EasySurvey

Taotao Li tl2453

Luyao Shi ls2899

Zaisheng Dai zd2140

Yifan Zhang yz 2365

Professor: Stephen A. Edwards

EASYSURVEY 1

CHAPTER 1 5

AN INTRODUCTION TO EASYSURVEY 5

1.1 BACKGROUND 5
1.2 GOALS OF EASYSURVEY 5
1.2.1 INTUITIVE 5
1.2.2 POWERFUL 6

CHAPTER2 6

TUTORIAL 6

2.1 FIRST EASYSURVEY SAMPLE 6
2.2 COMPILING AND RUNNING EASYSURVEY FILES 7
2.3 ONE MORE COMPLICATED EXAMPLE 7

CHAPTER 3 9

REFERENCE MANUAL 9
3.1 LEXICON 10
3.1.1 TOKEN SEPARATOR 10
3.1.2 COMMENTS 10
3.1.3 CONSTANTS 10
3.1.4 IDENTIFIER 10
3.1.5 KEYWORDS 10
3.1.6 SEPARATORS 10
3.1.6.1 ';' 10
3.1.6.2 '{' AND '}' 10
3.1.6.3 ',' 11
3.1.6.4 '(' AND ')' 11
3.1.7 OPERATORS 11
3.1.7.1 ASSIGN 11
3.2.7.2 POINT 11
3.2.7.3 COLON 12
3.2.8 ATTRIBUTES AND OPERATIONS 12
3.2.9 DATA TYPE 13
3.2.10 DISPLAY SET 13
3.4 DECLARATION 14
3.4.1 QUESTIONSET DECLARATION 14
3.4.2 OTHER DATATYPE DECLARATION 15
3.5 FUNCTION DEFINITION 15
3.6 STATEMENTS 16
3.6.1 COMMON STATEMENTS 16

3.6.2 BLOCK STATEMENTS 16
3.7 EXPRESSIONS 16
3.7.1 QUESTION ASSIGNMENT EXPRESSIONS 16
3.7. 2 IMAGE ASSIGNMENT EXPRESSION 16
3.7.3 JUMPBUTTON ASSIGNMENT EXPRESSION 16
3.7.4 ADDQUESTION EXPRESSIONS 17
3.7.5 CALL EXPRESSIONS 17
3.7.6 EVENT EXPRESSION 17
3.7.7 DISPLAY EXPRESSION 17
3.8 SCOPE 17

CHAPTER 4 18

PROJECT PLAN 18

4.1 TEAM RESPONSIBILITIES 18
4.2 PROJECT TIMELINE 18
4.3 SOFTWARE DEVELOPMENT ENVIRONMENT 19
4.4 PROJECT LOG 19

CHAPTER 5 19

ARCHITECTURAL DESIGN 20

5.1 ARCHITECTURE 20
5.2 THE RUNTIME ENVIRONMENT 20

CHAPTER 6 21

TESTING PLAN 21

6.1 GOALS 21
6.2 METHODS 21
6.2.1 PHASE I 21
6.2.2 PHASE II 21
6.3 IMPLEMENTATION 21
6.3.1 PHASE I 21
6.3.2 PHASE II 22
CHAPTER 7 22
LESSONS LEARNED 22

APPENDIX A 22

CODE STYLE CONVENTIONS 22

A.1 INTRODUCTION 22
A.2 GENERAL PRINCIPLES 22

A.3 DOCUMENTATION COMMENTS **23**

APPENDIX B CODE LISTING **23**

Chapter 1

An Introduction to EasySurvey

EasySurvey is a light weight language which is designed to make the online survey generation process more easy and fun. Our language builds a framework that allows user to define basic components of online survey. It allows user to define basic question type like Radio Button, Drop down list, check box and text filed. Also we allow the user to add some logic control on the displaying of questions, and question set operation.

EasySurvey is designed to be simple and natural. Every user with basic knowledge of Java or C/C++ can access to it without too much barriers.

1.1 Background

Online survey can find its importance in areas such as Market Research, Event Planning, Customer Feedback, Product Planning, Education & Training and so on. Unfortunately, the html or flex based online survey is not very easy for novice to develop such as meaningful survey within a short time period. Even for experienced developer, writing such survey is tedious and error-prone.

Thought, there are some survey-generate tool such as SurveyMonkey , Survey Gizmo that could facilitate user in designing online survey, they are not free and charge a lot for individual. The goal of *EasySurvey* is to simplify this develop process and make the generation of online survey more accessible. We will define several key words that are more familiar and easy to understand for inexperienced user. This will make this develop process easier and more available for users!

1.2 Goals of EasySurvey

The goal of EasySurvey is to simplify the programming process in creating an online survey so that it is easy for inexperienced programmer or even the amateur to handle with.

1.2.1 Intuitive

The most important reason to create this language is that we should make the complicated things simple. A programmer spends less time on programming, he/she will concentrates more on the structure of the survey and the functionality of the programs, which increase both the efficiency of coding and the accuracy of the survey.

1.2.2 Powerful

With the obvious key words and functions, EasySurvey is very friendly to programmers, which makes writing codes just like talking to the computer or giving commands to it.

Chapter2

Tutorial

2.1 First EasySurvey Sample

```
main ( ) {
/*Declaration of Questions*/
 Question Q1, Q2, Q3, Q4;
 /*Question 1 assignment*/
 Q1.Title= "what's your name ?";
 Q1.TextField=(10);
/*Question 2 assignment */
 Q2.Title="What's your gender?";
 Q2.DropdownList={"Male","Female"};
/*Question 3 assignment */
 Q3.Title="What's your major";
 Q3.SingleSelection={"Computer Science", "Electrical Engineer"};
/*Question 4 assignment */
 Q4.Title="Which courses you took this semester?";
 Q4.MultipleSelection={"Database","Analysis of Algorithm", "PLT", "ASE"};
/*display the related questions*/
 qDisplay Q1, Q2, Q3,Q4;
}
```

EasySurvey is designed to simplify the surveys generating process, so as we can see, the EasySurvey source file is very user-friendly. First of all, we need to declare the Questions. Secondly, EasySurvey offers four types of questions: text question, dropdownlist question, single-selection question and multiple-selection question. And we defined the corresponding keywords (TextField, DropdownList, SingleSelection and

MultipleSelection), and then we need to do the question assignment. In our first example, it includes all the four types questions. The last step is to display the related questions. It's easy to implement in EasySuvery, just put the questions after the keyword "qDisplay".

2.2 Compiling and Running EasySurvey Files

EasySurvey will accept the files with .es extension as the input file; the EasySurvey complier will compile the input file, which will be analyzed in lexically and grammatically. If everything is correct, the compiler will generate the flex page source code **(.mxml).** Then the output source code can be run and tested under Adobe Flash Builder. Then developer can export the application as a package, which can be run in any flash-support environment.

2.3 One more complicated Example

```
/* Declaration of Global Questions*/
Question GQ;
/*User defined function*/
FirstImg(Question Q, Question QT, Image Img) {
  iDisplay Img;
  qDisplay Q, QT;
}

/*main function*/
main ( ) {
/*Declaration of Local Questions*/
 Question Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9;
/*Declaration of Local images*/
Image Img;
 /*Declaration of Local QuestionSet*/
  QuestionSet QS = New QuestionSet();
/*Declaration of button*/
  JumpButton JP;
/*Declaration of Local QuestionSet*/
  QuestionSet Qee = New QuestionSet();
```

```
/*Image assignment*/
 Img.Link="C:/Adobe Flash Builder 4/PLTdemo/Survey_Select_Logo.jpg";
/*Button assignment*/
 JP.Jump="http://www.google.com";
 /*Global question assignment*/
 GQ.Title="Are you ready to start?";
 GQ.SingleSelection={"Yes", "NO"};
 /*Question 1(text question) assignment*/
Q1.Title= "what's your name ?";
Q1.TextField=(10);
 /*Question 2 (dropdownlist question) assignment*/
Q2.Title="What's your gender?";
Q2.DropdownList={"Male","Female"};
 /*Question 3 (single-selection question)assignment*/
Q3.Title="What's your major";
Q3.SingleSelection={"Computer Science", "Electrical Engineer"};
 /*Question 4 (single-selection question)assignment*/
Q4.Title="Which track?";
Q4.SingleSelection={"Machine Learning", "Software System", "Network System", "Others"};
 /*Question 5 (multiple-selection question)assignment*/
Q5.Title="Which courses you took this semester?";
Q5.MultipleSelection={"Database","Analysis of Algorithm", "PLT", "ASE"};
 /*Question 6 (single-selection question)assignment*/
Q6.Title="Which track?";
Q6.SingleSelection={"Networking","Communication","Circuits", "Others"};
 /*Question 7 (single-selection question)assignment*/
Q7.Title="What's your percentage?";
Q7.SingleSelection={"80-90%","below"};

 /*Question 8(text question) assignment*/
Q8.Title="Any Comment for this semester?";
```

```
Q8.TextField=(20);
/*Add Q7 and Q8 to QuestionSet*/
QS.AddQuestion(Q7,Q8);
/*If user choose answer 1 of Q3 (in this case, “Computer Science”)*, then we will display Q4 and Q5 but not display Q6*/
Q3.Event(1):True(Q4,Q5),False(Q6);
/*If user choose answer 2 of Q3 (in this case, “Electrical Engineer”)*, then we will display Q6 but not display Q4 and Q5*/
Q3.Event(2):True(Q6), False(Q4, Q5);
/*Call user-defined function “FirstImg”*/
FirstImg (Q1,Q2,Img);
/*Display Global Question*/
qDisplay GQ;
/*Display Global Question*/
qDisplay Q3;
/*Display Optional Question*/
oqDisplay Q4, Q5;
oqDisplay Q6;
/*Display QuestionSet*/
Display QS;
/*Display Button*/
jDisplay JP;
}
```

This example is the extension of the first EasySurvey sample and includes all the features EasySurvey offers. In this sample, we add global question (“GQ” in the example), image (“Img” in the example) and button (“JP” in the example). Also we add the **dynamic feature** of EasySurvey, that is, the visible properties of Q4, Q5 and Q6 are based on the answer of Q3. If the user selects the first answer in Q3, then Q4 and Q5 will display; otherwise, Q6 will show up in the survey.

Chapter 3

Reference Manual

3.1 Lexicon

3.1.1 Token Separator

White space ' ', New line '\n', Carriage return '\r' and Horizontal tab'\t' are the token separators.

3.1.2 Comments

Comments begins with '/*' and end with '*/', including everything between them.

3.1.3 Constants

Integer: a sequence of one or more digits.

String: a string starts with a double quote "followed by zero or more characters, ended by a double quote.

3.1.4 Identifier

An identifier is defined as a combination of alphanumeric characters [a-z][AZ][0-9] and must start with an alphabet character. Length of an identifier cannot exceed 32. This language is case sensitive.

3.1.5 Keywords

The following identifiers are reserved for use as key words, and many no be used otherwise:

Question	*QuestionSet*	*Image*
SingleSelection	*MultipleSelection*	*DropdownList*
TextField	*Title*	*Link*
JumpButton	*New*	*AddQuestion*
True	*False*	*Event*
Display	*oDisplay*	*oqDisplay*
qDisplay,	*iDisplay,*	*jDisplay*

3.1.6 Separators

3.1.6.1 ';'

';' is used to separate statements.

For example:

Question Q1, Q2;

QuestionSet qs1=*New QuestionSet* ();

3.1.6.2 '{' and '}'

'{' and '}' are used to separate sets like function declarations, gather the statements in Block, or gathering parameters when assigning the content of the question.

For example:

(1) QuestionLogo (*Question* que, *Image* logo) { }

Main() { } /*separating function declarations*/

(2) *Question* Address;

Address.*Title*="Where do you live?";

Address.*SingleSelection*={"Bronx","New Jersy","China"}; /*assigning the content of the question*/

3.1.6.3 ‘,’

‘,’ is used to separate the parameters and identifier.

For example: the third case above.

3.1.6.4 ‘(‘ and ‘)’

‘(‘ and ‘)’ are used to indicate the token before ‘(‘ is a function and gather the function parameters. Also, they can be used to contain the length of TextField.

For Example:

Q1.TextField(10);

FirstImg (Question Q1, Question Q2, Image img);

3.1.7 Operators

3.1.7.1 ASSIGN

‘=’ is used to represent assignment, usually used when constructing a new QuestionSet, setting the attribute values of each data type and so on.

For example:

Qu.Title = “Are you Columbia Student?”;

Qu.SingleSelection ={“Yes”, “No”};

QuestionSet QS = new QuestionSet( );

img.Link=”C:\Jason\Documents\easysurvey.jpeg”;

3.2.7.2 Point

‘.’ Is usually behind a union type like QuestionSet, Question, Image, and JumpButtion, following the

elements of that union type.

For example:

Q1.SingleSelection;

Img.Link;

3.2.7.3 COLON

The colon in our language is the show the visible *True* setting and visual *False* setting, which would decide what question can me shown based on that question.

For example:

```
Q1.Event(1):True(Q2, Q3), False(Q4);  /*If the informant choose the answer 1 of question Q1 and then
                                        question Q2, Q3 would be shown while Q4 would be hidden from
                                        user*/
```

3.2.8 Attributes and Operations

Title:

Title is one of the attribute of the question. It is a string, which will show the informant what the questions are.

SingleSelection:

The informant can only choose one of the answers.

MultipleSelection:

The informant can choose multiple answers.

DropdownList:

The answer will represent in the form of dropdown list.

Text Field:

The information should type his/her answer in text field.

Link:

Link is the attribute of Image, which would indicate the path of image that would be shown in the web page.

Jump:

Jump is the attribute of JumpButton, which would indicate the next page.

AddQuestion():

Add a Question or Questions to a QuestionSet

For Example:

qs1.AddQuestion(Q1, Q2);

***True*( ):**

Set the questions in True ( ) visible.

For Example:

True (Q1, Q2)

False();

False (Q1, Q2)

3.2.9 Data Type

Question:

Question is used to define the survey questions. It consists of Title and Type which might be one of RadioBox, DropdownList, MultipleBox, and TextField.

For example:

```
Question Name;
Name.Title = “What’s your name?”;
Name.TextField = (10);                    /* the TextField occupy 10 characters */
```

QuestionSet:

QuestionSet is used to define a set of related question and display them as a whole part.

Sample:

```
QuestionSet qs1 = new QuestionSet();
```

Image:

Claim an image for the use of displaying it.

JumpButton:

Jump to next or another page.

3.2.10 Display Set

Easy Survey provides a set of display action for QuestionSet, Question, Image, JumpButton.

Display:

Display a series of QuestionSet of Image.

For Example:

Display QS1, QS2; /*Display QuestionSet QS1, QS2*/

jDisplay:

Display the jump button in objective page.

For Example:

jDisplay JB1, JB2; /*Display jump button JB1, JB2*/

iDisplay:

Display the image in the corresponding location of the object page.

For Example:

iDisplay img1, img2; /*display image img1, img2*/

oqDisplay:

Insert temporary optional display question to object page. This question might be set as visible when other related answer of other question is chosen.

For Example:

oqDisplay Q1, Q2; /* set Question Q1, Q2 as optional question and insert them into page*/

3.4 Declaration

decl:

datatype var_opt SEMI

|datatype ID ASSIGN NEW QUESTIONSET LPAREN RPAREN SEMI

datatype:

QUESTION

|IMAGE

|JUMPBUTTON

|QUESTIONSET

3.4.1 QuestionSet Declaration

The QuestionSet Declaration gives a name to a *QuestionSet* .

The declaration of QuestionSet has the form

QUESTIONSET ID ASSIGN NEW QUESTIONSET LPAREN RPAREN SEMI

For example:

QuestionSet ID = new QuestionSet();

3.4.2 Other datatype declaration

The other datatype declaration is similar to other language like C/C++. It would assign names to each members of the var_decl list.

The declaration of other datatype has the form

expr:

datatype var_decl SEMI

The var_decl contains a list of variable names. The datatype has the form datatype:

QUESTION| IMAGE |JUMPBOTTON

For example:

Question q1, q2, q3;

3.5 Function definition

The function definition gives name to a function, define the data type and the number of its arguments, and the corresponding operations in the function body.

The function definition has the form

func_def :

ID LPAREN args_opt RPAREN LBRACE decl_list stmt_list RBRACE

For example:

FirstImage (Question Q1, Questin Q2, Image img)

{……}

The args_list is a list of arguments. In Easy Survey for the reason of implementation and data type, we do not allow QuestionSet as the parameter of the language.

For example: Question q1, QuestionSet qs1

The decl_list is a list of declarations.

The stmt_list is a list of statements.

3.6 Statements

3.6.1 Common statements

The common statements are composed with expression and a semicolon

3.6.2 Block statements

The block statements have the form

stmt:

expr SEMI

|LBRACE stmt_list RBRACE

3.7 Expressions

3.7.1 Question Assignment Expressions

The Question assignment expression assigns values to each question. The easy survey constrain that the number of potential answer cannot excess 8. Each question has two data attribute, the title of the name and the potential answers. To each of the two attributes, the user should assign corresponding value. When the inputting the potential answers, they should also indicate the attribute of answers, like multiple selection, single selection, drop down list, text field.

The assignment expressions have the form

expr:

|ID POINT MULTIPLE ASSIGN LBRACE string_opt RBRACE

|ID POINT SINGLE ASSIGN LBRACE string_opt RBRACE

|ID POINT DROP ASSIGN LBRACE string_opt RBRACE

|ID POINT TEXT ASSIGN LPAREN LITERAL RPAREN

|ID POINT TITLE ASSIGN STRING_LITERAL

|ID POINT ADDQUESTION LPAREN var_opt RPAREN

3.7. 2 Image Assignment Expression

Image assignment expression assigns value to an image. Normally, the value of the image data type is the path of the image.

expr:

ID POINT LINK ASSIGN STRING_LITERAL

3.7.3 JumpButton Assignment Expression

JumpButton assignment expression assigns a value to a jumpbutton. Normally, the value of the image is a url link, which would link the user to next survey page or other related page.

expr:

|ID POINT JUMP ASSIGN STRING_LITERAL

3.7.4 Addquestion Expressions

Adding a Question to a QuestionSet is a special expression. It has the form expr.

ID POINT ADDQUESTION LPARENT var_decl RPARENT

The var_decl is a list of variables(ID).

3.7.5 Call expressions

The call expression would call the function, and specify the argument of function.

expr.

ID LPAREN var_opt RPAREN

3.7.6 Event Expression

The event would allow the dynamic showing of questions.

They have the form

expr.

|ID POINT EVENT LPAREN LITERAL RPAREN COLON TRUE LPAREN var_opt RPAREN COMMA FALSE LPAREN var_opt RPAREN

3.7.7 Display Expression

The display expression specify the display of different data.

The display expressions have form:

expr

|DISPLAY var_opt

|ODISPLAY var_opt

|QDISPLAY var_opt

|OQDISPLAY var_opt

|JDISPLAY var_opt

|IDISPLAY var_opt

3.8 Scope

Global variables have a global scope. The scope of other variables is in their function declaration block.

Chapter 4

Project Plan

4.1 Team Responsibilities

Each team member will be in charge of different parts according to the interest and experience. In the process of finishing the task, everyone needs to cooperate with each other to test and debug the project code. The mainly task of each team member is showed below.

Taotao Li	Parser, AST, project setup(repository, build script), doc writing
Zaisheng Dai	Complier, byte-code design and implement, and ByteCode Test, doc writing
Yifan Zhang	Byte-code executer, doc writing
Luyao Shi	Scanner, Regression testing, doc writing

4.2 Project Timeline

10-10-2010	Language core function defined
10-30-2010	Project repository setup
11-03-2010	Language Reference Manual
11-05-2010	Scanner done
11-07-2010	Parser done
11-20-2010	Complier done, byte-code design done
11-27-2010	Byte-code done
12-05-2010	Byte-code executer done
12-15-2010	Regression testing done
12-22-2010	Bug fixing done, code freeze

4.3 Software Development Environment

The project is developed on Windows using Objective Caml version 3.11.0 (http://caml.inria.fr/download.en.html). The build script is based on Make(Windows version, which can be found at http://gnuwin32.sourceforge.net/packages/make.htm) . Google code is used to hold our projects (which can be found https://plt-columbia-easy-survey.googlecode.com/svn/trunk/).

4.4 Project Log

9-17-2010	Team formed
09-24-2010	Project initialization
09-25-2010	Proposal draft
09-29-2010	Proposal done
09-31-2010	Proposal revised due to the feedback of TA
10-02-2010	Investigate the output file format in mxml
10-08-2010	Sample input and output files done
10-10-2010	Language core function defined
10-22-2010	Scanner draft
10-29-2010	Parser draft
10-30-2010	Project repository setup
11-03-2010	LRM
11-05-2010	Scanner done
11-07-2010	Parser done
11-20-2010	Complier done, byte-code design done
12-05-2010	Byte-code done
12-07-2010	Byte-code executer done
12-15-2010	Regression testing done
12-21-2010	Demo preparation
12-22-2010	Bug fixing done, code freeze, final report done

Chapter 5

Architectural Design

5.1 Architecture

EasySurvey contains four major blocks: scanner, parser, compiler, byte code executer. In the pars

er part, it will generate the symbol table, and in compiler part, it will generate the byte code which is specifically designed to represent our project. The relationship between these components is shown in Figure5.1.

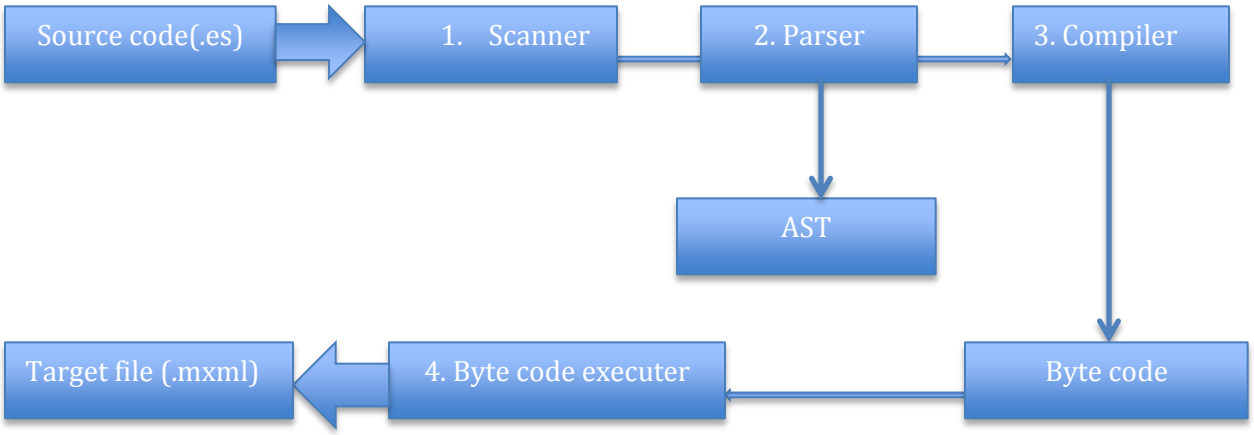

Figure5.1 The diagram of EasySurvey Compiler

The input file for EasySurvey Program is EasySurvey specification file (by convention, the suffix .es) and the final target file is Flex source code (.mxml). The source file will be translated to several tokens after scanner. In the parsing part, the compiler will build a symbol table, which contains all the symbols in the source file. The compiling part will generate the related byte code, and the byte code executer will translate these byte codes to the target file.

5.2 The Runtime Environment

The runtime environment provides the basic environment for running the target files (.mxml file). The generated target file will contain the basic initialization structure as shown in Figure 5.2. This default schema can guarantee that the target files are in flex specification and the compiler will generate the content part written by developer. Then the whole target files will be ready.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                    xmlns:s="library://ns.adobe.com/flex/spark"
                    xmlns:mx="library://ns.adobe.com/flex/mx"    minWidth="955" minHeight="600">
      <fx:Declarations>
    </fx:Declarations>
    <fx:script>
    </fx:script>
</s:Appication>
```

Chapter 6

Testing Plan

6.1 Goals

Testing is one of the most significant parts in software engineering; it should be involved in nearly all the developing and testing phases. In our project, in our developing process, we will use unit testing to assure that the code of small unit is correct. And during the regression testing, we first classify the input and select several representative examples to walk through the critical paths of our project.

6.2 Methods

6.2.1 Phase I

In the developing process, when we finished the first draft of the corresponding module, we will conduct the unit testing for each single module to make sure if it really works. The output should be strictly examined.

6.2.2 Phase II

In the regression process, we will choose several test cases which will cover all the critical paths of EasySurvey program. And first of all, we just work out a simple “Hello world” example and then we will simply add more complicated cases to examine the correctness of the output target file.

6.3 Implementation

6.3.1 Phase I

1. Syntax analysis- To test the output generated by scanner and parser, we wrote several simple but representative example. For example, there is one file named singleSelection.es and we expect the compiler will generate the correct symbol table and the right AST tree.

2. Compiler and byte code generator- In this section, we wrote some test method which will take the mock AST tree as the input and run the related methods in compiler and byte code generator to check if the output byte code is what we expected.

3. Byte code executor- The byte code executor will take the mock byte code (which has been defined as the interface specification of EasySurvey Program) as the input and call the generation methods to write the output content in the file system, in our case, we just write the output in a text file to exam its correctness.

6.3.2 Phase II

In the regression test, it is important that we need to remove all the mock code which is used in the unit testing. In this testing, we wrote several simple but representative source file (.es) and build (through make script) and run the compiler, then we will check the final output file (.mxml). Also, it is critical that our compiler can reject the invalid input such as invalid symbol or structure. So we also wrote some "clearly wrong" input file to check if our compiler can found and take the right action on these unexpected inputs.

Chapter 7

Lessons Learned

By taking this PLT course, we learned the structure of a practical compiler and also we gained some experience with the function language which is much more different form the normal language (C, C++, Java). Also when we were doing this project, we met regularly and discussed the language, revised the grammar, developed and debugged the language. The teamwork sprit and experience we gained from the project also is very valuable for us.

Appendix A

Code Style Conventions

A.1 Introduction

EasySurvey will not force user to choose the specific programming style, however, we wrote this appendix to offer a preferred programming style. The unified programming style will make the source code easy to read and maintain. Also it is critical to obey the same programming style in a team so that every single person can read and revise the code.

A.2 General Principles

We suggest user to define the variables and methods with Capital letter. For example, Question Q1 denotes the declaration of a question. And Func1 (Question Q1, Image Img) {} denotes that there is a function named Func1 and it will accept two arguments. Also, we suggest user to define the function brackets as below:

```
Func1 (Question Q1, Image Img){
        /*function implement*/
}
```

A.3 Documentation Comments

Comments in the source code can improve the readability of code, and in our project, EasySurvey supports the comment format like "/*This is a comment*/".

Appendix B Code Listing

1. Scanner(author: Luyao Shi)

```
{ open Parser }


rule token=parse
| "/*"    { comment lexbuf }          (* Comments *)
| [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| '('              {LPAREN}
| ')'              {RPAREN}
| '{'              {LBRACE}
| '}'              {RBRACE}
| ';'              {SEMI}
| ','               {COMMA}
| '='               {ASSIGN}
| '.'              {POINT}
| ':'              {COLON}

(*key words*)
|"Image"            {IMAGE}
|"JumpButton"       {JUMPBUTTON}
|"Question"           {QUESTION}
|"QuestionSet"      {QUESTIONSET}


|"SingleSelection"    {SINGLE}
|"MultipleSelection"   {MULTIPLE}
|"DropdownList"              {DROP}
|"TextField"          {TEXT}
|"Title"             {TITLE}
|"Link"            {LINK}
|"Jump"             {JUMP}
```

```
|"New"                    {NEW}
|"AddQuestion"        {ADDQUESTION}


|"True"              {TRUE}
|"False"             {FALSE}
|"SetTrue"            {SETTRUE}
|"Event"             {EVENT}


|"Display"                                  {DISPLAY}
|"oDisplay"           {ODISPLAY}
|"oqDisplay"          {OQDISPLAY}
|"qDisplay"           {QDISPLAY}
|"iDisplay"          {IDISPLAY}
|"jDisplay"          {JDISPLAY}

| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| ['0'-'9']+ as num { LITERAL(num) }
| '"'[^'"']*'"' as st { STRING_LITERAL(st) }


(*special character process*)
| _ as char { raise (Failure("illegal character " ^ Char.escaped char))}

| eof { EOF }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }
```

2. Parser (author: Taotao Li)

```
%{ open Ast %}


%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA POINT
%token MULTIPLE SINGLE DROP TEXT TITLE LINK JUMP
%token ASSIGN SETTRUE EVENT TRUE FALSE COLON
```

```
%token IMAGE
%token JUMPBUTTON
%token NEW
%token QUESTION
%token QUESTIONSET
%token DISPLAY ODISPLAY QDISPLAY OQDISPLAY IDISPLAY JDISPLAY
%token ADDQUESTION

%token <string> STRING_LITERAL
%token <string> LITERAL
%token <string> ID
%token EOF


%left ASSIGN

%start program
%type <Ast.program>  program

%%

program:
                                                                    { ([],[]) }
                                | program decl          { ($2 :: (fst $1)), snd $1 }
                                | program func_def                  { fst $1, ($2 :: snd $1) }


func_def :
    ID LPAREN args_opt RPAREN LBRACE decl_list stmt_list RBRACE { { fname = $1;

                                    formals = $3;

                                    locals = List.rev $6;

                                    body = List.rev $7 }}
args_opt:
    /*nothing*/         {[]}
    |args_list          {List.rev $1}


args_list :
                                def_args                    {[$1]}
        |args_list COMMA  def_args { $3::$1 }
```

```
def_args:
        datatype ID                 {($1,$2)}

decl_list:
                                                                                {[]}
                    |decl_list decl          { $2::$1 }
decl:
                    datatype var_opt SEMI                                       {($1,$2) }
                    |datatype ID ASSIGN NEW QUESTIONSET LPAREN RPAREN SEMI {($1,[$2])}



datatype:
   QUESTION                                                 { Question }
   |IMAGE                                                       { Image }
   |JUMPBUTTON                                              { Jump }
   |QUESTIONSET             { QuestionSet }


stmt_list:

          { [ ] }
                                |stmt_list stmt                                 { $2::$1
}

stmt:
                    expr SEMI                                                   { Expr($1) }
                    |LBRACE stmt_list RBRACE                                    { Block(List.rev $2) }


expr:
    /*assign value to question*/
     |ID POINT MULTIPLE ASSIGN LBRACE string_opt RBRACE                 {
Atype($1,Multiple,$6) }
          |ID POINT SINGLE ASSIGN LBRACE string_opt RBRACE                      {
Atype($1,Single,$6) }
          |ID POINT DROP ASSIGN LBRACE string_opt RBRACE                {
Atype($1,Drop,$6) }
          |ID POINT TEXT ASSIGN LPAREN LITERAL RPAREN                   { Text($1,$6)
}
          |ID POINT TITLE ASSIGN STRING_LITERAL                             {
Title($1,$5) }
```

```
    |ID POINT ADDQUESTION LPAREN var_opt RPAREN                                    {
Addquestion($1,$5) }
    |ID POINT LINK ASSIGN STRING_LITERAL                  {Link($1, $5)}
    |ID POINT JUMP ASSIGN STRING_LITERAL                  {Jup($1, $5)}

    |DISPLAY  var_opt
        { Display($2) }
    |ODISPLAY var_opt                                     {Odisplay($2)}
    |QDISPLAY var_opt                                     {Qdisplay ($2)}
    |OQDISPLAY var_opt                                    {Oqdisplay ($2)}
    |JDISPLAY var_opt                                     {Jdisplay($2)}
    |IDISPLAY var_opt                                     {Idisplay($2)}

    |ID POINT EVENT LPAREN LITERAL RPAREN COLON TRUE LPAREN var_opt RPAREN COMMA
FALSE LPAREN var_opt RPAREN {Event($1,$5,$10, $15)}

    |ID LPAREN var_opt RPAREN
        { Call($1, $3) }

    string_opt:
                                {[ ]}
                                    |stringlist                                   {
List.rev $1 }
    stringlist:
                                    STRING_LITERAL                               { [$1] }
                            | stringlist COMMA STRING_LITERAL  { $3::$1 }

var_opt:
                                    {[ ]}
         |var_decl
        {List.rev $1}

var_decl :
                             ID
        { [$1] }
                            |var_decl COMMA ID                                   { $3::$1
}
```

3. AST (author: Taotao Li)

```
type datatype=
        Question
        |QuestionSet
        |Image
  |Jump

type form= Multiple | Single | Drop


type expr=

        |Atype of string*form* (string list)
        |Title of string*string
        |Text of string* string
  |Link of string*string
  |Jup of string*string
        |Addquestion of string*(string list)


  |Display of string list
  |Odisplay of string list
  |Qdisplay of string list
  |Oqdisplay of string list
  |Jdisplay of string list
  |Idisplay of string list

  |Event of string*string*(string list)*(string list)


  |Call of string * (string list)


type stmt=
        Block of stmt list
```

```
        |Expr of expr
        (*|If of expr*stmt*stmt*)

type func_decl={
        fname:string;
        formals:(datatype*string) list;
        locals:(datatype*(string list)) list;
        body:stmt list;

}

type program= ( (datatype* (string list)) list )* (func_decl list)

let rec string_of_expr = function

  |Atype(e1, e2, e3)-> "the question name is " ^ e1 ^(match e2 with Multiple -> "multiple"
                                        | Single -> "single"
                                        | Drop -> "drop") ^ "answers"
  |Title(e1, e2) -> "the type is" ^ e1 ^ "and contents is" ^ e2
  |Text (e1, e2) -> "the question name is " ^ e1 ^ "and length is" ^ e2
  |Link (e1, e2) -> "the image's name is "^ e1 ^ "  and its value is " ^ e2
  |Jup(e1, e2) -> "the jumpbutton's name is " ^ e1 ^"  and its value is " ^ e2
  |Addquestion (e1, e2)->"the questionsetname is " ^ e1 ^ "and the contents is"

  |Display(e1)->"Display the questionset"
  |Odisplay(e1)-> "Display the optional questionset"
  |Qdisplay(e1)-> "Display the question"
  |Oqdisplay(e1)->"Display the optional question"
  |Jdisplay(e1)-> "Display the jumpbutton"
  |Idisplay(e1)-> "Display the image"

  |Event (e1,e2, e3,e4) -> "the question name is " ^e1^ " and input is "^e2 ^ "list of id..."

  |Call(e1, e2)->"call function" ^ e1

let rec string_of_stmt=function
  Block(stmts)->
    "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
    |Expr(expr) -> string_of_expr expr ^ ";\n"

let string_of_vdecl sam = ( match (fst(sam)) with Question -> "Question"
                        | QuestionSet -> "QuestionSet"
                        |Jump -> "Jumpbutton"
```

```
                    |Image -> "image")
                ^ String.concat "" (snd(sam)) ^ ";\n"


let string_of_formals fs= (match fst(fs) with Question -> "Question"
                    | QuestionSet -> "Questionset"
                    | Image-> "image"
                    | Jump -> "Jumpbutton")
            ^ snd(fs) ^";\n"

let string_of_fdecl fdecl =
 fdecl.fname ^ "(" ^ String.concat ", " (List.map string_of_formals fdecl.formals) ^ ")\n{\n" ^
 String.concat "" (List.map string_of_vdecl fdecl.locals) ^
 String.concat "" (List.map string_of_stmt fdecl.body) ^
 "}\n"

let string_of_program (vars, funcs) =
 String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
 String.concat "\n" (List.map string_of_fdecl funcs)
```

4. Compiler (author: Zaisheng Dai)

```
open Ast
open Bytecode


module StringMap = Map.Make(String)

(* Symbol table: Information about all the names in scope *)
type env = {
   function_index : int StringMap.t; (* Index for each function *)
   global_index   : int StringMap.t; (* "Address" for global variables *)
   local_index    : int StringMap.t; (* FP offset for args, locals *)
 }


(*this is the function to calculate the list inside*)

let rec inenum n x =function
                    []->[]
                     |hd::tl->(n, hd)::(inenum (n+x) x tl)
```

```
(*[(Question,[Q1;Q2]);(QuestionSet,[Qs1;Qs2])]*)

let rec enum n= function
                []->[]
                |hd::tl-> let le= (match fst hd with
                                Question->14
                                |QuestionSet->4
                                |Image->2
        |Jump->3) in
                  let next=(List.length (snd hd))*le+n
                   in (inenum n le (snd (hd)))@(enum next tl)


let rec enumf n= function
                []->[]
                |hd::tl-> let le= (match fst hd with
                                Question->(-14)
                                |QuestionSet->(-4)
                                |Image->(-2)
        |Jump ->(-2) ) in
                  let next=(List.length (snd hd))*le+n
                   in (inenum n le (snd (hd)))@(enum next tl)


let rec formenum n= function
                []->[]
                |hd::tl-> let le= (match fst hd with
                                Question->(-14)
                                |QuestionSet->(-4)
                                |Image->(-2)
        |Jump ->(-2)) in  (le+n, snd(hd))::(formenum (n+le) tl)




let arglenfun s e=
            let le=(match fst e with
                                    Question->14
                                    |QuestionSet->4
                                    |Image->2
```

```
        |Jump ->2) in
                                        let arglen=((List.length (snd e))*le) in
                                          arglen+s

let formlenfun s e=
          let le=(match fst e with
                                Question->14
                                |QuestionSet->4
                                |Image->2
        |Jump ->2)
                                        in  le+s   (*need to check*)



let string_map_pairs map pairs=
                List.fold_left ( fun m (i, n)-> StringMap.add n i m) map pairs


let translate (globals, functions)=

   let global_indexes =
                string_map_pairs StringMap.empty (enum 0 globals) in

   let function_indexes=string_map_pairs StringMap.empty
                                (inenum 1 1 (List.map (function f->f.fname) functions)) in

   let translate env fdecl=
    let formals_length= List.fold_left formlenfun 0 fdecl.formals
     and locals_length= List.fold_left arglenfun 0 fdecl.locals
     and locals_offsets= enum 1 fdecl.locals
     and formals_offsets= formenum (-1) fdecl.formals in

     let env={ env with local_index= string_map_pairs
                        StringMap.empty (locals_offsets@formals_offsets)
                } in

     let rec expr= function
     | Link (e1, e2) -> (try [LoLkwb ((StringMap.find e1 env.local_index), e2) ]
                    with Not_found-> try
                     [GlLkwb((StringMap.find e1 env.local_index), e2)]
                     with Not_found->
                     raise (Failure ("undeclared variable" ^ e1)))
```

```
    |Jup (e1, e2)-> (try [LoJpwb ((StringMap.find e1 env.local_index), e2)]
                    with Not_found-> try
                      [GlJpwb ((StringMap.find e1 env.local_index), e2)]
                    with Not_found->
                    raise (Failure ("undeclared variable" ^ e2)))
    | Title(e1,e2)->(try [Lowbt ((StringMap.find e1 env.local_index), e2, e1)]
                          with Not_found-> try
                      [Glwbt ((StringMap.find e1 env.global_index),e2, e1)]
                               with Not_found->
                                    raise (Failure ("undeclared variable" ^ e1)))

        |Atype(e1,form,e2)->  let atlist=(List.concat (List.map (function a->
                               try [Pushal ((StringMap.find e1 env.local_index), a)]
                                with Not_found-> try
                                  [Pushag ((StringMap.find e1 env.global_index), a)]
                                  with Not_found->
                                  raise (Failure ("undeclared variable" ^ e1 ))) e2))
                          and tp=( match form with
                                     Single->1
                                     |Multiple->2
                                     |Drop->3
                                   )
                                   in( try [Lowba ((StringMap.find e1 env.local_index),tp)]@atlist
                                       with Not_found-> try
                                            [Glwba ((StringMap.find e1 env.global_index),tp)]@atlist
                                   with Not_found->
                                        raise (Failure ("undeclared variable" ^ e1)))
    |Text(e1,e2)->( try [Lotwbt ((StringMap.find e1 env.local_index), int_of_string e2)]
                               with Not_found-> try
                                    [Gltwbt ((StringMap.find e1 env.global_index), int_of_string
e2)]
                               with Not_found->
                                    raise (Failure ("undeclared variable"^ e1))

                             )

        |Addquestion(e1,e2)->let len=(List.length e2)
                          and qlist=(List.map (function a-> try (Loquetrace (StringMap.find a
env.local_index))
                                                          with Not_found ->
                                                                raise (Failure ("undecluared
variable"^ a))
```

```
                                ) e2)
                                        in (try ([Loqswb ((StringMap.find e1
env.local_index),len)])@qlist
                                      with Not_found->
                                              raise (Failure ("undeclared variable"^ e1))
                                              )
  | Call (fname, actuals) -> (try
        (List.concat (List.map (function a -> try [Lfp (StringMap.find a env.local_index)]
                          with Not_found-> try
                          [Lod (StringMap.find a env.global_index)]
                          with Not_found->
                          raise (Failure ("undeclared variable" ^ a))
                        ) (List.rev actuals))) @
        [Jsr (StringMap.find fname env.function_index) ]
    with Not_found -> raise (Failure ("undefined function " ^ fname)))



  |Display(e1)-> (List.concat (List.map ( function a-> try [Loshow (StringMap.find a
env.local_index)]
                                                        with Not_found->
                                                          raise (Failure ("undeclared variable"^
a))) e1))



  |Odisplay (e1)-> (List.concat (List.map ( function a-> try [LoOshow (StringMap.find a
env.local_index)]
                                                        with Not_found->
                                                          raise (Failure ("undeclared variable"^
a))) e1))


  |Qdisplay (e1)->(List.concat (List.map ( function a-> try [LoQshow (StringMap.find a
env.local_index)]
                                                        with Not_found-> try
                                    [GlQshow (StringMap.find a env.global_index)]
                                  with Not_found->
                                                          raise (Failure ("undeclared variable"^
a))) e1))
```

```
  |Oqdisplay (e1)->(List.concat (List.map ( function a-> try [LoOqshow (StringMap.find a
env.local_index)]
                                                    with Not_found-> try
                                    [GlOqshow (StringMap.find a env.global_index)]
                                with Not_found->
                                                      raise (Failure ("undeclared variable"^
a))) e1))
  |Idisplay (e1)->(List.concat (List.map ( function a-> try [LoIshow (StringMap.find a
env.local_index)]
                                                    with Not_found-> try
                                    [GlIshow (StringMap.find a env.global_index)]
                                with Not_found->
                                                      raise (Failure ("undeclared variable"^
a))) e1))

  |Jdisplay (e1)->(List.concat (List.map ( function a-> try [LoJshow (StringMap.find a
env.local_index)]
                                                    with Not_found-> try
                                    [GlJshow (StringMap.find a env.global_index)]
                                with Not_found->
                                                      raise (Failure ("undeclared variable"^
a))) e1))



 |Event (e1, e2, e3,e4)-> (try [LoEvd ((StringMap.find e1 env.local_index), e2)]
                with Not_found->try
                    [GlEvd ((StringMap.find e1 env.global_index), e2)]
                    with Not_found->
                      raise (Failure ("undeclared variable " ^ e1)))@
                      (List.concat (List.map ( function a-> try [LoSetT (StringMap.find a
env.local_index)]
                                                    with Not_found-> try
                                    [GlSetT (StringMap.find a env.global_index)]
                                with Not_found->
                                                      raise (Failure ("undeclared variable "^
a))) e3))@
                          (List.concat (List.map ( function a-> try [LoSetF (StringMap.find a
env.local_index)]
                                                    with Not_found-> try
                                    [GlSetF (StringMap.find a env.global_index)]
                                with Not_found->
```

```
                                                    raise (Failure ("undeclared variable "^
a))) e4))@[EventEnd]



 in let rec stmt= function
     Expr e->expr e
         |Block s1-> List.concat (List.map stmt s1)

 in [Ent locals_length]@
  stmt (Block fdecl.body)@
         [Rts formals_length]

 in let env = { function_index = function_indexes;
                      global_index = global_indexes;
                      local_index = StringMap.empty } in

(* Code executed to start the program: Jsr main; halt *)
let entry_function = try
  [Jsr (StringMap.find "main" function_indexes); Hlt]
with Not_found -> raise (Failure ("no \"main\" function"))
in

(* Compile the functions *)
let func_bodies = entry_function :: List.map (translate env) functions in

(* Calculate function entry points by adding their lengths *)
let (fun_offset_list, _) = List.fold_left
   (fun (l,i) f -> (i :: l, (i + List.length f))) ([],0) func_bodies in
let func_offset = Array.of_list (List.rev fun_offset_list) in

{ num_globals =List.fold_left arglenfun 0 globals;
  (* Concatenate the compiled functions and replace the function
     indexes in Jsr statements with PC values *)
  text = Array.of_list (List.map (function
         Jsr i when i > 0 -> Jsr func_offset.(i)
    | _ as s -> s) (List.concat func_bodies))
}
```

5. Bytecode (author: Zaisheng Dai)

```
type bsmt=
```

```
Ent of int
|Rts of int
|Jsr of int
        |Push of string
|Pushal of int*string
|Pushag of int*string
|Lfp of int
|Lod of int
|LoLkwb of int*string
|GlLkwb of int*string
|LoJpwb of int*string
|GlJpwb of int*string
        |Lowbt of int*string*string    (*local title write back*)
        |Glwbt of int*string*string    (*global title write back*)

        |Lowba of int*int (*local answers write back*)
        |Glwba of int*int (*global answers write back*)
        |Lotwbt of int*int (*local text length write back*)
        |Gltwbt of int*int (*global text lenght write back*)
        |Loqswb of int*int (*local question set write back*)

        |Loquetrace of int (*trace the qeuestion locally*)



        |Loshow of int (*show the local questionset*)
|LoOshow of int (*print the optional local question set*)
|LoQshow of int (*show the local question, Q means question*)
|LoOqshow of int (*print the optional question, O means option, note that q is lower case*)
|LoIshow of int (*show the local image *)
|LoJshow of int (*show the local jump button*)


(*Global display*)

|GlQshow of int  (*show the global question*)
|GlOqshow of int (*show the global optionalqueston*)
|GlIshow of int  (*show the global image*)
|GlJshow of int  (*show the global jumpbutton*)

(*event*)

|LoEvd of int*string
```

```
   |GlEvd of int*string
   |LoSetT of int
   |GlSetT of int
   |LoSetF of int
   |GlSetF of int
   |EventEnd


   |Nop
   |Hlt

type prog={
    num_globals : int;
    text : bsmt array;
}

let string_of_stmt = function
   |Ent(i)->"ent " ^ string_of_int i
   |Lfp (i) ->"lfp " ^ string_of_int i
   |Lod(i)-> "Lod " ^ string_of_int i
   |Rts(i) -> "Rts " ^ string_of_int i
   |Jsr(i)->"Jsr  "^ string_of_int i
   |Push(i)->"Push " ^ i
   |Pushal (i, j)->"Pushal " ^ string_of_int i ^"  "^ j
   |Pushag (i,j)-> "Pushag " ^ string_of_int i ^"  "^ j

   (*Link and JumpButton*)

   |LoLkwb(i,j) -> "LoLkwb " ^ string_of_int i ^ "  " ^j
   |GlLkwb(i, j)-> "GlLkwb " ^ string_of_int i^ "  " ^ j
   |LoJpwb(i,j)-> "LoJpwb " ^ string_of_int i ^ " " ^ j
   |GlJpwb(i,j)-> "GlJpwb " ^ string_of_int i^ " "^ j

   |Lowbt(i,j,k)->"Lowbt " ^ string_of_int i ^ " "^j ^" "^ k
   |Glwbt(i,j,k)->"Glwbt " ^ string_of_int i ^ " "^ j ^" "^ k
   |Lowba(i,j)->"Lowwba " ^ string_of_int i ^ string_of_int j
   |Glwba(i,j)->"Glwba " ^ string_of_int i ^ string_of_int j
   |Lotwbt(i,j)->"Lotwbt " ^ string_of_int i ^ string_of_int j
   |Gltwbt (i, j)-> "Gltwbt " ^ string_of_int i ^ string_of_int j

   |Loqswb(i,j)->"Loqswb " ^ string_of_int i ^" "^ string_of_int j

   |Loquetrace(i)-> "Loquetrace " ^ string_of_int i
```

```
  (*All kinds of display*)
  (*there is not big difference between local image  jumpbutton show*)

  |Loshow(i)->"Loshow" ^ string_of_int i
  |LoOshow (i)-> "LoOshow " ^ string_of_int i
  |LoQshow (i)-> "LoQshow " ^ string_of_int i
  |LoOqshow (i)-> "LoOqshow " ^ string_of_int i
  |LoIshow (i) -> "LoIshow " ^ string_of_int i
  |LoJshow (i) -> "LOJshow " ^ string_of_int i

  (*global show*)

  |GlQshow (i)-> "QGlshow "^  string_of_int i
  |GlOqshow (i)-> "OqGlshow "^ string_of_int i
  |GlIshow (i) -> "GlIshow  " ^ string_of_int i
  |GlJshow (i) -> "GLJshow "^ string_of_int i

  (*event*)

  |GlEvd (i,j)-> "GlEvd "^ string_of_int i ^"  "^ j
  |LoEvd (i,j)-> "LoEvd "^ string_of_int i ^"  "^j
  |GlSetT (i)-> "GlSetT " ^ string_of_int i
  |LoSetT (i)-> "LoSetT " ^ string_of_int i
  |GlSetF (i)-> "GlSetF" ^ string_of_int i
  |LoSetF (i)-> "LoSetF " ^ string_of_int i

  |EventEnd -> "EventEnd"

  |Nop->"not yet available"
  |Hlt->"Hlt"

let string_of_prog p =
 string_of_int p.num_globals ^ " global variables\n" ^
 let funca = Array.mapi
    (fun i s -> string_of_int i ^ " " ^ string_of_stmt s) p.text
 in String.concat "\n" (Array.to_list funca)
```

6. Execute (author: Yifan Zhang)

```
open Ast
```

```
open Bytecode
open Printf

let curve s = let len=(String.length s) in s.[len-1] <- ' '; s.[0] <- ' '; s;;


let execute_prog prog=
let script =Array.make 2000 "0" in


let out=Array.make 2000 "0" in let stack=Array.make 500 "0"
and globals = Array.make prog.num_globals "0"
in let rec exec fp sp pc= match prog.text.(pc) with

|GlEvd (i,j)->
        let ii=ref 0 in let iii=while ((script.(!ii).[0]!='0')) do (ii:=!ii+1) done;
         !ii in script.(iii)<-"protected function "^ globals.(i+1) ^ j ^"(event:MouseEvent):void";
        script.(iii+1)<-"{"; globals.(i+13)<-"1";        exec fp sp (pc+1);


|GlSetT i-> let ii=ref 0 in let iii=while (script.(!ii).[0]!='0') do (ii:=!ii+1) done;
          !ii in let iiii= ref 0 in script.(iii)<-globals.(i+1)^".visible=true;";
          script.(iii+1)<-globals.(i+1)^".includeInLayout=true;";
          while(!iiii< (int_of_string globals.(i+3))) do
          ( script.(iii+2+(!iiii)*2)<-globals.(i+1)^(string_of_int ((!iiii)+1))^".visible=true;";
          script.(iii+3+(!iiii)*2)<-globals.(i+1)^(string_of_int ((!iiii)+1))^".includeInLayout=true;" ;
          iiii:=!iiii+1;) done;
          exec fp sp (pc+1);


|GlSetF i-> let ii=ref 0 in let iii=while (script.(!ii).[0]!='0') do
```

```
                    (ii:=!ii+1) done;
                    !ii in let iiii= ref 0 in script.(iii)<-globals.(i+1)^".visible=false;";
                    script.(iii+1)<-globals.(i+1)^".includeInLayout=false;";
                    while(!iiii< (int_of_string globals.(i+3))) do
                    ( script.(iii+2+(!iiii)*2)<-globals.(i+1)^(string_of_int ((!iiii)+1))^".visible=false;";
                     script.(iii+3+(!iiii)*2)<-globals.(i+1)^(string_of_int
((!iiii)+1))^".includeInLayout=false;" ;
                     iiii:=!iiii+1;) done;
                     exec fp sp (pc+1);



|LoEvd (i,j)-> let ii=ref 0 in
        let iii=while ((script.(!ii).[0]!='0')) do (ii:=!ii+1) done;
        !ii in script.(iii)<-"protected function "^ stack.(fp+i+1) ^ j ^"(event:MouseEvent):void";
         script.(iii+1)<-"{";
         stack.(fp+i+13)<-"1";
          exec fp sp (pc+1);



|LoSetT i-> let ii=ref 0 in
            let iii=while (script.(!ii).[0]!='0') do (ii:=!ii+1) done;
            !ii in let iiii= ref 0 in script.(iii)<-stack.(fp+i+1)^".visible=true;";
             script.(iii+1)<-stack.(fp+i+1)^".includeInLayout=true;";
             while(!iiii< (int_of_string stack.(fp+i+3))) do
             ( script.(iii+2+(!iiii)*2)<-stack.(fp+i+1)^(string_of_int ((!iiii)+1))^".visible=true;";
              script.(iii+3+(!iiii)*2)<-stack.(fp+i+1)^(string_of_int   ((!iiii)+1))^".includeInLayout=true;"
;
              iiii:=!iiii+1;) done;  exec fp sp (pc+1);

|LoSetF i-> let ii=ref 0 in
      let iii=while (script.(!ii).[0]!='0') do
      (ii:=!ii+1) done; !ii in let iiii= ref 0 in script.(iii)<-stack.(fp+i+1)^".visible=false;";
```

```
        script.(iii+1)<-stack.(fp+i+1)^".includeInLayout=false;";
         while(!iiii< (int_of_string stack.(fp+i+3))) do
         ( script.(iii+2+(!iiii)*2)<-stack.(fp+i+1)^(string_of_int ((!iiii)+1))^".visible=false;";
          script.(iii+3+(!iiii)*2)<-stack.(fp+i+1)^(string_of_int ((!iiii)+1))^".includeInLayout=false;" ;
          iiii:=!iiii+1;) done;
             exec fp sp (pc+1);

|EventEnd->let ii=ref 0 in
           let iii=while (script.(!ii).[0]!='0') do
           (ii:=!ii+1) done; !ii in script.(iii)<-"}";
             exec fp sp (pc+1);



| GlOqshow i-> let ii=ref 0 in
              let iiii= while((out.(!ii).[0])!='0') do
               (ii:=!ii+1) done;
                 !ii in

                   out.(iiii)<-"<mx:HBox id=\"" ^ globals.(i+1)^"\" visible=\"false\"
includeInLayout=\"false\">" ;
                   out.(iiii+1) <-"<mx:Label text=\"" ^ globals.(i+1)^":\"/>";

                   out.(iiii+2) <- "<mx:Label  text=" ^ globals.(i+2)^"/>";

                   out.(iiii+3)<- "</mx:HBox>";
                   if ( (int_of_string globals.(i+4))==1)
                   then (let j=ref 0 in
                   (while(!j<(int_of_string globals.(i+3)))
                   do (

                     if((int_of_string globals.(i+13))==0)
```

```
                then
                (
                   out.(iiii+(!j)+4)<- "<s:RadioButton id=\"" ^
                   globals.(i+1)^ (string_of_int ((!j)+1)) ^
                    "  groupName=\"  "^  globals.(i+1)^"group\"  \"  visible=\"false\"
includeInLayout=\"false\" label="
                    ^ globals.(i+5+(!j)) ^"/>"; )
                else(
                     out.(iiii+(!j)+4)<- "<s:RadioButton id=\"" ^ globals.(i+1)^ (string_of_int ((!j)+1)) ^
                      " groupName=\" "^ globals.(i+1)^
                      "group\" \" visible=\"false\" includeInLayout=\"false\" label=" ^
                      globals.(i+5+(!j)) ^" click=\"" ^
                      globals.(i+1)^(string_of_int ((!j)+1)) ^"(event)\"/>"; );


                        j:=!j+1;)
                done) ) else ();
                if((int_of_string globals.(i+4))==2) then
                (let jj=ref 0 in
                (while(!jj<(int_of_string globals.(i+3)))
                   do (out.(iiii+(!jj)+4)<- "<s:CheckBox id=\""^globals.(i+1)^
                   (string_of_int ((!jj)+1))^ "\" visible=\"false\" includeInLayout=\"false\" label=" ^
                    globals.(i+5+(!jj))^ "/>";jj:=!jj+1;)
                      done)
                ) else();


              if((int_of_string globals.(i+4))==3) then
              (let jj=ref 1 in out.(iiii+4)<-"<s:DropDownList id=\""^
                globals.(i+1)  ^  "\"  visible=\"false\"  includeInLayout=\"false\"
requireSelection=\"true\">";
                 out.(iiii+5)<-"<s:dataProvider>";
                 out.(iiii+6)<-"<s:ArrayList source=\"[";
                 out.(iiii+7)<-curve (globals.(i+5));
```

```
            while(!jj<(int_of_string globals.(int_of_string globals.(i+3))))
            do (out.(iiii+(!jj)+7)<- "," ^ (curve globals.(i+5+(!jj))));
            jj:=!jj+1;) done;
            out.(iiii+(!jj)+7)<-"]\"/>";
            out.(iiii+(!jj)+8)<-"</s:dataProvider>";
             out.(iiii+(!jj)+9)<-"</s:DropDownList>"
            ) else();
            if((int_of_string globals.(i+4))==4) then(out.(iiii+4) <-"<mx:TextArea id=\""
            ^ globals.(i+1)^
            "1 \" visible=\"false\" includeInLayout=\"false\" height=\"21\" width=\"" ^
              string_of_int ((int_of_string globals.(i+3))*10)     ^"\"/>"
            ) else(); exec fp sp (pc+1);

|LoOshow i-> exec fp sp (pc+1);

|Glwbt (i,j,k)-> globals.(i+1)<- k ;
          globals.(i+2) <- j;
          globals.(i) <- string_of_int 1; exec fp sp (pc+1)

(*write back the global answer, i is the length*)

|Glwba(i,j)-> globals.(i+4) <- string_of_int j; exec fp sp  (pc+1)

(*global text write back*)

|Gltwbt (i,j) -> globals.(i+3) <- string_of_int j;
          globals.(i+4) <- string_of_int 4;
          exec fp sp (pc+1)

(*3 is the place that we preserve the length information*)
```

```
| Pushag (i,j)-> let al = ref (i+5) in
                let a=while (globals.(!al).[0] !='0') do (al:=!al+1) done;
                  (!al) in globals.(a) <-j;
                  globals.(i+3) <-string_of_int ((int_of_string globals.(i+3))+1);
                exec fp sp (pc+1);



| Push i ->stack.(sp)<-i;
        exec fp (sp+1) (pc+1)
| Hlt-> let oc=open_out "test1.mxml" in fprintf oc "%s\n""<?xml version=\"1.0\" encoding=\"utf-8\"?>";
          fprintf oc "%s\n"  "<s:Application xmlns:fx=\"http://ns.adobe.com/mxml/2009\""; fprintf oc "%s\n"
          "xmlns:s=\"library://ns.adobe.com/flex/spark\""; fprintf oc "%s\n" "xmlns:mx=\"library://ns.adobe.com/flex/mx\" minWidth=\"955\" minHeight=\"600\">";
            fprintf oc "%s\n"  "<mx:VBox height=\"492\" x=\"48\" y=\"16\" width=\"1124\">";
            let w= ref 0 in
              while( out.(!w).[0]!='0') do
              (fprintf oc "%s\n" out.(!w); w:=!w+1) done;
                fprintf oc "%s\n"  "</mx:VBox>";
                fprintf oc "%s\n"  "<fx:Script>";
                fprintf oc "%s\n"  "<![CDATA[";

            let w= ref 0 in
            while( script.(!w).[0]!='0') do
            (fprintf oc "%s\n" script.(!w); w:=!w+1) done;

          fprintf oc "%s\n"  "]]>";
          fprintf oc "%s\n"  "</fx:Script>";
```

```
            fprintf oc "%s\n"  "</s:Application>";
            close_out oc;
            ()

| Lowbt (i,j,k)-> stack.(fp+i+1)<- k;
             stack.(fp+i+2)<- j;
             stack.(fp+i)<- string_of_int 1;
             exec fp sp (pc+1)

| Lowba(i, j)-> stack.(fp+i+4)<- string_of_int  j ;
            exec fp sp (pc+1)

| Lotwbt (i,j) -> stack.(fp+i+3) <- string_of_int j;
              stack.(fp+i+4)<- string_of_int 4;
              exec fp sp (pc+1)

| Loqswb (i,j) -> let ii= ref (i+fp) in
              let iii=  while( (int_of_string stack.(!ii))!= 0) do
              (ii:= (int_of_string stack.( (!ii))))  done ;
                   !ii in stack.(iii) <- string_of_int sp ;
                   stack.(iii+1)<- string_of_int j; exec fp (sp+2) (pc+1)

| Loquetrace i -> stack.(sp) <- string_of_int i;
              exec fp (sp+1) (pc+1)

|Jsr i-> stack.(sp)<-string_of_int  (pc+1);
         exec fp (sp+1) i;

| Ent i-> stack.(sp)<-string_of_int fp;
            exec sp (sp+i+1) (pc+1);
```

```
| Rts i ->let new_fp= stack.(fp) and new_pc=stack.(fp-1) in
          exec (int_of_string new_fp) (fp-i) (int_of_string new_pc)

| Pushal ( i, j)-> let al= ref (fp+i+5) in
          let a= while (stack.(!al).[0]!='0') do(al:=!al+1) done;
          (!al) in stack.(a)<- j;
          stack.(fp+i+3)<-string_of_int ((int_of_string stack.(fp+i+3))+1);
            exec fp sp (pc+1);

| Lfp i-> if(stack.(fp+i).[0]='1') then
         (let lf=ref 0 in while(!lf<14) do
         (stack.(sp+(!lf))<- stack.(fp+i+(!lf));lf:=!lf+1) done;
          exec  fp (sp+14) (pc+1);) else();
        if(stack.(fp+i).[0]='2') then
        (let lf=ref 0 in while(!lf<2) do
        (stack.(sp+(!lf))<- stack.(fp+i+(!lf));lf:=!lf+1) done;
        exec  fp (sp+2) (pc+1);) else();




| LoLkwb (i,j) -> stack.(fp+i)<-(string_of_int 2);
              stack.(fp+i+1)<- j;
             exec fp sp (pc+1);
| LoJpwb(i,j)-> stack.(fp+i)<-(string_of_int 3);
          stack.(fp+i+1)<- j;
          exec fp sp (pc+1);
|LoIshow i-> let ii=ref 0 in
          let iii= while(out.(!ii).[0]!='0') do(ii:=!ii+1;) done;
          !ii in out.(iii)<-"<mx:Image source ="^
          stack.(fp+i+1)^ " width=\"624\" height=\"114\"/>";
```

```
            exec fp sp (pc+1);
|LoJshow i->let ii=ref 0 in let iii= while(out.(!ii).[0]!='0') do
                (ii:=!ii+1;) done;
                !ii in out.(iii)<-"<s:Button label=\"Sumit\" click=\"navigateToURL(new URLRequest('"^
                 (curve stack.(fp+i+1))^ "'),'_top')\" />"; exec fp sp (pc+1)
|Nop -> exec fp sp (pc+1);

|Lod i-> exec fp sp (pc+1);

| GlLkwb (i,j) -> globals.(i)<-(string_of_int 2);
                globals.(i+1)<- j; exec fp sp (pc+1);
| GlJpwb(i,j)-> globals.(i)<-(string_of_int 3);
            globals.(i+1)<- j; exec fp sp (pc+1);

|GlIshow i-> let ii=ref 0 in
                let iii= while(out.(!ii).[0]!='0') do(ii:=!ii+1;) done;
                !ii in out.(iii)<-"<mx:Image source ="^ globals.(i+1)^
                 " width=\"624\" height=\"114\"/>";
                  exec fp sp (pc+1);
|GlJshow i->let ii=ref 0 in
                let iii= while(out.(!ii).[0]!='0') do
                (ii:=!ii+1;) done;
                !ii in out.(iii)<-"<s:Button label=\"Sumit\" click=\"navigateToURL(new URLRequest('"^
                 (curve globals.(i+1))^ "'),'_top         ')\" />";
                 exec fp sp (pc+1)


| GlQshow i-> let ii=ref 0 in
              let iiii= while((out.(!ii).[0])!='0') do
               (ii:=!ii+1) done;
               !ii in
```

```
out.(iiii)<-"<mx:HBox>";
out.(iiii+1)<- "<mx:Label text=\"" ^ globals.(i+1)  ^":\"/>";
out.(iiii+2)<-  "<mx:Label  text=" ^ globals. (i+2)
^"/>";
out.(iiii+3)<- "</mx:HBox>";
if ( (int_of_string globals.(i+4))==1)
then (let j=ref 0 in
(while(!j<(int_of_string globals.(i+3)))
do (
   if(int_of_string globals.(i+13)==0)
then
(
   out.(iiii+(!j)+4)<- "<s:RadioButton label=" ^ globals.(i+5+(!j)) ^
   " groupName=\" "^ globals.(i+1)^"group\" "
   ^"/>";
)
else(
out.(iiii+(!j)+4)<- "<s:RadioButton label=" ^
 globals.(i+5+(!j)) ^   " groupName=\" "^
 globals.(i+1)^"group\"" ^" click=\"" ^
  globals.(i+1)^(string_of_int ((!j)+1))
^"(event)\"/>" ;
   );


j:=!j+1;)
done) ) else ();


if((int_of_string globals.(i+4))==2) then
(let jj=ref 0 in
```

```
        (while(!jj<(int_of_string globals.(i+3)))
        do (out.(iiii+(!jj)+4)<- "<s:CheckBox label="
         ^ globals.(i+5+(!jj))
           ^ "/>";jj:=!jj+1;)
        done)
      ) else();

    if((int_of_string globals.(i+4))==3) then
        (let jj=ref 1 in out.(iiii+4)<-"<s:DropDownList requireSelection=\"true\">";
         out.(iiii+5)<-"<s:dataProvider>";
         out.(iiii+6)<-"<s:ArrayList source=\"[";
         out.(iiii+7)<-(curve globals.(i+5));

    while(!jj<(int_of_string globals.(int_of_string globals.(i+3))))
    do (out.(iiii+(!jj)+7)<- "," ^ (curve globals.(i+5+(!jj))));
    jj:=!jj+1;)
    done;
    out.(iiii+(!jj)+7)<-"]\"/>";
    out.(iiii+(!jj)+8)<-"</s:dataProvider>";
    out.(iiii+(!jj)+9)<-"</s:DropDownList>"
    ) else();

    if((int_of_string globals.(i+4))==4) then
      (out.(iiii+4) <-"<mx:TextArea height=\"21\" width=\"" ^
       string_of_int ((int_of_string globals.(i+3))*10)     ^"\"/>"
      ) else();
      exec fp sp (pc+1);

| LoOqshow i-> let ii=ref 0 in let iiii= while((out.(!ii).[0])!='0') do
          (ii:=!ii+1) done;
          !ii in
```

```
   out.(iiii)<-"<mx:HBox id=\"" ^
    stack.(fp+i+1)^"\" visible=\"false\" includeInLayout=\"false\">" ;
     out.(iiii+1)<- "<mx:Label text=\"" ^ stack.(fp+i+1)

   ^":\"/>";
 out.(iiii+2)<-  "<mx:Label  text=" ^ stack. (fp+i+2)
   ^"/>";
out.(iiii+3)<- "</mx:HBox>";
if ( (int_of_string stack.(i+4+fp))==1)
then (let j=ref 0 in
(while(!j<(int_of_string stack.(i+3+fp)))
   do (

     if(int_of_string stack.(i+13+fp)==0)
        then
     (
          out.(iiii+(!j)+4)<- "<s:RadioButton id=\"" ^ stack.(i+1+fp)^
           (string_of_int ((!j)+1)) ^ "\" visible=\"false\" includeInLayout=\"false\" label="
^
            stack.(fp+i+5+(!j)) ^ " groupName=\" "^ stack.(fp+i+1)^"group\" "
             ^"/>";
     )
   else(
        out.(iiii+(!j)+4)<- "<s:RadioButton id=\"" ^ stack.(fp+i+1)^
         (string_of_int ((!j)+1)) ^ " groupName=\" "^
         stack.(fp+i+1)^"group\" " ^
          "\" visible=\"false\" includeInLayout=\"false\" label=" ^
          stack.(fp+i+5+(!j)) ^" click=\"" ^
          stack.(fp+i+1)^(string_of_int ((!j)+1))
           ^"(event)\"/>";
           );
```

```
      j:=!j+1;)
    done) ) else ();
if((int_of_string stack.(i+4+fp))==2) then
(let jj=ref 0 in
   (while(!jj<(int_of_string stack.(i+3+fp)))
   do (out.(iiii+(!jj)+4)<- "<s:CheckBox id=\""^
   stack.(fp+i+1)^(string_of_int ((!jj)+1))^
    "\" visible=\"false\" includeInLayout=\"false\" label=" ^ stack.(i+5+fp+(!jj))
      ^ "/>";jj:=!jj+1;)
            done)
   ) else();
if((int_of_string stack.(i+4+fp))==3) then
(let jj=ref 1 in out.(iiii+4)<-"<s:DropDownList id=\"" ^stack.(fp+i+1) ^
"\" visible=\"false\" includeInLayout=\"false\" requireSeleetion=\"true\">";
out.(iiii+5)<-"<s:dataProvider>";out.(iiii+6)<-"<s:ArrayList source=\"[";
out.(iiii+7)<- (curve stack.(i+5+fp));

   while(!jj<(int_of_string stack.(i+3+fp)))
   do (out.(iiii+(!jj)+7)<- "," ^ (curve stack.(fp+i+5+(!jj))); jj:=!jj+1;)
   done;
   out.(iiii+(!jj)+7)<-"]\"/>";
   out.(iiii+(!jj)+8)<-"</s:dataProvider>";
    out.(iiii+(!jj)+9)<-"</s:DropDownList>"
   ) else();


if((int_of_string stack.(i+4+fp))==4) then
(out.(iiii+4) <-"<mx:TextArea id=\"" ^ stack.(fp+i+1)^
```

```
                "1 \" visible=\"false\" includeInLayout=\"false\" height=\"21\" width=\"" ^
                 string_of_int ((int_of_string stack.(i+3+fp))*10)     ^"\"/>"
                ) else(); exec fp sp (pc+1);


| LoQshow i-> let ii=ref 0 in
            let iiii= while((out.(!ii).[0])!='0') do
             (ii:=!ii+1) done;
              !ii in
                out.(iiii)<-"<mx:HBox>";
                out.(iiii+1)<- "<mx:Label text=\"" ^ stack.(fp+i+1)

                ^":\"/>";
              out.(iiii+2)<-  "<mx:Label  text=" ^ stack. (fp+i+2) ^"/>";
                   out.(iiii+3)<- "</mx:HBox>";
              if ( (int_of_string stack.(i+4+fp))==1)
              then (let j=ref 0 in
                   (while(!j<(int_of_string stack.(i+3+fp)))
                      do (
                      if(int_of_string stack.(fp+i+13)==0)
                      then
                      (
                         out.(iiii+(!j)+4)<- "<s:RadioButton label=" ^ stack.(fp+i+5+(!j)) ^
                          " groupName=\" "^ stack.(fp+i+1)^"group\" "
                            ^"/>";
                      )
              else(
                 out.(iiii+(!j)+4)<- "<s:RadioButton label=" ^
                  stack.(fp+i+5+(!j)) ^
                   " groupName=\" "^ stack.(fp+i+1)^
                   "group\" " ^" click=\"" ^ stack.(fp+i+1)^(string_of_int ((!j)+1))
```

```
      ^"(event)\"/>";
        );

j:=!j+1;)
 done) ) else ();
if((int_of_string stack.(i+4+fp))==2) then
(let jj=ref 0 in
   (while(!jj<(int_of_string stack.(i+3+fp)))
      do
      (out.(iiii+(!jj)+4)<- "<s:CheckBox label=" ^ stack.(i+5+fp+(!jj))
         ^ "/>";jj:=!jj+1;)
            done)
      ) else();

if((int_of_string stack.(i+4+fp))==3) then
(let jj=ref 1 in out.(iiii+4)<-"<s:DropDownList requireSelection=\"true\">";
 out.(iiii+5)<-"<s:dataProvider>";
 out.(iiii+6)<-"<s:ArrayList source=\"[";
   out.(iiii+7)<-(curve stack.(i+5+fp));
   while(!jj<(int_of_string stack.(i+3+fp)))
   do (out.(iiii+(!jj)+7)<- "," ^
    (curve stack.(fp+i+5+(!jj)));
    jj:=!jj+1;) done;
    out.(iiii+(!jj)+7)<-"]\"/>";
    out.(iiii+(!jj)+8)<-"</s:dataProvider>";
    out.(iiii+(!jj)+9)<-"</s:DropDownList>"
      ) else();

if((int_of_string stack.(i+4+fp))==4) then
(out.(iiii+4) <-"<mx:TextArea height=\"21\" width=\"" ^
 string_of_int ((int_of_string stack.(i+3+fp))*10)     ^"\"/>"
```

```
                ) else(); exec fp sp (pc+1);



|Loshow i-> let ijj=ref 0 in
        let ij=ref (fp+i) in while( int_of_string stack.(int_of_string stack.(!ij)+1)!=0)
        do (
            while((!ijj)< int_of_string stack.(int_of_string stack.(!ij)+1)) do
            (let ii=ref 0 in let iii= while((out.(!ii).[0])!='0') do
            (ii:=!ii+1) done;
              !ii in
                out.(iii)<-"<mx:HBox>";
                 out.(iii+1)<- "<mx:Label text=\"" ^
                 stack. (int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+1+fp)


                ^":\"/>";
              out.(iii+2)<-  "<mx:Label text=" ^
              stack. (int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+2+fp)^"/>";
               out.(iii+3)<- "</mx:HBox>";
            if ( (int_of_string stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+4+fp))==1)
              then (let j=ref 0 in
              (while(!j<(int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+3+fp)))


                do (


                    if((int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+13+fp))==0)
                  then
                    (
                        out.(iii+(!j)+4)<- "<s:RadioButton label=" ^
                        stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp+(!j)) ^
```

```
                    " groupName=\" "^
                     stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+1+fp)^
                     "group\" "
                       ^"/>";
                  )
                else(
                  out.(iii+(!j)+4)<- "<s:RadioButton label=" ^
                   stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp+(!j))^
                    " groupName=\" "^
                     stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+1+fp)^
                     "group\" "  ^" click=\"" ^
                     stack. (int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+1+fp)^
                     (string_of_int ((!j)+1))
                      ^"(event)\"/>";
                        );

                     j:=!j+1;)
                      done) ) else ();

                if((int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+4+fp))==2)
                then
                  (let jj=ref 0 in
                  (while(!jj<(int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+3+fp)))
                      do (out.(iii+(!jj)+4)<- "<s:CheckBox label="
                                ^ stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+5+fp+(!jj))
                      ^ "/>";jj:=!jj+1;)
                      done)
                    ) else();
```

```
                if((int_of_string stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+4+fp))==3)
                 then
                    (let jj=ref 1 in out.(iii+4)<-"<s:DropDownList requireSelection=\"true\">";
                     out.(iii+5)<-"<s:dataProvider>";
                       out.(iii+6)<-"<s:ArrayList source=\"[";
                          out.(iii+7)<- (curve stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp));

                   while(!jj<(int_of_string stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+3+fp)))
                       do (out.(iii+(!jj)+7)<- "," ^
                       (curve stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp+(!jj)));
                       jj:=!jj+1;)
                       done; out.(iii+(!jj)+7)<-"]\"/>";
                       out.(iii+(!jj)+8)<-"</s:dataProvider>";
                        out.(iii+(!jj)+9)<-"</s:DropDownList>"
                       ) else();

               if((int_of_string stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+4+fp))==4)

                  then(out.(iii+4)<-"<mx:TextArea height=\"21\" width=\"" ^
                   string_of_int ((int_of_string stack.(int_of_string stack.( int_of_string stack.((!ij))+2+(!ijj))+3+fp))*10)
                    ^"\"/>"

                 ) else();  ijj:=(!ijj)+1;) done;
                  ijj:=0; ij:=int_of_string stack.(!ij); ) done;

              while(!ijj< int_of_string stack.((!ij)+1)) do(let ii=ref 0 in
```

```
let iiii= while((out.(!ii).[0])!='0') do (ii:=!ii+1) done;
   !ii in
   out.(iiii)<-"<mx:HBox>";
    out.(iiii+1)<- "<mx:Label text=\"" ^
     stack.(fp+int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+1)


   ^":\"/>";
    out.(iiii+2)<-  "<mx:Label  text=" ^
     stack. (fp+int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+2)
       ^"/>";
    out.(iiii+3)<- "</mx:HBox>";
   if ( (int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+4+fp))==1)
     then (let j=ref 0 in
        (while(!j<(int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+3+fp)))
        do (


          if((int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+13+fp))==0)
          then
          (
            out.(iiii+(!j)+4)<- "<s:RadioButton label=" ^
            stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp+(!j))
^ " groupName=\" "^  stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+1+fp)^"group\"
"
          ^"/>";
          )
   else(
        out.(iiii+(!j)+4)<- "<s:RadioButton label=" ^
         stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp+(!j))^
          "  groupName=\"  "^  stack.(int_of_string  stack.(int_of_string
stack.((!ij))+2+(!ijj))+1+fp)^"group\" " ^
```

```
                    " click=\"" ^
                     stack. (int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+1+fp)^

                     (string_of_int ((!j)+1))
                     ^"(event)\"/>";
                    );

                     j:=!j+1;)
                     done)
                ) else();


                if((int_of_string stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+4+fp))==2) then
                (let jj=ref 0 in
                  (while(!jj<(int_of_string stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+3+fp)))
                     do (out.(iiii+(!jj)+4)<- "<s:CheckBox label=" ^ stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp+(!jj))
                     ^ "/>";jj:=!jj+1;)
                     done)
                ) else();


              if((int_of_string stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+4+fp))==3) then

              (let jj=ref 1 in out.(iiii+4)<-"<s:DropDownList requireSelection=\"true\">";
              out.(iiii+5)<-"<s:dataProvider>";out.(iiii+6)<-"<s:ArrayList source=\"[";
              out.(iiii+7)<-(curve stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp));
```

```
            while(!jj<(int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+3+fp)))
            do (out.(iiii+(!jj)+7)<- ","
             ^ (curve stack.(int_of_string stack.(int_of_string stack.((!ij))+2+(!ijj))+5+fp+(!jj)))
               ;jj:=!jj+1;)
               done;
               out.(iiii+(!jj)+7)<-"]\"/>";
               out.(iiii+(!jj)+8)<-"</s:dataProvider>";
                out.(iiii+(!jj)+9)<-"</s:DropDownList>"
               ) else();

            if((int_of_string stack.(int_of_string stack.(int_of_string
stack.((!ij))+2+(!ijj))+4+fp))==4) then
            (out.(iiii+4) <-"<mx:TextArea height=\"21\" width=\"" ^
              string_of_int ((int_of_string stack.(int_of_string stack.( int_of_string
stack.(!ij)+2+(!ijj))+3+fp))*10)
               ^"\"/>"
               ) else();   ijj:=!ijj+1;) done;
                    exec fp sp (pc+1)
in exec 0 0 0;;
```