+SetCylinderParameter:void
#operator():void
GraphicsText
-m_Geode:ref_ptr<Geode>
-m_text:ref_ptr<osgText::Text>
-_startPos:Vec3
-_endPos:Vec3
-_TextSize:float
-_TextFont:string
-_content:string
+GraphicsText
+~GraphicsText
+update:void
+draw:void
+CreateInstance:GraphicsEleme
+Start:void
+CreateElement:void
+SetTextPosition:void
+SetTextSize:void
#operator():void

Part VI Test Plan and Method

1.Plan

The test of DDRL includes following procedures. (Refer to appendix II for the test files)

1. Variable test
Create and define variable
.
2. Scope test
Retrieve, use and validate existing variable with in specific scopes, see if the scope functions well or not.

3. Arithmetic and logical operator test
arithmetic operators include: =,+,-,*,/
logical operators include: &&, ||, ==, !=, >,<, >=, <=

4. Control flow test
The test of control flow includes:
if…else, while, for, break, continue

5. Graphic Test
displaymodel define test
timetag test

6. Function Test
Check if the declaration, definition of function works well.

7. Overall Test
Run complete DDRL program.

2. Method of testing

To give an example of our testing, the way of test mathematical operator and logical operators are give as following:

Mathematical Operator Testing

The mathematical operator test consisted of applying math operators like ‘+’, ‘-‘, ‘*’, ‘/’, ‘^’. Each math operator test was applied to a set of variables testing combinations of math operators on various data types. For example, starting with the general case of adding two integers together and working to possibilities of exponent arithmetic on an integer and a float.

Note: int1 = 6;   int2 = 2;   flt1 = 6.0f;   flt2 = 2.0f;

	int1,int2	int1,flt1	flt1,flt2
+	8	12.0f	8.0f
-	4	0f	4.0f
*	12	36.0f	12.0f
/	3	1.0f	3.0f
^	36	46656.0f	36.0f

Logical Operator Testing

The logical operator test Each test consisted of applying logical operators like '==', '!=', '>', '<', '<=', '<=', '~', '&&', '||'.   Each logical operator test was applied to a set of variables testing combinations on various data types.

Note: int1 = 2, int2 = 4, flt1 = 2.0f, flt2 = 4.0f, str1 = ”hello”, str2 = ”world”

	==	!=	>	<	>=	<=	~	&&	||
int1, int2	False	True	False	True	False	True	False	True	True
int1, flt1	True	False	False	False	True	True	False	True	True
int1, str1	False	True	False	False	False	False	False	True	True
flt1, flt2	False	True	False	True	False	True	False	True	True
flt1, str1	True	True	False	False	False	False	False	True	True
str1, str2	False	True	False	False	False	False	False	True	True

In other parts of DDRL program, the same method is also used.

VII Lessons Learned

The first lesson we learned is to have a well-defined testing plan. We focused on implementing front-end and back-end's functionalities, but neglected the importance of integration testing. If we can start testing earlier, the project would have fewer bugs in the interface.

The second one is to build a small, functional model first, then include more advanced features from there. We started with a big scope and included many advanced features in the LRM, but it turned out to be larger than what our time and energy allow.

VIII Future Work

1. Improve on current back-end, making the language fully functional

By 0:00 5/8, our DDRL language is able to connect front-end and back-end and generate dynamic charts. The result is shown in part II, example 2 of this report.

2. Optimize the compiler by using by-code:

DDRL uses antler to generate a tree first and then walk the tree with back-end functions. Thus, the result of AST tree can be stored as by-code. Every time the tree is walked, the walker will just need to read the by-code rather than call parser to parse source file once again.

Appendix I: Source Code：

Basic Data Type:

1. DataType: The DataType define and implement all the data types which will be used in the system.

   There are three files in this part.

   1. DataType: define and implement the basic data type used in language, like num, string, array, struct.
   2. DataTypeFunction: used for function.
   3. DataTypeDisplayModel: this class implement the complex data type of

language, which can be considered as the combine of structure and the function, just like a C++ class.

DataType.h (contains all the define of DataType)

```
/************************************************************************/
/*     DataType:      Base Class for all data types
*
*    Yitao & Zhiyang
*/
/************************************************************************/

#ifndef PLT_DATATYPE_H
#define PLT_DATATYPE_H

#include <vector>
#include <map>
#include <string>
#include <stdlib.h>
#include <antlr/CommonAST.hpp>

using namespace std;

#define DATATYPE_NUM            1
#define DATATYPE_STRING             2
#define DATATYPE_ARRAY              3
#define DATATYPE_STRUCT             4
#define DATATYPE_DISPLAYM 5
#define DATATYPE_FUNCTION 6
#define DATATYPE_DATASET    7


typedef int      CLASSTYPE;

class DataTypeFactory;
class DataTypeNum;
class DataTypeString;
class DataTypeArray;
class DataTypeStruct;
class DataTypeDisplayModel;
class DataTypeDataset;


class DeclarationTable;
```

```
//DataType super Class define
//---------------------------------------------------------------------------------------------
class DataType
{
public:
        DataType(){ _trueFalse = 0;};
        ~DataType(){};

        /*Set the name for the variable*/
        void SetName(string name)
        {
                _DataName = name;
        }



        /*Get the name from the variable*/
        string GetName(){return _DataName;}

        /*Get the data type of the class*/
        int GetDataType(){return _DataType;}

        /*Set the data type of the class*/
        void SetDataType(CLASSTYPE type) { _DataType = type;}



        /*get the value for compare if the DataType is a Number of a string*/
        float get(){ return _trueFalse;}

        void set(float trueFalse) { _trueFalse = trueFalse; }

protected:
        float _trueFalse;
        int           _DataType;
        string    _DataName;
};
//---------------------------------------------------------------------------------------------
//end of DataType super Class Define






//the Data type class generate factory
//---------------------------------------------------------------------------------------------
```

```
class DataTypeFactory
{
private:
    /*constructor*/
    DataTypeFactory();

    /*de-constructor*/
    ~DataTypeFactory();

public:
    /*get the instance of this singleton class*/
    static DataTypeFactory* Instance();

    /*Create simple data type, includes DataTypeNum, DataTypeString, DataTypeArray*/
    inline DataTypeNum*         CreateNumber(string name);

    inline DataTypeString*   CreateString(string name);

    /*create the temp DataType for store */
    string      CreateTempName();

    DataTypeArray*     CreateArray(string name, string ElementType, size_t len);

    /*Create complex data type, includes DataTypeStruct, DataTypeDisplayModel*/
    DataTypeStruct*     CreateStruct(string name, string BaseTypeName);

    /*create the data struct for store the source data*/
    DataTypeDataset* CreateDataSet(string name);

    DataTypeDisplayModel*     CreateDisplayModel(string name, string BaseTypeName,
antlr::RefAST dmnode);

    /*generate the base data type class in use of creating other same data type class instance*/
    int    CreateBaseDataType(string BaseTypeName, CLASSTYPE type);

    /*Define the Member variable for complex data type*/
    int    AddMemberVariableNumber(string BaseTypeName, string name);

    int    AddMemberVariableString(string BaseTypeName, string name);

    int    AddMemberVariableArray(string BaseTypeName, string name, string
ElemenTypeName, size_t len);

    int    AddMemberVariableStruct(string BaseTypeName, string name, string
StructTypeName);

        int AddMemberVariableDisplayModel(string BaseTypeName, string name, string
ModelTypeName);

private:
        static DataTypeFactory*         _Instance;                              //instance of this class
        DeclarationTable*               _Declaration;                   //the declaration table

public:
        static      int                 _tempNameCounter;
};
//----------------------------------------------------------------------------------------------
//end of DataType factory





//Declaration of class should be modified to be inherited from DataType or not?
class DataTypeDataset : public DataType
{
private:
        /*constructor*/
        DataTypeDataset(string name);

public:
        /*destructor*/
        ~DataTypeDataset();

        /*pass the file name and feed the data into this class*/
        int     AssignData(string fileName);

        /*Get the data at specific column and row*/
        float GetDataValue(int x, int y);

        /*Get the name of specific column */
        string GetColumnName(int index);

        /*Get the name of specific row*/
        string GetRowName(int index);

        /*Get one column data*/
        vector<float>  GetColumnData(int column);

        /*Get one Row data*/
        vector<float>     GetRowData(int row);

        /*Get all data at*/
        vector<float>  GetDataList();


        /*Get all columns' name*/
        vector<string>GetAllColumnName();

        /*Get all rows' name*/
        vector<string>GetAllRowName();

private:

        vector<string> _ColumnList;
        vector<string> _RowList;
        vector<float> _DataList;


friend class DataTypeFactory;
};







//DataTypeNum      Class
//--------------------------------------------------------------------------------------------------
class DataTypeNum: public DataType
{

private:
        /*constructor*/
        DataTypeNum(string name);

public:
        /*De constructor*/
        ~DataTypeNum();
```

```
public:

        /*Set value for the Num type data */
        void SetNumValue(float value);

        /*Get value from the Num type data */
        float GetNumValue();

        /*Self increase*/
        void Increase();

        /*Self decrease*/
        void Decrease();

        /*Add a value to _NumValue*/
        void Add(float value);

        /*(Overloaded) Add a DataTypeNum obj to current object*/
        void Add(DataTypeNum* numObj);

        /*Multiply _NumValue by value */
        void Multiply(float value);

        /*(Overloaded) Multiply current DataTypeNum object by another object*/
        void Multiply(DataTypeNum* numObj);

private:
        float _NumValue;

        friend class DataTypeFactory;
};

//--------------------------------------------------------------------------------------------
//end of DataTypeNum class






//DataTypeString Class
//--------------------------------------------------------------------------------------------
class DataTypeString: public DataType
{
```

```
private:
        /*Private constructor */
        DataTypeString(string name);


public:
        /*De constructor*/
        ~DataTypeString();

        /*Set the value of String type data*/
        void SetStringValue(string value);

        /*Get the value of String type data*/
        string GetStringValue();

private:
        string      _StringValue;

        friend class DataTypeFactory;
};

//-------------------------------------------------------------------------------------------------
//end of DataTypeString class







//DataTypeArray class
//-------------------------------------------------------------------------------------------------
class DataTypeStruct;                     //first declare the DataTypeStruct class, cause the Array
will use it



class DataTypeArray: public DataType
{
private:
        /*constructor*/
        DataTypeArray(string name, string ElementTypeName,size_t len);
```

```
public:
	/*Destructor*/
	~DataTypeArray();

	/*Set value for the data element in the Array*/
	int SetNumberValue(unsigned int index,float num);
	int SetStringValue(unsigned int index,string str);

	/*Get value of the element from the Array */
	float GetNumberValue(unsigned int index);
	string	GetStringValue(unsigned int index);

	/*Get the element at the index position*/
	DataType* GetElement(unsigned int index);

	/*Get element type of the array element*/
	int GetElementType();

	string GetElementTypeName() { return _ElementTypeName;}

	/*Get the length of the Array*/
	size_t GetArrayLen();

private:
	/*add number of element into this array*/
	void	AddElement(unsigned int number);


public:

	string	_ElementTypeName;

private:
	//float	_FloatValue;

	vector<DataType*>_ArrayElementList;
	CLASSTYPE		_ElementType; //The type of elements in the array.
	size_t			_ArrayLen;

friend class DataTypeFactory;
};
//------------------------------------------------------------------------------------------------
//end of DataTypeArray class
```

```
//DataTypeStruct Class
//--------------------------------------------------------------------------------------------------
class DataTypeStruct : public DataType
{
private:
        /*constructor*/
        DataTypeStruct(string name);



public:

        /*method to create a same instance of this class*/
        DataTypeStruct*     Create(string name, string BaseTypeName);

        /*destructor*/
        ~DataTypeStruct();

        /*Get the BaseType Name*/
        string GetBaseTypeName();

        /*Set the BaseType Name*/
        void      SetBaseTypeName(string baseName) { _BaseTypeName = baseName;}

        int AddVariable(string name, DataType* Member);

        /*check whether member variable exist*/
        DataType*      IsVariableExist(string name);

        /*Set value for the data element in the Array*/
        int SetNumberValue(string name, float num);
        int SetStringValue(string name, string str);

        /*Get value of the element from the Array */
        float GetNumberValue(string name);
        string       GetStringValue(string name);
};

public:
        map<string,DataType*>        _MemberVariable;        //member variable list
        string                                  _BaseTypeName;          //the baseType name

friend class DataTypeFactory;
};

//--------------------------------------------------------------------------------------------
//end of DataTypeStruct class




class DDRLWalker;

// the class of the function
//--------------------------------------------------------------------------------------------
class DataTypeFunction : public DataType
{
private:
        /*constructor*/
        DataTypeFunction(string name, DDRLWalker* walker, CLASSTYPE returnType);

public:
        ~DataTypeFunction();

        /*add the store the parameters of this function call*/
        int   addFunctionParameter( CLASSTYPE type, string name);

        /*call this function*/
        int   CallFunction();

private:

        DDRLWalker*                         _walker;
        CLASSTYPE                   _returnType;

        DataTypeFactory*        _factory;               //use to create the new data type in current
symbol table as parameters

        map<string, CLASSTYPE>   _parameterList;
};
#endif
```

DataTypeNum.cpp (the data type for number)

```
/*********************************************************************/
/*
  *   DataTypeNum:          The data type in language for number type data
  *
  *   Yitao:                created
  *                         modified->04/08/2007          zhiyang
  */
/*********************************************************************/
#include "DataType.h"


/*Constructors of DataTypeNum type*/
//--------------------------------------------------------------------------------------------
DataTypeNum::DataTypeNum(string name):DataType()
{
      SetName(name);


      SetDataType(DATATYPE_NUM);

      _NumValue = 0;
      set(0);
}

DataTypeNum::~DataTypeNum()
{
}
//--------------------------------------------------------------------------------------------

/*Set the value */
void DataTypeNum::SetNumValue(float value)
{
      _NumValue=value;
      set(value);
}



/*Get the value */
float DataTypeNum::GetNumValue()
{
      return _NumValue;
}


/*Self-increase */
void DataTypeNum::Increase()
{
    _NumValue++;
}


/*Self-decrease*/
void DataTypeNum::Decrease()
{
    _NumValue++;
}


/*Addition*/
void DataTypeNum::Add(float value)
{
    _NumValue+=value;
}


void DataTypeNum::Add(DataTypeNum* numObj)
{
    _NumValue+=numObj->GetNumValue();
}


/*Multiply*/
void DataTypeNum::Multiply(float value)
{
    _NumValue*=value;
}


void DataTypeNum::Multiply(DataTypeNum* numObj)
{
    _NumValue*=numObj->GetNumValue();
}
```

DataTypeString.cpp (define the data type for string)

```
/***********************************************************************/
/*
 *      DataTypeString:         The data type for string in languange
 *
 *      Yitao & Zhiyang
 */
/***********************************************************************/
#include "DataType.h"
#include <string>

//constructor
//-----------------------------------------------------------------------------------------------
DataTypeString::DataTypeString(string name):DataType()
{
    SetName(name);

    SetDataType(DATATYPE_STRING);
}

DataTypeString::~DataTypeString()
{}
//-----------------------------------------------------------------------------------------------

//to set and get the value of this string
//first version ,may need to be updated.
//-----------------------------------------------------------------------------------------------
void DataTypeString::SetStringValue(string value)
{
    //clear the old value first
    _StringValue = _StringValue.substr(0,0);
    //here the string becomes an empty string, then append the new one to it
    _StringValue+=value;
}

string DataTypeString::GetStringValue()
{
    return _StringValue;
}
```

DataTypeStruct.cpp (for struct data type)

```
/*******************************************************************
****/
/*        DataTypeStruct:        the class for store the struct data type
 *                               which will be used in program as a data structure
 *        Zhiyang & Yitao
 */
/*******************************************************************
****/
#include "DataType.h"
#include "DeclarationTable.h"

using namespace std;

//constructor and destructor
//------------------------------------------------------------------------------------------------
DataTypeStruct::DataTypeStruct(string name):DataType()
{
    SetName(name);

    SetDataType(DATATYPE_STRUCT);
    _MemberVariable.clear();
}


DataTypeStruct::~DataTypeStruct()
{
    map<string,DataType*>::iterator index;
    for(index = _MemberVariable.begin();index != _MemberVariable.end();index++)
    {
        DataType* member = index->second;
        delete(member);
    }

    _MemberVariable.clear();
}
//------------------------------------------------------------------------------------------------


DataTypeArray* CreateArray(string name, string ElementType, size_t len)
{
    DataTypeArray*                              arr                              =
DataTypeFactory::Instance()->CreateArray(name,ElementType,len);
    return arr;
}



/*method to create a same instance of this class*/
//--------------------------------------------------------------------------------------------------
DataTypeStruct* DataTypeStruct::Create(string name, string BaseTypeName)
{
    DataTypeFactory* factory = DataTypeFactory::Instance();

    //first create a struct
    DataTypeStruct* NewStruct = new DataTypeStruct(name);
    NewStruct->SetBaseTypeName(BaseTypeName);

    //go through all the members in the list, create the same one as it
    map<string,DataType*>::iterator index;
    for(index = _MemberVariable.begin(); index != _MemberVariable.end();index++)
    {
        CLASSTYPE type = index->second->GetDataType();


        //for use, temporary
        DataTypeArray* arrayMember = NULL;
        size_t size = 0;
        DataTypeArray* ArrayVariable = NULL;            //for array

        DataTypeNum* NumberMember = NULL;
        DataTypeString* StringMember = NULL;

        string Basename;
        DataTypeStruct* StructMember = NULL;
        DataTypeStruct* baseStruct = NULL;

        switch(type)
        {

            case DATATYPE_ARRAY:
                arrayMember = ((DataTypeArray*)index->second);
                size = arrayMember->GetArrayLen();

                ArrayVariable                                                    =
factory->CreateArray(index->first,arrayMember->_ElementTypeName,size);
                NewStruct->AddVariable(index->first,ArrayVariable);
```

```
                    break;

                case DATATYPE_NUM:
                    NumberMember = factory->CreateNumber(index->first);
                    NewStruct->AddVariable(index->first,NumberMember);
                    break;

                case DATATYPE_STRING:
                    StringMember = factory->CreateString(index->first);
                    NewStruct->AddVariable(index->first,StringMember);
                    break;

                case DATATYPE_STRUCT:
                    Basename                                                =
((DataTypeStruct*)index->second)->GetBaseTypeName();

                    //get the base data type class from the declaration table
                    baseStruct                                              =
(DataTypeStruct*)DeclarationTable::Instance()->IsDeclarationExist(Basename);

                    StructMember = baseStruct->Create(index->first,Basename);
                    NewStruct->AddVariable(index->first,StructMember);
                    break;

                default:
                    break;

            }
    }

    return NewStruct;
}
//---------------------------------------------------------------------------------------------



/*Get the BaseType Name*/
string DataTypeStruct::GetBaseTypeName()
{
    return _BaseTypeName;
}


int DataTypeStruct::AddVariable(string name, DataType* Member)
```

```
{
      //check whether the name exist
      map<string,DataType*>::iterator index = _MemberVariable.find(name);
      if(index != _MemberVariable.end())
            return -1;                                    //same name , error

      _MemberVariable[name] = Member;

      return 0;
}

//-----------------------------------------------------------------------------------------------



/*check whether member variable exist*/
//-----------------------------------------------------------------------------------------------
DataType*    DataTypeStruct::IsVariableExist(string name)
{
      map<string,DataType*>::iterator index = _MemberVariable.find(name);
      if(index != _MemberVariable.end())
            return index->second;
      else
            return NULL;
}
//-----------------------------------------------------------------------------------------------



/*Set value for the data element in the Array*/
//to get the set the value of the array elements
//
//the return or the set should be the basic type, string and number,
//others should pass on to next level element data type to proceed.
//-----------------------------------------------------------------------------------------------
//before all the set and get, you should check the next level element to do that
//-----------------------------------------------------------------------------------------------
int DataTypeStruct::SetNumberValue(string name, float num)
{
      //check whether the member variable
      DataType* varaible = IsVariableExist(name);

      if(varaible != NULL && varaible->GetDataType() == DATATYPE_NUM)
      {
        DataTypeNum* number = (DataTypeNum*)varaible;
        number->SetNumValue(num);
        return 0;
    }
    else
        return -1;

}


int DataTypeStruct::SetStringValue(string name, string str)
{
    //check whether the member variable
    DataType* varaible = IsVariableExist(name);

    if(varaible != NULL && varaible->GetDataType() == DATATYPE_STRING)
    {
        DataTypeString* stringV = (DataTypeString*)varaible;
        stringV->SetStringValue(str);
        return 0;
    }
    else
        return -1;

}
//---------------------------------------------------------------------------------------------


/*Get value of the element from the Array */
//as the same as setting, you should test its element type first, only number and string type can
//be get directly
//---------------------------------------------------------------------------------------------
float DataTypeStruct::GetNumberValue(string name)
{
    //check whether the member variable
    DataType* varaible = IsVariableExist(name);

    if(varaible != NULL && varaible->GetDataType() == DATATYPE_NUM)
    {
        DataTypeNum* number = (DataTypeNum*)varaible;
        return number->GetNumValue();
    }
    else
```

```
        return 0.0f;

}


string DataTypeStruct::GetStringValue(string name)
{
    //check whether the member variable
    DataType* varaible = IsVariableExist(name);

    if(varaible != NULL && varaible->GetDataType() == DATATYPE_STRING)
    {
        DataTypeString* stringV = (DataTypeString*)varaible;
        stringV->GetStringValue();
        return 0;
    }
    else
        return "";

}
//----------------------------------------------------------------------------------------------
```

DataTypeArray.cpp (for array, can change size)

```
/*****************************************************************
****/
/*
 *  DataTypeString:       The data structure for storing the array in language
 *
 *  Yitao:                created
 *  Zhiyang:              modified 04/08/2007
 */
/*****************************************************************
****/
#include "DataType.h"
#include "DeclarationTable.h"
#include <vector>


/*Private Constructor*/
/*The dataT parameter comes from declaration table, searched by the name of it.*/
//------------------------------------------------------------------------------------------
DataTypeArray::DataTypeArray(string    name,string    ElementTypeName,    size_t
len):DataType()
{

    SetName(name);

    SetDataType(DATATYPE_ARRAY);
    _ArrayLen=0;

    /*Get the data type by type name*/
    //before this , upper level should make sure that TypeName exist


    DataType*                       dataT                          =
DeclarationTable::Instance()->IsDeclarationExist(ElementTypeName);

    /*Define the _ElementType attribute in the Array*/
    _ElementType = dataT->GetDataType();
    _ElementTypeName = ElementTypeName;

    AddElement(len);

}
```

```
DataTypeArray::~DataTypeArray()
{
	//release all the data memory created by this class
	for(size_t i=0;i<_ArrayElementList.size();i++)
		delete(_ArrayElementList[i]);

	_ArrayElementList.clear();
}
//------------------------------------------------------------------------------------------




void	DataTypeArray::AddElement(unsigned int number)
{
	DataTypeFactory*		_factory = DataTypeFactory::Instance();

	charelemntName[30];
	memset(elemntName,0,sizeof(elemntName));
	//create the instance in array, and initialize it
	DataType* instance = NULL;
	switch(_ElementType)
	{
		case DATATYPE_NUM:
			for(size_t i=0;i<number;i++)
			{
				sprintf(elemntName, "_NumElement%d",i+_ArrayLen);
				instance = _factory->CreateNumber(elemntName);
				_ArrayElementList.push_back(instance);
			}
			break;

		case DATATYPE_STRING:
			for(size_t i=0;i<number;i++)
			{
				sprintf(elemntName, "StringElement%d",i+_ArrayLen);
				instance = _factory->CreateString(elemntName);
				_ArrayElementList.push_back(instance);
			}
			break;

		case DATATYPE_STRUCT:
			for(size_t i=0;i<number;i++)
```

```
            {
                sprintf(elemntName, "StructElement%d",i+_ArrayLen);
                instance                                                                    =
_factory->CreateStruct(elemntName,_ElementTypeName);
                _ArrayElementList.push_back(instance);
            }
            break;

        case DATATYPE_DISPLAYM:

            break;

        default:
            break;

    }

    _ArrayLen = _ArrayElementList.size();          //update the len count
}





//to get the set the value of the array elements
//
//the return or the set should be the basic type, string and number,
//others should pass on to next level element data type to proceed.
//-----------------------------------------------------------------------------------------------

//before all the set and get, you should check the next level element to do that

/*Set value for the data element in the Array*/
//-----------------------------------------------------------------------------------------------
//set number type
int DataTypeArray::SetNumberValue(unsigned int index,float num)
{
    //check whether it is in scope
    if(index >= _ArrayElementList.size())
        return -1;

    DataTypeNum* element = (DataTypeNum*)(_ArrayElementList[index]);
    element->SetNumValue(num);
    return 0;
}

//set string type
int DataTypeArray::SetStringValue(unsigned int index,string str)
{
    //check whether it is in scope
    if(index >= _ArrayElementList.size())
        return -1;

    DataTypeString* element = (DataTypeString*)(_ArrayElementList[index]);
    element->SetStringValue(str);

    return 0;
}



/*Get the element at the index position*/
DataType* DataTypeArray::GetElement(unsigned int index)
{
    //check whether it is in scope
    if(index <0 || index >= _ArrayElementList.size())
    {
        int size = _ArrayElementList.size();
        int add = index - size + 1;

        AddElement(add);
    }

    return _ArrayElementList[index];
}




/*Get value of the element from the Array */
//as the same as setting, you should test its element type first, only number and string type can
//be get directly
//------------------------------------------------------------------------------------------------
float    DataTypeArray::GetNumberValue(unsigned int index)
{
    //check whether it is in scope
    if(index <0 || index >= _ArrayElementList.size())
        return 0.0f;

    DataTypeNum* element = (DataTypeNum*)(_ArrayElementList[index]);
    return element->GetNumValue();
}

string    DataTypeArray::GetStringValue(unsigned int index)
{
    //check whether it is in scope
    if(index <0 || index >= _ArrayElementList.size())
        return "";

    DataTypeString* element = (DataTypeString*)(_ArrayElementList[index]);

    return element->GetStringValue();
}
//------------------------------------------------------------------------------------------




/*Get element type of the array element*/
int DataTypeArray::GetElementType()
{
    return _ElementType;
}


/*Get the length of the Array*/
size_t DataTypeArray::GetArrayLen()
{
    return _ArrayElementList.size();
}
```

DataTypeDataSet.cpp

```
/*************************************************************************/
/*
  *   Dataset:          store the data from data file, the dataset provided
  *                             a serial method to retrieve the data and the name of
  *                             each column and row.
  *   Yitao
  */
/*************************************************************************/

#include "DataType.h"

#include <string>
#include <vector>
#include <iostream>
#include <fstream>

using namespace std;
//the construction function, which read the data from the source data file
//into the class
//-----------------------------------------------------------------------------------------------
DataTypeDataset::DataTypeDataset(string name):DataType()
{
        SetName(name);

        SetDataType(DATATYPE_DATASET);

        _ColumnList.clear();
        _DataList.clear();
        _RowList.clear();




}




/*destructor*/
DataTypeDataset::~DataTypeDataset()
{

}
```

```
//----------------------------------------------------------------------------------------------




//pass the file name and feed the data into this class
//----------------------------------------------------------------------------------------------
int   DataTypeDataset::AssignData(string fileName)
{
        ifstream istrm;
        istrm.open(fileName.c_str());
        if (istrm.is_open())
        {
                string line;
                string temp="";
                getline(istrm,line);

                /*Process the first line*/
                //Pre process string, get rid of the flags before string
                int count=0;
                for (;line[count]=='\t'; count++)
                {}//return the counts of blanks
                line.erase(0,count);

                //Pre process string, add a extra flag to the string
                if (line[line.size()]!='\t')
                {
                        line.append("\t");
                }//append a \t to the end of the whole

                //cout<<line<<endl;

                string::size_type loc;
                loc=line.find("\t",0);
                //cout<<loc<<endl;

                do
                {
                        temp=line.substr(0,loc);
                        cout<<temp<<endl;
                        _ColumnList.push_back(temp);
                        line.erase(0,++loc);
                        loc=line.find("\t",0);
                }while (loc!=line.length()&&loc!=string::npos);
                /*end of processing of the first line*/


                /* Process the rest lines*/

                while(!istrm.eof())
                {
                        count=0;
                        getline(istrm,line);
                        //Pre process string, add a extra flag to the string
                        if (line[line.size()]!='\t')
                        {
                            line.append("\t");
                        }//append a \t to the end of the whole

                        string::size_type loc;
                        loc=line.find("\t",0);
                        do
                        {   temp=line.substr(0,loc);
                        cout<<temp<<endl;
                        if (count==0)
                            _RowList.push_back(temp);
                        else
                        {
                            float data = atof(temp.c_str());
                            _DataList.push_back(data);
                        }

                        count++;
                        line.erase(0,++loc);
                        loc=line.find("\t",0);
                        }while (loc!=line.length()&&loc!=string::npos);

                }
        }
        else
        {
                cout<<"File Open Error! "<<endl;
                return -1;
        }

        return 0;
}
```

```
/*Get the data at specific column and row*/
float DataTypeDataset::GetDataValue(int column, int row)
{
	size_t index = row*_ColumnList.size() + column;
	float data = _DataList[index];

	return data;
}



/*Get the name of specific column */
string DataTypeDataset::GetColumnName(int index)
{
	string name = _ColumnList[index];

	return name;
}


/*Get the name of specific row*/
string DataTypeDataset::GetRowName(int index)
{
	string name = _RowList[index];

	return name;
}



/*Get one column data*/
vector<float> DataTypeDataset::GetColumnData(int column)
{
	vector<float>  datalist;
	datalist.clear();

	for(size_t row = 0; row < _RowList.size();row++)
	{
		size_t index = row*_ColumnList.size() + column;
		float data = _DataList[index];

		datalist.push_back(data);
	}
```

```
        return datalist;
}



/*Get one Row data*/
vector<float> DataTypeDataset::GetRowData(int row)
{
        vector<float>  datalist;
        datalist.clear();

        for(size_t column = 0; column < _ColumnList.size();column++)
        {
                size_t index = row*_ColumnList.size() + column;
                float data = _DataList[index];

                datalist.push_back(data);
        }

        return datalist;
}


/*Get all data at*/
vector<float> DataTypeDataset::GetDataList()
{
        vector<float> alldata = _DataList;

        return alldata;
}


/*Get all columns' name*/
vector<string> DataTypeDataset::GetAllColumnName()
{
        vector<string> allname = _ColumnList;

        return allname;
}
```

```
/*Get all rows' name*/
vector<string> DataTypeDataset::GetAllRowName()
{
        vector<string> allname = _ColumnList;

        return allname;
}
```

DataTypeDisplayModel.h

```
/*********************************************************************/
/*
 *        DataTypeDisplayModel:      Store the data for graphics display
 *                                          Model, and store the information and
 *                                          method to create instance for this class
 *        Zhiyang
 */
/*********************************************************************/
#ifndef PLT_DATATYPE_DISPLAYM_H
#define PLT_DATATYPE_DISPLAYM_H

#include <string>
#include "DataType.h"
#include "GraphicsElement.h"
#include "GraphicsDisplayModel.h"
#include <antlr/CommonAST.hpp>
#include <map>
#include <vector>

using namespace std;
using namespace osg;

class DataTypeDisplayModel : public DataType
{
private:
    //constructor
    DataTypeDisplayModel(string name);

    ~DataTypeDisplayModel();

public:
    //define the static method to create the instance for this class
    DataTypeDisplayModel*    Create(string name, string baseTypeName, antlr::RefAST
dmnode);


    /*Set the BaseType Name*/
    void    SetBaseTypeName(string baseName) { _BaseTypeName = baseName;}

    /*Get the BaseType Name*/
    string GetBaseTypeName() { return _BaseTypeName;}
    //the method to define and add the member variable to the class
    int AddVariable(string name, DataType* Member);

    //check whether member variable exist
    DataType*    IsVariableExist(string name);


    //Create the DispalyModel//
    void       createDisplayModel(string name, DDRLWalker* walker);

    GraphicsDisplayModel*   getDisplayModel();


    void SetDisplaySTMTNode(antlr::RefAST node) { _displayNode = node;}

    antlr::RefAST GetTreeNode() { return _displayNode;}


    //to add the position for the arg
    void AddArgName(string name, string type);


    //to set and pass the argument into model
    void SetArg(DataType* arg);



public:

    antlr::RefAST             _displayNode;

    DataTypeFactory*          _factory;         //use to create the new data type in current symbol table as parameters

    vector<string>            _argNameList;
    vector<string>            _argTypeList;
    vector<DataType*>         _parameterList;

    map<string,DataType*>         _declearList;         //member variable list
    string                                _BaseTypeName;         //the baseType name

public:
    //generate the model
    GraphicsDisplayModel*     _model;

friend class DataTypeFactory;
};



#endif
```

DataTypeFunction.h

```
/*********************************************************************/
/*
  *   DataTypeFunction:        the datatype for storing the function in the
  *                                    program, can be called and re walked the by
  *                                    the walker
  *   Zhiyang
  */
/*********************************************************************/
#ifndef   PLT_DATATYPE_FUNCTION_H
#define   PLT_DATATYPE_FUNCTION_H

#include <antlr/CommonAST.hpp>

#include "DataType.h"

//DataTypeNum      Class
//-----------------------------------------------------------------------------------------------
class DataTypeFunction: public DataType
{

private:
        /*constructor*/
        DataTypeFunction(string name, antlr::RefAST funcNode, string returnType);

public:
        /*De constructor*/
        ~DataTypeFunction();


public:

        /*add the definition of the function parameters */
        int AddFuncArgs(DataType*  para);

        /*pass the args value into function for executing*/
        int SetFuncArgsValue(DataType* para);

        /*execute the function */
        int   ExecuteFunction();

private:
	antlr::RefAST		_funcNode;

	vector<string>		_argNameList;
	vector<string>		_argTypeList;
	vector<DataType>	_passedInPara;
	string				_returnType;

	friend class DataTypeFactory;
};


#endif
```

DataTypeFunctionDisplayModel.cpp

```
/*********************************************************************
****/
/*
 *        This file contains the implementation of function and display model
 *        which is a kind of subprogram in our system. They will be create and
 *        built before using,
 *        Other programs of subprograms can call them.
 *
 *        Zhiyang
 */
/*********************************************************************
****/

#include "DataType.h"
#include "DataTypeDisplayModel.h"
#include "DeclarationTable.h"
#include "SymbolTable.h"

#include <vector>



//constructor
//----------------------------------------------------------------------------------------
DataTypeFunction::DataTypeFunction(string     name,     DDRLWalker*     walker,
CLASSTYPE returnType)
{
      DataType();
      SetName(name);

      _walker = walker;
      _returnType = returnType;

      _parameterList.clear();

      _factory = DataTypeFactory::Instance();
}


DataTypeFunction::~DataTypeFunction()
{
	_parameterList.clear();
}
//------------------------------------------------------------------------------------------------
//end of constructor




//add the store the parameters of this function call
//------------------------------------------------------------------------------------------------
int DataTypeFunction::addFunctionParameter( CLASSTYPE type, string name)
{
	//check whether there already has this name parameter
	map<string, CLASSTYPE>::iterator index = _parameterList.find(name);
	if( index != _parameterList.end())
		return -5;					//already have this name

	//we put this into function parameters list
	_parameterList[name] = type;

	return 0;
}



//call this function
//------------------------------------------------------------------------------------------------
int DataTypeFunction::CallFunction()
{

	map<string, CLASSTYPE>::iterator index;


	return 0;
}
```

```
DataTypeDisplayModel::DataTypeDisplayModel(string name):DataType()
{
    SetName(name);

    SetDataType(DATATYPE_DISPLAYM);
    _declearList.clear();

    _model = NULL;
}

DataTypeDisplayModel::~DataTypeDisplayModel()
{
    map<string,DataType*>::iterator index;
    for(index = _declearList.begin();index != _declearList.end();index++)
    {
        DataType* member = index->second;
        delete(member);
    }

    _declearList.clear();
}

//define the static method to create the instance for this class
DataTypeDisplayModel* DataTypeDisplayModel::Create(string name, string
baseTypeName, antlr::RefAST dmnode)
{
    DataTypeFactory* factory = DataTypeFactory::Instance();

    //first create a struct
    DataTypeDisplayModel* NewModel = new DataTypeDisplayModel(name);
    NewModel->SetBaseTypeName(baseTypeName);
    NewModel->_displayNode = dmnode;

    //go through all the members in the list, create the same one as it
    map<string,DataType*>::iterator index;

    for(index = _declearList.begin(); index != _declearList.end();index++)
    {
        CLASSTYPE type = index->second->GetDataType();


        //for use, temporary
        DataTypeArray* arrayMember = NULL;
        size_t size = 0;
        DataTypeArray* ArrayVariable = NULL;            //for array

        DataTypeNum* NumberMember = NULL;
        DataTypeString* StringMember = NULL;

        string Basename;
        DataTypeStruct* StructMember = NULL;
        DataTypeStruct* baseStruct = NULL;

        switch(type)
        {

        case DATATYPE_ARRAY:
            arrayMember = ((DataTypeArray*)index->second);
            size = arrayMember->GetArrayLen();

            ArrayVariable                                              =
factory->CreateArray(index->first,arrayMember->_ElementTypeName,size);
            NewModel->AddVariable(index->first,ArrayVariable);
            break;

        case DATATYPE_NUM:
            NumberMember = factory->CreateNumber(index->first);
            NewModel->AddVariable(index->first,NumberMember);
            break;

        case DATATYPE_STRING:
            StringMember = factory->CreateString(index->first);
            NewModel->AddVariable(index->first,StringMember);
            break;

        case DATATYPE_STRUCT:
            Basename = ((DataTypeStruct*)index->second)->GetBaseTypeName();

            //get the base data type class from the declaration table
            baseStruct                                                 =
(DataTypeStruct*)DeclarationTable::Instance()->IsDeclarationExist(Basename);
```

```
                StructMember = baseStruct->Create(index->first,Basename);
                NewModel->AddVariable(index->first,StructMember);
                break;

            default:
                break;

            }
        }



        //also pass the arg list into the model
        //-----------------------------------------------------------------------------------------------
        for(int nux = 0; nux < _argNameList.size(); nux++)
        {
            string name = _argNameList[nux];
            string type = _argTypeList[nux];

            NewModel->AddArgName( name,    type);
        }


        return NewModel;
    }





    //the method to define and add the member variable to the class
    int DataTypeDisplayModel::AddVariable(string name, DataType* Member)
    {
        //check whether the name exist
        map<string,DataType*>::iterator index = _declearList.find(name);
        if(index != _declearList.end())
            return -1;                              //same name , error

        _declearList[name] = Member;

        return 0;
    }
```

```
//to add the position for the arg
void DataTypeDisplayModel::AddArgName(string name, string type)
{
	_argNameList.push_back(name);
	_argTypeList.push_back(type);
}



//to set and pass the argument into model
void DataTypeDisplayModel::SetArg(DataType* arg)
{
	//we still have to check the correct type of the arg

	if(_parameterList.size() < _argNameList.size())
		_parameterList.push_back(arg);
	else
	{
		printf("the model doesn't take %d parameters..\n",_parameterList.size()+1);
		exit(-1);
	}
}





//check whether member variable exist
DataType* DataTypeDisplayModel::IsVariableExist(string name)
{
	map<string,DataType*>::iterator index = _declearList.find(name);
	if(index != _declearList.end())
		return index->second;
	else
		return NULL;
}


//Create the DispalyModel//
void DataTypeDisplayModel::createDisplayModel(string name, DDRLWalker* walker)
```

```
{



}



GraphicsDisplayModel*    DataTypeDisplayModel::getDisplayModel()
{
    return _model;
}
```

DataTypeFactory and Symbol table

DataTypeFactory.cpp

```
/*******************************************************************
****/
/*
  *   DataTypeFactory.cpp:            to generate the base class for differnet
  *                                   Datatype, then it can be store in the
  *                                   declaration table for further use to
  *                                   create the instance of class
  */
/*******************************************************************
****/

#include "DataType.h"
#include "DeclarationTable.h"
#include "DataTypeDisplayModel.h"

DataTypeFactory*     DataTypeFactory::_Instance = NULL;       //initialize           the
instance
int                  DataTypeFactory::_tempNameCounter = 0;

//constructor and destructor
DataTypeFactory::DataTypeFactory()
{
      _Declaration = DeclarationTable::Instance();

      //for basic type, "NUMBER", "STRING"
      //..

      //should defined in the declaration table
      DataTypeNum*   baseNumberType = CreateNumber("num");
      _Declaration->AddNewDeclaration("num",baseNumberType);

      DataTypeString* baseStringType = CreateString("string");
      _Declaration->AddNewDeclaration("string",baseStringType);

      DataTypeStruct* baseStructType = new DataTypeStruct("struct");
      _Declaration->AddNewDeclaration("struct",baseStructType);

      DataTypeDisplayModel*  baseModelType                =                    new
```

```
DataTypeDisplayModel("displaymodel");
        _Declaration->AddNewDeclaration("displaymodel",baseModelType);

}

DataTypeFactory::~DataTypeFactory()
{

}



//to get the singleton instance of this class
DataTypeFactory* DataTypeFactory::Instance()
{
        if(_Instance == NULL)
                _Instance = new DataTypeFactory();

        return _Instance;
}




//according to the name and data type to generate the base class,
//which can be put in the declaration table
//
//as initialization, the
//1.NUMBER:DataTypeNum
//2.STRING:DataTypeString
//3.ARRAY:DataTypeArray
//should already put in the declaration table
int DataTypeFactory::CreateBaseDataType(string BaseTypeName, CLASSTYPE type)
{
        DataType*   dataT = NULL;
        int val = 0;

        switch(type)
        {
                case DATATYPE_STRUCT:
                        dataT = CreateStruct(BaseTypeName, "struct");
                        if( dataT == NULL)
                                return -4;
```

```
            val = _Declaration->AddNewDeclaration(BaseTypeName, dataT);
            if(val != 0)
                return val;

            break;

        //-----------------------------------------------------------------
        case DATATYPE_DISPLAYM:
            dataT = CreateDisplayModel(BaseTypeName, "displaymodel", NULL);
            if( dataT == NULL)
                return -4;

            val = _Declaration->AddNewDeclaration(BaseTypeName, dataT);
            if(val != 0)
                return val;

            break;


        //-----------------------------------------------------------------
        case DATATYPE_FUNCTION:
            break;

        default:
            break;
    }

    return 0;
}



/*Create simple data type, includes DataTypeNum, DataTypeString, DataTypeArray*/
//-------------------------------------------------------------------------------------------
DataTypeNum* DataTypeFactory::CreateNumber(string name)
{
    DataTypeNum* num=new DataTypeNum(name);
    return num;
}


DataTypeString* DataTypeFactory::CreateString(string name)
{
        DataTypeString* str=new DataTypeString(name);
        return str;

}




/*Create complex data type, includes DataTypeStruct, DataTypeDisplayModel*/
//-----------------------------------------------------------------------------------------------
DataTypeStruct* DataTypeFactory::CreateStruct(string name, string BaseTypeName)
{

        //Get the base type from the declaration table
        DataType* dataT =
DeclarationTable::Instance()->IsDeclarationExist(BaseTypeName);

        if(dataT == NULL || dataT->GetDataType() != DATATYPE_STRUCT)
        {
                printf("Type mismatch:  The base type %s is not a struct type\n",
BaseTypeName);
                return NULL;
        }

        //
        DataTypeStruct* NewStruct = ((DataTypeStruct*)dataT)->Create(name,
BaseTypeName);

        return NewStruct;


}


DataTypeArray* DataTypeFactory::CreateArray(string name, string ElementType,
size_t len)
{

        DataTypeArray* arrayObj= new DataTypeArray(name, ElementType, len);
        return arrayObj;

}
//-----------------------------------------------------------------------------------------------
```

```
/*create the data struct for store the source data*/
DataTypeDataset* DataTypeFactory::CreateDataSet(string name)
{
    DataTypeDataset* dataset = new DataTypeDataset(name);
    return dataset;
}



DataTypeDisplayModel* DataTypeFactory::CreateDisplayModel(string name, string
BaseTypeName, antlr::RefAST dmmode)
{


    //Get the base type from the declaration table
    DataType* dataT =
DeclarationTable::Instance()->IsDeclarationExist(BaseTypeName);

    if(dataT == NULL || dataT->GetDataType() != DATATYPE_DISPLAYM)
    {
        printf("Type mismatch:  The base type %s is not a display type\n",
BaseTypeName);
        return NULL;
    }

    //
    DataTypeDisplayModel* NewModel =
((DataTypeDisplayModel*)dataT)->Create(name, BaseTypeName, dmmode);

    return NewModel;
}






/*create the temp DataType for store */
string DataTypeFactory::CreateTempName()
{
    charprefix[30];
    memset(prefix, 0, sizeof(prefix));
    sprintf(prefix, "__tempValue%d", _tempNameCounter);
    string tempName(prefix);        //generate a temp name for data

    _tempNameCounter++;
    return tempName;
}




/*Define the Member variable for complex data type*/
int DataTypeFactory::AddMemberVariableNumber(string BaseTypeName, string name)
{
    //find the data type name in declaration table
    DataType* index = _Declaration->IsDeclarationExist(BaseTypeName);

    if(index == NULL)   //not found
        return -1;

    //if this is a struct
    if(index->GetDataType() == DATATYPE_STRUCT)
    {
        DataTypeNum* number = CreateNumber(name);
        ((DataTypeStruct*)index)->AddVariable(name,number);
        return 0;
    }

    //if this is a display model

    if(index->GetDataType() == DATATYPE_DISPLAYM)
    {
        //DataTypeNum* number = CreateNumber(name);
        //((DataTypeStruct*)index)->AddVariable(name,number);
        return 0;
    }

    //wrong type
    return -1;
}
```

```
int DataTypeFactory::AddMemberVariableString(string BaseTypeName, string name)
{
    //find the data type name in declaration table
    DataType* index = _Declaration->IsDeclarationExist(BaseTypeName);

    if(index == NULL)    //not found
        return -1;

    //if this is a struct
    if(index->GetDataType() == DATATYPE_STRUCT)
    {
        DataTypeString* str = CreateString(name);
        ((DataTypeStruct*)index)->AddVariable(name,str);
        return 0;
    }

    //if this is a display model

    if(index->GetDataType() == DATATYPE_DISPLAYM)
    {
        //DataTypeString* str = CreateString(name);
        //((DataTypeStruct*)index)->AddVariable(name,str);
        return 0;
    }

    //wrong type
    return -1;
}
```

```
int DataTypeFactory::AddMemberVariableArray(string BaseTypeName, string name,
string ElemenTypeName, size_t len)
{
    //find the data type name in declaration table
    DataType* index = _Declaration->IsDeclarationExist(BaseTypeName);

    if(index == NULL)    //not found
        return -1;
    //if this is a struct
    if(index->GetDataType() == DATATYPE_STRUCT)
    {
        DataTypeArray* ary = CreateArray(name,ElemenTypeName,len);
        ((DataTypeStruct*)index)->AddVariable(name,ary);
        return 0;
    }

    //if this is a display model

    if(index->GetDataType() == DATATYPE_DISPLAYM)
    {
        //DataTypeArray* ary = CreateArray(name,ElemenTypeName,len);
        //((DataTypeStruct*)index)->AddVariable(name,ary);
        return 0;
    }

    //wrong type
    return -1;
}




int DataTypeFactory::AddMemberVariableStruct(string BaseTypeName, string name, string StructTypeName)
{
    //find the data type name in declaration table
    DataType* index = _Declaration->IsDeclarationExist(BaseTypeName);

    if(index == NULL)    //not found
        return -1;

    //if this is a struct
    if(index->GetDataType() == DATATYPE_STRUCT)
    {
        DataTypeStruct* dataStruct = CreateStruct(name,StructTypeName);
        ((DataTypeStruct*)index)->AddVariable(name,dataStruct);
        return 0;
    }

    //if this is a display model
    if(index->GetDataType() == DATATYPE_DISPLAYM)
    {
        //DataTypeStruct* dataStruct = CreateStruct(name,StructTypeName);
        //((DataTypeStruct*)index)->AddVariable(name,ary);
        return 0;
    }

    //wrong type
    return -1;
}

int DataTypeFactory::AddMemberVariableDisplayModel(string BaseTypeName,
string name, string ModelTypeName)
{
    return 0;
}
```

DeclarationTable.h

```
/**********************************************************************
****/
/*
  *        DeclarationTable:           The table stores all the declared struct
  *                                    display models and function, which will be
  *                                    used in the following program.
  *        Zhiyang
  */
/**********************************************************************
****/
#ifndef        PLT_DECLARATION_TABLE_H
#define        PLT_DECLARATION_TABLE_H

#include <map>
#include <string>

#include "plt.h"
#include "DataType.h"
#include "plt.h"

using namespace std;

typedef map<string,DataType*>          DeclarationList;

class DeclarationTable
{
private:
       /*constructor*/
       DeclarationTable();
       ~DeclarationTable();

public:
       /*get the singleton instance*/
       static DeclarationTable* Instance();

       /*Add new declaration*/
       int         AddNewDeclaration(string name, DataType* decl);

       /*Check whether the declaration exists*/
       DataType*   IsDeclarationExist(string name);
```

```
    /*Clear the declaration table*/
    void        ClearTable();

private:

    static DeclarationTable*        _Instance;                      //the table instance
    DeclarationList                         _DeclList;                      //the declaration list.

};

#endif
```

DeclarationTable.cpp

```
/*****************************************************************
****/
/*
 *      DeclarationTable:     The base type and defined data type are stored
 *                            here for using;
 *                            This is singleton class, you can access it from
 *                            Anywhere.
 */
/*****************************************************************
****/
#include "DeclarationTable.h"

DeclarationTable*        DeclarationTable::_Instance = NULL;

/*constructor*/
//-----------------------------------------------------------------------------------------------
DeclarationTable::DeclarationTable()
{
    _DeclList.clear();
}


DeclarationTable::~DeclarationTable()
{
    DeclarationList::iterator   index;
    for(index = _DeclList.begin();index != _DeclList.end();index++)
    {
        DataType* declear = index->second;
        delete(declear);
    }

    _DeclList.clear();
}


/*get the singleton instance*/
//-----------------------------------------------------------------------------------------------
DeclarationTable* DeclarationTable::Instance()
{
    if(_Instance == NULL)
        _Instance = new DeclarationTable();
	return _Instance;
}

/*Add new declaration*/
int DeclarationTable::AddNewDeclaration(string name, DataType* decl)
{
	//check whether there is a same name one
	DeclarationList::iterator index = _DeclList.find(name);
	if(index != _DeclList.end())
		return -3;

	_DeclList[name] = decl;

	return 0;
}

/*Check whether the declaration exists*/
//----------------------------------------------------------------------------------------------
DataType* DeclarationTable::IsDeclarationExist(string name)
{
	//check whether there is a same name one
	DeclarationList::iterator index = _DeclList.find(name);
	if(index != _DeclList.end())
		return index->second;
	else
		return NULL;
}


/*Clear the declaration table*/
void DeclarationTable::ClearTable()
{
	DeclarationList::iterator   index;
	for(index = _DeclList.begin();index != _DeclList.end();index++)
	{
		DataType* declear = index->second;
		delete(declear);
	}

	_DeclList.clear();
}
```

SymbolTable.h

```
/*******************************************************************
****/
/*
 *   SymbolTalbe:          The table to store the symbol and variable defined
 *                         in programe, one table is a scope.
 *                         The SymbolTables are managed by VariableList class
 *   Zhiyang
 */
/*******************************************************************
****/
#ifndef PLT_SYMBOLTABLE_H
#define  PLT_SYMBOLTABLE_H

#include <list>
#include <vector>
#include <map>
#include <string>
#include <osg/ref_ptr>
#include "plt.h"

#include "plt.h"
#include "DataType.h"


using namespace std;
using namespace osg;

typedef DataType*                    Variable;     //we define the variable class type
typedef  map<string,Variable>SymbolMap;     //define  the  data  type  which  will  be
put in symbol table



class SymbolTable
{
public:
	/*constructor*/
	SymbolTable(int typeFlag, int ID);
	~SymbolTable();

	/*add and remove the symbols from symbol table*/
```

```
    int   AddVariable(string name, Variable Var);

    /*Check whether there is one symbol*/
    Variable IsVariableExist(string name);

    /*Check whether the variable Type matches*/
    bool     IsTypeMatch(Variable var, int Type);

    /*Check whether the struct or display model match*/
    bool     IsTypeNameMatch(Variable var, char* typeName);

    /*Check whether this symbol can trace up level symbol table*/
    bool     IsTraceUp(){ return _TraceUp;}

    /*Set the table Trace Up property*/
    void     SetTraceUp(bool traceup){ _TraceUp = traceup;}

    /*add next level symbol table*/
    void     LinkNextLevelTable(SymbolTable* Next);

    /*Link previous level symbol table*/
    void     LinkPreviousLevelTable(SymbolTable* Previous);

    /*unLink Next Level symbol table*/
    void     UnLinkNextLevelTable() { _Next = NULL; }

    /*Get the symbol table of next or previous symbol table*/
    SymbolTable*     GetNextLevelTable() { return _Next;}
    SymbolTable*     GetPreviousLevelTable() { return _Previous;}

    /*Clear the symbol table and free the memory space*/
    void     ClearSymbolTalbe();

    /*get the type of scope, is it created by function or subprogram, or */
    int      GetScopeType() { return _typeFlag;}

private:

    SymbolTable*     _Next;             //next level symbol table
    SymbolTable*     _Previous;         //previous level symbol table

    bool             _TraceUp;          //whether this table can trace up
    int              _typeFlag;         //created by which kind of statement.
    int              _ID;               //the id of this symbol table
```

```
public:
    SymbolMap       _Variables;             //all variables in this table
};









class VariableList
{
private:
    /*constructor*/
    VariableList();
    ~VariableList();

public:
    /*get the singleton instance of this list*/
    static VariableList* Instance();

    /*Check whether there is the symbol with the name that can be used as variable*/
    Variable IsVariableExist(string name);

    /*Create and delete the symbol table*/
    void    AddNewSymbolTable(bool traceup, int typeFlag);
    void    DeleteSymbolTable();

    /*Add variable to current symbol table, we don't have to delete because it will
delete with table*/
    int  AddNewVariable(string name, Variable var);

    void RemoveVariable(string name);

    /*Clear the whole Variable list*/
    void    ClearVariableList();


    /*get the current symbol table type flag*/
    int     GetCurrentTableType() { return _Current->GetScopeType();}

private:
    static VariableList*        _Instance;                  //singleton instance

    SymbolTable*                    _TableList;             //the list of symbol table
    SymbolTable*                    _Current;               //the current scope table

    static    int                   _IDCounter;


};

#endif
```

SymbolTable.cpp

```
/*****************************************************************
****/
/*
*   SymbolTalbe:            The table to store the symbol and variable defined
*                           in programe, one table is a scope.
*                           The SymbolTables are managed by VariableList class
*                       Implementation
*   Zhiyang
*/
/*****************************************************************
****/

#include "SymbolTable.h"


//constructor
//------------------------------------------------------------------------------------------------
SymbolTable::SymbolTable(int typeFlag, int ID)
{
	_Next = NULL;			//next level symbol table
	_Previous = NULL;		//previous level symbol table

	_TraceUp = true;		//whether this table can trace up
	_typeFlag = typeFlag;
	_ID = ID;

	_Variables.clear();		//all variables in this table
}

SymbolTable::~SymbolTable()
{

}
//------------------------------------------------------------------------------------------------


//add and remove the symbols from symbol table
//the symbol name should not longer than NAMELENGTH characters
//
// @return 0:	successful
// @return -1:		variable already in the table
//------------------------------------------------------------------------------------------------
int SymbolTable::AddVariable(string name, Variable Var)
```

```
{
      SymbolMap::iterator index = _Variables.find(name);
      if(index != _Variables.end())
            return    -1;                              //the variable already exist

      //otherwise , put this variable into map
      _Variables[name] = Var;

      return 0;
}
//----------------------------------------------------------------------------------------------



//Check whether there is one symbol
//if exist return the pointer to the variable,
//otherwise return NULL
//----------------------------------------------------------------------------------------------
Variable SymbolTable::IsVariableExist(string name)
{
      Variable Var = NULL;

      SymbolMap::iterator index = _Variables.find(name);
      if(index != _Variables.end())    //the variable exist
      {
            Var = _Variables[name];
      }

      return Var;
}
//----------------------------------------------------------------------------------------------



//Check whether the variable Type matches
//----------------------------------------------------------------------------------------------
bool SymbolTable::IsTypeMatch(Variable var, int Type)
{


      return true;
}
//----------------------------------------------------------------------------------------------
```

```
//Check whether the struct or display model match
//-----------------------------------------------------------------------------------------------
bool SymbolTable::IsTypeNameMatch(Variable var, char* typeName)
{
    return true;
}
//-----------------------------------------------------------------------------------------------




//add next level symbol table
//-----------------------------------------------------------------------------------------------
void SymbolTable::LinkNextLevelTable(SymbolTable* Next)
{
    _Next = Next;
    Next->LinkPreviousLevelTable(this);
}
//-----------------------------------------------------------------------------------------------



//Link previous level symbol table
//-----------------------------------------------------------------------------------------------
void SymbolTable::LinkPreviousLevelTable(SymbolTable* Previous)
{
    _Previous = Previous;
}
//-----------------------------------------------------------------------------------------------


//Clear the symbol table and free the memory space
//-----------------------------------------------------------------------------------------------
void SymbolTable::ClearSymbolTalbe()
{
    //go through all the key in map, and clear the value, free the memory
    SymbolMap::iterator index;
    for(index = _Variables.begin();index != _Variables.end();index++)
    {
        Variable var = index->second;
        if(var != NULL)
```

```
                delete(var);
        }

        _Variables.clear();

        DataTypeFactory::_tempNameCounter = 0;                  //clear   the   temp   name
counter.
}
//--------------------------------------------------------------------------------------------------------------------------------




///////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////

//following is the implementation of class VariableList

VariableList* VariableList::_Instance = NULL;                   //singleton instance
int   VariableList::_IDCounter = 0;

//constructor
//--------------------------------------------------------------------------------------------------------------------------------
VariableList::VariableList()
{
        _TableList = NULL;                      //the list of symbol table
        _Current = NULL;                        //the current scope table

        AddNewSymbolTable(false, -1);
}


VariableList::~VariableList()
{

}
//--------------------------------------------------------------------------------------------------------------------------------


//get the singleton instance of this list
//--------------------------------------------------------------------------------------------------------------------------------
VariableList* VariableList::Instance()
{
	if(_Instance == NULL)
		_Instance = new VariableList();

	return _Instance;
}
//----------------------------------------------------------------------------------------------




//Check whether there is the symbol with the name that can be used as variable
// we will go trace every level table that can be traced
//----------------------------------------------------------------------------------------------
Variable VariableList::IsVariableExist(string name)
{
	SymbolTable*	index;
	index = _Current;
	Variable var = NULL;
	while (index != NULL)
	{
		var = index->IsVariableExist(name);

		if(var != NULL)			//we find the match
			return var;

		//otherwise we trace up if upper level traceable
		if(index->IsTraceUp())
			index = index->GetPreviousLevelTable();
		else
			break;
	}

	return var;
}
//----------------------------------------------------------------------------------------------



//Create and delete the symbol table
//traceup indicate whether this table can trace upper level table
//----------------------------------------------------------------------------------------------
void VariableList::AddNewSymbolTable(bool traceup , int typeFlag)
{
	SymbolTable*	newTable = new SymbolTable(typeFlag, _IDCounter);
```

```
    _IDCounter++;

    newTable->SetTraceUp(traceup);

    //link the header
    if(_TableList == NULL)
    {
        _TableList = newTable;
        _Current = newTable;
    }
    else
    {
        _Current->LinkNextLevelTable(newTable);
        _Current = newTable;
    }

}

void VariableList::DeleteSymbolTable()
{
    SymbolTable* index = _Current;
    _Current = index->GetPreviousLevelTable();          //move    current    pointer    to
upper level

    if( _Current != NULL)
        _Current->UnLinkNextLevelTable();                    //unlink

    index->ClearSymbolTalbe();
    delete(index);
    _IDCounter--;
}
//--------------------------------------------------------------------------------------------




//Add variable to current symbol table, we don't have to delete because it will delete
with table
//--------------------------------------------------------------------------------------------
int VariableList::AddNewVariable(string name, Variable var)
{
    int reval = _Current->AddVariable(name,var);

    return reval;
```

```
}
//-----------------------------------------------------------------------------------------------



//Clear the whole Variable list
//-----------------------------------------------------------------------------------------------
void VariableList::ClearVariableList()
{
	//the last level symbol table is at the tail of the list, and the current is the last
	while(_Current != NULL)
	{
		SymbolTable* index = _Current;
		_Current = index->GetPreviousLevelTable();

		index->ClearSymbolTalbe();
	}
}
//-----------------------------------------------------------------------------------------------




void VariableList::RemoveVariable(string name)
{
	map<string, DataType*>::iterator index = _Current->_Variables.find(name);
	if( index != _Current->_Variables.end())
		_Current->_Variables[name]=0;

}
```

Graphics Backend

GraphicsElement.h   (graphics super class)

```
/*****************************************************************
****/
/*   GraphicsElement:    The basic class for the graphics element, define
  *                          basic member and function
  *                          This class form a node in OSG tree
  *    Zhiyang
  */
/*****************************************************************
****/
#ifndef PLT_ELEMENT_H
#define PLT_ELEMENT_H

#include <osg/Node>
#include <osg/Geode>
#include <osg/Geometry>
#include <osg/Drawable>
#include <osg/Group>
#include <osg/Vec3>
#include <osg/Array>
#include <osg/LineWidth>
#include <osg/PositionAttitudeTransform>


#include "VRStimer.h"

#define DEG2RAD (PI / 180.0f)
#define RAD2DEG (180.0f / PI)

#define ANIMATION_STOP        0                    //play the animation only once
#define  ANIMATION_LOOP          1                    //loop  the  animation  when
finished
#define ANIMATION_CLEAR          2                    //play  the  animation  then
remove the model


//default element animation time
#define ANIMATION_POINT_TIME  1.0
#define ANIMATION_LINE_TIME         1.0
```

```
#define ANIMATION_BOX_TIME          2.0
#define ANIMATION_CYLINDER_TIME 2.0
#define ANIMATION_SLICE_TIME   2.0
#define ANIMATION_PLANE_TIME 3.0
#define ANIMATION_AXIS_TIME         1.0
#define ANIMATION_AXIS_MARKER    0.0


//define the model element type
#define ELEMENT_POINT               1
#define ELEMENT_LINE                2
#define ELEMENT_POLYGON                 3
#define ELEMENT_CYLINDER        4
#define ELEMENT_BOX                     5
#define ELEMENT_SLICE               6
#define ELEMENT_PLANE               7
#define ELEMENT_AXIS                8
#define ELEMENT_AXISMARKER          9
#define ELEMENT_TEXT                10
//-----------------------------------------------------------------------------------

using namespace osg;

class GraphicsElement: public NodeCallback,public Group
{
public:
    GraphicsElement();
    ~GraphicsElement();

    /*the callback function, for animation, implement this function*/
    virtual void update();

    /*the draw function to display the element*/
    virtual void draw() = 0;

    /*the virtual function for child class to implement, which create an instance of
this model class*/
    virtual GraphicsElement* CreateInstance()=0;

    /*Set and get the start time of the animation*/
    void            SetStartEndTime(float start, float end)
    {
        if(start > end)
        {
            _StartTime = end;
            _EndTime = end;
        }
        else
        {
            _StartTime = start;
            _EndTime = end;
        }
    }

    float           GetStartTime() { return _StartTime;}


    float           GetEndTime() { return _EndTime;}

    /*control the play*/
    virtual   void           Start();


    /*Set the animation flag*/
    void            SetAnimationFlag(int flag) { _AnimationFlag = flag;}

    /*generate and form the graphics node for    display*/
    virtual   void CreateElement()=0;


    /*Set and Get the color of this element*/
    void            SetColor(Vec4 color) { _Color = color;}
    Vec4            GetColor() { return _Color;}


protected:
    //implement the parent NodeCallback virtual function, to set the callback operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

protected:

     float          _StartTime;                    //start time for animation
     float          _EndTime;                      //end time for animation
```

```
	  float		_Duration;				//duration of the animation

	float		_PlayStartTime;		//actual start playing time
	float		_PlayEndTime;		//actual end playing time
	float		_CurrentTime;		//actual current playing time

	int		_AnimationFlag;			//the flag for animation
	bool		_Play;				//the flag for playing the animation

	Vec4		_Color;				//the color of this element

	VRStimer*	_timer;				//timer.

	bool		_isDone;			//finished animation display;
	bool		_isStarted;

public:
	//------------------------------------------------------------------------------
	int		_ElementType;

	float		_defaultLenght;			//for animation

};


#endif
```

GraphicsElement.cpp

```
/*********************************************************************
****/
/*    GraphicsElement:    The basic class for the graphics element, define
*                              basic member and function
*                              This class form a node in OSG tree
*    Zhiyang
*/
/*********************************************************************
****/
#include "GraphicsElement.h"

//constructor
//------------------------------------------------------------------------------------------------
GraphicsElement::GraphicsElement()
{
       _StartTime = 0.0;
       _EndTime = 0.0;                  //this will cause the element play directly

       _PlayStartTime = 0.0;                   //actual start playing time
       _PlayEndTime = 0.0;              //actual end playing time
       _CurrentTime = 0.0;              //actual current playing time

       _AnimationFlag = ANIMATION_STOP;           //by default, it play then stop
       _Play = false;                                        //not playing

       _Color = Vec4(1.0,1.0,1.0,1.0);                //by default the color is white

       _timer = VRStimer::Instance();                 //timer.

       _isDone = false;
       _isStarted = false;
};



GraphicsElement::~GraphicsElement()
{

}
//------------------------------------------------------------------------------------------------
```

```
void GraphicsElement::update()
{
    {
        if(_Play == true)
        {
            //update the timer
            _CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

            if(_CurrentTime > _Duration)        //finished play the animation single loop
            {
                //update the animation according to the flag
                if(_AnimationFlag == ANIMATION_LOOP)        //loop display
                {
                    //reset the current time, for loop back to beginning
                    _PlayStartTime = _timer->GetGraphicTime();
                    _PlayEndTime = _PlayStartTime + _Duration;
                    _CurrentTime = _CurrentTime - _Duration;

                    //we will loop back to beginning again.
                }

                if(_AnimationFlag == ANIMATION_CLEAR)        //remove the display, set the mode clear
                {
                    this->setNodeMask(0x00000000);
                    return;
                }

                if(_AnimationFlag == ANIMATION_STOP)        //stop playing the animation
                {
                    _CurrentTime = _Duration;
                }
            }

            draw();        //display
        }
    }
}



void GraphicsElement::Start()
{
	if(_Play == true)
		return;

	_Play = true;
	_PlayStartTime = _timer->GetGraphicTime();
	_PlayEndTime = _PlayStartTime + _Duration;

	_isDone = false;
}
```

GraphicsObject.h (define all shapes)

```
/*******************************************************************
****/
/*
 *	GraphicsObject:	This file defines all the graphics objects which
 *			can be displayed as an element in display model.
 *			includes Polygon, Box, Cylinder, Slice, Axis
 *			and Axis marker
 *
 *	Zhiyang
 */
/*******************************************************************
****/
#ifndef PLT_GRAPHICS_OBJECT
#define PLT_GRAPHICS_OBJECT

#include "plt.h"
#include "GraphicsElement.h"

#include <osg/Quat>
#include <osgText/String>
#include <osgText/Text>


//Graphics Polygon class
//----------------------------------------------------------------------------------------------
class GraphicsPolygon : public GraphicsElement
{
public:
	GraphicsPolygon();
	~GraphicsPolygon();

	/*the callback function, for animation, implement this function*/
	//void update();

	/*the draw function to display the element*/
	void draw();

	/*the virtual function for child class to implement, which create an instance of
this model class*/
	GraphicsElement*	CreateInstance();
```

```
         /*start the animation*/
        //void Start();


        /*generate and form the graphics node for    display*/
        void CreateElement();


        /*add point to the list*/
        void       AddNewVertex(Vec3 point);

protected:
        //implement the parent NodeCallback virtual function, to set the callback
operation
        virtual void operator()(Node* node, NodeVisitor* nv)
        {
                this->update();
                NodeCallback::traverse(node,nv);
        }

private:

        vector<Vec3>                  _VertexList;                  //the list of point to display
lines

        ref_ptr<Geode>              m_Geode;                //the node of the osg
        ref_ptr<Geometry>         m_Geometry;           //the geometry for points


};
//-----------------------------------------------------------------------------------------------
//end of Polygon





//Graphics Box class
//-----------------------------------------------------------------------------------------------
class GraphicsBox : public GraphicsElement
{
public:
        GraphicsBox();
        ~GraphicsBox();
```

```
    /*the callback function, for animation, implement this function*/
    //void update();

    /*the draw function to display the element*/
    void draw();

    /*the virtual function for child class to implement, which create an instance of this model class*/
    GraphicsElement*    CreateInstance();

    /*start the animation*/
    void Start();

    /*generate and form the graphics node for    display*/
    void CreateElement();

    /*Set box position*/
    void SetBoxPosition(Vec3 pos);

    /*Set the Value of box*/
    void SetBoxParameter(float width, float length, float value);

protected:
    //implement the parent NodeCallback virtual function, to set the callback operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

private:

    float               _width;
    float               _length;
    float               _value;

    Vec3                _position;

    ref_ptr<Geode>          m_Geode;         //the node of the osg
    ref_ptr<Geometry>       m_Geometry;      //the geometry for points
    ref_ptr<TessellationHints> _hints;
    PositionAttitudeTransform*    _Transform; //transform to control the position of this slice

    float                         _TempValue;                 //used to draw animation


};
//---------------------------------------------------------------------------------------------
//end of Box




//Graphics Cylinder class
//---------------------------------------------------------------------------------------------
class GraphicsCylinder : public GraphicsElement
{
public:
    GraphicsCylinder();
    ~GraphicsCylinder();

    /*the callback function, for animation, implement this function*/
    //void update();

    /*the draw function to display the element*/
    void draw();

    /*the virtual function for child class to implement, which create an instance of this model class*/
    GraphicsElement*     CreateInstance();

    /*start the animation*/
    void Start();

    /*generate and form the graphics node for    display*/
    void CreateElement();

    /*Set box position*/
    void SetCylinderPosition(Vec3 pos);

    /*Set the Value of box*/
    void SetCylinderParameter(float radius, float value);
```

```
protected:
    //implement the parent NodeCallback virtual function, to set the callback operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

private:

    float                    _radius;
    float                    _value;
    float                    _TempValue;

    Vec3                     _position;

    ref_ptr<Geode>               m_Geode;            //the node of the osg
    ref_ptr<Geometry>            m_Geometry;         //the geometry for points
    ref_ptr<TessellationHints> _hints;

    int                      _outlineCount;      //the outline segment count


};
//--------------------------------------------------------------------------------------------
//end of cylinder



//Graphics Slice class
//--------------------------------------------------------------------------------------------
class GraphicsSlice : public GraphicsElement
{
public:
    GraphicsSlice();
    ~GraphicsSlice();

    /*the callback function, for animation, implement this function*/
    //void update();

    /*the draw function to display the element*/
    void draw();
    /*the virtual function for child class to implement, which create an instance of
this model class*/
    GraphicsElement*    CreateInstance();

    /*start the animation*/
    void Start();

    /*generate and form the graphics node for   display*/
    void CreateElement();

    /*Set box position*/
    void SetSlicePosition(Vec3 pos);

    /*Set the Value of slice*/
    void SetSliceParameter(float radius, float startAngle, float angle, float height);

protected:
    //implement the parent NodeCallback virtual function, to set the callback
operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

private:

    float               _radius;
    float               _startAngle;
    float               _angle;
    float               _height;

    Vec3                _position;

    int                 _animationCounter;  //for animation control
    int                 _SideCount;


    ref_ptr<Geode>          m_Geode;            //the node of the osg
    ref_ptr<Geometry>       m_Geometry;         //the geometry for points
    ref_ptr<Geometry>       side_geom;

    unsigned short*         quadIndex;
    PositionAttitudeTransform*    _Transform; //transform to control the position of
this slice


};
//----------------------------------------------------------------------------------------------
//end of slice




//Graphics Plane class
//----------------------------------------------------------------------------------------------
class GraphicsPlane : public GraphicsElement
{
public:
    GraphicsPlane();
    ~GraphicsPlane();

    /*the callback function, for animation, implement this function*/
    //void update();

    /*the draw function to display the element*/
    void draw();

    /*the virtual function for child class to implement, which create an instance of
this model class*/
    GraphicsElement*     CreateInstance();

    /*start the animation*/
    void Start();


    /*generate and form the graphics node for    display*/
    void CreateElement();


    /*add point to the list*/
    void       AddNewVertex(Vec3 point);

protected:
    //implement the parent NodeCallback virtual function, to set the callback
operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

private:

    vector<Vec3f>           _VertexList;           //the list of point to display
lines

    ref_ptr<Geode>          m_Geode;             //the node of the osg
    ref_ptr<Geometry>       m_Geometry;          //the geometry for points



    float                   _PlaneLength;

};
//---------------------------------------------------------------------------------------------
//end of Plane



#define   X_AXIS          0
#define   Y_AXIS          1
#define   Z_AXIS          2

//Graphics Axis class
//---------------------------------------------------------------------------------------------
class GraphicsAxis : public GraphicsElement
{
public:
    GraphicsAxis(int axis);
    ~GraphicsAxis();

    /*the callback function, for animation, implement this function*/
    //void update();

    /*the draw function to display the element*/
    void draw();

    /*the virtual function for child class to implement, which create an instance of
this model class*/
    GraphicsElement*    CreateInstance();

    /*start the animation*/
    void Start();

    /*generate and form the graphics node for    display*/
    void CreateElement();


    /*set the text color*/
    void SetTextColor(Vec4 color) { _textColor = color;}

    /*set the title for this axis*/
    void SetAxisTitle(char* title);

    /*set the interval for the marker on this axis*/
    void SetAxisInterval(float interval);

    /*set the marker by the index*/
    void SetAxisMarker(int index, string mark);

private:
    inline Vec3 ConvertPosition(float pos);

protected:
    //implement the parent NodeCallback virtual function, to set the callback
operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

private:

    ref_ptr<Geode>            m_Geode;        //the node of the osg, each node for
one axis
    ref_ptr<Geometry>         m_Geometry;     //the geometry for displaying the axis
    ref_ptr<TessellationHints> _hints;

    bool                      _enabled;       //the enable flag for each axis
    float                     _interval;   //the interval of the marker on each axis

    vector<string>            _marker;     //the marker on each axis
    string                    _title;
    Vec4                       _textColor; //the color for text
    int                 _Aixs;
    Quat                _AxisRotation;          //to align the axis to desired
orientation
    float               _AxisRadius;

    float               _length;
};
//-----------------------------------------------------------------------------------------------
//end of Axis








//Graphics Text class
//-----------------------------------------------------------------------------------------------
class GraphicsText : public GraphicsElement
{
public:
    GraphicsText(char* text);
    ~GraphicsText();

    /*the callback function, for animation, implement this function*/
    void update();

    /*the draw function to display the element*/
    void draw();

    /*the virtual function for child class to implement, which create an instance of
this model class*/
    GraphicsElement*    CreateInstance();

    /*start the animation*/
    void Start();

    /*generate and form the graphics node for   display*/
    void CreateElement();


    /*set the text for this class*/
    void SetTextPosition(Vec3 startPos, Vec3 endPos) { _startPos = startPos; _endPos
= endPos;}

    /*set the font and size for the text*/
    void SetTextSize( float size) { _TextSize = size;}



protected:
    //implement the parent NodeCallback virtual function, to set the callback
operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

private:

    ref_ptr<Geode>          m_Geode;        //the node of the osg, each node for
one axis

    ref_ptr<osgText::Text> m_text;          //the text content

    Vec3                    _startPos;
    Vec3                    _endPos;
    float                   _TextSize;
    string                  _TextFont;

    string                  _content;       //this is will allocate the memory for
the string.

};
//------------------------------------------------------------------------------------------------
//end of Axis




#endif
```

GraphicsPoint.h

```
/**********************************************************************
****/
/*   GraphicsPoint:        The graphics element point, which display the points
*                          according to the input of the program
*
*   Zhiyang
*/
/**********************************************************************
****/
#ifndef PLT_POINT_H
#define PLT_POINT_H

#include "GraphicsElement.h"

using namespace std;

class GraphicsPoint : public GraphicsElement
{
public:
      GraphicsPoint();
      ~GraphicsPoint();

      /*the callback function, for animation, implement this function*/
      //void update();

      /*the draw function to display the element*/
      void draw();

      /*the virtual function for child class to implement, which create an instance of
this model class*/
      GraphicsElement*     CreateInstance();

      /*start the animation*/
      //void Start();

      /*generate and form the graphics node for    display*/
      void CreateElement();



      /*Set and Get the Point Size*/
      void       SetPointSize(float size) { _PointSize = size;}
```

```
    float       GetPointSize() { return _PointSize;}


    /*add point to the list*/
    void        AddNewPoint(Vec3 point);

protected:
    //implement the parent NodeCallback virtual function, to set the callback
operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

private:

    float               _PointSize;             //the size of the Point
    vector<Vec3>        _PointList;             //the list of point to display

    ref_ptr<Geode>              m_Geode;            //the node of the osg
    ref_ptr<Geometry>           m_Geometry;         //the geometry for points


    //intermediate variable for animation
    unsigned int        _Count;                 //animation index

};


#endif
```

GraphicsPoint.cpp

```
/******************************************************************
****/
/*   GraphicsPoint:       The graphics element point, which display the points
*                         according to the input of the program
*                         implementation
*   Zhiyang
*/
/******************************************************************
****/
#include "GraphicsPoint.h"
#include <osg/Point>

//construction
//------------------------------------------------------------------------------------------
GraphicsPoint::GraphicsPoint():GraphicsElement()
{
	_PointSize = 1.0;
	_PointList.clear();

	this->setUpdateCallback(this);

	_ElementType = ELEMENT_POINT;

	_defaultLenght = 1;
}


GraphicsPoint::~GraphicsPoint()
{

}
//------------------------------------------------------------------------------------------




//add points to the point list for display
//------------------------------------------------------------------------------------------
void GraphicsPoint::AddNewPoint(Vec3 point)
{
	_PointList.push_back(point);
```

```
}
//------------------------------------------------------------------------------------------



/*
//the callback function, for animation, implement this function for animation
//------------------------------------------------------------------------------------------
void GraphicsPoint::update()
{
    if(_Play == true)
    {
        //update the timer
        _CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;
    //change to unit of second

        if(_CurrentTime > _Duration)      //finished play the animation single loop

        {
            //update the animation according to the flag
            if(_AnimationFlag == ANIMATION_LOOP)        //loop display
            {
                //reset the current time, for loop back to beginning
                _PlayStartTime = _timer->GetGraphicTime();
                _PlayEndTime = _PlayStartTime + _Duration;
                _CurrentTime = _CurrentTime - _Duration;

                //we will loop back to beginning again.
            }

            if(_AnimationFlag == ANIMATION_CLEAR)       //remove the display,
set the mode clear
            {
                this->setNodeMask(0x00000000);
                return;
            }

            if(_AnimationFlag == ANIMATION_STOP)        //stop playing the
animation
            {
                _CurrentTime = _Duration;
            }
        }
```

```
		draw();			//display
	}


}
//-------------------------------------------------------------------------------------------------
*/




//the draw function to display the element
//-------------------------------------------------------------------------------------------------
void GraphicsPoint::draw()
{

	if(_isDone == true)
		return;


	if(_Duration != 0)
		_Count = (int)((_CurrentTime/_Duration)*_PointList.size());
	//calculate every frame increment
	else
		_Count	= _PointList.size();

	if( _Count < _PointList.size())
	{
		m_Geometry->setPrimitiveSet(0,new	DrawArrays(PrimitiveSet::POINTS,
0 ,_Count));
	}
	else
	{
		_Count = _PointList.size();
		m_Geometry->setPrimitiveSet(0,new	DrawArrays(PrimitiveSet::POINTS,
0 ,_Count));

		_isDone = true;
	}

}
//-------------------------------------------------------------------------------------------------



/*
```

```
//start to play the animation
//-----------------------------------------------------------------------------------------------
void GraphicsPoint::Start()
{
      _Play = true;
      _PlayStartTime = _timer->GetGraphicTime();
      _PlayEndTime = _PlayStartTime + _Duration;
}
//-----------------------------------------------------------------------------------------------
*/



//generate and form the graphics node for    display
//-----------------------------------------------------------------------------------------------
void GraphicsPoint::CreateElement()
{
      //create the node and geometry
      m_Geode = new Geode;
      this->addChild(m_Geode.get());

      m_Geometry = new Geometry;
      m_Geometry->setUseDisplayList(false);
      m_Geode->addDrawable(m_Geometry.get());

      //=====================================
      //=create the points
      Vec3Array* vertices = new Vec3Array;
      vector<Vec3>::iterator index;
      for(index = _PointList.begin();index != _PointList.end();index++)
      {
            vertices->push_back(*index);
      }

      m_Geometry->addPrimitiveSet(new  DrawArrays(PrimitiveSet::LINE_STRIP,  0,
0));
      m_Geometry->setVertexArray(vertices);

      Vec4Array* colors = new Vec4Array;
      colors->push_back(_Color);
      m_Geometry->setColorArray(colors);
      m_Geometry->setColorBinding(Geometry::BIND_OVERALL);
      StateSet* stateSet = m_Geometry->getOrCreateStateSet();
      Point* point = new Point;
```

```
    point->setSize(_PointSize);
    stateSet->setAttributeAndModes(point, StateAttribute::ON);
    stateSet->setMode(GL_LIGHTING, StateAttribute::OFF);
    stateSet->setRenderBinDetails(12, "RenderBin");
    //=====================================



    //=====================================
    //=calculate the intermediate value for animation
    _Duration = _EndTime - _StartTime;
    if(_Duration == 0)
    {
        //display the element totally
        _Count = _PointList.size();
    }
    else
    {
        //display animation
        _Count = 0;
    }

}
//---------------------------------------------------------------------------------------------



/*the virtual function for child class to implement, which create an instance of this
model class*/
//---------------------------------------------------------------------------------------------
GraphicsElement*    GraphicsPoint::CreateInstance()
{
    GraphicsElement* point = NULL;

    return point;
}
//---------------------------------------------------------------------------------------------
```

GraphicsLine.h

```
/**********************************************************************
****/
/*   GraphicsLine:          The graphics element Line, which display the Lines
*                          according to the input of the program
*
*   Zhiyang
*/
/**********************************************************************
****/
#ifndef PLT_LINE_H
#define PLT_LINE_H

#include "GraphicsElement.h"

using namespace std;

class GraphicsLine : public GraphicsElement
{
public:
	GraphicsLine();
	~GraphicsLine();

	/*the callback function, for animation, implement this function*/
	//void update();

	/*the draw function to display the element*/
	void draw();

	/*the virtual function for child class to implement, which create an instance of
this model class*/
	GraphicsElement*    CreateInstance();

	/*start the animation*/
	//void Start();


	/*generate and form the graphics node for   display*/
	void CreateElement();


	/*Set and Get the Point Size*/
```

```
    void      SetLineWidth(float width) { _LineWidth = width;}
    float     GetLineWidth() { return _LineWidth;}


    /*add point to the list*/
    void      AddNewPoint(Vec3 point);

protected:
    //implement the parent NodeCallback virtual function, to set the callback
operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }

private:

    float               _LineWidth;          //the size of the Point
    vector<Vec3>        _PointList;          //the list of point to display lines

    ref_ptr<Geode>              m_Geode;            //the node of the osg
    ref_ptr<Geometry>           m_Geometry;         //the geometry for points

    //intermediate variable for animation
    unsigned int        _Count;              //animation index
};


#endif
```

GraphicsLine.cpp

```
/******************************************************************
****/
/*   GraphicsLine:        The graphics element Line, which display the Lines
*                         according to the input of the program
*                         Implementation
*   Zhiyang
*/
/******************************************************************
****/
#include "GraphicsLine.h"


//constructor
//----------------------------------------------------------------------------------------------
GraphicsLine::GraphicsLine():GraphicsElement()
{
      _LineWidth = 1.0;
      _PointList.clear();

      this->setUpdateCallback(this);

      _ElementType = ELEMENT_LINE;
      _defaultLenght = 1;
}


GraphicsLine::~GraphicsLine()
{

}
//----------------------------------------------------------------------------------------------




//add points to the point list for display
//----------------------------------------------------------------------------------------------
void GraphicsLine::AddNewPoint(Vec3 point)
{
      _PointList.push_back(point);
}
```

```
//------------------------------------------------------------------------------------------------




/*
//the callback function, for animation, implement this function for animation
//------------------------------------------------------------------------------------------------
void GraphicsLine::update()
{
    if(_Play == true)
    {
        //update the timer
        _CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

        if(_CurrentTime > _Duration)      //finished play the animation single loop

        {
            //update the animation according to the flag
            if(_AnimationFlag == ANIMATION_LOOP)         //loop display
            {
                //reset the current time, for loop back to beginning
                _PlayStartTime = _timer->GetGraphicTime();
                _PlayEndTime = _PlayStartTime + _Duration;
                _CurrentTime = _CurrentTime - _Duration;

                //we will loop back to beginning again.
            }

            if(_AnimationFlag == ANIMATION_CLEAR)       //remove the display,
set the mode clear
            {
                this->setNodeMask(0x00000000);
                return;
            }

            if(_AnimationFlag == ANIMATION_STOP)         //stop playing the
animation
            {
                _CurrentTime = _Duration;
            }
        }

        draw();          //display
```

```
	}
}
//-----------------------------------------------------------------------------------------------
*/



//the draw function to display the element
//-----------------------------------------------------------------------------------------------
void GraphicsLine::draw()
{
	if(_isDone == true)
		return;

	if(_Duration != 0)
		_Count = (int)((_CurrentTime/_Duration)*_PointList.size());
	//calculate every frame increment
	else
		_Count    = _PointList.size();


	if( _Count < _PointList.size())
	{
		m_Geometry->setPrimitiveSet(0,new
DrawArrays(PrimitiveSet::LINE_STRIP, 0 ,_Count));
	}
	else
	{
		_Count = _PointList.size();
		m_Geometry->setPrimitiveSet(0,new
DrawArrays(PrimitiveSet::LINE_STRIP, 0 ,_Count));

		_isDone = true;
	}

}
//-----------------------------------------------------------------------------------------------



/*
//start to play the animation
//-----------------------------------------------------------------------------------------------
void GraphicsLine::Start()
```

```
{
    _Play = true;
    _PlayStartTime = _timer->GetGraphicTime();
    _PlayEndTime = _PlayStartTime + _Duration;
}
//-----------------------------------------------------------------------------------------------
*/



//generate and form the graphics node for    display
//-----------------------------------------------------------------------------------------------
void GraphicsLine::CreateElement()
{
    //create the node and geometry
    m_Geode = new Geode;
    this->addChild(m_Geode.get());

    m_Geometry = new Geometry;
    m_Geometry->setUseDisplayList(false);
    m_Geode->addDrawable(m_Geometry.get());

    //=====================================
    //=create the points
    Vec3Array* vertices = new Vec3Array;
    vector<Vec3>::iterator index;
    for(index = _PointList.begin();index != _PointList.end();index++)
    {
        vertices->push_back(*index);
    }

    m_Geometry->addPrimitiveSet(new  DrawArrays(PrimitiveSet::LINE_STRIP,  0,
0));
    m_Geometry->setVertexArray(vertices);

    Vec4Array* colors = new Vec4Array;
    colors->push_back(_Color);
    m_Geometry->setColorArray(colors);
    m_Geometry->setColorBinding(Geometry::BIND_OVERALL);
    StateSet* stateSet = m_Geometry->getOrCreateStateSet();
    LineWidth* width = new LineWidth;
    width->setWidth(_LineWidth);
    stateSet->setAttributeAndModes(width, StateAttribute::ON);
    stateSet->setMode(GL_LIGHTING, StateAttribute::OFF);
    stateSet->setRenderBinDetails(12, "RenderBin");
    //====================================



    //====================================
    //=calculate the intermediate value for animation
    _Duration = _EndTime - _StartTime;
    if(_Duration == 0)
    {
        //display the element totally
        _Count = _PointList.size();
    }
    else
    {
        //display animation
        _Count = 0;
    }

}
//-----------------------------------------------------------------------------------------------





/*the virtual function for child class to implement, which create an instance of this
model class*/
//-----------------------------------------------------------------------------------------------
GraphicsElement*    GraphicsLine::CreateInstance()
{
    GraphicsElement* point = NULL;

    return point;
}
//-----------------------------------------------------------------------------------------------
```

GraphicsBox.cpp

```
/***********************************************************************
****/
/*
  *     GraphicsBox:              Display the box , which is defined by data
  *                               and in animation sequence.
  *
  *    Zhiyang
  */
/***********************************************************************
****/
#include "GraphicsObject.h"
#include "plt.h"
#include <osg/Vec4>


//constructor
//---------------------------------------------------------------------------------------------
GraphicsBox::GraphicsBox():GraphicsElement()
{
       _width = 0.0f;
       _length = 0.0f;
       _value = 0.0f;

       _position = Vec3(0.0, 0.0, 0.0);
       _Transform = new PositionAttitudeTransform;

       _Color = Vec4(1.0, 1.0, 1.0, 1.0);

       this->setUpdateCallback(this);

       _ElementType = ELEMENT_BOX;

       _defaultLenght = 2;
}


GraphicsBox::~GraphicsBox()
{

}
```

```
/*
//the callback function, for animation, implement this function
//---------------------------------------------------------------------------------------------
void GraphicsBox::update()
{
    if(_Play == true)
    {
        //update the timer
        _CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

        if(_CurrentTime > _Duration)        //finished play the animation single loop

        {
            //update the animation according to the flag
            if(_AnimationFlag == ANIMATION_LOOP)        //loop display
            {
                //reset the current time, for loop back to beginning
                _PlayStartTime = _timer->GetGraphicTime();
                _PlayEndTime = _PlayStartTime + _Duration;
                _CurrentTime = _CurrentTime - _Duration;

                //we will loop back to beginning again.
            }

            if(_AnimationFlag == ANIMATION_CLEAR)        //remove the display, set the mode clear
            {
                this->setNodeMask(0x00000000);
                return;
            }

            if(_AnimationFlag == ANIMATION_STOP)        //stop playing the animation
            {
                _CurrentTime = _Duration;
            }
        }

        draw();            //display
    }
```

```
}
*/



//the draw function to display the element
//---------------------------------------------------------------------------------------------
void GraphicsBox::draw()
{
        printf("postion: %f\n",    _position._v[0]);

        if(_isDone == true)
                return;                                  //we already finished the animation, don't have
to do anything


        //otherwise, we ....have to update

        if(_Duration != 0)
                _TempValue = (_CurrentTime/_Duration)*_value;                        //calculate
every frame increment
        else
                _TempValue = _value;

        ref_ptr<ShapeDrawable> shape;                    //the box;

        if( _TempValue < _value)
        {
                m_Geode->removeDrawable(1,1);                //just remove the box

                Vec3 origin (0.0, 0.0, 0.0);
                origin._v[2] += _TempValue/2.0;

               shape = new ShapeDrawable(new Box(origin, _width, _length, _TempValue),
_hints.get());
                shape->setColor(_Color);
                StateSet* shapeset = shape->getOrCreateStateSet();
                shapeset->setMode(GL_BLEND,StateAttribute::ON);
                shapeset->setRenderingHint(StateSet::TRANSPARENT_BIN);
                m_Geode->addDrawable(shape.get());

        }
        else
        {
```

```
        m_Geode->removeDrawable(1,1);           //just remove the box

        Vec3 origin(0.0, 0.0, 0.0);
        origin._v[2] += _value/2.0;

        shape = new ShapeDrawable(new Box(origin, _width, _length, _value),
_hints.get());
        shape->setColor(_Color);
        StateSet* shapeset = shape->getOrCreateStateSet();
        shapeset->setMode(GL_BLEND,StateAttribute::ON);
        shapeset->setRenderingHint(StateSet::TRANSPARENT_BIN);
        m_Geode->addDrawable(shape.get());


        _isDone = true;
    }
}



//the virtual function for child class to implement, which create an instance of this
model class
//-----------------------------------------------------------------------------------------------
GraphicsElement* GraphicsBox::CreateInstance()
{
    return NULL;
}




//start the animation
//-----------------------------------------------------------------------------------------------
void GraphicsBox::Start()
{
    //start from beginning
    m_Geode->removeDrawable(1,1);           //just remove the box

    GraphicsElement::Start();
}



//generate and form the graphics node for  display
```

```
//----------------------------------------------------------------------------------------------
void GraphicsBox::CreateElement()
{
	//create the node and geometry
	m_Geode = new Geode;


	this->addChild(_Transform);
	_Transform->addChild(m_Geode.get());

	m_Geometry = new Geometry;
	m_Geometry->setUseDisplayList(false);
	m_Geode->addDrawable(m_Geometry.get());


	//==========================================================
===================
	//create the box
	_hints = new TessellationHints;
	_hints->setDetailRatio(2.0f);


	Vec3 origin(0.0, 0.0, 0.0);
	origin._v[2]+= 0.0 / 2.0f ;

	ref_ptr<ShapeDrawable> shape;			//the box;
	shape = new ShapeDrawable(new Box(origin, _width, _length, 0), _hints.get());
	shape->setColor(Vec4(1.0, 1.0, 1.0, 0.0));			//make it invisible first
	StateSet* shapeset = shape->getOrCreateStateSet();
	shapeset->setMode(GL_BLEND,StateAttribute::ON);
	shapeset->setRenderingHint(StateSet::TRANSPARENT_BIN);

	m_Geode->addDrawable(shape.get());
	//==========================================================
====================




	//=====================================
	//=calculate the intermediate value for animation
	_Duration = _EndTime - _StartTime;
	if(_Duration == 0)
	{
        //display the element totally
        _TempValue = _value;
    }
    else
    {
        //display animation
        _TempValue = 0.0f;
    }
}



//Set box position
//---------------------------------------------------------------------------------------------
void GraphicsBox::SetBoxPosition(Vec3 pos)
{
    _position = pos;
    _Transform->setPosition(_position);
}



//Set the Value of box
//---------------------------------------------------------------------------------------------
void GraphicsBox::SetBoxParameter(float width, float length, float value)
{
    _width = width;
    _length = length;
    _value = value;
}
```

GraphicsCylinder.cpp

```
/*****************************************************************
****/
/*
*     GraphicsCylinder:          Display the cylinder , which is defined by data
*                                and in animation sequence.
*
*      Zhiyang
*/
/*****************************************************************
****/
#include "GraphicsObject.h"
#include "plt.h"
#include <osg/Vec4>


//constructor
//--------------------------------------------------------------------------------------------
GraphicsCylinder::GraphicsCylinder():GraphicsElement()
{
      _radius = 0.0f;
      _value = 0.0f;

      _position = Vec3(0.0, 0.0, 0.0);

      _Color = Vec4(1.0, 1.0, 1.0, 1.0);
      _outlineCount = 60;


      this->setUpdateCallback(this);

      _ElementType = ELEMENT_CYLINDER;
}


GraphicsCylinder::~GraphicsCylinder()
{

}
```

```
/*
//the callback function, for animation, implement this function
//-----------------------------------------------------------------------------------------
void GraphicsCylinder::update()
{
    if(_Play == true)
    {
        //update the timer
        _CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

        if(_CurrentTime > _Duration)     //finished play the animation single loop

        {
            //update the animation according to the flag
            if(_AnimationFlag == ANIMATION_LOOP)        //loop display
            {
                //reset the current time, for loop back to beginning
                _PlayStartTime = _timer->GetGraphicTime();
                _PlayEndTime = _PlayStartTime + _Duration;
                _CurrentTime = _CurrentTime - _Duration;

                //we will loop back to beginning again.
            }

            if(_AnimationFlag == ANIMATION_CLEAR)       //remove the display, set the mode clear
            {
                this->setNodeMask(0x00000000);
                return;
            }

            if(_AnimationFlag == ANIMATION_STOP)        //stop playing the animation
            {
                _CurrentTime = _Duration;
            }
        }

        draw();         //display
    }
}
*/
```

```
//the draw function to display the element
//------------------------------------------------------------------------------------------------
void GraphicsCylinder::draw()
{

	if(_isDone == true)
		return;				//we already finished the animation, don't have
to do anything


	//otherwise, we ....have to update


	if(_Duration != 0)
		_TempValue = (_CurrentTime/_Duration)*_value;			//calculate
every frame increment
	else
		_TempValue = _value;

	ref_ptr<ShapeDrawable> shape;			//the box;

	if( _TempValue < _value)
	{
		m_Geode->removeDrawable(1,1);		//just remove the box

		Vec3 origin = _position;
		origin._v[2] += _TempValue/2.0;

		shape = new ShapeDrawable(new Cylinder(origin,_radius,_TempValue),
_hints.get());
		shape->setColor(_Color);
		StateSet* shapeset = shape->getOrCreateStateSet();
		shapeset->setMode(GL_BLEND,StateAttribute::ON);
		shapeset->setRenderingHint(StateSet::TRANSPARENT_BIN);
		m_Geode->addDrawable(shape.get());



	}
	else
	{
		m_Geode->removeDrawable(1,1);		//just remove the box
```

```
            Vec3 origin = _position;
            origin._v[2] += _value/2.0;

            shape = new ShapeDrawable(new Cylinder(origin,_radius,_TempValue),
_hints.get());
            shape->setColor(_Color);
            StateSet* shapeset = shape->getOrCreateStateSet();
            shapeset->setMode(GL_BLEND,StateAttribute::ON);
            shapeset->setRenderingHint(StateSet::TRANSPARENT_BIN);
            m_Geode->addDrawable(shape.get());



            _isDone = true;

      }
}



//the virtual function for child class to implement, which create an instance of this
model class
//-----------------------------------------------------------------------------------------------
GraphicsElement* GraphicsCylinder::CreateInstance()
{
      return NULL;
}




//start the animation
//-----------------------------------------------------------------------------------------------
void GraphicsCylinder::Start()
{


      m_Geode->removeDrawable(1,1);            //just remove the cylinder

      GraphicsElement::Start();

}
```

```
//generate and form the graphics node for    display
//-------------------------------------------------------------------------------------------------
void GraphicsCylinder::CreateElement()
{
    //create the node and geometry
    m_Geode = new Geode;
    this->addChild(m_Geode.get());

    m_Geometry = new Geometry;
    m_Geometry->setUseDisplayList(false);
    m_Geode->addDrawable(m_Geometry.get());

    //=========================================================
==================
    //create the cylinder
    _hints = new TessellationHints;
    _hints->setDetailRatio(0.5f);


    Vec3 origin = _position;
    origin._v[2]+= _value / 2.0f ;

    ref_ptr<ShapeDrawable> shape;              //the cylinder;
    shape = new ShapeDrawable(new Cylinder(origin,_radius,_value), _hints.get());
    shape->setColor(Vec4(1.0, 1.0, 1.0, 0.0));
    //make it invisible first
    StateSet* shapeset = shape->getOrCreateStateSet();
    shapeset->setMode(GL_BLEND,StateAttribute::ON);
    shapeset->setRenderingHint(StateSet::TRANSPARENT_BIN);

    m_Geode->addDrawable(shape.get());
    //=========================================================
=====================


    //======================================
    //=calculate the intermediate value for animation
    _Duration = _EndTime - _StartTime;
    if(_Duration == 0)
    {
        //display the element totally
        _TempValue = _value;
```

```
	}
	else
	{
		//display animation
		_TempValue = 0.0f;
	}
}



//Set box position
//-----------------------------------------------------------------------------------------------
void GraphicsCylinder::SetCylinderPosition(Vec3 pos)
{
	_position = pos;
}



//Set the Value of box
//-----------------------------------------------------------------------------------------------
void GraphicsCylinder::SetCylinderParameter(float radius, float value)
{
	_radius = radius;
	_value = value;
}
```

GraphicsSlice.cpp

```
/*******************************************************************
****/
/*
 *       GraphicsSlice:        the fan slice for the pie chart in graphics
 *                             this implement the graphics construction and
 *                             animation control
 *       Zhiyang
 */
/*******************************************************************
****/

#include "GraphicsObject.h"

#include <osg/Array>
#include <osg/Math>

#define    ANGLE_PER_VERTEX        10


using namespace osg;
//constructor
//-----------------------------------------------------------------------------------------
GraphicsSlice::GraphicsSlice():GraphicsElement()
{
    _radius = 0.0f;
    _startAngle = 0.0f;
    _angle = 0.0f;
    _height = 0.1f;
    _position = Vec3(0.0, 0.0, 0.0);

    _animationCounter = 0;
    _SideCount = 0;

    _Color = Vec4(1.0, 1.0, 1.0, 1.0);
    _Transform = new PositionAttitudeTransform();

    quadIndex = NULL;

    this->setUpdateCallback(this);

    _ElementType = ELEMENT_SLICE;
```

```
}


GraphicsSlice::~GraphicsSlice()
{
        if(quadIndex != NULL)
                free(quadIndex);

}



/*

//the callback function, for animation, implement this function
//-----------------------------------------------------------------------------------------------
void GraphicsSlice::update()
{
        if(_Play == true)
        {
                //update the timer
                _CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

                if(_CurrentTime > _Duration)        //finished play the animation single loop

                {
                        //update the animation according to the flag
                        if(_AnimationFlag == ANIMATION_LOOP)            //loop display
                        {
                                //reset the current time, for loop back to beginning
                                _PlayStartTime = _timer->GetGraphicTime();
                                _PlayEndTime = _PlayStartTime + _Duration;
                                _CurrentTime = _CurrentTime - _Duration;

                                //we will loop back to beginning again.
                        }

                        if(_AnimationFlag == ANIMATION_CLEAR)          //remove the display,
set the mode clear
                        {
                                this->setNodeMask(0x00000000);
                                return;
                        }
```

```
			if(_AnimationFlag == ANIMATION_STOP)		//stop	playing	the
animation
			{
				_CurrentTime = _Duration;
				_isDone = true;
			}
		}

		draw();			//display
	}
}
*/




//the draw function to display the element
//-----------------------------------------------------------------------------------------------
void GraphicsSlice::draw()
{

	if(_isDone == true)
		return;				//we already finished the animation, don't have
to do anything


	//otherwise, we ....have to update



	int _Count;


	if(_Duration != 0)
		_Count = (int)((_CurrentTime/_Duration)*(_SideCount+1));
	//calculate every frame increment
	else
		_Count	= _SideCount+1;


	if( _Count < _SideCount+1)
	{
		m_Geometry->setPrimitiveSet(0,						new
DrawArrays(PrimitiveSet::TRIANGLE_FAN, 0, _Count+1));
```

```
        m_Geometry->setPrimitiveSet(1, new
DrawArrays(PrimitiveSet::TRIANGLE_FAN, _SideCount+2, _Count+1));

        if(_Count == 2)
        {
            unsigned short indices[] = { 0, 2+_SideCount, 3+_SideCount, 1};
            m_Geometry->setPrimitiveSet(2, new
DrawElementsUShort(PrimitiveSet::QUADS,4,indices));

            unsigned short indices2[] = { 0, _Count+1, (_Count+1)*2 - 1, _Count };
            m_Geometry->setPrimitiveSet(3, new
DrawElementsUShort(PrimitiveSet::QUADS,4,indices2));
        }
        else
        {
            m_Geometry->setPrimitiveSet(2, new
DrawElementsUShort(PrimitiveSet::QUAD_STRIP,(_Count+1)*2,quadIndex));

            unsigned short indices2[] = { 0, 2 +_SideCount, _Count+2+_SideCount,
_Count };
            m_Geometry->setPrimitiveSet(3, new
DrawElementsUShort(PrimitiveSet::QUADS,4,indices2));
        }

    }
    else
    {
        _Count = _SideCount+1;
        m_Geometry->setPrimitiveSet(0, new
DrawArrays(PrimitiveSet::TRIANGLE_FAN, 0, _Count+1));
        m_Geometry->setPrimitiveSet(1, new
DrawArrays(PrimitiveSet::TRIANGLE_FAN, _SideCount+2, _Count+1));


        m_Geometry->setPrimitiveSet(2, new
DrawElementsUShort(PrimitiveSet::QUAD_STRIP,(_Count+1)*2,quadIndex));

        unsigned short indices2[] = { 0, _Count+1, (_Count+1)*2 - 1, _Count };
        m_Geometry->setPrimitiveSet(3, new
DrawElementsUShort(PrimitiveSet::QUADS,4,indices2));


        m_Geometry->addPrimitiveSet( new
DrawArrays(PrimitiveSet::LINE_LOOP,0, _SideCount+2));
```

```
            m_Geometry->addPrimitiveSet(                                          new
DrawArrays(PrimitiveSet::LINE_LOOP,_SideCount+2, _SideCount+2));

            _isDone = true;
      }

}



//the virtual function for child class to implement, which create an instance of this
model class
//---------------------------------------------------------------------------------------------
GraphicsElement* GraphicsSlice::CreateInstance()
{
      return NULL;
}



//start the animation
//---------------------------------------------------------------------------------------------
void GraphicsSlice::Start()
{

      GraphicsElement::Start();

      m_Geometry->addPrimitiveSet(                                                new
DrawElementsUShort(PrimitiveSet::QUAD_STRIP, 2, quadIndex));
      m_Geometry->addPrimitiveSet(                                                new
DrawElementsUShort(PrimitiveSet::QUADS, 2, quadIndex));


}



//generate and form the graphics node for    display
//---------------------------------------------------------------------------------------------
void GraphicsSlice::CreateElement()
{
      //create the node and geometry
      m_Geode = new Geode;
      this->addChild(_Transform);
```

```
_Transform->addChild(m_Geode.get());
//this->addChild(m_Geode.get());

m_Geometry = new Geometry;
m_Geometry->setUseDisplayList(false);
m_Geode->addDrawable(m_Geometry.get());

side_geom = new Geometry;
m_Geode->addDrawable(side_geom.get());

//==============================================
//create the slice


Vec3Array*  vertices = new Vec3Array();



vector<Vec3>      tempVertexTop;
vector<Vec3>      tempVertexBottom;
//create the vertex on the side, first calculate the vertex except the last one
for(int i=0; i< _SideCount;i++)
{
      float angle = _startAngle + i*ANGLE_PER_VERTEX;
      float x =    _radius*cos(angle*DEG2RAD);
      float y =    _radius*sin(angle*DEG2RAD);

      tempVertexBottom.push_back(Vec3(x,y,0.0f));
      tempVertexTop.push_back(Vec3(x,y,_height));
}

float angle = _startAngle + _angle - 0.01;
float x =    _radius*cos(angle*DEG2RAD);
float y =    _radius*sin(angle*DEG2RAD);

tempVertexBottom.push_back(Vec3(x,y,0.0f));
tempVertexTop.push_back(Vec3(x,y,_height));

//push the origin point
vertices->push_back(Vec3(0.0f,0.0f,0.0f));
//put the vertices into array correctly in order
for(size_t i =0; i < tempVertexBottom.size();i++)
{
      vertices->push_back(tempVertexBottom[i]);
}

vertices->push_back(Vec3(0.0f,0.0f,_height));
for(size_t i =0; i < tempVertexTop.size();i++)
{
    vertices->push_back(tempVertexTop[i]);
}


//calculate the normal for all face
//==========================================================
==========
vector<Vec3> FaceNormal;
FaceNormal.push_back(Vec3(0.0, 0.0, -1.0));         //bottom one
FaceNormal.push_back(Vec3(0.0, 0.0, 1.0));          //top one


//the side 1
Vec3f v1 = (*vertices)[1] -   (*vertices)[0];
v1.normalize();
Vec3f v2 = Vec3f(0.0, 0.0, 1.0);
Vec3f normal = v1^v2;
normal.normalize();
FaceNormal.push_back(normal);

//face on curve
for(int i=0; i < _SideCount; i++)
{
    float angle = _startAngle + ANGLE_PER_VERTEX*(0.5+i);
    float x =   _radius*cos(angle*DEG2RAD);
    float y =   _radius*sin(angle*DEG2RAD);

    normal = Vec3f(x, y, 0);
    normal.normalize();
    FaceNormal.push_back(normal);
}

//the side 2 normal
v1 = (*vertices)[_SideCount+1] -   (*vertices)[0];
v1.normalize();
normal = v2^v1;
normal.normalize();
FaceNormal.push_back(normal);
```

```
    int FaceNumber = FaceNormal.size();
    //--------------------------------------------------------------------------
    //end of calculate the normal for each face



    //calculate the normal for all vertex
    //=======================================================
====================
    Vec3Array*    vertexNormal = new Vec3Array();

    //index 0:    the origin point, it connect faces with index 0, 2 , FaceNumber-1;
    normal = ( FaceNormal[0] + FaceNormal[2] + FaceNormal[FaceNumber - 1]);
    normal.normalize();
    vertexNormal->push_back(normal);

    //index 1 - (count+1),    it connect faces with index i+1, i, 0
    for(int i=1; i<= _SideCount+1; i++)
    {
        normal = ( FaceNormal[i+1] + FaceNormal[i+2] + FaceNormal[0]);
        normal.normalize();
        vertexNormal->push_back(normal);
    }

    //index count+2,    the top origin point, index with 1, 2, FaceNumber -1
    normal = ( FaceNormal[1] + FaceNormal[2] + FaceNormal[FaceNumber - 1]);
    normal.normalize();
    vertexNormal->push_back(normal);

    //index count + 3    to (count+2)*2,
    for(int i=1; i<= _SideCount+1; i++)
    {
        normal = ( FaceNormal[i+1] + FaceNormal[i+2] + FaceNormal[1]);
        normal.normalize();
        vertexNormal->push_back(normal);
    }


    //---------------------------------------------------------------------------
    //end of calculate the normal for each vertex


    m_Geometry->addPrimitiveSet(new
DrawArrays(PrimitiveSet::TRIANGLE_FAN, 0,0));
```

```
    m_Geometry->addPrimitiveSet(new
DrawArrays(PrimitiveSet::TRIANGLE_FAN, _SideCount+1, 0));
    m_Geometry->setNormalArray(vertexNormal);
    m_Geometry->setNormalBinding(Geometry::BIND_PER_VERTEX);


    m_Geometry->setVertexArray(vertices);

    quadIndex = (unsigned short*)malloc((vertices->size()+2)*sizeof(unsigned
short));


    int j = 0;
    for( int i = 0; i< _SideCount+2; i++)
    {
        j = i*2;
        quadIndex[j] = i;
        quadIndex[j+1] = i+_SideCount+2;

    }
    quadIndex[j+2] = 0;
    quadIndex[j+3] = _SideCount+2;              //for the loop for a closure model




    Vec4Array* colors = new Vec4Array;
    colors->push_back(_Color);
    m_Geometry->setColorArray(colors);
    m_Geometry->setColorBinding(Geometry::BIND_OVERALL);
    StateSet* stateSet = m_Geometry->getOrCreateStateSet();
    stateSet->setMode(GL_LIGHTING, StateAttribute::ON);
    stateSet->setMode(GL_BLEND,StateAttribute::ON);
    stateSet->setRenderingHint(StateSet::TRANSPARENT_BIN);


    /*
    //===========================test the
normal====================================
    Vec3Array*   normaltestV = new Vec3Array();

    int size = vertexNormal->size();
    size = vertexNormal->size();
    for(size_t i=0; i< vertexNormal->size(); i++)
    {

        Vec3f n = (*vertexNormal)[i];
        Vec3f start = (*vertices)[i];
        Vec3f end = start + n*1.0;
        normaltestV->push_back(start);
        normaltestV->push_back(end);

    }

    size = normaltestV->size();
    side_geom->addPrimitiveSet( new DrawArrays(PrimitiveSet::LINES,
0,normaltestV->size()));
    side_geom->setVertexArray(normaltestV);


    Vec4Array* Bcolors = new Vec4Array;
    Bcolors->push_back(Vec4(1.0, 1.0, 1.0, 1.0));
    side_geom->setColorArray(Bcolors);
    side_geom->setColorBinding(Geometry::BIND_OVERALL);
    stateSet = side_geom->getOrCreateStateSet();
    stateSet->setMode(GL_LIGHTING, StateAttribute::OFF);
    stateSet->setMode(GL_BLEND,StateAttribute::ON);
    stateSet->setRenderingHint(StateSet::TRANSPARENT_BIN);
    //==================================================
======================
    */




    //=================================
    //=calculate the intermediate value for animation
    _Duration = _EndTime - _StartTime;
    if(_Duration == 0)
    {
        //display the element totally
        _animationCounter = _SideCount;
    }
    else
    {
        //display animation
        _animationCounter = 0;
    }
```

```
}



//Set box position
//---------------------------------------------------------------------------------------------
void GraphicsSlice::SetSlicePosition(Vec3 pos)
{
        _position = pos;
        _Transform->setPosition(_position);
}



//Set the Value of slice
//---------------------------------------------------------------------------------------------
void GraphicsSlice::SetSliceParameter(float radius, float startAngle, float angle, float
height)
{
        _radius = radius;
        _startAngle = startAngle;
        _angle = angle;
        _height = height;

        _SideCount = (int)angle/ANGLE_PER_VERTEX;

        if( angle - (int)angle < 0.001)
                ;
        else
                _SideCount += 1;
}
```

GraphicsPlane.cpp

```
/*****************************************************************
****/
/*
 *      GraphicsPlane:       display a plane, defined by three point
 *                           and define the animation.
 *
 *      Zhiyang
 */
/*****************************************************************
****/
#include "GraphicsObject.h"

using namespace std;

//constructor
//-------------------------------------------------------------------------------------------
GraphicsPlane::GraphicsPlane():GraphicsElement()
{
    _VertexList.clear();

    this->setUpdateCallback(this);

    _ElementType = ELEMENT_PLANE;
}


GraphicsPlane::~GraphicsPlane()
{

}
//-------------------------------------------------------------------------------------------




//add points to the vertex list for define the plane
//-------------------------------------------------------------------------------------------
void GraphicsPlane::AddNewVertex(Vec3 point)
{
    if( _VertexList.size() < 3)
        _VertexList.push_back(point);
```

```
}
//--------------------------------------------------------------------------------------------



/*
//the callback function, for animation, implement this function for animation
//--------------------------------------------------------------------------------------------
void GraphicsPlane::update()
{
    if(_Play == true)
    {
        //update the timer
        _CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

        if(_CurrentTime > _Duration)        //finished play the animation single loop

        {
            //update the animation according to the flag
            if(_AnimationFlag == ANIMATION_LOOP)        //loop display
            {
                //reset the current time, for loop back to beginning
                _PlayStartTime = _timer->GetGraphicTime();
                _PlayEndTime = _PlayStartTime + _Duration;
                _CurrentTime = _CurrentTime - _Duration;

                //we will loop back to beginning again.
            }

            if(_AnimationFlag == ANIMATION_CLEAR)        //remove the display,
set the mode clear
            {
                this->setNodeMask(0x00000000);
                return;
            }

            if(_AnimationFlag == ANIMATION_STOP)        //stop playing the
animation
            {
                _CurrentTime = _Duration;
            }
        }
```

```
            draw();               //display
      }
}
//-----------------------------------------------------------------------------------------------
*/




//the draw function to display the element
//-----------------------------------------------------------------------------------------------
void GraphicsPlane::draw()
{
      if(_isDone == true)
            return;                           //we already finished the animation, don't have
to do anything


      //otherwise, we ....have to update


      float ratio =0.0;

      if(_Duration != 0)
            ratio = (_CurrentTime/_Duration);                 //calculate every frame
increment
      else
            ratio   = 1.0;

      if(ratio > 1)
            ratio = 1;

      //update the position of
      Vec3f temp2, temp3;


      temp2 = _VertexList[1]*(1-ratio) + _VertexList[2]*ratio;
      temp3 = _VertexList[0]*(1-ratio) + _VertexList[3]*ratio;

      Vec3Array* vertexList =
dynamic_cast<osg::Vec3Array*>(m_Geometry->getVertexArray());
      (*vertexList)[2] = temp2;
      (*vertexList)[3] = temp3;
```

```
    if(ratio == 1)
        _isDone = true;

}
//---------------------------------------------------------------------------------------------



//start to play the animation
//---------------------------------------------------------------------------------------------
void GraphicsPlane::Start()
{
    GraphicsElement::Start();

    m_Geometry->setPrimitiveSet(0, new DrawArrays(PrimitiveSet::QUADS, 0, 4));
    m_Geometry->setPrimitiveSet(1,                                         new
DrawArrays(PrimitiveSet::LINE_LOOP,0,4));
}
//---------------------------------------------------------------------------------------------




//generate and form the graphics node for   display
//---------------------------------------------------------------------------------------------
void GraphicsPlane::CreateElement()
{
    //create the node and geometry
    m_Geode = new Geode;
    this->addChild(m_Geode.get());

    m_Geometry = new Geometry;
    m_Geometry->setUseDisplayList(false);
    m_Geode->addDrawable(m_Geometry.get());


    //calculate the four point for this plane
    //==========================================
    //      0           3
    //   1          2
    //
    Vec3 ve = (_VertexList[0] - _VertexList[1]);
    float dist = sqrt( ve*ve);
    ve.normalize();
    Vec3 vertex4 = _VertexList[2] + ve*dist;
    _VertexList.push_back(vertex4);

    Vec3Array* normal = new Vec3Array;
    Vec3 va = (_VertexList[2] - _VertexList[1]);
    va.normalize();

    Vec3 norm = va^ve;
    norm.normalize();
    normal->push_back(norm);

    //======================================
    //=create the points
    Vec3Array* vertices = new Vec3Array;
    for(size_t index = 0; index < _VertexList.size();index++)
    {
        Vec3 vert = _VertexList[index];
        vertices->push_back(Vec3(vert._v[0], vert._v[1], vert._v[2]));
    }

    m_Geometry->addPrimitiveSet(new DrawArrays(PrimitiveSet::QUADS, 0, 0));
    m_Geometry->addPrimitiveSet(new
DrawArrays(PrimitiveSet::LINE_LOOP,0,0));
    m_Geometry->setVertexArray(vertices);
    m_Geometry->setNormalArray(normal);
    m_Geometry->setNormalBinding(Geometry::BIND_OVERALL);


    Vec4Array* colors = new Vec4Array;
    colors->push_back(_Color);
    colors->push_back(Vec4(1.0, 1.0, 1.0, 0.8));
    m_Geometry->setColorArray(colors);
    m_Geometry->setColorBinding(Geometry::BIND_PER_PRIMITIVE_SET);
    StateSet* stateSet = m_Geometry->getOrCreateStateSet();
    LineWidth* width = new LineWidth;
    width->setWidth(2.0);
    stateSet->setMode(GL_LIGHTING, StateAttribute::ON);
    stateSet->setMode(GL_BLEND,StateAttribute::ON);
    stateSet->setRenderingHint(StateSet::TRANSPARENT_BIN);
    //======================================
    //======================================
    //=calculate the intermediate value for animation
    _Duration = _EndTime - _StartTime;


}
//-----------------------------------------------------------------------------------------------






/*the virtual function for child class to implement, which create an instance of this
model class*/
//-----------------------------------------------------------------------------------------------
GraphicsElement*    GraphicsPlane::CreateInstance()
{
    return NULL;
}
//-----------------------------------------------------------------------------------------------
```

GraphicsText.cpp

```
/**************************************************************************
****/
/*
 *        GraphicsText:          display the 2D text in 3D position for information
 *                               in model.
 */
/**************************************************************************
****/

#include "GraphicsObject.h"

/**************************************************************************
****/
/*
*    GraphicsAxis:          To display the axis and its marker
*/
/**************************************************************************
****/

#include "GraphicsObject.h"

#include <osg/Array>
#include <osg/Math>
#include <osg/Shape>



using namespace osg;
//constructor
//-----------------------------------------------------------------------------------------------
GraphicsText::GraphicsText(char* text):GraphicsElement()
{

     _startPos = Vec3(0.0, 0.0, 0.0);
     _endPos = Vec3(0.0, 0.0, 0.0);

     _TextSize = 0.5;

     _content = _content.substr(0, 0);

     _content += text;
```

```
    _TextFont += "fonts/simhei.ttf";

    this->setUpdateCallback(this);

    _ElementType = ELEMENT_TEXT;
}


GraphicsText::~GraphicsText()
{

}





//the callback function, for animation, implement this function
//-----------------------------------------------------------------------------------------------
void GraphicsText::update()
{
    if(_Play == true)
    {
        //update the timer
        _CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

        if(_CurrentTime > _Duration)      //finished play the animation single loop

        {
            //update the animation according to the flag
            if(_AnimationFlag == ANIMATION_LOOP)          //loop display
            {
            //reset the current time, for loop back to beginning
            _PlayStartTime = _timer->GetGraphicTime();
            _PlayEndTime = _PlayStartTime + _Duration;
            _CurrentTime = _CurrentTime - _Duration;

            //we will loop back to beginning again.
            }

            if(_AnimationFlag == ANIMATION_CLEAR)        //remove the display,
set the mode clear
            {
                this->setNodeMask(0x00000000);
                return;
            }

            if(_AnimationFlag == ANIMATION_STOP)        //stop   playing   the
animation
            {
                _CurrentTime = _Duration;
                _isDone = true;
            }
        }

        draw();         //display
    }
}




//the draw function to display the element
//--------------------------------------------------------------------------------------------
void GraphicsText::draw()
{

    //if(_isDone == true)
    //    return;                       //we already finished the animation, don't have
to do anything


    //otherwise, we ....have to update
    float ratio;
    if(_Duration != 0)
        ratio = _CurrentTime/_Duration;
    else
        ratio = 1.0;


    Vec3 position = _startPos*(1-ratio) + _endPos*ratio;
    m_text->setPosition(position);

    if( ratio == 1.0)
        _isDone = true;
```

```
}



//the virtual function for child class to implement, which create an instance of this model class
//---------------------------------------------------------------------------------------------
GraphicsElement* GraphicsText::CreateInstance()
{
    return NULL;
}



//start the animation
//---------------------------------------------------------------------------------------------
void GraphicsText::Start()
{

    GraphicsElement::Start();

    m_Geode->setNodeMask(0xFFFFFFFF);

}



//generate and form the graphics node for  display
//---------------------------------------------------------------------------------------------
void GraphicsText::CreateElement()
{

    //create the node and geometry
    m_Geode = new Geode;

    this->addChild(m_Geode.get());

    m_text = new osgText::Text;
    m_text->setFont(_TextFont.c_str());
    m_text->setCharacterSize(_TextSize);
    m_text->setFontResolution(64, 64);
    m_text->setPosition(_startPos);
    m_text->setColor(_Color);
    m_text->setCharacterSizeMode(osgText::Text::OBJECT_COORDS );
    m_text->setAlignment(osgText::Text::CENTER_CENTER);
    m_text->setAutoRotateToScreen(true);
    m_text->setText(_content.c_str());
    m_Geode->addDrawable(m_text.get());

    StateSet* stateSet = m_text->getOrCreateStateSet();
    stateSet->setMode(GL_LIGHTING, StateAttribute::OFF);
    stateSet->setMode(GL_BLEND, StateAttribute::ON);
    stateSet->setRenderBinDetails(12, "RenderBin");

    //m_Geode->setNodeMask(0x00000000);

    //=======================================
    //=calculate the intermediate value for animation
    _Duration = _EndTime - _StartTime;

}
```

GraphicsAxis.cpp

```
/*******************************************************************
****/
/*
  *   GraphicsAxis:            To display the axis and its marker
  */
/*******************************************************************
****/

#include "GraphicsObject.h"

#include <osg/Array>
#include <osg/Math>
#include <osg/Shape>



using namespace osg;
//constructor
//-----------------------------------------------------------------------------------------------
GraphicsAxis::GraphicsAxis(int axis):GraphicsElement(),_Aixs(axis)
{
	_enabled = false;

	_interval = 1.0;

	this->setUpdateCallback(this);

	switch(_Aixs)
	{
		case X_AXIS:
			_AxisRotation.makeRotate(PI/2, 0.0, 1.0, 0.0);
			break;

		case Y_AXIS:
			_AxisRotation.makeRotate(-PI/2, 1.0, 0.0, 0.0);
			break;

		case Z_AXIS:
		default:
			_AxisRotation.makeRotate(0.0f, 0.0, 0.0, 1.0);
			break;
```

```
    }

    _AxisRadius = 0.05;

    _textColor = Vec4( 1.0f, 1.0f, 1.0f, 1.0f);


    _ElementType = ELEMENT_AXIS;
}


GraphicsAxis::~GraphicsAxis()
{


}




/*

//the callback function, for animation, implement this function
//-------------------------------------------------------------------------------------------------
void GraphicsAxis::update()
{
if(_Play == true)
{
//update the timer
_CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

if(_CurrentTime > _Duration)        //finished play the animation single loop
{
//update the animation according to the flag
if(_AnimationFlag == ANIMATION_LOOP)          //loop display
{
//reset the current time, for loop back to beginning
_PlayStartTime = _timer->GetGraphicTime();
_PlayEndTime = _PlayStartTime + _Duration;
_CurrentTime = _CurrentTime - _Duration;

//we will loop back to beginning again.
}
if(_AnimationFlag == ANIMATION_CLEAR)        //remove the display, set the
mode clear
{
this->setNodeMask(0x00000000);
return;
}

if(_AnimationFlag == ANIMATION_STOP)         //stop playing the animation
{
_CurrentTime = _Duration;
_isDone = true;
}
}

draw();              //display
}
}
*/




//the draw function to display the element
//-----------------------------------------------------------------------------------------------
void GraphicsAxis::draw()
{

    if(_isDone == true)
        return;                          //we already finished the animation, don't have
to do anything

}




//the virtual function for child class to implement, which create an instance of this
model class
//-----------------------------------------------------------------------------------------------
GraphicsElement* GraphicsAxis::CreateInstance()
{
    return NULL;
}
```

```
//start the animation
//------------------------------------------------------------------------------------------
void GraphicsAxis::Start()
{

    GraphicsElement::Start();

    m_Geode->setNodeMask(0xFFFFFFFF);

}



//generate and form the graphics node for   display
//------------------------------------------------------------------------------------------
void GraphicsAxis::CreateElement()
{

    //create the node and geometry
    m_Geode = new Geode;

    this->addChild(m_Geode.get());

    m_Geometry = new Geometry;
    m_Geometry->setUseDisplayList(false);
    m_Geode->addDrawable(m_Geometry.get());


    //===================================================
    //create the axis
    _hints = new TessellationHints;
    _hints->setDetailRatio(0.5f);

    ref_ptr<ShapeDrawable> shapeAxis;           //the axis;
    ref_ptr<ShapeDrawable> tipAxis;

    Vec3 center;
    _length = _marker.size()*_interval + 2;
    if( _Aixs == X_AXIS)
        center._v[0] = _length / 2;
    else if( _Aixs == Y_AXIS)
        center._v[1] = _length / 2;
    else
```

```
        center._v[2] = _length / 2;


ref_ptr<Cylinder>      axis = new Cylinder(center, _AxisRadius, _length);
axis->setRotation(_AxisRotation);
shapeAxis = new ShapeDrawable(axis.get(), _hints.get());
shapeAxis->setColor(_Color);
StateSet* shapeset = shapeAxis->getOrCreateStateSet();
shapeset->setMode(GL_BLEND,StateAttribute::ON);
shapeset->setRenderingHint(StateSet::TRANSPARENT_BIN);
m_Geode->addDrawable(shapeAxis.get());

float height = 2.0;
float coneRadiu = _AxisRadius+0.05;
Vec3    coneCenter;

if( _Aixs == X_AXIS)
        coneCenter._v[0] = _length + 0.5;
else if( _Aixs == Y_AXIS)
        coneCenter._v[1] = _length + 0.5;
else
        coneCenter._v[2] = _length + 0.5;

ref_ptr<Cone>      axistop = new Cone(coneCenter, coneRadiu, height);
axistop->setRotation(_AxisRotation);
tipAxis = new ShapeDrawable(axistop.get(),_hints.get());
tipAxis->setColor(_Color);
shapeset = tipAxis->getOrCreateStateSet();
shapeset->setMode(GL_BLEND,StateAttribute::ON);
shapeset->setRenderingHint(StateSet::TRANSPARENT_BIN);
m_Geode->addDrawable(tipAxis.get());

//------------------------------------------------------
//create the title for the axis
if( _title.size() ==0)
        _title += "NO TITLE";

osgText::Text* title = new osgText::Text;
title->setFont("fonts/simhei.ttf");
title->setCharacterSize(1.0);
title->setFontResolution(64, 64);
title->setPosition(ConvertPosition(_length+3));
title->setColor(_textColor);
title->setCharacterSizeMode(osgText::Text::OBJECT_COORDS );
```

```
    title->setAlignment(osgText::Text::CENTER_CENTER);
    title->setAutoRotateToScreen(true);
    title->setText(_title.c_str());
    m_Geode->addDrawable(title);

    StateSet* stateSet = title->getOrCreateStateSet();
    stateSet->setMode(GL_LIGHTING, StateAttribute::OFF);
    stateSet->setMode(GL_BLEND, StateAttribute::ON);
    stateSet->setRenderBinDetails(12, "RenderBin");


    //==================================================================
    //create the marker for axis
    for( int i=1; i< _marker.size();i++)
    {


        osgText::Text* marker = new osgText::Text;
        marker->setFont("fonts/simhei.ttf");
        marker->setCharacterSize(0.5);
        marker->setFontResolution(64, 64);
        marker->setPosition(ConvertPosition(i*_interval));
        marker->setColor(_textColor);
        marker->setCharacterSizeMode(osgText::Text::OBJECT_COORDS );
        marker->setAlignment(osgText::Text::CENTER_CENTER);
        marker->setAutoRotateToScreen(true);
        marker->setText(_marker[i].c_str());
        m_Geode->addDrawable(marker);

        StateSet* stateSet = marker->getOrCreateStateSet();
        stateSet->setMode(GL_LIGHTING, StateAttribute::OFF);
        stateSet->setMode(GL_BLEND, StateAttribute::ON);
        stateSet->setRenderBinDetails(12, "RenderBin");

    }

    m_Geode->setNodeMask(0x00000000);

    //=======================================
    //=calculate the intermediate value for animation
    _Duration = _EndTime - _StartTime;

}
```

```
//set the title for this axis
void GraphicsAxis::SetAxisTitle(char* title)
{
	_title.substr(0,0);
	_title += title;
}



//set the interval for the marker on this axis
void GraphicsAxis::SetAxisInterval(float interval)
{
	_interval = interval;
}



//set the marker by the index
void GraphicsAxis::SetAxisMarker(int index, string mark)
{
	if( _marker.size() <= index)
		_marker.resize(index+1);


	_marker[index] = mark;
}



Vec3 GraphicsAxis::ConvertPosition(float pos)
{
	Vec3 center(0.2,0.2,0.2);

	center._v[_Aixs] = pos;

	return center;
}
```

GraphicsDisplayModel.h

```
/*********************************************************************
****/
/*
 *      GraphicsDisplayModel:       the graphics elements to store the data
 *                                  and the models for display and animation
 *                                  this    model    is    generated    by
DataTypeDispalyModel
 *                                  according to the information and data.
 *      Zhiyang
 */
/*********************************************************************
****/

#ifndef PLT_GRAPHICS_DISPLAYMODEL_H
#define PLT_GRAPHICS_DISPLAYMODEL_H

#include <vector>
#include "plt.h"
#include "GraphicsElement.h"
#include "ModelData.h"

using namespace osg;
using namespace std;

class GraphicsDisplayModel : public GraphicsElement
{
public:
    /*constructor*/
    GraphicsDisplayModel();
    ~GraphicsDisplayModel();

public:
    //////////////////////////////////////////////////////////////////////
    //graphics
    /*the callback function, for animation, implement this function*/
    void update();

    /*the draw function to display the element*/
    void draw();

    /*Set and get the start time of the animation*/
    void            SetStartTime(float time) { _StartTime = time;}
    float           GetStartTime() { return _StartTime;}

    /*Set and get the end time of the animation*/
    void            SetEndTime(float time) { _EndTime = time;}
    float           GetEndTime() { return _EndTime;}

    /*control the play*/
    void            Start();

    GraphicsElement*    CreateInstance();

    void            CreateElement();

    /*Set the animation flag*/
    void            SetAnimationFlag(int flag) { _AnimationFlag = flag;}

    /*put the created model into to list*/
    void            AddElement(int type, int addtional);

    /*get the last element in list*/
    GraphicsElement*        GetCurrentElement();


    void            SetBoxParameter(float x, float y, float z);
    void            SetCylinderParameter(float r, float h);


    void            addOneVertex(float x, float y, float z);

    void            SetTimeForCurrent(float s, float d, float mode);

    void            GenerateCurrent();

    void            SetShowTime(float start, float duration);
    void            SetUnShowTime(float start, float duration);
    void            SetPositionTime(float start, float duration);
    void            SetRotationTime(float start, float duration);
    void            SetScalingTime(float start, float duration);
    void            SetMode(int index, int mode) { _mode[index] = mode;}

protected:
    //implement the parent NodeCallback virtual function, to set the callback operation
    virtual void operator()(Node* node, NodeVisitor* nv)
    {
        this->update();
        NodeCallback::traverse(node,nv);
    }


public:

    ref_ptr<Group>                          _ModelNode;
    //used to connect all graphics elements node
    ref_ptr<PositionAttitudeTransform>      _transform;
    vector<GraphicsElement*>                _ElementList;    //graphics element
    vector<GraphicsElement*>                _StartList;   //graphics element
    vector<ModelData*>                      _DataList;

    float                                   _cumulativeTime;

    //global variable for this display model
    Vec4                                    _gColor;
    float                                   _gPointSize;
    float                                   _gLineWidth;

    float                                   _showTime[2];
    float                                   _unShow[2];
    float                                   _positionTime[2];
    float                                   _rotationTime[2];
    float                                   _scalingTime[2];
    int                                     _mode[5];


    bool                                    _finishCreated;



};

#endif
```

GraphicsDisplayModel.cpp

```
/*****************************************************************
****/
/*
 *   GraphicsDisplayModel:          the element that display the graphics element
 *                                  to organize and manage the displaying and
 *                                  animation
 *   Zhiyang
 */
/*****************************************************************
****/

#include "GraphicsDisplayModel.h"
#include "GraphicsLine.h"
#include "GraphicsPoint.h"
#include "GraphicsObject.h"

#include "Global.h"

/*constructor*/
//-----------------------------------------------------------------------------------------------
GraphicsDisplayModel::GraphicsDisplayModel():GraphicsElement()
{
	_transform = new PositionAttitudeTransform();
	_ModelNode = new Group();
	this->addChild(_transform.get());
	this->addChild(_ModelNode.get());

	this->setUpdateCallback(this);

	_ElementList.clear();
	_gColor = Vec4(1.0,1.0, 1.0, 1.0);
	_gLineWidth = 1.0;
	_gPointSize = 1.0;


	_finishCreated = false;
	_cumulativeTime = 0;
}


GraphicsDisplayModel::~GraphicsDisplayModel()
```

```
{

}

/*the callback function, for animation, implement this function*/
void GraphicsDisplayModel::update()
{
	if(_Play == true)
	{
		//update the timer
		_CurrentTime = (_timer->GetGraphicTime() - _PlayStartTime)/1000;

		//update the element graphics
		vector<GraphicsElement*>::iterator index;
		for(index=_StartList.begin(); index < _StartList.end(); index++)
		{
			float sx = (*index)->GetStartTime();
			if(_CurrentTime >= sx )
			{
				(*index)->Start();
				_StartList.erase(index);
				printf("start one.....\n");
			}
		}
	}


}


/*the draw function to display the element*/
void GraphicsDisplayModel::draw()
{

}


/*control the play*/
void GraphicsDisplayModel::Start()
{
	_Play = true;
	_PlayStartTime = _timer->GetGraphicTime();
	_PlayEndTime = _PlayStartTime + _Duration;
```

```
    _StartList = _ElementList;

    _isDone = false;
}





GraphicsElement* GraphicsDisplayModel::CreateInstance()
{

    return NULL;

}



void    GraphicsDisplayModel::CreateElement()
{
    for(int i=0; i< _DataList.size();i++)
    {
        BoxData* x = ((BoxData*)_DataList[i]);

        int j = 0;
    }
}




/*put the created model into to list*/
void GraphicsDisplayModel::AddElement(int type, int additonal)
{
    //create and add the graphics model to the list
    BoxData* boxData = NULL;


    switch(type)
    {
        case ELEMENT_POINT:

            break;
```

```
		case ELEMENT_LINE:

			break;

		case ELEMENT_BOX:
			boxData = new BoxData();
			boxData->_Color = _gColor;

			_DataList.push_back(boxData);

			break;

		case ELEMENT_CYLINDER:

			break;

		case ELEMENT_SLICE:

			break;

		case ELEMENT_PLANE:

			break;

		case ELEMENT_AXIS:

			break;


		default:
			break;
	}


}

/*get the last element in list*/
GraphicsElement* GraphicsDisplayModel::GetCurrentElement()
{
	int index = _ElementList.size()-1;
	if(index > 0)
		return _ElementList[0];

	return NULL;
```

```
}



void      GraphicsDisplayModel::SetShowTime(float start, float duration)
{

}


void      GraphicsDisplayModel::SetUnShowTime(float start, float duration)
{

}


void      GraphicsDisplayModel::SetPositionTime(float start, float duration)
{

}


void      GraphicsDisplayModel::SetRotationTime(float start, float duration)
{

}

void      GraphicsDisplayModel::SetScalingTime(float start, float duration)
{

}



void GraphicsDisplayModel::addOneVertex(float x, float y, float z)
{
     if(_DataList.size()==0)
          return;

     int index = _DataList.size() -1;
     //------------------------------------------------------------check different type
     ModelData* current = _DataList[index];
```

```
	switch(current->_ModelType)
	{

		case MODEL_BOX:
			((BoxData*)current)->_position = Vec3(x,y,z);
			break;

		case MODEL_CYLINDER:
			break;



		default:
			break;
	}
}







void GraphicsDisplayModel::SetBoxParameter(float x, float y, float z)
{
	if(_DataList.size()==0)
		return;

	int index = _DataList.size() -1;
	//------------------------------------------------------------check different type
	ModelData* current = _DataList[index];
	if( current->_ModelType != MODEL_BOX)
	{
		printf("wrong order in storing the elements data\n");
		exit(-1);
	}


	((BoxData*)current)->_height = z;
	((BoxData*)current)->_length = x;
	((BoxData*)current)->_width = y;

	return;
```

```
}




void GraphicsDisplayModel::SetCylinderParameter(float r, float h)
{
	if(_ElementList.size()==0)
		return;

	int index = _ElementList.size() -1;
	//------------------------------------------------------------check different type
	GraphicsElement* current = _ElementList[index];
	if( current->_ElementType != ELEMENT_CYLINDER)
		return;
	else
		((GraphicsCylinder*)current)->SetCylinderParameter(r, h);

	return;
}





void GraphicsDisplayModel::SetTimeForCurrent(float s, float d, float mode)
{
	float start;
	float end;

	if( s <= 0)	//use cumulative time
		start = _cumulativeTime;
	else
		start = s;

	if( d < 0)
		end = _cumulativeTime + 1.0;	//default length	1 s
	else
		end = start + d;



	if(_ElementList.size()==0)
		return;
```

```
        int index = _ElementList.size() -1;
        //-----------------------------------------------------------check different type
        GraphicsElement* current = _ElementList[index];
        current->SetStartEndTime(start, end);
        current->SetAnimationFlag((int)mode);




        //update the current cumulative time
        if( end > _cumulativeTime)
                _cumulativeTime = end;

        return;
}




void GraphicsDisplayModel::GenerateCurrent()
{

}
```

Interface of front-end & back-end.

DDRL.hpp

```
#ifndef    DDRL_APPOBJ_H
#define    DDRL_APPOBJ_H

#include <vector>
#include <string>
#include <antlr/CommonAST.hpp>
#include <antlr/ASTFactory.hpp>

#include "../SymbolTable.h"
#include "../DataType.h"
#include "../DeclarationTable.h"

class DataTypeDisplayModel;


using namespace std;

using namespace antlr;


#define    SCOPE_MAIN                    -3
#define    SCOPE_DM                 -2
#define    SCOPE_GLOBAL             -1
#define    SCOPE_FUNC                    0
#define    SCOPE_SUB                1
#define    SCOPE_LOOP                    2
#define    SCOPE_STRUCT             3

#define    NONE_DEFINE              0
#define     MODEL_DEFINING               1
#define    STRUCT_DEFINING          2




//the operand type
#define    OPERAND_NUM              0
#define    OPERAND_STR              1
```

```
#define    OPERAND_ID                  2

#define    OPERAND_OTHER               -1


#define    TABLE_DECL                  0
#define    TABLE_SYMBOL            1



typedef    DataType*               pDataType;

class      DDRLWalker;


class    AppObj
{

public:

        /*constructor*/
        AppObj();
        ~AppObj();


        /*output the information */
        void output(pDataType data);


        //basic mathematic operation,
        //----------------------------------------------------------------------
        pDataType GetpDataType(string operandName);

        pDataType GetpDataTypeComplexID(string operandName);

        pDataType GetDataTypeInStruct(DataType* structData, string member);

        pDataType GetpDataTypeArray(string operandName, pDataType size);

        pDataType GetpDataTypeArray(string operandName, float index);

        /*add two number, can apply to Number and String*/
        pDataType add(pDataType a, pDataType b);
```

```
/*do minus, can be both Number or String,    a - b*/
pDataType minus(pDataType a, pDataType b);


/*do multiply, only can applied to both Number*/
pDataType mult(pDataType a, pDataType b);


/*do divide, only can applied to both Number*/
pDataType div(pDataType a, pDataType b);


/*exponential operation*/
pDataType exp(pDataType a, pDataType b);


/*do equal test, can apply to both Number and String*/
pDataType equal(pDataType a, pDataType b);


/*do not equal test, can apply to both Number and String*/
pDataType nequal(pDataType a, pDataType b);


/*check whether a is less than b, can apply to both Number and String*/
pDataType less(pDataType a, pDataType b);


/*check whether a is less than or equal to b, can apply to both Number and String*/
pDataType lessequal(pDataType a, pDataType b);


/*check whether a is greater than b, can apply to both Number and String*/
pDataType greater(pDataType a, pDataType b);


/*check whether a is greater than or equal to b, can apply to both Number and String*/
pDataType greatequal(pDataType a, pDataType b);


/* negate the a, can only apply to Number*/
pDataType negate(pDataType a);
```

```
/* get the value of the variable, can apply to Number and String*/
pDataType getValue(pDataType varName);

/* get the value of the array element*/
pDataType getArrValue(pDataType arrayName, float index);

/*Set the value to a pDataType variable*/
void assignValue(pDataType varName, pDataType value);


/*start a new scope, 0 - non-track up, 1 - track up*/
float newScope(int type);


/*end of the current scope*/
float closeScope();

float returnStmt();

float breakStmt();

float continueStmt();


/*Create a new Variable, and put it in the current scope symbol table*/
void createNewVar(string name, int type);


/*Add a struct variable to the symbol table*/
void addStructVar(string structName, string name, int type);


/*Create an Array variable , put into current scope symbol table*/
void addArray(string type, string name,int size);


/*Create a struct definition, put into declaration table*/
void addStruct(string name, int type);


/*Enter the definition section of a struct*/
void enterStruct(string structName);
```

```
/*Leave the definition section of a struct*/
void leaveStruct();


/*for each element id1    in    array id2, loop */
float foreach(string id1, string id2, DDRLWalker* walker, antlr::RefAST loop_body);


/*Define a function, and store the AST tree for recall*/
void addFunc(string Name, antlr::RefAST subprogram, string r);


/*Set the parameters for function call, which will be called following*/
int    setFuncArg(pDataType argValue);
//void setFuncArg(std::pDataType arg)


/*get the arg from the stack*/
int ConsumeArg(string name, int type);


/*Call a function, to execute the code in this predefined function*/
pDataType funcCall(DDRLWalker* walker, string funcName);




void    DeclArg(string name, CLASSTYPE type);

void    DeclArg(string baseTypename, string name, CLASSTYPE type);




//graphic display function are declared below
//--------------------------------------------------------------------
void    addDisplayVar(string basename, string varname, int type);

void    enterDmfunction(string modelName, int table);

void    leaveDmfunction();

void    setDispStmt(DDRLWalker*, antlr::RefAST node);
```

```
/*Set the Time tag for begin time and duration of animation*/
int setTimeTag(pDataType begin_time, pDataType duration, float repeatnode);

/*Set the Color for the following graphics element, color will remain until next setting*/
int setColor(pDataType red, pDataType green, pDataType blue, pDataType alpha);

/*Set the Point Size for graphics point element, will remain until next setting*/
int setPointSize(pDataType PointSize);

/*Set the Line width for graphics line element, will remain until next setting*/
int setLineWidth(pDataType lineWidth);

/*Add a Vertex to current context and current defining graphics element*/
int addVertex(pDataType x, pDataType y, pDataType z);

/**/
int setPointScope(float);

int setLineScope(float);


int   setTmeshScope(float);


int setBoxScope(float);

/*set the parameter for box element*/
int setBoxSize(pDataType lenght, pDataType bwidth, pDataType height);

int setCylinderScope(float);

/*Set the parameter for the cylinder element*/
```

```
int    setCylinderSize(pDataType radius, pDataType height);

int setSliceScope(float);

/*set the parameter for Slice*/
int    setSliceSize(pDataType radius, pDataType height, pDataType start, pDataType angle);

int    setPlaneScope(float);

/*set the parameter for Axis*/
int setAxis(float axis , pDataType title, pDataType length,    pDataType interval);

/*set the marker on the axis with text*/
int setAxisMarker(pDataType index, pDataType text);

/*setup a text on 3D position*/
int    setText(pDataType text, pDataType sx, pDataType xy, pDataType sz, pDataType ex, pDataType ey, pDataType ez);

int controlGraphicTimeTag(pDataType begin_time,pDataType duration,float repeatnode);

int controlGraphic(float show, string modelName);

int    controlPosition(string modelName, pDataType x, pDataType y, pDataType z);

int    controlRotation(string model, pDataType x, pDataType y, pDataType z);

int    controlScale(string model, pDataType scale);

pDataType    addDisp(string Name, DDRLWalker* walker, RefAST node);
```

```
        void    dispNow(DDRLWalker* walker);


private:

        DeclarationTable*               _Declaration;
        VariableList*                   _SymbolTalbe;
        DataTypeFactory*                _dataFactory;



        int                                     _whichdefining;

        static      DataTypeStruct*             _tempDefiningStruct;                //to store the struct that
is being defined.
        static    DataTypeDisplayModel* _tempDefiningModel;
        static    float                 _tempStartTime;
        static    float                 _tempDuration;
        static    int                   _tempShowMode;

        DDRLWalker*                             _walker;
        vector<DataType*>               _argList;

public:
        static ASTFactory                               my_factory;


};

void antlr_main(std::string);

#endif
```

DDRLback.cpp

```
#include "ddrl.hpp"
#include <antlr/CommonAST.hpp>
#include <fstream>
#include "DDRLLexer.hpp"
#include "DDRLParser.hpp"
#include "DDRLWalker.hpp"
#include "DDRLTokenTypes.hpp"

#include "../plt.h"
#include "../pch.h"
#include "../DataTypeDisplayModel.h"

#include "../Global.h"



using namespace std;

ASTFactory              AppObj::my_factory;
DataTypeStruct*         AppObj::_tempDefiningStruct = NULL;
DataTypeDisplayModel*           AppObj::_tempDefiningModel = NULL;
float                   AppObj::_tempStartTime = 0.0;
float                   AppObj::_tempDuration = 0.0;
int                         AppObj::_tempShowMode = 0;





void antlr_main(std::string filename)
{

    fstream input;
    input.open(filename.c_str());

    if(!input.is_open())
    {
        printf("DDRL ERROR: can't find the source code file %s\n", filename.c_str());
        exit(-1);
    }
```

```
	DDRLLexer lexer(input);
	DDRLParser* parser = new DDRLParser(lexer);
	parser->initializeASTFactory(AppObj::my_factory);
	parser->setASTFactory(&AppObj::my_factory);

	DDRLWalker* walker = new DDRLWalker();

	parser->program();

	antlr::RefAST t = parser->getAST();



	walker->program(t.get());


	delete(parser);

//	delete(walker);

}




//=================================================================
==========================
// the application object , the interface for the front end and back end.
//=================================================================
==========================

//constructor
//---------------------------------------------------------------------------------------------
AppObj::AppObj()
{
	_Declaration = DeclarationTable::Instance();
	_SymbolTalbe = VariableList::Instance();
	_dataFactory = DataTypeFactory::Instance();
	_whichdefining = 0;

}

AppObj::~AppObj()
```

```
{
        //_SymbolTalbe->ClearVariableList();
        //_Declaration->ClearTable();
}
//----------------------------------------------------------------------------------------------




void AppObj::output(pDataType data)
{
        if(data == NULL)
                return;

        if( data->GetDataType() == DATATYPE_STRING)
        {
                DataTypeString*    outputstring = (DataTypeString*)data;
                printf("%s\n",outputstring->GetStringValue().c_str());
                return;
        }
        else if( data->GetDataType() == DATATYPE_NUM)
        {
                DataTypeNum*    outputnum = (DataTypeNum*)data;
                printf("%f\n",outputnum->GetNumValue());
                return;
        }
        else
        {
                printf("can only output the string and number\n");
        }
}




// Create a new variable into the symbol table
//
// @return 0:    success
// @return -1: if data type not found
// @return -2: if variable name already exists
//----------------------------------------------------------------------------------------------
void AppObj::createNewVar(string name, int type)
{
```

```
//at very first, we can not use the '[' ']' '.' in name
string::size_type pos1 = name.find('.',0);
string::size_type pos2 = name.find('[',0);
string::size_type pos3 = name.find(']',0);

if(pos1 != string::npos   || pos2 != string::npos   || pos3 != string::npos)
{
    printf("illegal charator in variable name:   %s\n", name.c_str());
    exit(-1);
}


//check the name and the type of the variable
//1. check the name, whether there is a same name variable in current scope
Variable   var;
if( _whichdefining == NONE_DEFINE)
    var = _SymbolTalbe->IsVariableExist(name);
else if( _whichdefining == STRUCT_DEFINING)
    var = _tempDefiningStruct->IsVariableExist(name);
else
    var = _tempDefiningModel->IsVariableExist(name);


if( var != NULL)                    //variable redefine
{
    printf("variable redefine:   %s\n", name.c_str());
    exit(-1);
}

//2. according to the type create the variable
DataType* data = NULL;
switch(type)
{
    case DATATYPE_NUM:
        data = _dataFactory->CreateNumber(name);

        //if it is normal declare, put in symbol table,
        //else, it is struct define, put in the struct define
        if( _whichdefining == NONE_DEFINE)
            _SymbolTalbe->AddNewVariable(name, data);
        else if( _whichdefining == STRUCT_DEFINING)
        {
            _tempDefiningStruct->AddVariable(name, data);
            printf("                    --:" );
        }
        else
        {
            _tempDefiningModel->AddVariable(name, data);
            printf("                    --:" );
        }

        break;


    case DATATYPE_STRING:
        data = _dataFactory->CreateString(name);

        //if it is normal declare, put in symbol table,
        //else, it is struct define, put in the struct define
        if( _whichdefining == NONE_DEFINE)
            _SymbolTalbe->AddNewVariable(name, data);
        else if( _whichdefining == STRUCT_DEFINING)
        {
            _tempDefiningStruct->AddVariable(name, data);
            printf("                    --:" );
        }
        else
        {
            _tempDefiningModel->AddVariable(name, data);
            printf("                    --:" );
        }

        break;


    case DATATYPE_DATASET:
        data = _dataFactory->CreateDataSet(name);

        //if it is normal declare, put in symbol table,
        //else, it is struct define, put in the struct define
        if( _whichdefining == NONE_DEFINE)
            _SymbolTalbe->AddNewVariable(name, data);
        else if( _whichdefining == STRUCT_DEFINING)
        {
            _tempDefiningStruct->AddVariable(name, data);
            printf("                    --:" );
        }
        else
```

```
            {
                _tempDefiningModel->AddVariable(name, data);
                printf("                              --:" );
            }
            break;


        default:
            break;
    }

    printf("---define variable: %s\n", name.c_str());
    return;
}



/*Add a struct variable to the symbol table*/
void AppObj::addStructVar(string structName, string name, int type)
{

    //at very first, we can not use the '[' ']' '.' in name
    string::size_type pos1 = name.find('.',0);
    string::size_type pos2 = name.find('[',0);
    string::size_type pos3 = name.find(']',0);

    if(pos1 != string::npos   || pos2 != string::npos   || pos3 != string::npos)
    {
        printf("illegal charator in variable name:   %s\n", name.c_str());
        exit(-1);
    }



    //1. check whether the struct base name is defined.
    DataType* baseType = _Declaration->IsDeclarationExist(structName.c_str());
    if(baseType == NULL)
    {
        printf("Struct base type not defined:   %s\n", structName.c_str());
        return;
    }

    Variable   var;
    if( _whichdefining == NONE_DEFINE)
```

```
        var = _SymbolTalbe->IsVariableExist(name);
    else if( _whichdefining == STRUCT_DEFINING)
        var = _tempDefiningStruct->IsVariableExist(name);
    else
        var = _tempDefiningModel->IsVariableExist(name);

    if( var != NULL)                    //variable redefine
    {
        printf("variable redefine:    %s\n", name.c_str());
        return;
    }


    DataTypeStruct*     data = _dataFactory->CreateStruct(name, structName);

    if( data != NULL)
    {
        //if it is normal declare, put in symbol table,
        //else, it is struct define, put in the struct define
        if( _whichdefining == NONE_DEFINE)
            _SymbolTalbe->AddNewVariable(name, data);
        else if( _whichdefining == STRUCT_DEFINING)
        {
            _tempDefiningStruct->AddVariable(name, data);
            printf("                    --:" );
        }
        else
        {
            _tempDefiningModel->AddVariable(name, data);
            printf("                    --:" );
        }
    }

    printf("---define struct variable: %s\n", name.c_str());
    return;
}




/*Create an Array variable , put into current scope symbol table*/
void AppObj::addArray( string type, string name, int size)
{
```

```
//at very first, we can not use the '[' ']' '.' in name
string::size_type pos1 = name.find('.',0);
string::size_type pos2 = name.find('[',0);
string::size_type pos3 = name.find(']',0);

if(pos1 != string::npos    || pos2 != string::npos    || pos3 != string::npos)
{
        printf("illegal charator in variable name:    %s\n", name.c_str());
        exit(-1);
}



//1. check the name, whether there is a same name variable in current scope

//check the type whether defined in declaration table
DataType* baseType = _Declaration->IsDeclarationExist(type);
if(baseType == NULL)
{
        printf("the type of array elements not defined:    %s\n", type.c_str());
        return;
}

Variable    var;
if( _whichdefining == NONE_DEFINE)
        var = _SymbolTalbe->IsVariableExist(name);
else if( _whichdefining == STRUCT_DEFINING)
        var = _tempDefiningStruct->IsVariableExist(name);
else
        var = _tempDefiningModel->IsVariableExist(name);


if( var != NULL)                            //variable redefine
{
        printf("variable redefine:    %s\n", name.c_str());
        return;
}


DataTypeArray* data = _dataFactory->CreateArray(name, type, size);


{
        //if it is normal declare, put in symbol table,
        //else, it is struct define, put in the struct define
        if( _whichdefining == NONE_DEFINE)
            _SymbolTalbe->AddNewVariable(name, data);
        else if( _whichdefining == STRUCT_DEFINING)
        {
            _tempDefiningStruct->AddVariable(name, data);
            printf("                    --:" );
        }
        else
        {
            _tempDefiningModel->AddVariable(name, data);
            printf("                    --:" );
        }
    }

    printf("---defined Array [%s][%d:   %s\n", type.c_str(), size, name.c_str());
    return;
}



/*Create a struct definition, put into declaration table*/
void AppObj::addStruct(string name, int type)
{

    //at very first, we can not use the '[' ']' '.' in name
    string::size_type pos1 = name.find('.',0);
    string::size_type pos2 = name.find('[',0);
    string::size_type pos3 = name.find(']',0);

    if(pos1 != string::npos   || pos2 != string::npos   || pos3 != string::npos)
    {
        printf("illegal charator in variable name:   %s\n", name.c_str());
        exit(-1);
    }


    //1. check whether this kind of struct is defined
    DataType* baseType = _Declaration->IsDeclarationExist(name);
    if(baseType != NULL)
    {
        printf("the type of struct is already defined:   %s\n", name);
        return;
        }


        //2. create the definition of this struct in declaration table
        int reval = _dataFactory->CreateBaseDataType(name, DATATYPE_STRUCT);
        if(reval != 0)
        {
                printf("--can not declear the struct: %s\n",name.c_str());
                exit(-1);

        }

        printf("---decleared struct base type: %s\n",name.c_str());
        return;
}



/*Enter the definition section of a struct*/
void AppObj::enterStruct(string structName)
{
        //1. check whether this kind of struct is defined
        DataType* baseType = _Declaration->IsDeclarationExist(structName);
        if(baseType == NULL || baseType->GetDataType() != DATATYPE_STRUCT)
        {
                printf("the type of struct is not defined or not a struct:    %s\n", structName.c_str());
                return;
        }

        //create the scope of this structure definition.
        _SymbolTalbe->AddNewSymbolTable(false,SCOPE_STRUCT);

        _whichdefining = STRUCT_DEFINING;

        _tempDefiningStruct = (DataTypeStruct*)baseType;

}



/*Leave the definition section of a struct*/
void AppObj::leaveStruct()
{
        _tempDefiningStruct = NULL;
```

```
    _whichdefining = NONE_DEFINE;
    //close the scope of the structure being defined.
    _SymbolTalbe->DeleteSymbolTable();

}




//basic mathematic operation,
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//create the temp memory of DataType, to store the constant operand, which is passed into
function
//----------------------------------------------------------------------------------------------
pDataType AppObj::GetpDataType(string operandName)
{
    pDataType    reval = NULL;


    //support string adding
    if(operandName[0] == '"')            //this is a string, create the temp dataType to store
    {
        string value = operandName.substr(1,operandName.size()-2);
        string tempName = _dataFactory->CreateTempName();

        DataTypeString* str = _dataFactory->CreateString(tempName);

        str->SetStringValue(value);

        _SymbolTalbe->AddNewVariable(tempName, str);
        reval = str;
        return reval;
    }

    //support number adding
    if((operandName[0] >= '0' && operandName[0] <= '9'))            //this is a const number,
create temp
    {
        float val = (float)atof(operandName.c_str());
        string tempName = _dataFactory->CreateTempName();
        DataTypeNum*    num = _dataFactory->CreateNumber(tempName);

        num->SetNumValue(val);
        _SymbolTalbe->AddNewVariable(tempName, num);

        reval = num;
        return reval;
    }


    //here must be an Identifier, we get its pointer
    //check the struct and array and ID.

    //....
    //....
    //....
    // deal with the operandName
    //////////////////////////////////////////////////////////////////////////////////
    string subname = operandName;
    DataType*    data = NULL;
    string::size_type loc1 = subname.find("[",0);
    string::size_type loc2 = subname.find(".",0);

    //if no [] and . in name
    if( loc1 == string::npos && loc2 == string::npos)
    {
        pDataType data=NULL;


        //decide to look in symbol table or local struct member variable
        if( _whichdefining == NONE_DEFINE)
            data = _SymbolTalbe->IsVariableExist(operandName);
        else if( _whichdefining == STRUCT_DEFINING)
            data = _tempDefiningStruct->IsVariableExist(operandName);
        else
            data = _tempDefiningModel->IsVariableExist(operandName);

        if(data == NULL)
        {
            printf("variable not defined:    %s\n",operandName.c_str());
            exit(-1);
        }
        return data;
    }
    else
    {
        data = GetpDataTypeComplexID(operandName);
        return data;
    }




}








//get the data type pointer of a ID type variable, not the constant
//--------------------------------------------------------------------------------------------
pDataType AppObj::GetpDataTypeComplexID(string operandName)
{
    pDataType    structData = NULL;


    //check    and the first '.'
    string::size_type dotpos = operandName.find(".", 0);

    if( dotpos == 0)
    {
        printf("illegal format of variable, '.' can not be the first charactor:    %s\n", operandName.c_str());
        exit(-1);
    }


    // format
    // xxxx.xx.xxx
    // xxx[y].xxx.xxx.xxx.

    string rightID;
    string firstID;
    bool    contiueGo = true;
    if( dotpos == string::npos)
    {
        contiueGo = false;                        //we can return now, we find the end
        firstID = operandName;
    }
```

```
else
{
    firstID = operandName.substr(0,dotpos);
    rightID = operandName.substr(dotpos+1);
}
//check whether has [] in firstID
string::size_type apos = firstID.find("[",0);


if(apos == string::npos)
{
    //firstID is not an array, we check the symbol table as a simple ID

    //decide to look in symbol table or local struct member variable
    if( _whichdefining == NONE_DEFINE)
        structData = _SymbolTalbe->IsVariableExist(firstID);
    else if( _whichdefining == MODEL_DEFINING)
        structData = _tempDefiningModel->IsVariableExist(firstID);
    else
        structData = _tempDefiningStruct->IsVariableExist(firstID);




    if(structData == NULL)
    {
        printf("variable not defined:    %s\n", firstID.c_str());
        exit(-1);
    }

    pDataType member;
    if(contiueGo == true)
        return member = GetDataTypeInStruct((DataTypeStruct*)structData, rightID);
    else
        return structData;

}
else        //firstID is an array variable..
{
    string::size_type loc1 = firstID.find("[",0);
    string::size_type loc2 = firstID.find("]",0);

    if( loc2 == string::npos || loc2 != firstID.size()-1)
    {
```

```
    printf("illegal format of array variable: %s\n", firstID.c_str());
    exit(-1);
}


//get the pDataType of this array's element
//----------------------------------------------------------------
//get the index number between []
int elementIndex = 0;
string varname;

string number = firstID.substr(loc1+1,loc2-loc1-1);
if(number[0] >= '0' && number[0] <= '9')
{
    if(loc1 == string::npos)
        elementIndex = 0;
    else
        elementIndex = atoi(number.c_str());
}
else
{
    pDataType n = _SymbolTalbe->IsVariableExist(number);
    if(n==NULL || n->GetDataType() != DATATYPE_NUM)
    {
        printf("variable not defined: %s\n", number.c_str());
        exit(-1);
    }

    elementIndex = ((DataTypeNum*)n)->GetNumValue();
}


varname = firstID.substr(0,loc1);
pDataType arrayData = _SymbolTalbe->IsVariableExist(varname);
if(arrayData == NULL)
{
    printf("Variable not defined: %s\n", firstID.c_str());
    exit(-1);
}

if(arrayData->GetDataType() == DATATYPE_ARRAY)
{
    structData = ((DataTypeArray*)arrayData)->GetElement(elementIndex);
}
```

```
        else
        {
            printf("attemp to use [], but not an array: %s\n", varname.c_str());
            exit(-1);
        }

        pDataType member;
        if( contiueGo == true)
            return member = GetDataTypeInStruct((DataTypeStruct*)structData, rightID);
        else
            return structData;

    }



}






//get the data type pointer of struct member variable
//-------------------------------------------------------------------------------------------
pDataType AppObj::GetDataTypeInStruct(DataType* structData, string member)
{
    pDataType varmember = NULL;

    if( structData->GetDataType() != DATATYPE_STRUCT && structData->GetDataType() !=
DATATYPE_DISPLAYM)
    {
        printf(" Not an struct type:    %s\n", structData->GetName().c_str());
        exit(-1);
    }


    bool    contiueGo = true;
    string::size_type dotpos = member.find(".", 0);

    if( dotpos == 0)
    {
        printf("illegal format of variable, '.' can not be the first charactor:    %s\n",
member.c_str());
```

```
        exit(-1);
}

string rightID;
string firstID;

if( dotpos == string::npos)
{
        contiueGo = false;                       //we can return now, we find the end
        firstID = member;
}
else
{
        firstID = member.substr(0,dotpos-1);
        rightID = member.substr(dotpos+1);
}




DataType*       nextStruct = NULL;
//deal as the same as the complexID, but find the variable in struct inside
// format
// xxxx.xx.xxx
// xxx[y].xxx.xxx.xxx.


//check whether has [] in firstID
string::size_type apos = firstID.find("[",0);

if(apos == string::npos)
{
        //firstID is not an array, we check the symbol table as a simple ID
        if(structData->GetDataType() == DATATYPE_STRUCT)
                nextStruct = ((DataTypeStruct*)structData)->IsVariableExist(firstID);
        else if(structData->GetDataType() == DATATYPE_DISPLAYM)
                nextStruct = ((DataTypeDisplayModel*)structData)->IsVariableExist(firstID);


        if(structData == NULL)
        {
                printf("variable not defined:    %s\n", firstID.c_str());
                exit(-1);
        }
        pDataType nextLevel = NULL;
        if( contiueGo == false)
                return nextStruct;
        else
                nextLevel = GetDataTypeInStruct(nextStruct, rightID);

        return nextLevel;

}
else            //firstID is an array variable..
{
        string::size_type loc1 = firstID.find("[",0);
        string::size_type loc2 = firstID.find("]",0);

        if( loc2 == string::npos || loc2 != firstID.size()-1)
        {
                printf("illegal format of array variable: %s\n", firstID.c_str());
                exit(-1);
        }


        //get the pDataType of this array's element
        //---------------------------------------------------------------
        //get the index number between []
        int elementIndex = 0;
        string varname;

        string number = firstID.substr(loc1+1,loc2-loc1-1);
        if(number[0] >= '0' && number[0] <= '9')
        {
                if(loc1 == string::npos)
                        elementIndex = 0;
                else
                        elementIndex = atoi(number.c_str());
        }
        else
        {
                pDataType n = _SymbolTalbe->IsVariableExist(number);
                if(n==NULL || n->GetDataType() != DATATYPE_NUM)
                {
                        printf("variable not defined: %s\n", number.c_str());
                        exit(-1);
                }
        elementIndex = ((DataTypeNum*)n)->GetNumValue();
    }


    varname = firstID.substr(0,loc1);


    pDataType arrayData = NULL;
    if(structData->GetDataType() == DATATYPE_STRUCT)
        arrayData = ((DataTypeStruct*)structData)->IsVariableExist(varname);
    else if(structData->GetDataType() == DATATYPE_DISPLAYM)
        arrayData = ((DataTypeDisplayModel*)structData)->IsVariableExist(varname);



    if(arrayData == NULL)
    {
        printf("Variable not defined: %s\n", firstID.c_str());
        exit(-1);
    }

    if(arrayData->GetDataType() == DATATYPE_ARRAY)
    {
        nextStruct = ((DataTypeArray*)arrayData)->GetElement(elementIndex);
    }
    else
    {
        printf("attemp to use [], but not an array: %s\n", varname.c_str());
        exit(-1);
    }

    pDataType nextLevel = NULL;
    if( contiueGo == false)
        return nextStruct;
    else
        nextLevel = GetDataTypeInStruct((DataTypeStruct*)nextStruct, rightID);

    return nextLevel;


}
```

```
}




//get the data type pointer in an array variable
//-----------------------------------------------------------------------------------
pDataType AppObj::GetpDataTypeArray(string operandName, pDataType index)
{
	string arrayName = operandName.substr(0, operandName.size()-1);

	DataType*		arrayData = NULL;


	if( _whichdefining == NONE_DEFINE)
		arrayData = _SymbolTalbe->IsVariableExist(arrayName);
	else if( _whichdefining == MODEL_DEFINING)
		arrayData = _tempDefiningModel->IsVariableExist(arrayName);
	else
		arrayData = _tempDefiningStruct->IsVariableExist(arrayName);


	if( arrayData == NULL)
	{
		printf(" Variable doesn't exist:    %s\n", arrayName);
		exit(-1);
	}

	if( arrayData->GetDataType() != DATATYPE_STRING)
	{
		printf(" Not an array type:    %s\n", arrayName);
		exit(-1);
	}

	if(index->GetDataType() != DATATYPE_NUM)
	{
		printf("array ' index ' format not correct:\n" );
		exit(-1);
	}

	DataTypeNum*    number = (DataTypeNum*)index;
	pDataType                                        element                        =
((DataTypeArray*)arrayData)->GetElement(number->GetNumValue());
```

```
    return element;
}








pDataType AppObj::GetpDataTypeArray(string operandName, float index)
{

    string arrayName = operandName.substr(0, operandName.size()-1);

    int intIndex = (int)index;
    DataType*           arrayData = NULL;
    if( _whichdefining == NONE_DEFINE)
        arrayData = _SymbolTalbe->IsVariableExist(arrayName);
    else if( _whichdefining == MODEL_DEFINING)
        arrayData = _tempDefiningModel->IsVariableExist(arrayName);
    else
        arrayData = _tempDefiningStruct->IsVariableExist(arrayName);


    if( arrayData == NULL)
    {
        printf(" Variable doesn't exist:    %s\n", arrayName);
        exit(-1);
    }

    pDataType    element = ((DataTypeArray*)arrayData)->GetElement(intIndex);

    return element;
}




//----------------------------------------------------------------------
/*add two number, can apply to Number and String*/
pDataType AppObj::add(pDataType a, pDataType b)
{
    //1. get the type of first operand
    if( a->GetDataType() != b->GetDataType())
    {
        printf("two variables are not same type, can not add!\n");
        exit(-1);
    }

    //2. if it is the same, we now only support Number add and String add
    if(a->GetDataType() == DATATYPE_NUM)
    {
        float add_value = ((DataTypeNum*)a)->GetNumValue() +
((DataTypeNum*)b)->GetNumValue();

        string tempName = _dataFactory->CreateTempName();
        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
        reval->SetNumValue(add_value);
        _SymbolTalbe->AddNewVariable(tempName, reval);

        return (DataType*)reval;
    }
    else if( a->GetDataType() == DATATYPE_STRING)
    {
        string add_value = ((DataTypeString*)a)->GetStringValue() +
((DataTypeString*)b)->GetStringValue();

        string tempName = _dataFactory->CreateTempName();
        DataTypeString* reval = _dataFactory->CreateString(tempName);
        reval->SetStringValue(add_value);
        _SymbolTalbe->AddNewVariable(tempName, reval);

        return (DataType*)reval;
    }
    else
    {
        //we not support this type of operation
        printf("this data type doesn't not support adding:   %s     %s\n", a->GetName().c_str(),
b->GetName().c_str());
        exit(-1);
    }


}
```

```
/*do minus, can be both Number or String,   a - b*/
pDataType AppObj::minus(pDataType a, pDataType b)
{


    //1. get the type of first operand
    if( a->GetDataType() != b->GetDataType())
    {
        printf("two variables are not same type, can not minus!\n");
        exit(-1);
    }

    //2. if it is the same, we now only support Number add and String add
    if(a->GetDataType() == DATATYPE_NUM)
    {
        float minus_value = ((DataTypeNum*)a)->GetNumValue() - ((DataTypeNum*)b)->GetNumValue();

        string tempName = _dataFactory->CreateTempName();
        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
        reval->SetNumValue(minus_value);
        _SymbolTalbe->AddNewVariable(tempName, reval);
        return (DataType*)reval;
    }
    else
    {
        //we not support this type of operation
        printf("this data type doesn't not support minus :   %s   %s\n", a->GetName().c_str(), b->GetName().c_str());
        exit(-1);
    }


}
```

```
/*do multiply, only can applied to both Number*/
pDataType AppObj::mult(pDataType a, pDataType b)
{


	//1. get the type of first operand
	if( a->GetDataType() != b->GetDataType())
	{
		printf("two variables are not same type, can not multiply!\n");
		exit(-1);
	}

	//2. if it is the same, we now only support Number add and String add
	if(a->GetDataType() == DATATYPE_NUM)
	{
		float mutiply_value = ((DataTypeNum*)a)->GetNumValue()*((DataTypeNum*)b)->GetNumValue();

		string tempName = _dataFactory->CreateTempName();
		DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
		reval->SetNumValue(mutiply_value);
		_SymbolTalbe->AddNewVariable(tempName, reval);
		return (DataType*)reval;
	}
	else
	{
		//we not support this type of operation
		printf("this data type doesn't not support mutiply :	%s	%s\n", a->GetName().c_str(), b->GetName().c_str());
		exit(-1);
	}


}




/*do divide, only can applied to both Number*/
pDataType AppObj::div(pDataType a, pDataType b)
{

	//1. get the type of first operand
	if( a->GetDataType() != b->GetDataType())
	{
		printf("two variables are not same type, can not divide!\n");
		exit(-1);
	}

	//2. if it is the same, we now only support Number add and String add
	if(a->GetDataType() == DATATYPE_NUM)
	{
		float second = ((DataTypeNum*)b)->GetNumValue();

		if( 0.0 == second)
		{
			printf("divide by zeor:    exceptions %s", b->GetName().c_str());
			exit(-1);
		}

		float div_value = ((DataTypeNum*)a)->GetNumValue()/second;

		string tempName = _dataFactory->CreateTempName();
		DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
		reval->SetNumValue(div_value);
		_SymbolTalbe->AddNewVariable(tempName, reval);
		return (pDataType)reval;
	}
	else
	{
		//we not support this type of operation
		printf("this data type doesn't not support divide :    %s      %s\n", a->GetName().c_str(),
b->GetName().c_str());
		exit(-1);
	}

	return NULL;
}




/*exponential operation*/
pDataType AppObj::exp(pDataType a, pDataType b)
{
	//1. get the type of first operand
	if( a->GetDataType() != b->GetDataType())
	{
		printf("two variables are not same type, can not use exponent!\n");
		exit(-1);
	}

	//2. if it is the same, we now only support Number add and String add
	if(a->GetDataType() == DATATYPE_NUM)
	{
		float first = ((DataTypeNum*)a)->GetNumValue();
		float ex = ((DataTypeNum*)b)->GetNumValue();

		float result = pow(first, ex);

		string tempName = _dataFactory->CreateTempName();
		DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
		reval->SetNumValue(result);
		_SymbolTalbe->AddNewVariable(tempName, reval);
		return (pDataType)reval;
	}
	else
	{
		//we not support this type of operation
		printf("this data type doesn't not support exponent : %s %s\n",
a->GetName().c_str(), b->GetName().c_str());
		exit(-1);
	}

	return NULL;
}



/*do equal test, can apply to both Number and String*/
pDataType AppObj::equal(pDataType a, pDataType b)
{

	//1. get the type of first operand
	if( a->GetDataType() != b->GetDataType())
	{
		printf("DDRL Error: two variables are not same type, can not check EQUAL!\n");
		exit(-1);
	}
```

```
//2. if it is the same, we now only support Number add and String add
if(a->GetDataType() == DATATYPE_NUM)
{
    float second = ((DataTypeNum*)a)->GetNumValue();
    float first = ((DataTypeNum*)b)->GetNumValue();

    if( first == second)
    {
        string tempName = _dataFactory->CreateTempName();
        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
        reval->SetNumValue(1);
        _SymbolTalbe->AddNewVariable(tempName, reval);
        return (pDataType)reval;
    }
    else
    {
        string tempName = _dataFactory->CreateTempName();
        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
        reval->SetNumValue(0);
        _SymbolTalbe->AddNewVariable(tempName, reval);
        return (pDataType)reval;
    }


}
else if( a->GetDataType() == DATATYPE_STRING)
{
    string second = ((DataTypeString*)a)->GetStringValue();
    string first = ((DataTypeString*)b)->GetStringValue();

    if( first == second)
    {
        string tempName = _dataFactory->CreateTempName();
        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
        reval->SetNumValue(1);
        _SymbolTalbe->AddNewVariable(tempName, reval);
        return (pDataType)reval;
    }
    else
    {
        string tempName = _dataFactory->CreateTempName();
        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
        reval->SetNumValue(0);
```

```
                        _SymbolTalbe->AddNewVariable(tempName, reval);
                        return (pDataType)reval;
                }



        }
        else
        {
                //we not support this type of operation
                printf("DDRL Error:   this two types can not check equality :   %s      %s\n",
a->GetName().c_str(), b->GetName().c_str());
                exit(-1);
        }



}



/*do not equal test, can apply to both Number and String*/
pDataType AppObj::nequal(pDataType a, pDataType b)
{



        //1. get the type of first operand
        if( a->GetDataType() != b->GetDataType())
        {
                printf("two variables are not same type, can not NOTEQUAL!\n");
                exit(-1);
        }

        //2. if it is the same, we now only support Number add and String add
        if(a->GetDataType() == DATATYPE_NUM)
        {
                float second = ((DataTypeNum*)a)->GetNumValue();
                float first = ((DataTypeNum*)b)->GetNumValue();

                if( first != second)
                {
                        string tempName = _dataFactory->CreateTempName();
                        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
                        reval->SetNumValue(1);
                        _SymbolTalbe->AddNewVariable(tempName, reval);
                        return (pDataType)reval;
                }
        else
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(0);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }

    }
    else if( a->GetDataType() == DATATYPE_STRING)
    {
        string second = ((DataTypeString*)a)->GetStringValue();
        string first = ((DataTypeString*)b)->GetStringValue();

        if( first != second)
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(1);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }
        else
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(0);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }

    }
    else
    {
        //we not support this type of operation
        printf("this two types can not check non-equality:    %s      %s\n", a->GetName().c_str(),
b->GetName().c_str());
        exit(-1);
    }


}
```

```
/*check whether a is less than b, can apply to both Number and String*/
pDataType AppObj::less(pDataType a, pDataType b)
{
    pDataType reval = NULL;

    //1. get the type of first operand
    if( a->GetDataType() != b->GetDataType())
    {
        printf("two variables are not same type, can not LESS!\n");
        exit(-1);
    }

    //2. if it is the same, we now only support Number
    if(a->GetDataType() == DATATYPE_NUM)
    {
        float second = ((DataTypeNum*)b)->GetNumValue();
        float first = ((DataTypeNum*)a)->GetNumValue();

        if( first < second)
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(1);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }
        else
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(0);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }

    }
    else
    {
        //we not support this type of operation
        printf("this two types can not check less :   %s     %s\n", a->GetName().c_str(),
b->GetName().c_str());
        exit(-1);
    }
```

```
        return reval;
}



/*check whether a is less than or equal to b, can apply to both Number and String*/
pDataType AppObj::lessequal(pDataType a, pDataType b)
{
        pDataType reval = NULL;

        //1. get the type of first operand
        if( a->GetDataType() != b->GetDataType())
        {
                printf("two variables are not same type, can not LESSEQUAL!\n");
                exit(-1);
        }

        //2. if it is the same, we now only support Number
        if(a->GetDataType() == DATATYPE_NUM)
        {
                float second = ((DataTypeNum*)b)->GetNumValue();
                float first = ((DataTypeNum*)a)->GetNumValue();

                if( first <= second)
                {
                        string tempName = _dataFactory->CreateTempName();
                        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
                        reval->SetNumValue(1);
                        _SymbolTalbe->AddNewVariable(tempName, reval);
                        return (pDataType)reval;
                }
                else
                {
                        string tempName = _dataFactory->CreateTempName();
                        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
                        reval->SetNumValue(0);
                        _SymbolTalbe->AddNewVariable(tempName, reval);
                        return (pDataType)reval;
                }


        }
        else
```

```
    {
        //we not support this type of operation
        printf("this two types can not check less equal :   %s     %s\n", a->GetName().c_str(),
b->GetName().c_str());
        exit(-1);
    }

    return reval;
}



/*check whether a is greater than b, can apply to both Number and String*/
pDataType AppObj::greater(pDataType a, pDataType b)
{
    pDataType reval = NULL;

    //1. get the type of first operand
    if( a->GetDataType() != b->GetDataType())
    {
        printf("two variables are not same type, can not GREATER!\n");
        exit(-1);
    }

    //2. if it is the same, we now only support Number
    if(a->GetDataType() == DATATYPE_NUM)
    {
        float second = ((DataTypeNum*)b)->GetNumValue();
        float first = ((DataTypeNum*)a)->GetNumValue();

        if( first > second)
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(1);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }
        else
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(0);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }
    }
    else
    {
        //we not support this type of operation
        printf("this two types can not check greater :  %s    %s\n", a->GetName().c_str(), 
b->GetName().c_str());
        exit(-1);
    }

    return reval;
}



/*check whether a is greater than or equal to b, can apply to both Number and String*/
pDataType AppObj::greatequal(pDataType a, pDataType b)
{
    pDataType reval = NULL;

    //1. get the type of first operand
    if( a->GetDataType() != b->GetDataType())
    {
        printf("two variables are not same type, can not GREATEQUAL!\n");
        exit(-1);
    }

    //2. if it is the same, we now only support Number
    if(a->GetDataType() == DATATYPE_NUM)
    {
        float second = ((DataTypeNum*)b)->GetNumValue();
        float first = ((DataTypeNum*)a)->GetNumValue();

        if( first >= second)
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(1);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }
        else
        {
            string tempName = _dataFactory->CreateTempName();
            DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
            reval->SetNumValue(0);
            _SymbolTalbe->AddNewVariable(tempName, reval);
            return (pDataType)reval;
        }
    }
    else
    {
        //we not support this type of operation
        printf("this two types can not check greater equal:    %s      %s\n", a->GetName().c_str(),
b->GetName().c_str());
        exit(-1);
    }

    return reval;
}


/* negate the a, can only apply to Number*/
pDataType AppObj::negate(pDataType a)
{
    pDataType reval = NULL;

    //2. if it is the same, we now only support Number
    if(a->GetDataType() == DATATYPE_NUM)
    {
        float value = ((DataTypeNum*)a)->GetNumValue();

        string tempName = _dataFactory->CreateTempName();
        DataTypeNum* reval = _dataFactory->CreateNumber(tempName);
        reval->SetNumValue(-value);
        _SymbolTalbe->AddNewVariable(tempName, reval);
        return (pDataType)reval;
    }
    else
    {
        //we not support this type of operation
        printf("this type can not negeate:    %s    \n", a->GetName().c_str());
        exit(-1);
    }

    return reval;
}
```

```
/* get the value of the variable, can apply to Number and String*/
pDataType AppObj::getValue(pDataType varName)
{
    pDataType reval;

    return reval;
}



/* get the value of the array element*/
pDataType AppObj::getArrValue(pDataType arrayName, float index)
{
    pDataType reval = NULL;

    if(arrayName->GetDataType() != DATATYPE_ARRAY)
    {
        printf("the variable is not an array: %s\n", arrayName->GetName().c_str());
        exit(-1);
    }

    reval = ((DataTypeArray*)arrayName)->GetElement((unsigned int)index);


    return reval;
}

/*Set the value to a pDataType variable*/
void AppObj::assignValue(pDataType varName, pDataType value)
{
    //right not we only support the Number, String,
    if(varName->GetDataType() == DATATYPE_NUM)
    {
         if( value->GetDataType() == DATATYPE_NUM)
         {
             float data = ((DataTypeNum*)value)->GetNumValue();
             ((DataTypeNum*)varName)->SetNumValue(data);
             return ;
         }
         else
         {
            printf("can not assign different data type: %s %s\n", varName->GetName().c_str(),
value->GetName().c_str());
             exit(-1);
         }
    }


    if(varName->GetDataType() == DATATYPE_STRING)
    {
        if( value->GetDataType() == DATATYPE_STRING)
        {
            string data = ((DataTypeString*)value)->GetStringValue();
            ((DataTypeString*)varName)->SetStringValue(data);
            return ;
        }
        else
        {
            printf("can not assign different data type: %s %s\n", varName->GetName().c_str(),
value->GetName().c_str());
            exit(-1);
        }
    }
}
```

```
/*start a new scope, 0 - non-track up, 1 - track up*/
//--------------------------------------------------------------------------------------------
float AppObj::newScope(int type)
{
    if( type    <=0)
        _SymbolTalbe->AddNewSymbolTable(false, type);
    else
        _SymbolTalbe->AddNewSymbolTable(true, type);

    return 0;
}



/*end of the current scope*/
```

```
//-----------------------------------------------------------------------------------------------
float AppObj::closeScope()
{

	//just close one symbol table
	_SymbolTalbe->DeleteSymbolTable();
	return 0;
}




float AppObj::returnStmt()
{
	//we have to check the current symbol table type, the return
	//we exit the
	return 0;
}


float AppObj::breakStmt()
{
	//we have to check the current symbol table type, the return
	//we exit the
	int symboType = _SymbolTalbe->GetCurrentTableType();
	while(symboType != SCOPE_LOOP)
	{
		_SymbolTalbe->DeleteSymbolTable();
		symboType = _SymbolTalbe->GetCurrentTableType();
	}

	return 0;
}


float AppObj::continueStmt()
{

	//we have to check the current symbol table type, the return
	//we exit the
	//we have to check the current symbol table type, the return
	//we exit the
	int symboType = _SymbolTalbe->GetCurrentTableType();
	while(symboType != SCOPE_LOOP)
	{
        _SymbolTalbe->DeleteSymbolTable();
        symboType = _SymbolTalbe->GetCurrentTableType();
    }


    return 0;
}








/*for each element id1    in    array id2, loop */
float AppObj::foreach(string id1, string id2, DDRLWalker* walker, antlr::RefAST loop_body)
{
    pDataType       arrayList = _SymbolTalbe->IsVariableExist(id2);
    if( arrayList == NULL)
    {
        printf("variable not defined:    %s\n",id2.c_str());
        exit(-1);
    }

    if( arrayList->GetDataType() != DATATYPE_ARRAY)
    {
        printf("foreach can only apply to Array: %s is not an array!\n", id2.c_str());
        exit(-1);
    }

    //ok, we have to put the id1 into symbol table for using
    DataTypeArray*     dataList = (DataTypeArray*)arrayList;
    //string    elementType = dataList->GetElementType();

    //get the first pointer
    pDataType forloop = getArrValue(dataList,0);
    _SymbolTalbe->AddNewVariable(id1, forloop);

    for(unsigned int i=0; i<dataList->GetArrayLen()&&forloop != NULL; i++)
    {
        walker->program(loop_body);
    }

    return 0;
```

```
}


/*Define a function, and store the AST tree for recall*/
void AppObj::addFunc(string Name, antlr::RefAST subprogram, string r)
{

}


/*Set the parameters for function call, which will be called following*/
int    AppObj::setFuncArg(pDataType argValue)
{
        _argList.push_back(argValue);

        return 0;
}


int AppObj::ConsumeArg(string name, int type)
{
        DataType*       var;
        if(type == TABLE_SYMBOL)
        {
                var = _SymbolTalbe->IsVariableExist(name);

                if( var == NULL)
                {
                        printf("variable or function not defined: %s\n", name.c_str());
                        exit(-1);
                }

                //this must be a display model
                for(int i=0; i< _argList.size();i++)
                        ((DataTypeDisplayModel*)var)->SetArg(_argList[i]);

                _argList.clear();
        }
        else
        {
                var = _Declaration->IsDeclarationExist(name);
                if( var == NULL)
                {
                        printf("variable or function not defined: %s\n", name.c_str());
                exit(-1);
            }

            //must be function, set the data

        }

        return 0;

}


/*Call a function, to execute the code in this predefined function*/
pDataType AppObj::funcCall(DDRLWalker* walker, string funcName)
{
        return NULL;
}






void    AppObj::DeclArg(string name, CLASSTYPE type)
{
        if( _tempDefiningModel == 0)
        {
            printf("DDRL ERROR: unkown, backend or antler\n");
            exit(-1);
        }

        switch(type)
        {
            case DATATYPE_NUM:
                _tempDefiningModel->AddArgName(name,"num");
                break;

            case DATATYPE_STRING:
                _tempDefiningModel->AddArgName(name,"string");
                break;

            case DATATYPE_DATASET:
                _tempDefiningModel->AddArgName(name,"dataset");
                break;
        }

}


void    AppObj::DeclArg(string baseTypename, string name, CLASSTYPE type)
{
        if( _tempDefiningModel == 0)
        {
                printf("DDRL ERROR: unkown, backend or antler\n");
                exit(-1);
        }

        switch(type)
        {
                case DATATYPE_ARRAY:
                        _tempDefiningModel->AddArgName(name,"array");
                        break;

                case DATATYPE_STRUCT:
                        _tempDefiningModel->AddArgName(name,baseTypename);
                        break;
        }
}





////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void    AppObj::addDisplayVar(string basename, string varname, int type)
{
        //at very first, we can not use the '[' ']' '.' in name
        string::size_type pos1 = varname.find('.',0);
        string::size_type pos2 = varname.find('[',0);
        string::size_type pos3 = varname.find(']',0);

        if(pos1 != string::npos    || pos2 != string::npos    || pos3 != string::npos)
        {
                printf("illegal charator in variable name:    %s\n", varname.c_str());
                exit(-1);
        }
```

```
    //1. check whether the struct base name is defined.
    DataType* baseType = _Declaration->IsDeclarationExist(basename);
    if(baseType == NULL)
    {
        printf("Struct base type not defined:    %s\n", basename.c_str());
        return;
    }

    Variable    var;
    if( _whichdefining == NONE_DEFINE)
        var = _SymbolTalbe->IsVariableExist(varname);
    else if( _whichdefining == STRUCT_DEFINING)
        var = _tempDefiningStruct->IsVariableExist(varname);
    else
        var = _tempDefiningModel->IsVariableExist(varname);

    if( var != NULL)                        //variable redefine
    {
        printf("variable redefine:    %s\n", varname.c_str());
        return;
    }


    DataTypeDisplayModel*       data = _dataFactory->CreateDisplayModel(varname, basename,
((DataTypeDisplayModel*)baseType)->GetTreeNode());

    if( data != NULL)
    {
        //if it is normal declare, put in symbol table,
        //else, it is struct define, put in the struct define
        if( _whichdefining == NONE_DEFINE)
            _SymbolTalbe->AddNewVariable(varname, data);
        else if( _whichdefining == STRUCT_DEFINING)
        {
            _tempDefiningStruct->AddVariable(varname, data);
            printf("                              --:" );
        }
        else
        {
            _tempDefiningModel->AddVariable(varname, data);
            printf("                              --:" );
        }
        }

        printf("---define displaymodel variable: %s\n", varname.c_str());
}




void    AppObj::enterDmfunction(string modelName, int table)
{

        //1. check whether this kind of struct is defined

        DataType* baseType ;
        if(table == TABLE_DECL)
                baseType = _Declaration->IsDeclarationExist(modelName);
        else
                baseType = _SymbolTalbe->IsVariableExist(modelName);

        if(baseType == NULL || baseType->GetDataType() != DATATYPE_DISPLAYM)
        {
                printf("DDRL ERROR:    display model not defined:    %s\n", modelName.c_str());
                exit(-1);
        }

        _whichdefining = MODEL_DEFINING;

        _tempDefiningModel = (DataTypeDisplayModel*)baseType;
}



void    AppObj::leaveDmfunction()
{
        _whichdefining = NONE_DEFINE;
        _tempDefiningModel = NULL;
}
```

```
void    AppObj::setDispStmt(DDRLWalker* walker, antlr::RefAST node)
{
        //we must at the definition of the display model, so we
        //can get the data
        if( _walker == NULL)
                _walker = walker;


        if(_tempDefiningModel != NULL)
                _tempDefiningModel->SetDisplaySTMTNode(node);
        else
        {
                printf("Display statement error, backend part error!\n");
                exit(-1);
        }
}





//graphic display function are declared below
//---------------------------------------------------------------------
/*Set the Time tag for begin time and duration of animation*/
int AppObj::setTimeTag(pDataType begin_time, pDataType duration, float repeatnode)
{
        if(_tempDefiningModel != NULL)
        {
                if( _tempDefiningModel->_model->_finishCreated != true)
                {
                        _tempDefiningModel->_model->SetTimeForCurrent(begin_time->get(),
duration->get(), repeatnode);
                }
                else
                {
                        //set the time for the whole model
                }
        }

        return 0;
}
```

```
/*Set the Color for the following graphics element, color will remain until next setting*/
int AppObj::setColor(pDataType red, pDataType green, pDataType blue, pDataType alpha)
{
        if(_tempDefiningModel != NULL)
        {
                _tempDefiningModel->_model->_gColor._v[0] = red->get()/255.0;
                _tempDefiningModel->_model->_gColor._v[1] = green->get()/255.0;
                _tempDefiningModel->_model->_gColor._v[2] = blue->get()/255.0;
                _tempDefiningModel->_model->_gColor._v[3] = alpha->get()/255.0;
        }

        return 0;
}


/*Set the Point Size for graphics point element, will remain until next setting*/
int AppObj::setPointSize(pDataType PointSize)
{
        if(_tempDefiningModel != NULL)
        {
                _tempDefiningModel->_model->_gPointSize = PointSize->get();
        }

        return 0;
}


/*Set the Line width for graphics line element, will remain until next setting*/
int AppObj::setLineWidth(pDataType lineWidth)
{
        if( _tempDefiningModel != NULL)
        {
                _tempDefiningModel->_model->_gLineWidth = lineWidth->get();
        }

        return 0;
}


/*Add a Vertex to current context and current defining graphics element*/
int AppObj::addVertex(pDataType x, pDataType y, pDataType z)
{

        if(_tempDefiningModel != NULL)
```

```
                _tempDefiningModel->_model->addOneVertex(x->get(),y->get(),z->get());

        return 0;
}


/**/
int AppObj::setPointScope(float)
{
        return 0;
}


int AppObj::setLineScope(float)
{
        return 0;
}



int     AppObj::setTmeshScope(float)
{
        return 0;
}



int AppObj::setBoxScope(float flag)
{
        if(flag == 0)
        {
                //finished define and feed data, we generate the model
                _tempDefiningModel->_model->GenerateCurrent();
        }
        else
        {
                //we entered the box creation
                if(_tempDefiningModel == NULL)
                        return -1;

                _tempDefiningModel->_model->AddElement(ELEMENT_BOX, 0);
        }

        return 0;
```

```
}


/*set the parameter for box element*/
int AppObj::setBoxSize(pDataType lenght, pDataType bwidth, pDataType height)
{
	if( _tempDefiningModel != NULL)
		_tempDefiningModel->_model->SetBoxParameter(lenght->get(),	bwidth->get(),
height->get());

	return 0;
}



int AppObj::setCylinderScope(float flag)
{
	if(flag == 0)
	{
		//finished define and feed data, we generate the model
		_tempDefiningModel->_model->GenerateCurrent();
	}
	else
	{
		//we entered the box creation
		if(_tempDefiningModel == NULL)
			return -1;

		_tempDefiningModel->_model->AddElement(ELEMENT_CYLINDER, 0);
	}

	return 0;
}


/*Set the parameter for the cylinder element*/
int	AppObj::setCylinderSize(pDataType radius, pDataType height)
{
	if( _tempDefiningModel != NULL)
		_tempDefiningModel->_model->SetCylinderParameter(radius->get(), height->get());

	return 0;
}
```

```
int AppObj::setSliceScope(float)
{
	return 0;
}


/*set the parameter for Slice*/
int   AppObj::setSliceSize(pDataType  radius,  pDataType  height,  pDataType  start,  pDataType
angle)
{
	return 0;
}



int    AppObj::setPlaneScope(float)
{
	return 0;
}


/*set the parameter for Axis*/
int AppObj::setAxis(float axis , pDataType title, pDataType length,    pDataType interval)
{
	return 0;
}


/*set the marker on the axis with text*/
int AppObj::setAxisMarker(pDataType index, pDataType text)
{
	return 0;
}


/*setup a text on 3D position*/
int    AppObj::setText(pDataType text, pDataType sx, pDataType xy, pDataType sz, pDataType ex,
pDataType ey, pDataType ez)
{
	return 0;
}
```

```
int AppObj::controlGraphicTimeTag(pDataType begin_time,pDataType duration,float repeatnode)
{
    //repeat node , 0. stop,   1. loop,     2, clear
    _tempShowMode = repeatnode;


    if(begin_time->GetDataType()!=DATATYPE_NUM     ||     duration->GetDataType()     !=
DATATYPE_NUM)
    {
        printf("Time format not correct: \n");
        exit(-1);
    }




    _tempStartTime = ((DataTypeNum*)begin_time)->GetNumValue();
    _tempDuration = ((DataTypeNum*)duration)->GetNumValue();
    return 0;
}





int AppObj::controlGraphic(float type, string modelID)
{

    DataType*     model = _SymbolTalbe->IsVariableExist(modelID);
    if( model == NULL)
    {
        printf("DDRL ERROR: graphics model not defined   %s\n", modelID.c_str());
        exit(-1);
    }

    if( model->GetDataType() != DATATYPE_DISPLAYM)
    {
        printf("DDRL ERROR: not a display model %s\n", modelID.c_str());
        exit(-1);
    }




    //we can create the modelNow, but first check the model whether created before
    DataTypeDisplayModel*   displayModel = ((DataTypeDisplayModel*)model);
```

```
        if(displayModel == NULL)
        {
                printf("display model not defined: %s\n", model->GetName().c_str());
                exit(-1);
        }

        _tempDefiningModel = displayModel;

        if(type == 1)                           //this is the show operation;
        {
                displayModel->_model->SetShowTime(_tempStartTime, _tempDuration);
                _tempStartTime = 0.0;
                _tempDuration = 0.0;
        }
        else
        {
                displayModel->_model->SetUnShowTime(_tempStartTime, _tempDuration);
                _tempStartTime = 0.0;
                _tempDuration = 0.0;
        }


        return 0;
}


int     AppObj::controlPosition(string modelName, pDataType x, pDataType y, pDataType z)
{
        return 0;
}


int     AppObj::controlRotation(string modelName, pDataType x, pDataType y, pDataType z)
{
        return 0;
}


int     AppObj::controlScale(string modelName, pDataType scale)
{
        return 0;
}
```

```
//add the displayModel into declaration table, wait for further use
//------------------------------------------------------------------------------------------------
pDataType    AppObj::addDisp(string Name, DDRLWalker* walker, RefAST node)
{
        //at very first, we can not use the '[' ']' '.' in name
        string::size_type pos1 = Name.find('.',0);
        string::size_type pos2 = Name.find('[',0);
        string::size_type pos3 = Name.find(']',0);

        if(pos1 != string::npos    || pos2 != string::npos    || pos3 != string::npos)
        {
                printf("illegal charator in variable name:    %s\n", Name.c_str());
                exit(-1);
        }


        //1. check whether this kind of struct is defined
        DataType* baseType = _Declaration->IsDeclarationExist(Name);
        if(baseType != NULL)
        {
                printf("the type of struct is already defined:    %s\n", Name);
                exit(-1);
        }


        //2. create the definition of this struct in declaration table
        int reval = _dataFactory->CreateBaseDataType(Name, DATATYPE_DISPLAYM);
        if(reval != 0)
        {
                printf("--can not declear the struct: %s\n",Name.c_str());
                exit(-1);

        }

        printf("---decleared struct base type: %s\n",Name.c_str());
        return NULL;
}
//------------------------------------------------------------------------------------------------
//end of add the display model
```

```
void    AppObj::dispNow(DDRLWalker* walker)
{
    if( _tempDefiningModel == NULL)
        return;



    GlobalVar* gl = GlobalVar::Instance();
    if(_tempDefiningModel->_model == NULL)
    {
        _tempDefiningModel->_model = new GraphicsDisplayModel();
        //displayModel->createDisplayModel("model", _walker);

        gl->_ModelList.push_back(_tempDefiningModel->_model);
    }
    else
        return;






    //create the model
    //-----------------------------------------------------------------------------------------------------
    //set the appobj temp model

    //first new a scope for this operation
    VariableList* _sym = VariableList::Instance();
    _sym->AddNewSymbolTable(false, -2);

    //put all variable into this symbol table
    map<string,DataType*>::iterator    index;
    for(index          =          _tempDefiningModel->_declearList.begin();          index          !=
_tempDefiningModel->_declearList.end();index++)
    {
        _sym->AddNewVariable(index->first, index->second);
    }

    //put all parameter into the symbol table
    for(int i=0; i<_tempDefiningModel->_argNameList.size();i++)
    {
        _sym->AddNewVariable(_tempDefiningModel->_argNameList[i],
_tempDefiningModel->_parameterList[i]);
```

```
        }


        walker->displaymodelstmt(_tempDefiningModel->_displayNode);


        //move the parameter out
        //put all parameter into the symbol table
        for(int i=0; i<_tempDefiningModel->_argNameList.size();i++)
        {
                _sym->RemoveVariable(_tempDefiningModel->_argNameList[i]);
        }


        //suppose after build one , the variable member will be be deleted.
        _sym->DeleteSymbolTable();
        _tempDefiningModel->_model->CreateElement();
        _tempDefiningModel = NULL;


}
```

Main

Main.cpp

```
#include <osg/Geode>
#include <osg/TexGen>
#include <osg/Texture2D>

#include <osgDB/ReadFile>

#include <osgProducer/Viewer>
#include <osg/Node>
#include <osg/Group>
#include <osg.h>
#include <string>

#include "GameCamera.h"
#include "CameraAdjust.h"
#include "GraphicsPoint.h"
#include "GraphicsLine.h"
#include "GraphicsObject.h"
#include "VRStimer.h"
#include "GraphicsDisplayModel.h"

#include <stdlib.h>

#include "Parser/ddrl.hpp"

using namespace osg;
using namespace osgProducer;



#include "Global.h"
```

```
int main( int argc, char **argv )
{



    //read and parser the source file
    //------------------------------------------------------------------------------------------------
    std::string     sourcefile(argv[1]);
    antlr_main(sourcefile);

    //------------------------------------------------------------------------------------------------

    // construct the viewer.
    Viewer *viewer = new Viewer;

    unsigned        viewerOptions        =        Viewer::SKY_LIGHT_SOURCE        |
    Viewer::HEAD_LIGHT_SOURCE;
    // set up the value with sensible default event handlers.
    //viewer->setUpViewer(viewerOptions);
    viewer->setUpViewer(osgProducer::Viewer::STANDARD_SETTINGS);
    Group* scene = new Group;
    Node* plane = osgDB::readNodeFile("./arrow.flt");
    scene->addChild(plane);

    CameraAdjust* CameraIndicator = CameraAdjust::Instance(5.0, 30.0);

    //scene->addChild(CameraIndicator);


    // add model to viewer.

    //*
    //------------------------------------------------------------------------------------------------
    VRStimer::Instance()->StartTimer();



    //------------------------------------------------------------------------------------------------
    GraphicsDisplayModel     ourTest;
    vector<float>            data;
    for(int i = 0; i<20; i++)
    {
        data.push_back(i*2);
    }
```

```
for(int j=0;j<20;j++)
{
    if(j>10)
    {
        GraphicsBox* b = new GraphicsBox();
        b->SetBoxParameter(2.0, 2.0, 1+j*1.0);
        b->SetColor(Vec4(1.0, 155/225.5, 155/255.0, 155/255.0));
        b->SetBoxPosition(Vec3(j*(2.0+0.01), 0.0, 0.0));
        b->SetStartEndTime(j*1 , j*1+1);
        b->CreateElement();
        ourTest.addChild(b);
        ourTest._ElementList.push_back(b);
    }
    else
    {
        GraphicsCylinder* b = new GraphicsCylinder();
        b->SetCylinderParameter(1.0, j*2);
        b->SetColor(Vec4(1.0, 1.0, 1.0, 100/255.0));
        b->SetCylinderPosition(Vec3(j*(2.0+0.01), 0.0, 0.0));
        b->SetStartEndTime(j*1 , j*1+1);
        b->CreateElement();
        ourTest.addChild(b);
        ourTest._ElementList.push_back(b);
    }
}

ourTest.Start();
viewer->setSceneData(&ourTest);


//----------------------------------------------------------------------------------------------
//viewer->setSceneData(CameraIndicator);

GlobalVar* gl = GlobalVar::Instance();
for(int i=0;i< gl->_ModelList.size();i++)
{
    gl->_ModelRoot->addChild(gl->_ModelList[i]);

    GraphicsDisplayModel* mod = ((GraphicsDisplayModel*)gl->_ModelList[i]);

    for(int j=0; j< gl->_ModelList.size();j++)
    {
        GraphicsBox* box = (GraphicsBox*)gl->_ModelList[j];
        box->GetStartTime();
        }

        gl->_ModelList[i]->Start();
    }

    viewer->setSceneData(gl->_ModelRoot.get());

    //*/

    //viewer->setSceneData(scene);
    //--------------------------------------------------------------------------------------------
    //CameraIndicator->addChild(grahics);
    //viewer->setSceneData(CameraIndicator);
    //CameraIndicator->addChild(scene);
    //GameCamera *myCamera = GameCamera::Create(viewer,Vec3(PI/4.0, 0.0, 0.0),Vec3(0.0f,-30.0f,30.0f));
    //GameCamera *myCamera = GameCamera::Create(viewer);

    // create the windows and run the threads.
    viewer->realize();

    while( !viewer->done() )
    {
        // wait for all cull and draw threads to complete.
        viewer->sync();

        // update the scene by traversing it with the the update visitor which will
        // call all node update callbacks and animations.
        viewer->update();

        // fire off the cull and draw traversals of the scene.
        viewer->frame();

    }

    // wait for all cull and draw threads to complete before exit.
    viewer->sync();

    return 0;
}
```

Appendix II Test cases

1. Antlr Test

Parser.g

```
//author: Yan Zhang, Alexander Ling Lee
options{ language="Cpp";
}

class DDRLParser extends Parser;
options {
        buildAST=true;
        k=2;
        exportVocab=DDRL;
}

tokens{
        PROGRAM;
        FUNCTION;
        SUBPROGRAM;
        FOREXPR;
        DECLS;
        DMFUNCTION;
        SUBDMFUNCTION;
        CGSTMT;
        MARKERSTMTS;
        MARKERSTMT;
        NEGATE;
        TIMETAGSTMT;
        GCSTMT;
        DISPLAYSTMT;
        RET;
        FUNCCALL;
}

program   : (dmfunction|function|ddrlstruct)* main EOF!
        {#program = #([PROGRAM, "PROGRAM"], program);};

main        : ret MAIN^ LPAREN! RPAREN! subprogram;
```

```
function: ret ID^ LPAREN! decls RPAREN! subprogram
        {#function = #([FUNCTION,"FUNCTION"], function);}; //zh

dmfunction     : DISPLAYMODEL ID^ LPAREN! decls RPAREN!
                    LBRACE!
                    subdmfunction
                    RBRACE!
        {#dmfunction = #([DMFUNCTION,"DMFUNCTION"], dmfunction);}; //zh

subdmfunction: (stmt)* displaystmt
               {#subdmfunction = #([SUBDMFUNCTION,"SUBDMFUNCTION"], subdmfunction);};
//al

displaystmt:    DOLLAR! LTH! DISPLAY!
                        displaymodel
                    DOLLAR! DISPLAY! GT!
        {#displaystmt = #([DISPLAYSTMT,"DISPLAYSTMT"], displaystmt);}; //zh

displaymodel: (stmt | (timetagstmt gdstmt))+ ;

ret    : (NUM^|STRING^|VOID^|ID^);

subprogram: LBRACE! (stmt|gdstmt|gcstmt)* RBRACE
        {#subprogram = #([SUBPROGRAM,"SUBPROGRAM"], subprogram);}; //zh

stmt :    varstmt SC!
            |dispVarstmt SC!
                    |asgstmt SC!
            |breakstmt SC!
            |continuestmt SC!
            |array SC!
            |ddrlstruct SC!
            |forstmt
            |whilestmt
            |ifstmt
            |foreachstmt
            |outputstmt SC!
            ;

// statements
outputstmt: OUTPUT^ fbool;

//varstmt  : (NUM^|STRING^|DATASET^|(DISPLAYMODEL^ ID)|(NEWSTRUCT^ ID)) args;
```

```
varstmt    : (NUM^|STRING^|DATASET^|(NEWSTRUCT^ ID)) args;

dispVarstmt : (DISPLAYMODEL^ ID) arg LPAREN! (expr (COMMA! expr)*)? RPAREN!;

asgstmt : ID ASSIGN^ expr;

breakstmt : BREAK;

forstmt    : FOR^ LPAREN! forexpr SC!
        forexpr SC!
        forexpr RPAREN! subprogram;

forexpr    : (boolean)?
        {#forexpr = #([FOREXPR, "FOREXPR"], forexpr); }; //zh

whilestmt: WHILE^ LPAREN! boolean RPAREN! subprogram;

ifstmt      : IF^ LPAREN! boolean RPAREN! subprogram
        (options{greedy=true;}: ELSE! subprogram)?;

foreachstmt : FOREACH^ LPAREN! ID "in" ID RPAREN! subprogram;

continuestmt: CONTINUE;


//gdstmt
gdstmt     : colorstmt SC!
        | pointsizestmt SC!
        | linewidthstmt SC!
        | pointstmt SC!
        | linestmt SC!
        | trianglemeshstmt SC!
        | boxstmt SC!
        | cylinderstmt SC!
        | slicestmt SC!
        | planestmt SC!
        | axisstmt SC!
        | xmstmt SC!
        | textstmt SC!
        ;

cgstmt     : (stmt|(vertexstmt SC!))*
        {#cgstmt = #([CGSTMT, "CGSTMT"], cgstmt); }; //zh
```

```
vertexstmt : VERTEX^ expr COMMA! expr COMMA! expr;

colorstmt : COLOR^ expr COMMA! expr COMMA! expr COMMA! expr;

pointsizestmt   : POINTSIZE^ expr;

linewidthstmt  : LINEWIDTH^ expr;

pointstmt : POINT^ cgstmt timetagstmt ENDPOINT;

linestmt   : LINE^ cgstmt timetagstmt ENDLINE;

trianglemeshstmt : TRIANGLEMESH^ cgstmt timetagstmt ENDTRIANGLEMESH;

boxstmt          : BOX^ cgstmt timetagstmt expr COMMA! expr COMMA! expr SC! ENDBOX;

cylinderstmt    : CYLINDER^ cgstmt timetagstmt expr COMMA! expr SC! ENDCYLINDER;

slicestmt  : SLICE^ cgstmt timetagstmt expr COMMA! expr COMMA! expr COMMA! expr SC!
ENDSLICE;

planestmt: PLANE^ cgstmt timetagstmt ENDPLANE;

axisstmt   : AXIS^ "X" expr COMMA! expr COMMA! expr SC! ENDAXIS!
           | AXIS^ "Y" expr COMMA! expr COMMA! expr SC! ENDAXIS!
           | AXIS^ "Z" expr COMMA! expr COMMA! expr SC! ENDAXIS!
           ;

xmstmt           : AXISMARKER^ markerstmts ENDAXISMARKER!;

markerstmts     : (markerstmt)+
        {#markerstmts = #([MARKERSTMTS, "MARKERSTMTS"], markerstmts); }; //zh

markerstmt      : expr COMMA! expr SC!
        {#markerstmt = #([MARKERSTMT, "MARKERSTMT"], markerstmt); }; //zh

textstmt   : TEXT^ expr SC! vertexstmt COMMA! vertexstmt SC! ENDTEXT!;

//gcstmt
timetagstmt    :   LTH!   (DOLLAR|NUMBER|ID)   COMMA!   (POUND|NUMBER|ID)   GT!
COLON! ( /*nothing*/ |LOOP | CLEAR )
        {#timetagstmt = #([TIMETAGSTMT, "TIMETAGSTMT"], timetagstmt); }; //zh
```

```
gcstmt          : timetagstmt
          (showstmt
          |unshowstmt
          |positionstmt
          |rotationstmt
          |scalingstmt
          )
          SC!
     {#gcstmt = #([GCSTMT, "GCSTMT"], gcstmt); }; //zh

showstmt: SHOW^ ID;

unshowstmt     : UNSHOW^ ID;

positionstmt   : POSITION^ ID NUMBER COMMA! NUMBER COMMA! NUMBER;

rotationstmt   : ROTATION^ ID NUMBER COMMA! NUMBER COMMA! NUMBER;

scalingstmt    : SCALING^ ID NUMBER;


//struct and array
array     : ARRAY^ type ID //(LBRACK! (NUMBER) RBRACK!)+
     ;

ddrlstruct: STRUCT^ ID
     LBRACE! (decl SC!)+ RBRACE!
     ;

type : NUM
     | STRING
     | DATASET
     | DISPLAYMODEL
     | ID
     ;

// declarations
decls     : (decl    (COMMA! decl )*
          | /*NOTHING*/)
     { #decls = #([DECLS, "DECLS"], decls); } ; //zh

decl : (( NUM^
     | STRING^
     | DATASET^
```

```
        //| DISPLAYMODEL^
        | NEWSTRUCT^ ID
        )
        ID)
        | array;

// basic blocks
args  : arg (COMMA! arg)*;

arg   : ID; //(ASSIGN^ boolean)?;

boolean    : fbool ((AND^ | OR^) fbool)*;

fbool      : gbool ((EQUAL^ | NEQUAL^ | LTH^ | GT^ | LTE^ | GTE^) gbool)?;

gbool      : (ID ASSIGN^ expr) | expr;

expr : eexpr ((EXP^) eexpr)*;

eexpr      : mexpr((PLUS^|MINUS^)mexpr)*;

mexpr      : unary ((STAR^|DIV^) unary)*;

unary      : (MINUS^atom)
        {#unary->setType(NEGATE);} //zh
        | atom
        ;

atom: NUMBER
        | STRING
        | LPAREN! expr RPAREN!
        | ID //var
        | ID LBRACK^ (NUMBER|ID) RBRACK! //(/*nothing*/|("." ID "." ID)) //var
        | fcstmt
        ;

// to implement function call
fcstmt : ID LPAREN! varlist RPAREN!
        {#fcstmt = #([FUNCCALL,"FUNCCALL"],fcstmt);};

varlist : (((atom)(COMMA! (atom))*)|/*nothing*/);

class DDRLLexer extends Lexer;
```

```
options {
        testLiterals =false;
        k=2;
        charVocabulary = '\3'..'\377';
        exportVocab=DDRL;
        }

tokens {
        /* keywords */
        /* Basic */
        IF="if";
        ELSE="else";
        WHILE="while";
        FOR="for";
        FOREACH="foreach";
        RETURN="return";
        BREAK="break";
        CONTINUE="continue";
        NUM="num";//zh
        DATASET="dataset";
        STRING="string";
        ARRAY="array";
        STRUCT="struct";
        NEWSTRUCT="newstruct";
        //IMPORT="import";
        MAIN="main";
        OUTPUT="output";

        /* Graphics Definition */
        VERTEX="vertex";
        COLOR="color";
        POINTSIZE="pointsize";
        LINEWIDTH="linewidth";
        POINT="point";
        ENDPOINT="endpoint";
        LINE="line";
        ENDLINE="endline";
        TRIANGLEMESH="trianglemesh";
        ENDTRIANGLEMESH="endtrianglemesh";
        BOX="box";
        ENDBOX="endbox";
        CYLINDER="cylinder";
        ENDCYLINDER="endcylinder";
        SLICE="slice";
        ENDSLICE="endslice";
        PLANE="plane";
        ENDPLANE="endplane";
        AXIS="axis";
        ENDAXIS="endaxis";
        AXISMARKER="axismarker";
        ENDAXISMARKER="endaxismarker";
        TEXT="text";
        ENDTEXT="endtext";
        DISPLAYMODEL="displaymodel";
        DISPLAY="display";

        /* Time Tag */
        LOOP="loop";
        CLEAR="clear";



        /* Graphics Control */
        SHOW="show";
        UNSHOW="unshow";
        POSITION="position";
        ROTATION="rotation";
        SCALING="scaling";
}

/* Expression */
LPAREN  : '(';
RPAREN          : ')';
PLUS    : '+';
MINUS   : '-';
STAR    : '*';
DIV : '/';
EXP : '^';
LTE     : "<=";
GTE : ">=";
LTH : '<';
GT  : '>';
EQUAL   : "==";
NEQUAL          : "!=";
AND     : "&&";
OR  : "||";
ASSIGN  : '=';
COMMA   : ',';
DOT     : '.';
```

```
SC	: ';';
COLON	: ':';
LBRACE: '{';
RBRACE		: '}';
LBRACK		: '[';
RBRACK		: ']';
DOLLAR		: '$';
POUND	: '#';




/* identifier */
protected DIGIT: ('0'..'9');
protected LETTER: ('a'..'z'|'A'..'Z');

//INT		: (DIGIT)+;

NUMBER		:(DIGIT)+ ('.' (DIGIT)*)? (('E'|'e')('+'|'-')? (DIGIT)+)?;

STRING options {testLiterals=false;}: '"'(~('"'|'\n'))*'"';

ID	options {testLiterals=true;}
	: (('a'..'z')+(DIGIT)*('['(NUMBER|ID)']')?)("."('a'..'z')+(DIGIT)*('['(NUMBER|ID)']')?)*;

/* others */
COMMENT :"//"
	(~('\n'|'\r'))* ('\n'|'\r'('\n')?)?
	{
		//C++
		$setType(ANTLR_USE_NAMESPACE(antlr)Token::SKIP); //zh
		//Java
		//$setType(Token.SKIP);
		//newline();
	};

WS :		( ' '
	| '\t'
	| '\n' { newline(); }
	| '\r'
	)
	{$setType(ANTLR_USE_NAMESPACE(antlr)Token::SKIP);}; //zh
```

walker.g

```
//author: Yan Zhang, Alexander Ling Lee
header "pre_include_hpp" {
        #include <stdio.h>
        #include <stdlib.h>
        #include <string>
        #include "ddrl.hpp"
        #include "../DataType.h"
}
options{
        language ="Cpp";
}
{
        #define RETURN_TRUE 1
        #define BREAK_TRUE 2
        #define CONTINUE_TRUE 3
        AppObj appobj;
}
class DDRLWalker extends TreeParser;
options{
        importVocab=DDRL;
}

program : #(PROGRAM (function|dmfunction|ddrlstruct)* main) {printf("      found program
");};

dmfunction {float s;} : #(DMFUNCTION {appobj.newScope(SCOPE_DM); }
                        #(i:ID DISPLAYMODEL { appobj.addDisp(i->getText(), this,
node);appobj.enterDmfunction(i->getText(), TABLE_DECL);} decls s=node:subdmfunction ) )
                          {appobj.leaveDmfunction();appobj.closeScope();};

subdmfunction returns [float r=0;] : #(SUBDMFUNCTION (r=stmt)* displaystmt) {};

displaystmt     : #(DISPLAYSTMT node:.) {appobj.setDispStmt(this, node);};

displaymodelstmt returns [float r=0;] : (r=stmt | (timetagstmt gdstmt))+;

main {float s; std::string t;}    : #(MAIN {appobj.newScope(SCOPE_MAIN);}
                                t=ret s=subprogram)
                                {printf("found main");
                                  appobj.closeScope();};

function {std::string r;}  : #(FUNCTION {appobj.newScope(SCOPE_FUNC);}
                                    #(i:ID r=ret decls node:.))
                                    {appobj.addFunc(i->getText(), node, r);
                                      appobj.closeScope();};

decls        : #(DECLS (decl)*) {};

subprogram returns [float r=0;]      : #(SUBPROGRAM {appobj.newScope(1);}
                                  (r=stmt|gcstmt|gdstmt)*
                                  RBRACE {appobj.closeScope();}) {printf("found subprogram ");};

//ret returns [std::string r;]: (VOID {r="void";}) | (NUM {r="num";}) | (STRING {r="string";}) |
(ID {r="id";});
ret returns [std::string r;]: name:. {r=name->getText();};

stmt returns [float r=0] {float a=0, b=0;} :
          varstmt
        |asgstmt
        |dispVarstmt
        //|breakstmt
        |BREAK {r=BREAK_TRUE;}
        |CONTINUE {r=CONTINUE_TRUE;}
        //|array
        |ddrlstruct
        |r=forstmt
        |r=whilestmt
        |r=ifstmt
        |r=foreachstmt
        |outputstmt
        ;

outputstmt: #(OUTPUT i:.) {appobj.output(expr(i));};

varstmt    : (#(NUM (i:ID {appobj.createNewVar(i->getText(),DATATYPE_NUM);})*))
          |(#(STRING (j:ID {appobj.createNewVar(j->getText(), DATATYPE_STRING);} )*))
          |(#(DATASET (k:ID {appobj.createNewVar(k->getText(),DATATYPE_DATASET);} )*))
          |(#(NEWSTRUCT        type:ID        (name:ID        {appobj.addStructVar(type->getText(),
name->getText(), DATATYPE_STRUCT);} )*))
          | array
          ;

dispVarstmt:#(DISPLAYMODEL   t:ID   l:ID   {appobj.addDisplayVar(t->getText(),   l->getText(),
DATATYPE_DISPLAYM);} pass_args )
                { appobj.ConsumeArg( l->getText(), TABLE_SYMBOL ); };
```

```
//arg {std::string a;}: (i:ID) {};
                                    //need to add

asgstmt {pDataType a,i;}: #(ASSIGN i=expr a=expr) {appobj.assignValue(i,a);};
                        //add assigning a to ID in c++};

//breakstmt: BREAK {break;};

//continuestmt: CONTINUE {continue;};

array       : #(ARRAY i:type j:ID)    {appobj.addArray(i->getText(), j->getText(), 1);};

ddrlstruct :        #(STRUCT        i:ID{appobj.addStruct(i->getText(),        DATATYPE_STRUCT);
appobj.enterStruct(i->getText());}
                  (varstmt)+
                  {appobj.leaveStruct();});

forstmt returns [float r=0;] {float a;} :
                  #(FOR {a=1; appobj.newScope(SCOPE_LOOP);}
                  #(FOREXPR expr1:.) {if(#expr1!=NULL) {this->stmt(#expr1);};}
                  #(FOREXPR expr2:.) {if(#expr2!=NULL) {a=(this->expr(#expr2))->get();};}
                  #(FOREXPR expr3:.)
                  //#(SUBPROGRAM body:.)
                  body:.
                  )
      {
                     while(a!=0)
                           {
                              if((#body)!=NULL)
                              {
                                    r=subprogram(#body);
                        if(r==RETURN_TRUE) {return RETURN_TRUE; appobj.returnStmt();}
                        if(r==BREAK_TRUE) {break; appobj.breakStmt();}
                        if(r==CONTINUE_TRUE) {continue; appobj.continueStmt();}
                                    //stmt((#body));
                              }
                              if((#expr3)!=NULL)
                              {
                                    stmt((#expr3));
                              }
                              if((#expr2)!=NULL)
                              {
                                    a=(expr((#expr2)))->get();
```

```
                    }
                  }

              appobj.closeScope();
    };

whilestmt   returns[float   r=0;]:   #(WHILE   {appobj.newScope(SCOPE_LOOP);}   while_expr:.
loop_body:.)
          {
                while (((expr(#while_expr))->get())!=0)
                {
                    r=subprogram(#loop_body);
            if(r==RETURN_TRUE) {return RETURN_TRUE; appobj.returnStmt();}
            if(r==BREAK_TRUE) {break; appobj.breakStmt();}
            if(r==CONTINUE_TRUE) {continue; appobj.continueStmt();}
                }
            r=appobj.closeScope();
          };

foreachstmt  returns  [float  r=0;]:  #(FOREACH  {appobj.newScope(SCOPE_LOOP);}  id1:.  id2:.
loop_body:.)
        {
            r = appobj.foreach(id1->getText(), id2->getText(), this, #loop_body);
                r=appobj.closeScope();
        };

ifstmt returns [float r=0;] {pDataType a;} :
    #(IF a=expr temp1:. (temp2:.)?)
    {
          if((a->get())!=0)
          {
                r = subprogram(#temp1);
            if(r==RETURN_TRUE) return RETURN_TRUE;
            if(r==BREAK_TRUE) return BREAK_TRUE;
            if(r==CONTINUE_TRUE) return CONTINUE_TRUE;
          }
          else
          {     if(#temp2!=NULL)
                    r = subprogram(#temp2);
            if(r==RETURN_TRUE) return RETURN_TRUE;
            if(r==BREAK_TRUE) return BREAK_TRUE;
            if(r==CONTINUE_TRUE) return CONTINUE_TRUE;
          }
    }
```

```
        ;

type : NUM
        | STRING
        | DATASET
        | DISPLAYMODEL
        | ID
        ;

decl : (#(NUM i:ID)) {appobj.DeclArg(i->getText(), DATATYPE_NUM);}
        | (#(STRING j:ID)) {appobj.DeclArg(j->getText(), DATATYPE_STRING);}
        | (#(DATASET k:ID)) {appobj.DeclArg(k->getText(), DATATYPE_DATASET);}
        | (#(NEWSTRUCT l:ID m:ID)) {appobj.DeclArg(l->getText(), m->getText(),
DATATYPE_STRUCT);}
        | (#(ARRAY n:. o:ID)) {appobj.DeclArg(n->getText(), o->getText(), DATATYPE_ARRAY);}
        ;

gdstmt     : colorstmt
        | pointsizestmt
        | linewidthstmt
        | pointstmt
        | linestmt
        | tmeshstmt
        | boxstmt
        | cylinderstmt
        | slicestmt
        | planestmt
        | axisstmt
        | xmstmt
        | textstmt
        ;

colorstmt {pDataType r, g, b, a;} : #(COLOR r=expr g=expr b=expr a=expr)
        {appobj.setColor(r,g,b,a);};

pointsizestmt {pDataType s;} : #(POINTSIZE s=expr) {appobj.setPointSize(s);};

linewidthstmt {pDataType w;} : #(LINEWIDTH w=expr) {appobj.setLineWidth(w);};

pointstmt : #(POINT {appobj.setPointScope(1);} cgstmt timetagstmt endpoint) ;
endpoint : ENDPOINT {appobj.setPointScope(0);};

linestmt : #(LINE {appobj.setLineScope(1);} cgstmt timetagstmt endline) ;
endline : ENDLINE {appobj.setLineScope(0);};
```

```
tmeshstmt : #(TRIANGLEMESH {appobj.setTmeshScope(1);} cgstmt timetagstmt endtmesh) ;
endtmesh : ENDTRIANGLEMESH {appobj.setTmeshScope(0);};

boxstmt {pDataType x,y,z;} : #(BOX {appobj.setBoxScope(1);} cgstmt timetagstmt x=expr
y=expr z=expr {appobj.setBoxSize(x,y,z);} endbox);

endbox : ENDBOX {appobj.setBoxScope(0);};

cylinderstmt {pDataType r,h;} : #(CYLINDER {appobj.setCylinderScope(1);} cgstmt timetagstmt
r=expr h=expr    {appobj.setCylinderSize(r,h);} endcylinder);

endcylinder : ENDCYLINDER {appobj.setCylinderScope(0);};

slicestmt {pDataType r,h,sa,a;} : #(SLICE {appobj.setSliceScope(1);} cgstmt timetagstmt r=expr
h=expr sa=expr a=expr endslice)
        {
          appobj.setSliceSize(r,h,sa,a);
        };
endslice : ENDSLICE {appobj.setSliceScope(0);};

planestmt : #(PLANE {appobj.setPlaneScope(1);} cgstmt timetagstmt endplane) ;
endplane: ENDPLANE    {appobj.setPlaneScope(0);};

axisstmt {pDataType t,l,i; float d;}: #(AXIS d=dim_decide t=expr l=expr i=expr)
        {appobj.setAxis(d,t,l,i);};

dim_decide returns [float r=0]
        : "x" {r=1;}
        | "y" {r=2;}
        | "z" {r=3;}
        ;

xmstmt : #(AXISMARKER markerstmts ){};

markerstmts : #(MARKERSTMTS (markerstmt)+){};

markerstmt {pDataType i,j;}: #(MARKERSTMT i=expr j=expr) {appobj.setAxisMarker(i, j);};

textstmt {pDataType t,x1,y1,z1, x2,y2,z2;}: #(TEXT t=expr VERTEX x1=expr y1=expr z1=expr
x2=expr y2=expr z2=expr)
        {appobj.setText(t, x1, y1, z1, x2, y2, z2);};

gcstmt    {pDataType b, e; float r=0;} : #(GCSTMT controlTimeTag controlstmt)
```

```
        {appobj.dispNow(this);};

controlTimeTag{pDataType b, e; float r;} :
        #(TIMETAGSTMT b=begin_time e=duration r=repeatnode)
        {appobj.controlGraphicTimeTag(b,e,r);};

controlstmt :
          #(SHOW s:ID) {appobj.controlGraphic(1, s->getText());}
        |#(UNSHOW u:ID) {appobj.controlGraphic(2, u->getText());}
        |#(POSITION p:ID x1:NUMBER y1:NUMBER z1:NUMBER)
                {appobj.controlPosition(p->getText(),appobj.GetpDataType(x1->getText()),

        appobj.GetpDataType(y1->getText()),appobj.GetpDataType(z1->getText()));}
        |#(ROTATION r:ID x2:NUMBER y2:NUMBER z2:NUMBER)
                {appobj.controlRotation(r->getText(),appobj.GetpDataType(x2->getText()),

        appobj.GetpDataType(y2->getText()),appobj.GetpDataType(z2->getText()));}
        |#(SCALING                          sc:ID                          m:NUMBER)
{appobj.controlScale(sc->getText().c_str(),appobj.GetpDataType(m->getText()));}
        ;

cgstmt {float r=0;}: #(CGSTMT (r=stmt|vertexstmt)*);

vertexstmt {pDataType x, y, z; } : #(VERTEX x=expr y=expr z=expr)
        {appobj.addVertex(x, y, z);};

timetagstmt {pDataType b, e; float r;} :
        #(TIMETAGSTMT b=begin_time e=duration r=repeatnode)
        {appobj.setTimeTag(b,e,r);};
begin_time returns[pDataType r=0] {pDataType a;}
        : i:NUMBER   {r=appobj.GetpDataType(i->getText());}
        | j:ID       {r=appobj.GetpDataType(j->getText());}
        | DOLLAR     {a=appobj.GetpDataType("1"); r=appobj.negate(a);}   //constant
        ;

duration returns[pDataType r=0] {pDataType a;}
        : i:NUMBER   {r=appobj.GetpDataType(i->getText());}
        | j:ID       {r=appobj.GetpDataType(j->getText());}
        | POUND      {a=appobj.GetpDataType("1"); r=appobj.negate(a);}   //constant
        ;
repeatnode returns[float r=0]
        : LOOP       {r=1;}
        | CLEAR      {r=2;}
        |        {r=0;}    //nothing
        ;

/*
timetagstmt {pDataType b, e, r;} :
        #(TIMETAGSTMT b=begin_time e=duration r=repeatnode)
        {appobj.setTimeTag(b,e,r);};
begin_time returns[pDataType r=NULL]
        : i=expr    {r=appobj.GetpDataType(i->getText());i;}
        | DOLLAR      {r=-1;}    //constant
        ;

duration returns[pDataType r=0]
        : i=expr    {r=i;}
        | POUND        {r=-1;}    //constant
        ;
repeatnode returns[pDataType r=0]
        : LOOP           {r=1;}
        | CLEAR         {r=2;}
        |              {r=0;}     //nothing
        ;
*/

expr returns [pDataType r=NULL]
{ pDataType a,b; }
        : #(EXP a=expr b=expr) {r=appobj.exp(a,b);}
        | #(PLUS a=expr b=expr) {r=appobj.add(a,b);}
        | #(MINUS a=expr b=expr) {r=appobj.minus(a,b);}
        | #(STAR a=expr b=expr) {r=appobj.mult(a,b);}
        | #(DIV a=expr b=expr) {r=appobj.div(a,b);}
        | #(EQUAL a=expr b=expr) {r=appobj.equal(a,b);}
        | #(NEQUAL a=expr b=expr) {r=appobj.nequal(a,b);}
        | #(LTH a=expr b=expr) {r=appobj.less(a,b);}
        | #(LTE a=expr b=expr) {r=appobj.lessequal(a,b);}
        | #(GT a=expr b=expr) {r=appobj.greater(a,b);}
        | #(GTE a=expr b=expr) {r=appobj.greatequal(a,b);}
        | i:NUMBER   {r=appobj.GetpDataType(i->getText());}
        | k:STRING     {r=appobj.GetpDataType(k->getText());}
        | #(NEGATE a=expr) {r=appobj.negate(a);}
        | #(FUNCALL n:ID pass_args ) { appobj.newScope(0);
              r=appobj.funcCall(this,n->getText());
              appobj.closeScope();
        }

        | j:ID        {r=appobj.GetpDataType(j->getText());} //empty for now
        |     #(LBRACK     m:ID     ((l:NUMBER)     {r=appobj.GetpDataTypeArray(m->getText(),
atof(l->getText().c_str()));}
                |(o:ID)                              {r=appobj.GetpDataTypeArray(m->getText(),
appobj.GetpDataType(o->getText()));})
            )
        ;

pass_args: (value)*;

value {pDataType r;} : (r=expr) {appobj.setFuncArg(r);};
```

main.java

```
//author: Yan Zhang
import antlr.*;
import antlr.collections.*;
import java.io.*;
import antlr.debug.misc.ASTFrame;

public class Main {
        public static void main(String[] args) throws Exception
        {

                FileInputStream fileInput = null;
                fileInput = new FileInputStream(args[0]);
                DataInputStream input = new DataInputStream(fileInput);
                DDRLLexer lexer = new DDRLLexer(input);
                DDRLParser parser = new DDRLParser(lexer);
                parser.program();
                CommonAST t=(CommonAST)parser.getAST();
                //AST t= parser.getAST();
                //System.out.println(t.toStringTree());
                System.out.println("=============AST Structure=============");
                System.out.println( t.toStringList() );
                ASTFrame frame = new ASTFrame ("AST from the DDRL parser", t);
                frame.setVisible(true);
                System.out.println("=============        END        =============");
                DDRLWalker walker = new DDRLWalker();
                walker.program(t);

                /*
                try {
                        FileInputStream fileInput = null;
```

```
			fileInput = new FileInputStream(args[0]);
			DataInputStream input = new DataInputStream(fileInput);
			DDRLLexer lexer = new DDRLLexer(input);
			DDRLParser parser = new DDRLParser(lexer);
			CommonAST tree = (CommonAST)parser.getAST();
			System.out.println("============AST Structure============");
			System.out.println( tree.toStringList() );
			System.out.println("============	END	============");
		}
		catch(Exception e)
		{
			System.err.println(e.getMessage());
		}*/
		//ExprTreeParser treeParser = new ExprTreeParser();
	//	int x = treeParser.expr(t);
	//	System.out.println(x);
	}
}
```

2. DDRL Program Test

Test1

```
//author: Yan Zhang
num main()
{
	//testing stmt in walker
	struct abc {
		num a;
		string b;
	};
	//for (a=0; a<5; a=a+1)
	//{
	//	b=0;
	//}
```

```
    //while (a<5)
    //{
    //    b=0;
    //}

    //foreach (a in b)
    //{
    //    b=0;
    //}

    if (a<5)
    {
        b=0;
    }

    array num arr[3];
    num a;
    string b;
    a=1+2;
    break;
    continue;
}
```

Test 2

```
//author: Yan Zhang
num main()
{
        num a,b;
        //a = 1;
}
```

Test 3

```
//author: Yan Zhang
num main()
{
        // comment
        num a=2,b;
        a = 1;
}
```

Test 4

```
//author: Yan Zhang
num function(num c)
{

        //comment
        num d;
        d=1;
}
num function2(num c, string a)
{

        //comment
        num d;
        d=1;

}
//comment
num main()
{
        // comment
        num a=2,b;
        a = 1;
}
```

Test 5

```
//author: Yan Zhang
num function(num c)
{

        //comment
        num d;
        d=1;
}
num function2(num c)
{

        //comment
        num d;
        d=1;
}
//comment
num main()
{
        struct new {
        num a;
        num b;
        };
        // comment
        num a=2,b;
        a = 1;
}
```

Test 6

```
//author: Yan Zhang
num function(num c)
{

        //comment
        num d;
        d=1;
}
num function2(num c)
{

        //comment
```

```
    num d;
    d=1;
}
//comment
num main()
{
    // comment
    num a=2,b;
    a = 1;
    array num c[3];
}
```

Test 7

```
//author: Yan Zhang
num function(num c)
{

    //comment
    num d;
    d=1;
    break;
}
num function2(num c)
{

    //comment
    num d;
    d=1;
    break;
}
//comment
num main()
{
    // comment
    num a=2,b;
    a = 1;
    break;
    continue;
}
```

Test 8

```
//author: Yan Zhang
num function(num c)
{

    //comment
    num d;
    d=1;
    break;
}
num function2(num c)
{

    //comment
    num d;
    d=1;
    break;
}
//comment
num main()
{
    // comment
    num a=2,b;
    a = 1;
    for (a=1;a<8;a=a+1)
    {
        a=0;
        break;
        continue;
    }
}
```

Test 9

```
//author: Yan Zhang
num function(num c)
{

    //comment
    num d;
    d=1;
    break;
}
num function2(num c)
{

        //comment
        num d;
        d=1;
        break;
}
//comment
num main()
{
        // comment
        num a=2,b;
        a = 1;
        while (a<8)
        {
                a=0;
                break;
                continue;
                a = a+1;
        }
}
```

Test 10

```
//author: Yan Zhang
num function(num c)
{

        //comment
        num d;
        d=1;
        break;
}
num function2(num c)
{

        //comment
        num d;
        d=1;
        break;
}
//comment
```

```
num main()
{
	// comment
	num a=2,b;
	a = 1;
	if (a<8)
	{
		a=0;
		break;
		continue;
		a = a+1;
	}
	else {
		a=a+1;
	}
}
```

Test 11

```
//author: Yan Zhang
num function(num c)
{

	//comment
	num d;
	d=1;
	break;
}
num function2(num c)
{

	//comment
	num d;
	d=1;
	break;
}
//comment
num main()
{
	// comment
	num a=2,b;
	array num c[3];
	a = 1;
	foreach (a in c)
```

```
        {
                a=0;
                break;
                continue;
                a = a+1;
        }
}
```

Test 12

```
//author: Yan Zhang
num function(num c)
{

        //comment
        num d;
        d=1;
        break;
}
num function2(num c)
{

        //comment
        num d;
        d=1;
        break;
}
//comment
num main()
{
        // comment
        num a=2,b;
        array num c[3];
        a = 1;
        color 255, 255, 0, 255;
        color r, g, b, a;
        pointsize 1;
        linewidth 1;
}
```

Test 13

```
//author: Yan Zhang
```

```
num function(num c)
{

    //comment
    num d;
    d=1;
    break;
}
num function2(num c)
{

    //comment
    num d;
    d=1;
    break;
}
//comment
num main()
{
    // comment
    num a=2,b;
    array num c[3];
    a = 1;

    point
        vertex 1,2,3;
        vertex 2,3,4;
        num a=2;
    endpoint;

    line
        vertex 1,2,3;
        vertex 2,3,4;
        num a=2;
    endline;

    polygon
        vertex 1,2,3;
        vertex 2,3,4;
        num a=2;
    endpolygon;
}
```

Test 14

```
//author: Yan Zhang
num function(num c)
{

        //comment
        num d;
        d=1;
        break;
}
num function2(num c)
{

        //comment
        num d;
        d=1;
        break;
}
//comment
num main()
{
        // comment
        num a=2,b;
        array num c[3];
        a = 1;

        box
                vertex 1.0,0.0,0.0;
                vertex 1.0,0.0,0.0;
                <$,#>:
                //2.0,2.0,data[1];
                //2.0,2.0,data[i];
                2.0,2.0,2.0;
        endbox;
}
```

Test 15

```
//author: Yan Zhang
num function(num c)
{
        //comment
        num d;
        d=1;
        break;
}
num function2(num c)
{

        //comment
        num d;
        d=1;
        break;
}
//comment
num main()
{
        // comment
        num a=2,b;
        array num c[3];
        a = 1;

        cylinder
                vertex 1.0,0.0,0.0;
                <$,#>:
                //2.0,2.0,data[1];
                //2.0,2.0,data[i];
                2.0,2.0;
        endcylinder;

        slice
                vertex 1.0,0.0,0.0;
                <$,#>:
                //2.0,2.0,data[1];
                //2.0,2.0,data[i];
                2.0,2.0, data[i];
        endslice;

        plane
                vertex 1.0,0.0,0.0;
                <$,#>:
                //2.0,2.0,data[1];
                //2.0,2.0,data[i];
                //2.0,2.0, data[i];
        endplane;
```

```
	axis x
		//vertex 1.0,0.0,0.0;
		//2.0,2.0,data[1];
		//2.0,2.0,data[i];
		2.0,2.0, data[i];
	endaxis;
	axis y
		//vertex 1.0,0.0,0.0;
		//2.0,2.0,data[1];
		//2.0,2.0,data[i];
		2.0,2.0, data[i];
	endaxis;
	axis z
		//vertex 1.0,0.0,0.0;
		//2.0,2.0,data[1];
		//2.0,2.0,data[i];
		2.0,2.0, data[i];
	endaxis;

	axismarker
		2, 2;
		2, a;
	endaxismarker;

	text
		a;
		vertex 2.0,2.0,data[i];
	endtext;
}
```

Test 16

```
//author: Yan Zhang
num function(num c)
{

	//comment
	num d;
	d=1;
	break;
}
num function2(num c)
{
        //comment
        num d;
        d=1;
        break;
}
//comment
num main()
{
        // comment
        num a=2,b;
        array num c[3];
        a = 1;

        //<10,#>: show mychart;
        <$,#>: show mychart;
        <10,20>: show mychart;
        <10,20>: unshow mychart;
        <10,20>: position mychart 10.0, 10.0, 10.0;
        <10,20>: rotation mychart 10.0, 10.0, 10.0;
        <10,20>: scaling mychart 10;
}
```

Test 17

```
//author: Yan Zhang, Alexandre Ling Lee
displaymodel columnchart(num size)
{
        num b=2;
        b = 1;
        num threshold = 50.0;
        array num data[size];
        $<display
                num i;
                for (i=0; i<size; i=i+1)
                {
                        if (data[i]>threshold)
                        {
                                color r,g,b,a;
                                box
                                        vertex 1.0+i*2,0,0;
                                        <$,#>:
                                        2,2,data[i];
                                endbox;
                        }
                        else
                        {
                                color 255,255,255,100;
                                box
                                        vertex 1.0+i*2,0.0,0.0;
                                        <$,#>:
                                        2.0,2.0,data[i];
                                endbox;
                        }
                        if (data[i]>threshold)
                        {
                                color r,g,b,a;
                                box
                                        vertex 1.0+i*2,0,0;
                                        <$,#>:
                                        2,2,data[i];
                                endbox;
                        }
                        else
                        {
                                color 255,255,255,100;
                                box
                                        vertex 1.0+i*2,0.0,0.0;
                                        <$,#>:
                                        2.0,2.0,data[i];
                                endbox;
                        }
                }
        $display>
}
num function(num c)
{

        //comment
        num d;
        d=1;
        break;
}
num function2(num c)
{

        //comment
        num d;
        d=1;
        break;
}
//comment
num main()
{
        // comment
        num a=2,b;
        array num c[3];
        a = 1;

}
```

Test 18

```
//author: Yan Zhang
num main()
{
        2+3;
}
```

Test 19

```
//author: Yan Zhang
num main()
{
        //num x=5;
        //if (x<6)
        //{
        //      x=x+1;
        //}
        //else
        //{
        //      x=x;
        //}

        //a.b.c = 1;
        string a;
        num r;
        r = a+3;
}
```

Test 20

```
//author: Yan Zhang
num main()
{
	a=b+c+d;
}
```

Test 21

```
//Alexandre Ling Lee	-al2537@columbia.edu

displaymodel columnchart(num size)
{
	num a;
	num c;
	$<display
		text
			a+1;
			vertex a+1, 2, c;
			<1,2>:
		endtext
	$display>
}

int main()
{}
```

Test 22

```
//Alexandre Ling Lee	-al2537@columbia.edu

displaymodel columnchart(num size)
{
	$<display
		box
			vertex 1.0,0.0,0.0;
			<$,#>:
			2.0,2.0,data[1];
			//2.0,2.0,data[i];
			//2.0,2.0;
```

```
        endbox;
    $display>
}

int main()
{}
```

Test 23

```
//Alexandre Ling Lee    -al2537@columbia.edu

displaymodel columnchart(num size)
{
    num a;
    num c;
    $<display
        text
            a+1;
            vertex a+1, 2, c;
        endtext
    $display>
}

int main()
{}
```

Test 24

```
//Alexandre Ling Lee    -al2537@columbia.edu

displaymodel columnchart(num size)
{
    $<display
        slice
            vertex 1.0,0.0,0.0;
            <$,#>:
            2.0,2.0,data[1];
            //2.0,2.0,data[i];
            //2.0,2.0;
        endslice;
    $display>
}
```

```
int main()
{}
```

Test 25

```
//Alexandre Ling Lee      -al2537@columbia.edu

displaymodel columnchart(num size)
{
        num a;
        num c;
        $<display
                num a,c;
                point
                        a+1;
                        vertex a+1, 2, c;
                        <2,3>:
                endpoint
        $display>
}

int main()
{}
```

Test 26

```
//Alexandre Ling Lee      -al2537@columbia.edu

displaymodel columnchart(num size)
{
        $<display
                slice
                        vertex 1.0,0.0,0.0;
                        <$,#>:
                        //2.0,2.0,data[1];
                        //2.0,2.0,data[i];
                        //2.0,2.0;
                endslice;
        $display>
}

int main()
```

```
{}
```

Test 27

```
//Alexandre Ling Lee	-al2537@columbia.edu

displaymodel columnchart(num size)
{
	$<display
		cylinder
			vertex 1.0,0.0,0.0;
			<$,#>:
			//2.0,2.0,data[1];
			//2.0,2.0,data[i];
			2.0,2.0;
		endcylinder;
	$display>
}

int main()
{}
```

Test 28

```
//Alexandre Ling Lee	-al2537@columbia.edu

displaymodel columnchart(num size)
{
	num a;
	num c;
	$<display
		color a,1+2,c;

		pointsize a;

		linewidth 1;

	$display>
}

int main()
```

```
{}
```

Test 29

```
//Alexandre Ling Lee     -al2537@columbia.edu

displaymodel columnchart(num size)
{
        num a;
        num c;
        $<display
                axis
                        X a+1, 2, c*2;
                endaxis
        $display>
}

int main()
{}
```

Test 30

```
//Alexandre Ling Lee     -al2537@columbia.edu

displaymodel columnchart(num size)
{
        num a;
        num c;
        $<display
                axismarker
                        a+1, 2;
                endaxismarker
        $display>
}

int main()
{}
```

Test 31

```
//Alexandre Ling Lee     -al2537@columbia.edu

displaymodel columnchart(num size)
{
        num a;
        num c;
        $<display
                num a,c;
                point
                        a+1;
                        vertex a+1, 2, c;
                        <2,3>:
                endpoint
        $display>
}

int main()
{
        displaymodel columnchart x(10);
        <1,1>: show x;
        <1,1>: unshow x;
        <1,1>: postion x 1, 2,3;
        <1,1>: rotation x 2,3,4;
        <1,1>: scaling x 5,;
}
```

Test 32

```
//for test
//Yitao Wang
num main()
{
        num x;
        for(x=1;x<10;x=x+1)
        {
                output 1;
        }

}
```

Test 33

```
//add test
//Yitao Wang
num main()
{
        string a;
        num b;

        a = "b";
        b = a + "test";

        num c;
        c = 98.0;
        num d;

        d = c + 87;
}
```

Test 34

```
//while test
//Yitao Wang
num main()
{
      num n,m;
      n=0;
      m=0;

      while (n<10)
      {
            output n;
            continue;


      }
}
```

Test 35

```
//if test
//Yitao Wang
```

```
num main()
{
    num x;

    if(1==0)
    {
        output 1;
    }
    else
    {
        output 2;
    }
}
```

Test 36

```
//multiply test
//Yitao Wang
num main()
{
        num a;
        num b;

        b = 5*4;

        a = b*3;

        //a = b*"test";

        a = b*c;
}
```

Test 37

```
//divide test
//Yitao Wang
num main()
{
        num a;
        num b;

        b = 5/4;
```

```
	a = b/3;

	//a = b/"test";

	//a = b/c;

	a = b/0;
}
```

Test 38

```
//minus test
//Yitao Wang
num main()
{
	num a;
	num b;

	b = 5 - 4;

	a = b - 3;

	//a = b - "test";

	a = b - c;
}
```

Test 39

```
//negate symbol test
//Yitao Wang
num main()
{
	string a;
	num b;

	a = "test";

	b = -0;

	b = -5;

	b = -(-5);
```

```
        //b = -a;

        b = -"test";


}
```

Test 40

```
//equal test
//Yitao Wang
num main()
{
      string a;
      a = "b";

      num b;

      b = 4;

      output    1==1;

      output    1==2;

      output    "test"=="test";

      output    "test"=="ab";



      output    a=="test";



}
```

Test 41

```
//noequal
//Yitao Wang
num main()
{
    string a;
    a = "b";

    num b[6];
}
```

Test 42

```
//exponent test
//Yitao Wang
num main()
{
        string a;
        num b;

        a = "test";
        output a;

        b = 2;

        b = b^2;
        output b;

        b = a^2;


        b = "test"^2;


}
```

Test 43

```
//varaible create and definition test
//Zhiyang Cao

struct struct1
{
                num a;
                string b;

                array num aa;
                array string bb;
}

struct struct2
{
        num ddnum;
        string aastring;
        array num arrays2;
        array string arrays2;

        newstruct struct1 struinstr;

        array struct1 structarrayinstruct2;
}


num main()
{
    num abc[];
      string cde;
      array num efg;
      array string dfd;

        newstruct struct1 test;
        array struct1 tt;
        newstruct struct2 test2;
        array struct2 god;

      test.a = 5;
      output test.a;

      test.aa[10] = 10;

      test.aa = 65;

      output test.aa[0];
      output test.aa[10];

        god[2].ddnum = 6;
        output god[2].ddnum;

        god[2].arrays2[2] = 7;
        output god[2].arrays2[2];
}
```

Test 44

```
//array test
//Yitao Wang
num main()
{
        num x, add, that[], ded;

        array num list;
        x = 6;

        list[0] = 5;

        output     "testtste";

        output   list[0];

        list[x] = 20;

        output list[x];

}
```

Test 45

```
//Test
//The embed of struct and array
//Yitao WAang

struct ST
{
        num a;
        string b;
        //array num c;
        //array string d;

}

num main()
{
        newstruct ST x;
        x.a=1;
        x.b="hello";
        //num i;


        output x.a;
        output x.b;

}
```

Test 46

```
//dmfunction test
//Zhiyang Cao, Yitao Wang
displayModel columnchart(int size)
{
        Array float data[size]; //define the data array
        int r = 255, g=0, b=0, a = 255; //the color for our selection
        float threshold = 50.0; //threshold
        $<Display
                int i;
                for(i=0;i<size;i++) //loop the array for all the data
                {
                        if(data[i]>threshold) //if the data larger than threshold
                        {
                                Color r,g,b,a; //use our defined color
                                Box //draw the box, height as value
                                        Vertex 1.0+i*2,0.0,0.0;//start at current time of this model
                                        <$,#>:
                                        2.0,2.0,data[i]; //duration is default time for Box
                                EndBox;
                        }
                        else
                        {
                                Color 255,255,255,100;
                                Box
                                        Vertex 1.0+i*2,0.0,0.0;
                                        <$,#>:
                                        2.0,2.0,data[i];
                                EndBox;
                        }
                }
        $Display>
}

num main()
```

```
{
}
```