An introduction to KP Language

Tae Yano
tae.yano@hunter.cuny.edu
ty2142@columbia.edu

Dec/21/2004

CSMS W4115
Programming Language and Translator
Fall 2004

Document Type: Language Release Note -- Version 0.1

1.1 Introduction

The KP Language is developed to help knitters of all level of computer literacy to write knitting patterns, or applications to generate knitting patterns, with ease. Using KP, knitters can create new patterns, combine or adjust existing ones for their liking, their display the graphic representations. Any pattern written in legitimate KP is by itself an abstraction of a "knittable" item. The accompanied KP interpreter takes source codes written in KP and, typically, prints out, "blue prints", and/or row-by-row instructions in commonly accepted knitter's notation. If the given instruction is un-knittable, it was detected by KP interpreter and is rejected. Any knittable KP object can actually be made with (for this version, two[1]) needles and yearn.

[1] Some knitting technique uses more than two needles. Cable knitting, for example, needs at least three needles to do it. As needles are in essence a temporary "memory" to hold unbounded hooks, KP language abstructed them as stacks. Current version only has two of them.

1.2 Feature: What it is and is not

KP is a tool to serve one narrowly defined purpose. It is not a language to write general, all-mighty CAD application or such. Its aim is to do one kind of job and do it well.

The focuses of the language design here are more on simplicity and intuitiveness. Attention is diverted more to build solid, robust founding than add versatility.

The KP Language is also highly OOP conscious and, as such, knit objects are created/manipulated as are aggregations of smaller knits. This, however, is not a goal itself. It is actually the matter of convenience. OOP happen to be a suitable concept to abstract the subject at hand (e.g., knitted items). OOP also offers solutions to some problems in the conventional knitters notations (manipulation of objects, reusability of object etc), which is the more ambitious goal of KP language.

1.3 Background: Knitter's Language

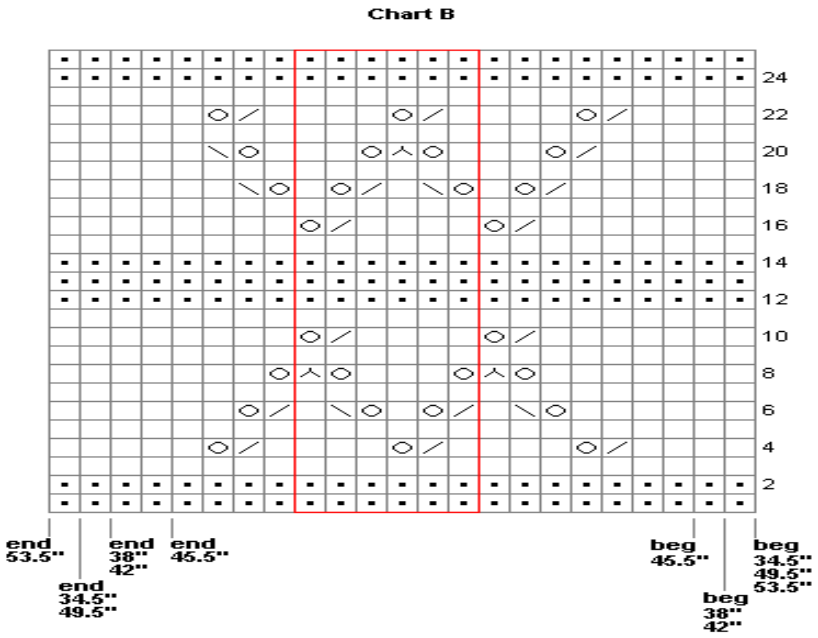
Knitters use the well defined and widely known notation to describe their projects. There are some dialect, and some creative vernaculars seen occasionally, but the core part of the notation is invariable almost everywhere.

This is how the notaion looks like:

```
ssk:  Sl2 sts knitways one at a time then insert left-hand
       needle into fronts of these 2 sts from the left and knit them
       together.
PATTERN:
       1st Row: K1, * k1, inc tgl, yo, k1, yo, ssk, k2;(rep from* to end)
       2nd and every alt Row: p.
       3rd Row: K1, * k2 inc tgl, yo, k3, yo, ssk, k1; (rep from* to end)
       5th Row: K2tgl, * yo, k5, yo, s1, k2 tgl;(rep from* to end)
       7th Row: K1, * yo, ssk, k3, k2, tgl, yo, k1; (rep from* to end)
Rep these 12 rows.
```

This row-by-row type description is the only reference you need for actual knitting, however, very few people can imagine the finished product from this. For this end, pictograms on grits are often accompanied with written instructions, though rarely no one use pictogram alone when knitting.

This is how such a picture looks like (the pattern is different):



1.4 Observation : Strength of knitter's language

KP language borrows heavily from the common knitters' notation. The targeted audience (knitters, knitting application designers) are already familiar with those symbols; therefore they can learn the language with relative ease. Moreover, knitter's notation, despite (or because?) it is spontaneously developed over the course of the long time, is surprisingly reasonable, understandable, and intuitive - that is, agreeable to many. It manages to be efficient yet succinct (though a bit tares, maybe). [2]

Not only it already have established users, the common knitter's notation is linguistically sound/robust. It is capable to describe any items that can be knitted. (though I have not found out if it is NOT capable to describe what can not be knit, but it is a cute notion).

In fact, the first challenge in KP language development is not "how" to define the language, but "how not" to spoil the existing notation and its strength while formalizing it into "language".

[2] Basically, the motivation behind KP language project was "well, those notations already look SO like a computer language." It is already so cryptic that it seemed nonsensical not to use them, in order to make computer understand knitting.

1.5 Another Observation : Problem of knitter's language

The knitter's notation, of course, is not without a problem.

While the notation is very powerful in describing patterns, it does not have a built-in mechanism to generate new ones. (let me explain)

Previously in this document, two styles of knit pattern description were shown --- a row-by-row instruction written in knitter's notation and a pictogram. the later, while not really useful as a knitting instruction, is good for visualizing the item. knitters often use it when they start new projects or modify the existing ones.

Often, knitters wish to scale up/down, interpose, or combine given patterns. They first draw a pictogram, or convert the existing instruction to one and modify it. They then convert the picture back to the row-by-row instruction when it is ready for knitting. This process, unless in the case of the simplest patterns, almost always involves tedious circulations and lengthy trial-and-error. For the most complicated items, the given patterns are almost un-alterable by hand; even small modifications require rewriting the pattern over.

The source of the problem is that, while knit production needs to crunch information sequentially (row-by-row), human's mind does not perceive a knitted item as a stack of strings. It is in their mind conceived as a whole object.

The better half of KP's development goal is to provide the knitters with solutions for this problem - in other word, to upgrade the knitter's notation into an Object Oriented Language. [3]

[3] This goal wasn't quite achieved in the first version of KP - it was rather ambitious, that turned out - but the foundation is laid with this goal in mind, so that future expansions will be easier . Call it for the day.

KP Programming Tutorial

Tae Yano
tae.yano@hunter.cuny.edu
ty2142@columbia.edu

Dec/21/2004

CSMS W4115
Programming Language and Translator
Fall 2004

Document Type: language Tutorial -- version 0.1

2.1 An example (to illustrate the language)

The KP language compiler takes a KP source code as a input and produces a Java executable file as output. When the executable runs on JVM, it creates unique "knit" objects and does actions (e.g., printing, resizing, saving, etc) on them as instructed in the source. This is about all, by design, that the KP language is good for.

The "Knit" object is, in its most basic sense, an abstract data structure representing knitted/knittable items. This data structure is a collection of elements called "Stitch". Stitch, like Knit, is a data type unique to KP language. It is not unlike the Boolean type but has three states instead of two: it is either "k" (for knit stitch), "p" (for pearl stitch), or "B" (empty). All the properties of a given Knit object is defined by the values, or absence, of its stitches and how they are connected to each other.

An easier way to picture knit data structure is to think it as a string of characters from a character set with only three values (k and p and s), with capacity to extend vertically as well as horizontally. Probably the readiest abstract data type is a multi-dimensional, doubly-linked, list such as this one:

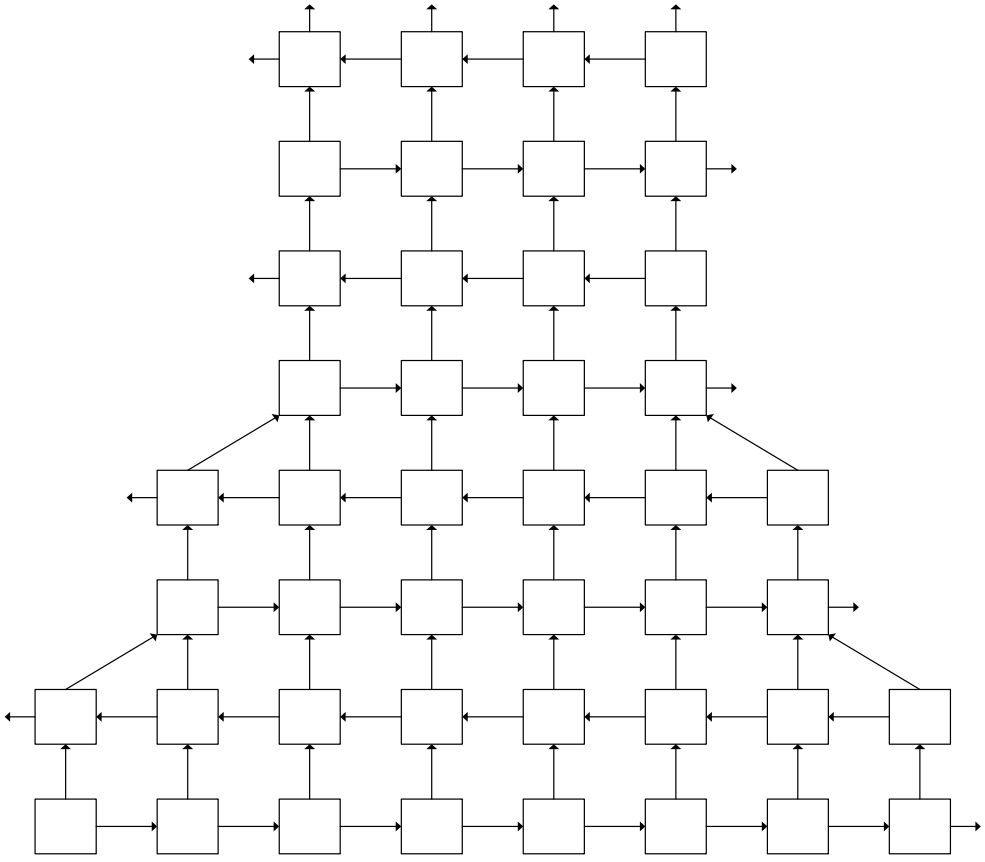


figure 1. knitting represented as a 2d-list

A code to create this instance of the data structure would look something like the code snippet below. Here I used so-do c++ code. It shows pointer operations which, with java, would be hidden by API. This is not the only implementation there is.

```
struct knit_element{
       char sts,
       knit_element *up,
       knit_element *side
}
knit_element
       stitch0[8], stitch1[8],
       stitch2[6], stitch3[6],
       stitch4[4], stitch5[4], stitch6[4], stitch7[4];

knit_element *this_knit = stitch0;

stitch0[0].sts = 'k';            //this is the very first stitch
stitch0[0].side = & stitch0[1];
stitch0[0].up = & stitch1[7];

//"up" of the first stitch is the last stitch of the 2nd row...
//this is how knitting works.

stitch0[1].sts = 'p';
stitch0[1].side = & stitch0[2];
stitch0[1].up = & stitch1[6];

stitch0[2].sts = 'p';
stitch0[2].side = & stitch0[3];
stitch0[2].up = & stitch1[5];

//and so on and on...

stitch0[7].sts = 'p';
stitch0[7].side =NULL;           //turn the row
stitch0[7].up = &stitch1[0];

//and so on and on...

stitch7[2].sts = 'p';
stitch7[2].side = & stitch7[3];
stitch7[2].up = NULL;

stitch7[3].sts = 'p';            //this is the last stitch
stitch7[3].side =NULL;
stitch7[3].up = NULL;            //end of the binding (i.e., last row)
```

The KP language definition of this object would look like this:

```
//note--spacing is for ease of read. no semantic meaning in them.
//k, p, dec are all build-in (or atomic) knit objects.
//only build-in knit object can appear in the rows

knit this_knit = {
       row1:   k, p, k, p, k, p, k, p;
       row2:   k, p, k, p, k, p, k, p;
       row3: dec, p, k, p, k, p, k, dec;
       row4:    p, k, p, k, p, k;
       row5:  dec, k, p, k, p, dec;
       row6:       k, p, k, p;
       row7:       k, p, k, p;
       row8:       k, p, k, p;
}
```

The object, as it is now, does not do much that users can see. A data structure is constructed, and just sits somewhere in the memory.

From the knitter's stand point, one very useful thing would be to print out a graphical representation of this item. There is a well-defined, widely-used common set of notations among knitters (commonly referred as "symbolcraft" or "knitting chart") and KP has a built-in function to do just this:

```
printk( this_Knit );
```

or

```
printk( this_Knit, PIC );
```

will print out a knitting chart like this (those symbols are abbreviated version):

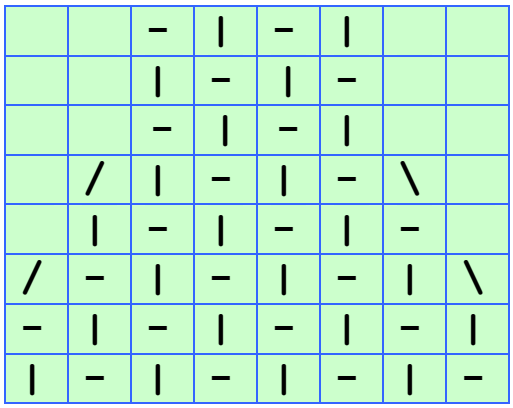


figure 2. knitting chart to knit figure 1

2.2 How to Play with KP

0. make sure your java is 1.4 or up (for swing)
1. untar smlkp.tgz
2. cd to ~/smlkp
3. if you have not java class path set, pls do so.
4. type "java kp < xxxx" (xxxx is the name of kp program you are executing)

the tar file include a sample kp program name "test". If you are executing this one, the command would be:

```
java kp < test
```

The following is the first part of the "test". This includes sufficient information to write very simple KP program. For full detail of the language function, please read the remaining of "test" file.

***

KP language looks kinds of like C's array initialization. but not entirely. Note the followings differences from the C and C-cousins that you are familiar with:

1. There exists only single line comment. (like this comment)
2. Function's arguments are NOT comma separated
3. " * x y" does not mean "x times y" but "x more of y".
4. same goes to "**"

Example 1:

The very simple knit pattern definition:

```
knit ada = {
      row 1: k, k, k, k;
      row 2: k, k, k, k;
}

knit bill = {
      row 1: p, p, p, p;
      row 2: p, p, p, p;
}
```

A few more rules to follow here:

1. When defining knit pattern with primitive stitches, start each row with "row" keyword
2. "row" keywords are followed by the row number, and ":".
3. Each stitches are terminated by "," and each row is by ";"
4. Row number has to appear in order. you can't skip to row 3 from 1.
5. Only primitives can appear in rows.

Primitives defined in KP are: "k","p","tgl","yo","inc", and "dec". Primitives represent basic Knitting stitches:

k	knit stitch
p	pearl stitch
yo	yearn over
tgl	toggle
inc	add one more stitch to the row
dec	decrease one stitch from the row.

All those symbols, as well as the way they are put together, should be familiar to those who knits. Indeed, all those are borrowed from commonly used by knitters notations.

"printk" function prints out knit patter.

```
printk(ada TEXT);
printk(bill TEXT);
```

the examples here requesting its TEXT mode printout. To print out instructions, use "INST" instead of "TEXT". Similarly, use "PIC" to print out pictogram.

KP Language Reference Manual

Tae Yano
tae.yano@hunter.cuny.edu
ty2142@columbia.edu

Dec/21/2004

CSMS W4115
Programming Language and Translator
Fall 2004

Document Type: language Reference Manual -- version 0.1

3.1 KP language Lexicon

3.1.1 Character Set

The KP source code is a sequence of characters from a character set defined in Basic Latin block of ISO/IEC 10646. Any character from this set can appear in the source, though not all of them are meaningful in the language.

3.1.2 White Space and Comments

(Blank) spaces, vertical and horizontal tabs, new lines, and carriage returns are known, collectively, as white space. All white space will be ignored except insofar as they are used to separate adjacent "tokens" or when they appear in character or string constants.

Only C++'s single line style comments are supported: "//" start a single-line comment. Texts appeared as comments will be treated as white space and ignored similarly. Comments are not nested.

3.1.3 Tokens

The sequence of characters making up a KP source code are grouped into atomics elements, called *tokens*, according to the rules explained in this document. There are five classes of tokens: *operators*, *separators*, *identifiers*, *keywords*, and *constants*. Tokens are separated from adjacent ones with one or more of above mentioned white space.

3.1.4 Operators and Separators

The following character sequences when appears in a KP source code are considered as operator / punctuation tokens:

```
= ** *
( ) [ ] { } , ; :
```

3.1.5 Identifiers

An identifier is a sequence of upper and lower case letters, digits and under score ("_") character. An identifier must start with non-digit and not spell same as a keyword. All user defined identifiers are created as "knit" type.

3.1.6 Keywords

The character sequences listed below are keywords n KP language:

```
knit row
k p dec inc tg yo
printk PIC INST TEXT
```

3.1.7 Constants

The only constant class tokens appear in KP is the integer.

3.2 KP language Syntax

3.2.1 Statement

A KP program is a collection of "statements". A statement is an executable block of code which, if appears independently in the body of the program, executed sequentially at the invocation in the order of appearance. There are three kinds of statement: *Knit definition*, *build-in function call*, and *assignment*. Those statements are explained farther below.

3.2.2 Data Type

There are not many data types in KP that users manipurate directly. There are "**`int`**" for integer, and "**`knit`**" data types, which is unique to the KP language.

As mentioned, **`knit`** is an abstract data structure of knitting/knittable items. Internally it is a collective of objects of type "`eye`". An eye consist of a "`stitch`" object, and pointers (or references) to two other eyes. Stitch is a trinaly data type whose state is either '`k`', '`p`', or '`s`'.

The KP language has a set of pre-defined short-hands (**`k`**, **`p`**, **`dec`**, **`inc`**, **`tg`**, **`yo`**) for programmers to use to make up a rows of knit. [4]

3.2.3 Declaration and Scope

To declare is to give an entity a name - *identifier*- by which it can be referred throughout a given domain. In the KP language a block consists of one domain, or *Scope*. Identifiers are visible only within the block they are declared. Named block (such as knit definitions, below), variables or function declaration outside of any block are therefore globally visible.

All *Knit definitions* are named block that declaring and initializing an knit object. A Knit object can also be declared by *assignment*. Identifiers can not be used before it declared and must be unique throughout its scope.

3.2.4 Knit Definition

Knit definition is a named block which defines the properties of a knit object. There have to be at least one `entry` in a given definition (i.e., empty definition is illegal). There are more than one way to define knit object within KP language. The list below is the specification for construction:

knit_definition :
'**`knit`**' id '=' '{' *unit_definition_list*
unit_definition_list:
(*row_definition)*+ | (*knit_reference*)+
row_definition:

[4] Originally, the language envisioned the tools for programmers to define their own eyes, connect them together and make up a knit item "from scratch", but the idea was aborted. Although quite a few knitting projects are possible with only this set, I still think it should be there for future version

'**row**' digit ':' ('**k**'|'**p**'|'**s**'|'**yo**'|'**tg**'|'**dec**'|'**inc**'|id) (',' '**k**'|'**p**'|'**s**'|'**yo**'|'**tg**'|'**dec**'|'**inc**'| id)* ';'

knit_reference:
('**row**' digit ':')? id (',' id)* ';'

For example, a simplest knit definition looks like this:

```
knit garter = { row 1: k; }
```

or:

```
knit skitt = {
      row 1: k, p, k;
      row 2: k, p, k;     }
knit r-skitt = {
      row 1: p, k, p;
      row 2: p, k, p;     }
knit combine = {
      r-skitt, skitt;
      skitt, r-skitt;     }
```

3.2.5 Expression and Assignment

There are two kinds of quis-arithmetic you can perform on Knit objects, "*" and "**".

"*" is used as:

```
myknit =  * ope1 ope2 ;
```

This means "add *ope2* more of the object to *ope1, horizontally*". *ope1* must be other knit object and *ope2* an integer literal.

"**" is used as in the same way and means "add *ope2* more of the object to *ope1, vertically*"

For this version of KP, it does not allow "unknitting" of the object. once the object is declared and attached to an identifier, it is not be overwritten by the assignment operation.

3.2.6 Function Call

The language does not offer means for programmers to define there own functions therefore all the *function call* appears in a source code is call to a built-in function which is explained in the next section.

For this version of KP, one function, three internal functions are inplemented, all called as printk():

```
int printk( knit item, format_id );

(Example: printk( this_knit PIC );)
```

This function prints a knit object on the console (or, STDOUT the running program is associated to). printk function takes one, or two arguments. The first of the argument is the name (identifier) of the knit object to be displayed. The Optional "format_id" is either PIC, TEXT or INST. This decides if the knit is printed as a knitting chart image (PIC or Text) or row-by-row instruction (INST).

KP Language Development

Tae Yano
tae.yano@hunter.cuny.edu
ty2142@columbia.edu

Dec/21/2004

CSMS W4115
Programming Language and Translator
Fall 2004

Document Type: software development history -- version 0.1

4.1 Project Plan: Overview

The KP language development was conducted in, roughly, "waterfall" software development model. The project was divided into more or less autonomous sub-modules and, at the first stage, quick-and-dirty prototypes of each which, though very crudely and sometime not quite honestly, did most of what it supposed to do. Then on, modules were coded and tested independently for a duration. Periodically, some modules were put together and examined as a suite, then the inputs from the experiments were fed back into each. Modules were merged into one unit after such sufficient amount of such tests. Thereafter they were, while development was concerned, treated as one unit.

I had four modules at the end: Lexer/Parser, Tree Walker (semantic analysis and code generation), Abstract data types for Knitting, and graphics with Swing. Lexer/Parser and Tree Walker, naturally, were merged first, then, knitting and graphics (see "Architectural Design"). The two compounds were put together at the final stage of the development. The integration test was applied on this suite (see "Testing Plan").

A few circumstances compelled me to settle with this model: first, at the beginning, I couldn't estimate the project well. There were too many unknown elements for designing well defined, unambiguous interfaces for each module. Meanwhile, I was anxieties to do some coding in order to get into shape. Second, being one person project, the price for flexible, fluid, yet unorganized development model. Protocols benefits when two parties interact, and my project was not a case. Also, I wanted to put as much as possible into the project, yet did not want find me in a middle of untied ends. It worked OK, more or less (except for the documentation, I suppose).

4.2 Project Timeline and Log

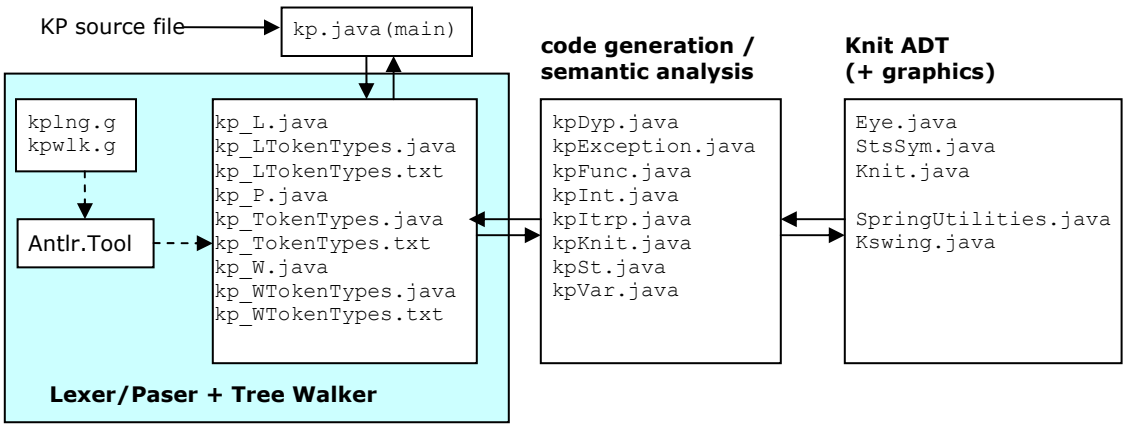
The tasks required for the deliverable were divided into programming (marked "p") and designing/documentation ("d"). Programming tasks are further divided into aforementioned four.

Milestone	Date	Status
White Paper, first release (d)	Sep/24/2004	delivered
LRM, first release (d)	Oct/21/2004	delivered
Knit ADT:1st (qdp)version (p)	Oct/21/2004	
Lexer/Parser:1st (qdp)version (p)	Nov/01/2004	
Graphic :1st (qdp)version (p)	Nov/15/2004	
Tree Walker: 1st (qdp)version (p)	Dec/12/2004	
Merger :Knit ADT and Graphic (p)	Dec/14/2004	
Merger :L/P and TW	Dec/14/2004	
Merger : all (p)	Dec/16/2004	
Final Integration Test (p)	Dec/17/2004	
Source code freeze (p)	Dec/20/2004	
Final Report release (d)	Dec/21/2004	

5.1 Architectural Design

The KP language interpreter is a java bytecode executable ("kp.class"). It is compiled with Java version "1.4.2_05" and run on the java's runtime environment of the same version (build 1.4.2_05-b04).

The distribution tar file ("smlkp.frz.tgz") includes all the codes to rebuild kp interpreter:



(Files prefixed with "KP_" are generated by Antlr Tool, from "*.g" files.)

figure 1. Organization of the Source Tree

`kp.java`, which contains `"main()"`, reads in data from kp source codes, and passes it to the tokenizer. The returned data was then fed to the parser. Then, It proceeds to create AST of the given token stream.

Upon successful return from the parser. main thread invokes the tree walker on the AST. As it recursively "walks" the AST, it matches the sub trees with the pre-defined patterns and, as it happens, executes Java code associated to the patter.

All the codes invoked from the tree walker is generated by the Java source appears in the above diagram, under "code generation / semantic analysis", and in specific, the interpreter class from, `kpItrp.java` . Actual manipulation of the object of interest, knit object, is done as calls to the functions from knit-wrapping class, `kpKnit.java`. From there, the calls to the "Knit ADT" and its graphical component are dispatched.

7.1 Test Plan

After all the modules were integrated, I wrote a set of KP programming to run through after change/modification. If the interpreter works correctly, the result of the each run should be the same.

I was able to run those "complete" KP programming only after every modules were put together, but each modules were small enough and had relatively few points of entry. It was not difficult to declare them kosher.

The following is the contents of this test suite. This was written in KP language from head to toe so that you can feed it directly to the interpreter. The same document ("test") is found in the distribution tar file.

```
// README HERE FIRST
//
// KP language looks kinds of like C's array initialization.
// but not so entirely.
//
// BEFORE EVEN START
// note the followings:
//
// 1. There exists only single line comment. (like this comment)
// 2. Function's arguments are NOT comma separated
// 3. " * x y" does not mean "x times y" but "x more of y".
// 4. same goes to "**"
//
// now, examples of KP programs.
//
// to feed them to the interpreter, type:
//
// "java kp < test"
//
// on command line.
//
//
// A bit of excuse for not having multiple comment
//
// I think multiple comment, without notion of nesting, is prone to errors.
// single line commenting are bothersome sometime.
// (but at least I would know where it end).
// when I need multiple comment, I use vi's substitution command.
// it works ok with me.


// Example 1.
// The very simple knit pattern definition:

knit ada = {
       row 1: k, k, k, k;
       row 2: k, k, k, k;
}
knit bill = {
       row 1: p, p, p, p;
       row 2: p, p, p, p;
}

// A few rules to follow here:
// 1.When defining knit pattern with primitive stitches, start each row with
```

```
"row" keyword, follwed by the row number, and ":".
// 2. Each stitches are terminated by "," and each row is by ";"
// 3. row number have to appear in order. you can't skip to row 3 from 1.
// 4. only primitives can appear in rows.
// 5. primitives are: "k","p","tgl","yo","inc",and "dec".
// primitives reoresent basic Knitting stitches:
// k - knit stitch, p - paerl stitch, yo - yearn over, tgl - toggle:
// inc - add one more stitch to the row
// dec - decrease one stitch from the row.
// (those are all notation commonly used by knitters).
// 6. only k or P can appear in the "casting" row (the first row)



// "printk" function prints out knit patter.

printk(ada TEXT);
printk(bill TEXT);


// (here its TEXT mode. to print out instruction, use "INST" instead of "TEXT")
// (to print out pictogram, use "PIC" instead of "TEXT")




// Example 2.
// using other primitives beside k or p:
// increasing and decreasing the row size.
// (spacing if only for ease of read)

knit charls = {
       row 1:     k, k, k, k;
       row 2:   inc, k, k, inc;
       row 3:inc, k, k, k, k, inc;
       row 4: k, k, k, k, k, k, k, k;
       row 5:dec,k, k, k, k, k, k,dec;
       row 6:  dec, k, k, k, k, dec;
       row 7:       k, k, k, k;
}

printk(charls TEXT);

// use "inc" stitch  when increasing the width of knitting
// similary, use "dec" stitch to decrease.
// knitting more/less than the privious one without inc/dec marker is illigal.


// Example 3.
// repeating previous rows:
// patterns can be a reference to other rows.
//(this, by the way, called "moss" pattern)

knit dennis = {
       row 1:   p, k, p, k, p, k;
       row 2:   k, p, k, p, k, p;
       row 3:   row 1;
       row 4:   row 2;
       row 5:   row 3;
       row 6:   row 4;
       row 7:   row 5;
}
printk(dennis TEXT);
```

```
//"dennis" result in the same pattern as "eric" below:

knit eric = {
       row 1:   p, k, p, k, p, k;
       row 2:   k, p, k, p, k, p;
       row 3:   p, k, p, k, p, k;
       row 4:   k, p, k, p, k, p;
       row 5:   p, k, p, k, p, k;
       row 6:   k, p, k, p, k, p;
       row 7:   p, k, p, k, p, k;
}

printk(eric TEXT);



// Example 4.
// combining the knit patterns to create new one:
//
//pattern can be expressed in terms of other knit.
//

knit foo = {
       row 1: k,k,k,k;
       row 2: p,yo,p,yo;
       row 3: k,k,k,k;
}

boo = ** foo 2;
printk(boo TEXT);

knit gibson = { foo; }
printk(gibson TEXT);

knit hal = { gibson, gibson; }
printk(hal TEXT);

knit ivan = {  hal; hal; }
printk(ivan TEXT);

// gibson and hal is joinig knits horizontally
// ivan is joining hal vertically



// Example 5.
// combining the knit patterns to create new one -- in other way
//

john = gibson;
printk(john TEXT);

ken = * gibson 1;
printk(ken TEXT);

larry = ** hal 1;
printk(larry TEXT);

//john print same pattern as gibson
// ken print the same pattern as hal
//larry print the same pattern as ivan
```

```
// Example 6.
// caution on joining knits
//
// to join vertically, knits' joining edge's length have to be matched.
// meaning, only the knits with same row counts can be joined horizontally

knit cloud = {
       row 1:     k, k, k, k;
       row 2:   inc, k, k, inc;
       row 3:inc, k, k, k, k, inc;
       row 4: k, k, k, k, k, k, k, k;
       row 5:dec,k, k, k, k, k, k,dec;
       row 6:  dec, k, k, k, k, dec;
       row 7:       k, k, k, k;
}

//can be joined with

knit evi = {
       row 1:   p, k, p, k, p, k;
       row 2:   k, p, k, p, k, p;
       row 3:   p, k, p, k, p, k;
       row 4:   k, p, k, p, k, p;
       row 5:   p, k, p, k, p, k;
       row 6:   k, p, k, p, k, p;
       row 7:   p, k, p, k, p, k;
}

//horizontally, but not vertically, while

knit al = {
       row 1: k, k, k, k;
       row 2: k, k, k, k;
}

// or

knit bjorn = {
       row 1: p, p, p, p;
       row 2: p, p, p, p;
}

//can be joined them no problem.

knit matt = {
       al;
       cloud;
       bjorn;
       }
printk( matt TEXT );

knit norm = {
       evi,cloud, evi;
       }
printk( norm TEXT );
```

7.1 Lessen Learned

As a CVN participant, I did this project by myself and was, in general, grad of this arrangement.

Software development team of more than three persons would experience diminishing rate of return to its labor: one-person project is better off in comparison. Adding it the "remote access only" communication barrier, it would be a nightmare. I was spared with considerable overhead in organization and logistics - book keepings, basically - for being a sole member of my little project.

That said, there were times when I wished for other heads, ideally two more at a time, to bounce off and scrutinize ideas. Having only one mind to look at problems, I often found errors in planning well into implementation stage... then had to go back to redo the beginning. I know "discussion" over designs or specifications sometimes would look messy, useless and inefficient. But it is much better to take time at the designing stage than find out later that somehow, something or others, were not thought through. Also, this is my first real Java programming and I sometimes spent ridiculous amount of time in figuring out the most basic thing, (why can't they declare array like anyone else?) with which more experienced Java programmers would spent half a second. It would have not been bad if the problem is something I have some idea about, but PTL was not one of them.

Also, toward the end of the projects, I seemed to have too much of a "paper works" left to deal. There were a lot of rather time consuming and tedious works at the end. In estimating work amount, I consulted with project reports from previous years but all of them are 3 - 4 person project and had to calibrate the figures to one person project. The fact is, some kind of work is done three times faster if there are three times more people. Documenting or testing is one of them.

The project is relatively small in terms of coding - less than 3000 in SLOK - so I did not use version control measures such as CVS or SourceSafe, though I would normally do so for the bigger, coding intensive ones. I would definitely use it if there are person altering codes besides me, for even very small projects.

I tried scripting but, I have to say, it sometimes does not worth it. It is fun writing scripts, and I used bits and pieces of the unquoted from my past works. But if you are spending 2 hrs debugging the scripts that saves 10 words of typing, maybe it's your being carried away.

Writing Laxer and Parser was straight forward enough, but AST parser was little confusing and I wish the lecture spent more time on the subject. In specific, I was not too clear how semantic checker and code generation should be meshed in AST parser implementation. There seems to be more than one way to do and I think I was not quite equipped to make informed decision. In the end I did in the way I did, but still am not sure that was the "right" way.

Appendix A. KP Source Codes Listing

(Those files are included in the distribution tar file, "smlkp.frz.tgz", under "/src")

```
total 164
//*.g files, and java source generated from them:
drwxr-xr-x    5 sysadm   games        4096 Dec 22 14:49 ..
-rwxr-xr-x    1 root     root         2326 Dec 22 14:49 kpwlk.g
-rwxr-xr-x    1 root     root         2764 Dec 22 14:49 kplng.g
-rw-r--r--    1 root     root          971 Dec 22 14:49 kp_WTokenTypes.java
-rw-r--r--    1 root     root        10327 Dec 22 14:49 kp_W.java
-rw-r--r--    1 root     root          970 Dec 22 14:49 kp_TokenTypes.java
-rw-r--r--    1 root     root        17642 Dec 22 14:49 kp_P.java
-rw-r--r--    1 root     root         1126 Dec 22 14:49 kp_LTokenTypes.java
-rw-r--r--    1 root     root        24169 Dec 22 14:49 kp_L.java

//Knit ADT and its graphic components:
-rwxr-xr-x    1 root     root         1269 Dec 22 14:54 kp.java
-rwxr-xr-x    1 root     root         8109 Dec 22 14:55 SpringUtilities.java
-rwxr-xr-x    1 root     root         3162 Dec 22 14:55 Kswing.java
-rwxr-xr-x    1 root     root          768 Dec 22 14:56 StsSym.java
-rwxr-xr-x    1 root     root        15386 Dec 22 14:56 Knit.java
-rwxr-xr-x    1 root     root         1363 Dec 22 14:56 Eye.java

//code for generation and semantic analysis
-rwxr-xr-x    1 root     root          490 Dec 22 14:57 kpVar.java
-rwxr-xr-x    1 root     root         2339 Dec 22 14:57 kpSt.java
-rwxr-xr-x    1 root     root         5017 Dec 22 14:57 kpKnit.java
-rwxr-xr-x    1 root     root         8019 Dec 22 14:57 kpItrp.java
-rwxr-xr-x    1 root     root          697 Dec 22 14:57 kpInt.java
-rwxr-xr-x    1 root     root          577 Dec 22 14:57 kpFunc.java
-rwxr-xr-x    1 root     root          168 Dec 22 14:57 kpException.java
-rwxr-xr-x    1 root     root         3100 Dec 22 14:57 kpDyp.java
drwxr-xr-x    2 root     root         4096 Dec 22 14:57 .
```

//main
rwxr-xr-x   1 root   root   1269 Dec 22 14:54 kp.java

```
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;


public class kp {

static boolean dbgtree = false;
//static boolean dbgtree = true;

public static void main(String args[]) {
       try {
       DataInputStream input = new DataInputStream(System.in);

       kp_L lexer = new kp_L(input);
       kp_P parser = new kp_P(lexer);

       parser.program();
       CommonAST parseTree = (CommonAST)parser.getAST();

       if( dbgtree){
       System.out.println(parseTree.toStringList());
       ASTFrame frame = new ASTFrame("AST from the Simp parser", parseTree);
       frame.setVisible(true);
       }
       else{
       kp_W walker = new kp_W();
       kpDyp r = walker.knit_expr( parseTree );
       if ( null != r )
             r.print();
       }

       //} catch( IOException e ) {
          //   System.err.println( "Error: I/O: " + e );
         } catch( RecognitionException e ) {
             System.err.println( "Error: Recognition: " + e );
         } catch( TokenStreamException e ) {
             System.err.println( "Error: Token stream: " + e );
         } catch( Exception e ) {
             System.err.println( "Error: " + e );
         }

}//endofmain

}//endofmainclass
```

```
//*.g files, and java source generated from them:
drwxr-xr-x    5 sysadm   games        4096 Dec 22 14:49 ..
-rwxr-xr-x    1 root     root         2326 Dec 22 14:49 kpwlk.g
-rwxr-xr-x    1 root     root         2764 Dec 22 14:49 kplng.g



//kplng.g

{
import java.io.*;
import java.util.*;
}

class kp_L extends Lexer;

options{
    k = 2;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
    exportVocab = kp_;
}

{
    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}

protected ALPHA   : 'a'..'z' | 'A'..'Z' | '_' ;
protected DIGIT   :   '0'..'9';

WS      : (' ' | '\t')
       { $setType(Token.SKIP); }
        ;
NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
            { $setType(Token.SKIP); newline(); }
        ;
COMMENT : (  "//" (~( '\n' | '\r' ) )* (NL) )
       { $setType(Token.SKIP); }
        ;

LPAREN  : '(';
RPAREN  : ')';
MULT    : '*';
MULTROW : "**";
PLUS    : '+';
MINUS   : '-';
DIV     : '/';
MOD     : '%';
SEMI    : ';';
LBRACE  : '{';
RBRACE  : '}';
LBRK    : '[';
RBRK    : ']';
ASGN    : '=';
COMMA   : ',';
```

```
GE      : ">=";
LE      : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
NEQ     : "!=";
COLON   : ':';
DOT    : '.';

ID  options { testLiterals = true; }
        : ALPHA (ALPHA|DIGIT)*
        ;
NUMBER  : (DIGIT)+
       ;
STRING  : '"'!  (  ~('"' | '\n') | ('"'!'"') )* '"'!
        ;



class kp_P extends Parser;

options{
    k = 2;
    buildAST = true;
    exportVocab = kp_;
}

tokens {
   PROG;
   STATEMENT;
   KNIT;
   KNIT_REF;
   KREF_LIST;
   ROW_DEF;
   RDEF_LIST;
   VAR_LIST;
   FUNC_CALL;
}

{
    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}

program: ( statement )* EOF!
           {#program = #([STATEMENT,"PROG"], program); }
        ;
statement:
       knit_definition
       | func_call
       | assignment
       ;

assignment:
       ID ASGN^ knit_mult SEMI!
       ;
```

```
knit_mult:
       ( MULT^ | MULTROW^ ) ID NUMBER | ID
       ;

//function argument are NOT COMMA SEPARATED!!!
func_call:
       ID LPAREN! (knit_expr)* RPAREN!SEMI!
        {#func_call = #([FUNC_CALL,"FUNC_CALL"], func_call); }
       ;

knit_definition:
       "knit"! ID ASGN! LBRACE!
       ( kref_list | rdef_list )
       RBRACE!
       {#knit_definition = #([KNIT,"KNIT"],knit_definition ); }
       ;
kref_list:
       (knit_ref)+
       {#kref_list = #([KREF_LIST,"KREF_LIST"], kref_list);}
       ;

knit_ref:
       knit_expr (COMMA! knit_expr)* SEMI!
       {#knit_ref = #([KNIT_REF,"KNIT_REF"], knit_ref); }
       ;

rdef_list:
       (row_def)+
       {#rdef_list = #([RDEF_LIST, "RDEF_LIST"], rdef_list);}
       ;
row_def:
       "row"! NUMBER COLON! ( ("row"! NUMBER SEMI!)| knit_ref )
       {#row_def = #([ROW_DEF, "ROW_DEF"], row_def);}
       ;
knit_expr:
       basek | ID
       ;
basek:
       ( "k" | "p" | "tg" | "yo" | "inc" | "dec" )
       ;
```

```
//kpwlk.g



{
import java.io.*;
import java.util.*;
}

class kp_W extends TreeParser;

options{
       importVocab = kp_;
}

{
    static kpDyp null_data = new kpDyp( "<NULL>" );
    kpItrp itr = new kpItrp();
}

knit_expr  returns [ kpDyp r ]
{
    r = null_data;
    kpDyp a, b, tmp;
    Vector v;
    kpDyp[] ar;
    String s = null;
    String[] sx;
    kpInt dbg;
    kpKnit x;
    kpKnit y;
}

       : #(KNIT a=knit_atm b=knit_expr)
           { s = a.name; //get the name of the idlabel
           ((kpKnit)b).setName(s);
           r = itr.setVariable(s, b);
           }

       | #(KREF_LIST
           {a = null;}
           ( ar=knit_exprl { a=itr.joinKnit(a,ar);} )+ )
           { //x.endKnit();
           r = a;
           }

       | #(RDEF_LIST {x = new kpKnit();}
           ( ar=knit_exprl {a=itr.joinRow(x, ar);} )+ )
           { x.endKnit();
           r = x;
           }

       | #(STATEMENT (a=knit_expr {r=a;})+ )
              { //System.out.println( "found prog:well, statement");
              //r = null_data;
              }
       | #(FUNC_CALL {v= new Vector();} a=knit_atm
              (b=knit_atm {v.add(b);})* )
              { ar = itr.convertExprList(v);
              r = itr.funcCall(a, ar);
              }
       | #(ASGN a=knit_atm ( b=knit_expr | b=knit_atm ) )
```

```
            { s = a.name;
            tmp = ((kpKnit)b).deepCopy();
            ((kpKnit)tmp).setName(s);
            r = itr.setVariable(s, tmp);
            }
        //{ r = itr.assign(a,b); }
        | #(MULT a=knit_atm b=knit_atm )
               { r = itr.mult(a, b); }
        | #(MULTROW a=knit_atm b=knit_atm )
               { r = itr.multrow(a, b); }
        //| knit_atm
        ;

knit_exprl  returns [ kpDyp[] rv]
{
    Vector v;
    rv = null;
    String s = null;
    kpDyp a, b;
    kpDyp[] tmp;
    kpInt dbg;
}
       : #(KNIT_REF {v= new Vector();} (a=knit_atm {v.add(a);})+ )
            { rv = itr.convertExprList(v);
            }

       | #(ROW_DEF a=knit_atm
              ( (b=knit_atm {rv = itr.convertExprListRow(a,b);} )
              | (tmp=knit_exprl {rv = itr.convertExprListRow(a,tmp); })
              )
            )
       ;

knit_atm  returns [ kpDyp r ]
{
    kpInt dbg;
    r = null_data;
    String s = null;
}
       :id:ID { r = itr.getVariable( id.getText() ); }
         | num:NUMBER { r = itr.getNumber( num.getText() ); }
         | k:"k" { r = itr.getVariable( k.getText() ); }
       | p:"p" { r = itr.getVariable( p.getText() ); }
       | tg:"tg" { r = itr.getVariable( tg.getText() ); }
       | yo:"yo" { r = itr.getVariable( yo.getText() ); }
       | inc:"inc" { r = itr.getVariable( inc.getText() ); }
       | dec:"dec" { r = itr.getVariable( dec.getText() ); }
       ;
```

```
//code for generation and semantic analysis
-rwxr-xr-x    1 root     root          490 Dec 22 14:57 kpVar.java
-rwxr-xr-x    1 root     root         2339 Dec 22 14:57 kpSt.java
-rwxr-xr-x    1 root     root         5017 Dec 22 14:57 kpKnit.java
-rwxr-xr-x    1 root     root         8019 Dec 22 14:57 kpItrp.java
-rwxr-xr-x    1 root     root          697 Dec 22 14:57 kpInt.java
-rwxr-xr-x    1 root     root          577 Dec 22 14:57 kpFunc.java
-rwxr-xr-x    1 root     root          168 Dec 22 14:57 kpException.java
-rwxr-xr-x    1 root     root         3100 Dec 22 14:57 kpDyp.java
drwxr-xr-x    2 root     root         4096 Dec 22 14:57 .
```

```
//kpItrp.java
//
//
//
import java.util.*;
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;


class kpItrp {

    kpSt symt;

    final static int PIC = 1; //used by internal functions
    final static int TEXT = 2;
    final static int INST = 3;
    final static int RIGHT = 4;
    final static int LEFT = 5;
    final static int TOP = 6;
    final static int BTM = 7;


    public kpItrp() {
        symt = new kpSt( null );
        registerInternal();
    }


    public void registerInternal(){

       //remark:below is not used for now. future use.
        //B = 3; L_DEC = 7; C_DEC = 8; R_D_DEC = 9;
        //L_D_DEC = 10; L_INC = 12;

       symt.put( "k", new kpInt(StsSym.K) );
       symt.put( "p", new kpInt(StsSym.P) );
       symt.put( "yo", new kpInt(StsSym.YO) );
       symt.put( "tg", new kpInt(StsSym.TGL) );
       symt.put( "inc", new kpInt(StsSym.R_INC) ); //11
       symt.put( "dec", new kpInt(StsSym.R_DEC) ); //6
       symt.put( "l_inc", new kpInt(StsSym.L_INC) );
       symt.put( "l_dec", new kpInt(StsSym.L_DEC) );
       symt.put( "r_inc", new kpInt(StsSym.R_INC) );
       symt.put( "r_dec", new kpInt(StsSym.R_DEC) );
       symt.put( "row", new kpInt(StsSym.ROWREF) );
       symt.put("printk",new kpFunc("printk",
```

```
            (new String[] {"knit","int"}),2));
      symt.put("trim",new kpFunc("trimk",
            (new String[] {"knit","int"}),2));
      symt.put("spil",new kpFunc("growk",
            (new String[] {"knit","int"}),2));
      symt.put( "PIC", new kpInt(kpItrp.PIC) );
      symt.put( "INST", new kpInt(kpItrp.INST) );
      symt.put( "TEXT", new kpInt(kpItrp.TEXT) );
      symt.put( "RIGHT", new kpInt(kpItrp.RIGHT) );
      symt.put( "LEFT", new kpInt(kpItrp.LEFT) );
      symt.put( "TOP", new kpInt(kpItrp.TOP) );
      symt.put( "BTM", new kpInt(kpItrp.BTM) );

   }


   public kpDyp assign( kpDyp a, kpDyp b ){
      kpDyp x;
      kpKnit z;
      System.out.println( "assign");
      if( !(b instanceof kpKnit) )
            return a.error("inconsistant data");
      if( a.name == null )
            return a.error("inconsistant data");
      System.out.println( "assign 2");

       x = ((kpKnit)b).deepCopy();
       ((kpKnit)x).setName( a.name );
       symt.setValue( x.name, x, true, 0 );
       return x;
   }


   public kpDyp mult( kpDyp a, kpDyp b ){
      kpKnit x, y, z;
      kpInt n;
      if( !( a instanceof kpKnit) )
            return a.error("inconsistnat data type");
      if( !( b instanceof kpInt) )
            return a.error("inconsistnat data type");
      n = (kpInt)b;
      y = (kpKnit)a;
      x = y.joinHorizontal(y);
      for(int i=1; i< n.var; i++){
            z = x.joinHorizontal(y);
            x = z;
      }
      return x;
   }

   public kpDyp multrow( kpDyp a, kpDyp b ){
      kpDyp x;
      kpKnit  y, z;
      kpInt n;
      if( !( a instanceof kpKnit) )
            return a.error("inconsistnat data type");
      if( !( b instanceof kpInt) )
            return a.error("inconsistnat data type");
      n = (kpInt)b;
x = ((kpKnit)a).joinVertical((kpKnit)a);
      for(int i=1; i< n.var; i++){
            z = ((kpKnit)x).joinVertical((kpKnit)a);
            x = z;
```

```
   }
   return x;
}


public kpDyp getVariable( String s ){
    kpDyp x = symt.getValue( s, true, 0 );
   //if getValue do not find it, it is null, i think.
    if ( x == null ){
       //System.out.println( "      getVariable -- novalue found");
        return new kpVar( s );
   }
   else
        return x;
}


public static kpDyp getNumber( String s ) {
    return new kpInt( Integer.parseInt( s ) );
}


//include semantic check
public kpDyp setVariable( String s, kpDyp b ) {
    kpDyp x = symt.getValue( s, true, 0 ); //makesure its new name
    if( x == null ){
       if( b instanceof kpKnit )
          x = ((kpKnit)b).copy();
        x.setName(s);
        symt.setValue( s, x, true, 0 );  // bother scope!
   }
   else if( x instanceof kpVar ){
       //need this bc st does not necessarily return collect one
       if( b instanceof kpKnit )
          x = ((kpKnit)b).copy();
        x.setName(s);
        symt.setValue( s, x, true, 0 );  // bother scope!
   }
   else{
       //System.out.println( "    previously defined : x.name "+ x.name);
          return x.error("this variable is defined before ");
   }
   return x;
}



//note: leave the semantic checks to later func
public kpDyp[] convertExprList( Vector v ) {
    kpDyp[] x = new kpDyp[v.size()];
    for ( int i=0; i<x.length; i++ )
        x[i] = (kpDyp) v.elementAt( i );
    return x;
}


//note: leave the semantic check to later func
public kpDyp[] convertExprListRow( kpDyp a, kpDyp[] tmp ) {
    kpDyp[] x = new kpDyp[tmp.length +1];
   x[0] = a;
    for ( int i=1; i<x.length; i++ ){
        x[i] = tmp[i-1];
   }
```

```
    return x;
}

//note: leave the semantic check to later func
public kpDyp[] convertExprListRow( kpDyp a, kpDyp b ) {
    kpDyp[] x = new kpDyp[3];
   x[0] = a;
    x[1] = new kpInt(StsSym.ROWREF);
    x[2] = b;
    return x;
}



public kpDyp joinKnit(kpDyp x, kpDyp[] klist){

   kpKnit y, z;
   kpInt n;
   for( int i=0; i< klist.length; i++){
       if( !(klist[i] instanceof kpKnit) ){
          System.out.println( "not knit" );
          return x.error("the argument have to be a knit");
       }
   }
   y = (kpKnit)klist[0];
   for(int i=1; i < klist.length; i++){
       z = y.joinHorizontal( (kpKnit)klist[i] );
       if( z == null )
          return ((kpDyp)z).error( "Error while knitting" );
       y = z;
   }
   if( x == null ){
          x = y.copy();
          return x;
   }
   else{
          x = ((kpKnit)x).joinVertical(y);
           if(x == null)
              return x.error( "Error while knitting" );
          return x;
   }
}


//semantic check here
public kpDyp joinRow(kpKnit x, kpDyp[] klist){

   int knit_err  = 0;
   boolean first_row = false;
   kpInt rowid = (kpInt)(klist[0]);
   if( rowid.var == 1)
          first_row = true;
   kpInt isrowref = (kpInt)(klist[1]);
   if( isrowref.var == StsSym.ROWREF ){
          kpInt rowref = (kpInt)(klist[2]);
          knit_err = x.repeatRow( rowid.var, rowref.var, first_row);
   }
   else{
          knit_err = x.addRow( klist, first_row );
   }
   if(knit_err < 0){
          return ((kpDyp)x).error( "Error while knitting" );
   }
```

```
   return x;
}


//this should really be called interfuncCall
//note: include semantic check here
public kpDyp funcCall( kpDyp a, kpDyp[] ar){

   kpKnit arg1, arg3;
   kpInt arg2, arg4;
   kpFunc func;

   kpDyp x = symt.getValue( a.name,true,0 );

   //sematic check
   if( !(x instanceof kpFunc) )
          return a.error( "this is not function" );
   if( !( ar[0] instanceof kpKnit) )
          return ar[0].error( "first argument should be a knit" );
   //System.out.println( "function call to: "+ a.name);

   func = (kpFunc)a;
   arg1 = (kpKnit)ar[0];
   if( func.name == "printk" ){
       if( ar.length != 2)
          return ar[1].error( "printk need two arguments" );
       if( !( ar[1] instanceof kpInt ) )
          return ar[1].error( "1st arg of printk should be int" );
       arg2 = (kpInt)ar[1];
       if( arg2.var == PIC ){
          arg1.swingKnit();
       }
       else if( arg2.var == INST ){
          arg1.instKnit();
       }
       else if( arg2.var == TEXT){
          arg1.printKnit();
       }
       else{
          return ar[1].error( "2nd arg of printk is PIC/INST/TEXT" );
       }
       //if( arg2.var != PIC && arg2.var!= INST )
       //arg1.printKnit(arg2.var);
   }
   if( func.name == "trimk" ){
       if( ar.length != 2)
          return ar[1].error( "trimk need two arguments" );
       if( !( ar[1] instanceof kpInt ) )
          return ar[1].error( "2nd arg of trimk is RIGHT/LEFT/TOP/BTM" );
       arg2 = (kpInt)ar[1];
       if( arg2.var !=RIGHT && arg2.var !=LEFT
          && arg2.var !=TOP &&  arg2.var !=BTM )
          return ar[1].error( "2nd arg of trimk is RIGHT/LEFT/TOP/BTM");
       arg1.trimKnit( arg2.var );
   }
   if( func.name == "growk" ){
       if( ar.length != 2)
          return ar[1].error( "growk need two arguments" );
       if( !( ar[1] instanceof kpInt ) )
          return ar[1].error( "2nd arg of growk is RIGHT/LEFT/TOP/BTM" );
       arg2 = (kpInt)ar[1];
       if( arg2.var !=RIGHT && arg2.var !=LEFT
          && arg2.var !=TOP &&  arg2.var !=BTM )
```

```
            return ar[1].error( "2nd arg of growk is RIGHT/LEFT/TOP/BTM");
          arg1.growKnit( arg2.var );
       }
       return x;
    }


} //endof kpItrp class



//kpKnit.java
//
//
//
//
import java.io.PrintWriter;

//wrapper for the Knit data structure...

class kpKnit extends kpDyp {

    Knit ptrn;
    boolean read_only = false;
    static int mygrid[];
    static int grid_hight = 0;


    kpKnit() {
        ptrn = new Knit();
    }
    kpKnit( Knit ptrn ) {
        this.ptrn = ptrn;
    }

    public String typename() {
        return "knit";
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.println( name + " = " );
    }
    public void what( PrintWriter w ) {
        w.print( "<" + typename() + ">  " );
        if ( name != null )
            w.print( name + "  " );
    }


    //which copy should i use?
    public kpDyp copy() {
        return new kpKnit( ptrn );
    }
    public kpDyp deepCopy() {
        return new kpKnit( ptrn.copy() );
    }

    public void setName(String s) {
        ptrn.set_name(s);
    }
```

```
private int getLength() {
   return( ptrn.get_Length() );
}
private  int getAcrs() {
   return( ptrn.get_Acrs() );
}
private int get_last_acrs() {
   return( ptrn.get_last_acrs() );
}
private int get_first_acrs() {
   return( ptrn.get_first_acrs() );
}



public int addRow( kpDyp [] x, boolean first_row ){

   int knit_err = 0;
   kpInt dbgtmp;
    if ( x.length == 0 )
        return -1;
   dbgtmp = (kpInt)x[0];
   if( dbgtmp.var != ptrn.get_current_row() )
        return -1;

   //start from the second element of arr. the first is the label
   if(first_row){ //casting-rows
       for( int i=1; i<x.length; i++ ){
          if( !(x[i] instanceof kpInt) )
              return -1;
          dbgtmp = (kpInt)x[i];
          if( dbgtmp.var != StsSym.K && dbgtmp.var != StsSym.P){
              System.out.println( "illigal casting. exit");
              return -1;
          }
          ptrn.cast_one( dbgtmp.var );
       }
       knit_err = ptrn.rownd_row();
       return knit_err;
   }
   else{   //knitting-rows
       for ( int i=1; i<x.length; i++ ){
          if( !(x[i] instanceof kpInt) )
              return -1;
          dbgtmp = (kpInt)x[i];
          if( dbgtmp.var == StsSym.K || dbgtmp.var == StsSym.P
           ||dbgtmp.var == StsSym.YO || dbgtmp.var == StsSym.TGL){
              knit_err = ptrn.knit_one( dbgtmp.var );
              if( knit_err < 0)
                 return knit_err;
          }
          else if( dbgtmp.var == StsSym.R_INC
              || dbgtmp.var == StsSym.L_INC){
              ptrn.inc_one( StsSym.K, StsSym.K );
          }
          else if( dbgtmp.var == StsSym.R_DEC
              || dbgtmp.var == StsSym.L_DEC){
              ptrn.dec_one();
          }
          else{
              System.out.println( "not knitting symbol. exit");
              return -1;
          }
```

```
        }
        knit_err = ptrn.rownd_row();
        return knit_err;
    }
}



public int repeatRow( int rowid, int rowref, boolean first_row ){

//reapeatRow does NOT repeat kntting instruction
//BUT PATTERN that result from whatever instruction.

    int knit_err = 0;
    kpInt dbgtmp;
    int rowptrn[];

    if(first_row){
        System.out.println( "not knitting symbol. exit");
        return -1;
    }
    else if( rowid != ptrn.get_current_row() ){
        System.out.println( "inconsistan row. exit");
        return -1;
    }
    else if( rowref > ptrn.get_current_row() ){
        System.out.println( "cant repeat future row. exit");
        return -1;
    }
    else{
        rowptrn = ptrn.get_row_ptrn( rowref );
        for ( int i=0; i< rowptrn.length; i++ ){
           if( rowptrn[i] == StsSym.R_DEC) //not go in here
               ptrn.dec_one();
           else if( rowptrn[i] == StsSym.R_INC)
               ptrn.inc_one( StsSym.K, StsSym.K);
           else if( rowptrn[i] != 0 && rowptrn[i] != StsSym.B ){
               knit_err = ptrn.knit_one( rowptrn[i] );
               if( knit_err < 0)
                  return knit_err;
           }
        }
        knit_err = ptrn.rownd_row();
        return knit_err;
    }
}


public kpKnit joinVertical(kpKnit a){
    Knit dbg =  ptrn.stack_knit(a.ptrn);
    if( dbg == null )
           return null;
    return new kpKnit( dbg );
}


public kpKnit joinHorizontal(kpKnit a){
    Knit dbg = ptrn.join_knit(a.ptrn);
    if( dbg == null )
           return null;
    return new kpKnit( dbg );
}
```

```
    public void endKnit(){
       ptrn.end_knit();
       read_only = true;
    }

    public void setReadOnly(){
       read_only = true;
    }

    public void unsetReadOnly(){
       read_only = false;
    }


    //note on senmantic: all the argument to function is
    //of corrct type at this point


    public void swingKnit(){

       mygrid = ptrn.createKnitGrid();
        grid_hight = ptrn.get_grid_hight();
       //System.out.println( "the hight of the grid is " + grid_hight );
        //System.out.println( "the size of the grid is " + mygrid.length );

        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                Kswing.createAndShowGUI(mygrid, grid_hight);
            }
        });
    }


    public void instKnit(){
       ptrn.inst_knit();
    }

    public void printKnit(){
       ptrn.print_knit();
    }

    public void trimKnit(int a){
       System.out.println( "function trimkni");
    }

    public void growKnit(int a){
       System.out.println( "function growknit");
    }

    public void checkKnit(){
       System.out.println( "function checkknit");
    }



}//endof kpKnit class



//kpSt.java
//
//
```

```
//
import java.util.*;
import java.io.PrintWriter;

class kpSt extends HashMap {

    kpSt static_parent;
    boolean read_only;

    public kpSt( kpSt sparent ) {
        static_parent = sparent;
        read_only = false;
    }
    public void setReadOnly() {
        read_only = true;
    }
    public final kpSt staticParent() {
        return static_parent;
    }
    public final kpSt parent( boolean is_static ) {
        return static_parent;
    }
    public final boolean containsVar( String name ) {
        return containsKey( name );
    }
    private final kpSt gotoLevel( int level, boolean is_static ) {
        kpSt st = this;
        if ( level < 0 ){
            while ( null != st.static_parent )
                st = st.parent( is_static );
        }
        else{ //local
            for ( int i=level; i>0; i-- ){
                while ( st.read_only ){
                    st = st.parent( is_static );
                }
                if ( null != st.parent( is_static ) )
                    st = st.parent( is_static );
                else
                    break;
            }
        }
        return st;
    }


    public final kpDyp getValue(String name, boolean is_static, int level ) {
       kpDyp r;
        kpSt st = gotoLevel( level, is_static );
        Object x = st.get( name ); //thisis hash's function
        while ( null == x && null != st.parent( is_static ) ){
           System.out.println( "should't be here atall ");
            st = st.parent( is_static );
            x = st.get( name ); //my function do not go in here...
        }
       r = (kpDyp)x; //if it does not find it, it is null, i think
        return r;
    }


    public final void setValue( String name, kpDyp data,
                                boolean is_static, int level ) {
        kpSt st = gotoLevel( level, is_static );
```

```
        while ( st.read_only ){
            st = st.parent( is_static );
        }
        st.put( name, data );
    }


    public void what( PrintWriter output ) {
        for ( Iterator it = values().iterator() ; it.hasNext(); )
        {
            kpDyp d = ((kpDyp)(it.next()));
            //if( !( d instanceof MxFunction && ((MxFunction)d).isInternal() ) )
                //d.what( output );
        }
    }

    public void what() {
        what( new PrintWriter( System.out, true ) );
    }
}//end of kpSt class




//kpInt.java
//
//
//
//
import java.io.PrintWriter;

//kpInt.java

class kpInt extends kpDyp {

    int var;

    public kpInt( int x ) {
        var = x;
    }

    public String typename() {
        return "int";
    }

    public kpDyp copy() {
        return new kpInt( var );
    }

    public static int intValue( kpDyp b ) {
        if ( b instanceof kpInt )
            return ((kpInt)b).var;
        b.error( "ERR -dont' do that" );
        return 0;
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.println( Integer.toString( var ) );
    }

    public kpDyp times( kpDyp b ) {
```

```
            return new kpInt( var * intValue(b) );
    }
}
//endof kpInt.class




//kpVar.java
//
//
//
import java.io.PrintWriter;

//The wrapper class for unsigned variables

class kpVar extends kpDyp {
    public kpVar( String name ) {
        super( name );
       //System.out.println("new variable" +name);
    }
    public String typename() {
        return "undefined-variable";
    }
    public kpDyp copy() {
        throw new kpException( "Variable " + name + " has not been defined" );
    }
    public void print( PrintWriter w ) {
        w.println( name + " = <undefined>" );
    }
}
//end of kpVar class



//kpDyp.java
//
//
//
//
import java.io.PrintWriter;

//The base data type class (also a meta class)
//kpDyp.java

public class kpDyp
{
    public final static int TYPE_INT = 1;
    public final static int TYPE_KNIT = 2;
    public final static int TYPE_EYE = 3;
    public final static int TYPE_STR = 5; //string
    public final static int TYPE_FUNC = 6;

    String name;   // used in hash table
    int typeid = 0; //meaning, its unknown

    public kpDyp() {
        name = null;
    }

    public kpDyp( String name ) {
```

```
    this.name = name;
}

public kpDyp( String name, int typeid ) {
    this.name = name;
    this.typeid = typeid;
}
public String typename() {
    return "unknown";
}
public kpDyp copy() {
    return new kpDyp();
}
public void setName( String name ) {
    this.name = name;
}

public kpDyp error( String msg ) {
    throw new kpException( "illegal operation: " + msg
                           + "( <" + typename() + "> "
                           + ( name != null ? name : "<?>" )
                           + " )" );
}

public kpDyp error( kpDyp b, String msg ) {
    if ( null == b )
        return error( msg );
    throw new kpException(
        "illegal operation: " + msg
        + "( <" + typename() + "> "
        + ( name != null ? name : "<?>" )
        + " and "
        + "<" + typename() + "> "
        + ( name != null ? name : "<?>" )
        + " )" );
}

public void print( PrintWriter w ) {
    if ( name != null )
        w.print( name + " = " );
    w.println( "<undefined>" );
}

public void print() {
    print( new PrintWriter( System.out, true ) );
}

public void what( PrintWriter w ) {
    w.print( "<" + typename() + ">  " );
    print( w );
}

public void what() {
    what( new PrintWriter( System.out, true ) );
}

public kpDyp multrow( kpDyp b ) {
   //vertical multi
    return error( b, "**" );
}
public kpDyp dott( kpDyp b ) {
   //dot operator is only for eyes
    return error( b, "." );
```

```
    }
    public kpDyp assign( kpDyp b ) {
        return error( b, "=" );
    }
    public kpDyp uminus() {
        return error( "-" );
    }
    public kpDyp plus( kpDyp b ) {
        return error( b, "+" );
    }
    public kpDyp minus( kpDyp b ) {
        return error( b, "-" );
    }
    public kpDyp times( kpDyp b ) {
        return error( b, "*" );
    }
    public kpDyp dv( kpDyp b ) {
        return error( b, "/" );
    }
    public kpDyp modulus( kpDyp b ) {
        return error( b, "%" );
    }
    public kpDyp gt( kpDyp b ) {
        return error( b, ">" );
    }
    public kpDyp ge( kpDyp b ) {
        return error( b, ">=" );
    }
    public kpDyp lt( kpDyp b ) {
        return error( b, "<" );
    }
    public kpDyp le( kpDyp b ) {
        return error( b, "<=" );
    }
    public kpDyp eq( kpDyp b ) {
        return error( b, "==" );
    }
    public kpDyp ne( kpDyp b ) {
        return error( b, "!=" );
    }

} //endof class kpDype



//kpFunc.java
//
//
//
//
import java.io.PrintWriter;
import antlr.collections.AST;


class kpFunc extends kpDyp {

    String[] args;
    int num_arg;

    public kpFunc( String name, String[] args, int num_arg) {
        super( name );
        this.args = args;
```

```
        this.num_arg = num_arg;
    }
    public kpFunc( String name ) {
        super( name );
       num_arg = 0;
    }
    public String typename() {
        return "function";
    }
    public void print( PrintWriter w ) {
            w.println( name + " = <function> " );
    }
    public String[] getArgs() {
        return args;
    }

}
```

//Knit ADT and its graphic components:

```
-rwxr-xr-x    1 root     root         8109 Dec 22 14:55 SpringUtilities.java
-rwxr-xr-x    1 root     root         3162 Dec 22 14:55 Kswing.java
-rwxr-xr-x    1 root     root          768 Dec 22 14:56 StsSym.java
-rwxr-xr-x    1 root     root        15386 Dec 22 14:56 Knit.java
-rwxr-xr-x    1 root     root         1363 Dec 22 14:56 Eye.java
```

```
//StsSym.java
//
//
//
//This class hold the basic stitch and short hand to combine those stitches.
import java.io.*;
import java.util.*;

public class StsSym{
       //this is the key to the graphics for build-in symbols
       //0-16
       final static int S = 0;
       final static int K = 1;
       final static int P = 2;
       final static int B = 3;
       final static int D = 4;
       final static int YO = 5;
       final static int R_DEC = 6;
       final static int L_DEC = 7;
       final static int C_DEC = 8;
       final static int R_D_DEC = 9;
       final static int L_D_DEC = 10;
       final static int R_INC = 11;
       final static int L_INC = 12;
       final static int TGL = 13;
       final static int ROWREF = 14;

       final static int BEG = 20;
        final static int DEC_ONE = 21;
        final static int INC_ONE = 22;
        final static int INC_TWO = 23;
        final static int RWND_B = 24;
        final static int RWND_T = 25;
}



//Eye.java
//
//
//
//
import java.io.*;
import java.util.*;

public class Eye{
       public int stitch= 0;
       public int id= 0;
       public Eye hook = null; //pointer to the above
       public Eye side = null; //pointer to the left

       public int sts_instruction = 0; //whereever sts goes, it goes.

       public Eye(){
              stitch = StsSym.S;
```

```
		sts_instruction = StsSym.S;
		hook = null;
		side = null;
	}

	public Eye(int in_sts){
		this();
		stitch = in_sts;
	}
	public Eye(int in_sts, int in_sts_inst){
		this();
		stitch = in_sts;
		sts_instruction = in_sts_inst;
	}

	public Eye(int in_sts, Eye hook_sts, Eye side_sts, int in_id){
		this();
		stitch = in_sts;
		hook = hook_sts;
		side = side_sts;
		id = in_id;
	}
	public Eye(int in_sts, Eye hook_sts,
		Eye side_sts, int in_id, int in_sts_inst){
		this();
		sts_instruction = in_sts_inst;
		stitch = in_sts;
		hook = hook_sts;
		side = side_sts;
		id = in_id;
	}
	public void set_id( int in_id ){
		id = in_id;
	}
	public void set_eye( int in_sts ){
		stitch =  in_sts;
	}
	public void set_sts_inst( int in_sts_inst ){
		sts_instruction =  in_sts_inst;
	}
	public void set_side( Eye side_sts ){
		side = side_sts;
	}
	public void set_hook( Eye hook_sts ){
		hook =  hook_sts;
	}
	public Eye copy(){
		Eye r = new Eye(stitch, sts_instruction);
		//do not copy the hook and side. it should stay null
		//at this stage
		return r;
	}


}//endofeye class



//Knit.java
//
//
//
//
```

```
import java.io.*;
import java.util.*;

public class Knit{

static final int KNIT_ERR = -1;
private String myname = "foo";
private boolean FIN = false;
private int top_mrg = 2; //int side_mrg= 2; //used for GUI

private int Acrs = 0;            //the total eye in the widest row so far
private int current_acrs = 0;    //current position in the current row
private int MaxInc = 0;

private int Length = 0;    //total row, so far
private int current_row = 0;     //current row
private int next_id = 0;
private int first_acrs = 0;
private int last_acrs = 0;

//Caution. when counting eyes in the row. it does not include the
//2 "B"s at the both end... may need to add 2+ at time.

private Eye left_btm = new Eye();
private Eye current_sts = null;
private Eye pre_sts = null;

//those are knitting needles... two needles for now
private Stack needle_L = new Stack();   //Left knitting needle
private Stack needle_R = new Stack();   //Right needle
private Stack pushto_ndl = needle_R;
private Stack popfrm_ndl = needle_L;



public Knit(){

       //bookkeeping...
       myname = "NO NAME";
       Acrs = 0;
       Length = 1;
       current_row = 1;
       current_acrs = 0;

       next_id = 1;
       first_acrs= 0;
       last_acrs = 0;

       //DBG System.out.println( "   start_knit: ");
       Eye tmp_side = new Eye();
       Eye tmp_above = new Eye();
       left_btm = new Eye( StsSym.B, tmp_side, tmp_above, next_id );
       pre_sts = left_btm;
       current_sts = left_btm.side;
}

public Knit(String x){
       this();
       myname = x;
}


public void set_name(String x){
```

```
        myname = x;
}


public void cast_one(int in_sts){

        //note: "casting" is to making of the very first row

        //this check is taken care at kpKnit
        //if( in_sts != StsSym.P || in_sts != StsSym.K )
        //     return KNIT_ERR;

        current_sts.set_eye( in_sts );
        current_sts.set_id( ++next_id );

        Eye tmp_side = new Eye();
        Eye tmp_above = new Eye();
        current_sts.set_side( tmp_side );
        current_sts.set_hook( tmp_above );

        //push the "hook" for later use
        pushto_ndl.push( current_sts.hook );

        //advance the pointer
        pre_sts = current_sts;
        current_sts = current_sts.side;

        //bookkeeping
        Acrs++;
        current_acrs++;
        first_acrs++;
}



public int rownd_row(){

        current_sts.set_eye(StsSym.B);    //first, set the last eye
        current_sts.set_id( ++next_id );
        Eye tmp_above = new Eye(); //create eye above
        current_sts.set_hook( tmp_above );

        //then, advance to the above stitch
        pre_sts = current_sts;
        current_sts = current_sts.hook;

        //set this eye, the first of the next row
        current_sts.set_eye(StsSym.B);
        current_sts.set_id( ++next_id );
        Eye tmp_side = new Eye();
        Eye tmp_above_above = new Eye();
        current_sts.set_side( tmp_side );
        current_sts.set_hook( tmp_above_above );

        //advance the pointer
        pre_sts = current_sts;
        current_sts = current_sts.side;

        //now, check the sanity and change the needle
        if( popfrm_ndl.empty() !=  true ){
              return(KNIT_ERR); //ERR
        }
        if( pushto_ndl.empty() ){
```

```
            return(KNIT_ERR); //ERR
        }
        //switch the needle popfrm points what pushto pointed
        Stack tmp_stk = popfrm_ndl;
        popfrm_ndl = pushto_ndl;
        pushto_ndl = tmp_stk;

        //bookkeeping
        if( current_acrs > Acrs ){
            Acrs = current_acrs;
        }
        last_acrs = current_acrs;
        current_acrs = 0;
        Length++;
        current_row++;
        return(0); //OK
}



public int knit_one(int in_sts){

        current_sts.set_eye( in_sts );
        current_sts.set_id( ++next_id );
        if( popfrm_ndl.empty() ){
            System.out.println( "you can't knit that ... " );
            return(KNIT_ERR); //ERR
        }
        current_sts.set_side( (Eye)(popfrm_ndl.pop()) );

        Eye tmp_above = new Eye();
        current_sts.set_hook( tmp_above );
        pushto_ndl.push( current_sts.hook );

        //now, advance the pointer...
        pre_sts = current_sts;
        current_sts = current_sts.side;

        //bookkeeping...
        current_acrs++;

        return(0); //OK
}



public int dec_one(){

        current_sts.set_eye( StsSym.B );
        //current_sts.set_eye( StsSym.D );
        current_sts.set_sts_inst( StsSym.DEC_ONE );
        current_sts.set_id( ++next_id );

        //dec is same as knit_one, except it not create hook
        if( popfrm_ndl.empty() ){
            System.out.println( "you can't knit that ... " );
            return(KNIT_ERR); //ERR
        }
        current_sts.set_side( (Eye)(popfrm_ndl.pop()) );
        //then advance the pointer...
        pre_sts = current_sts;
        current_sts = current_sts.side;
```

```
        //current_acrs++;

        return(0); //OK
}


public int inc_one(int in_sts, int add_sts){

        current_sts.set_eye( in_sts );
        current_sts.set_sts_inst( StsSym.INC_ONE );
        current_sts.set_id( ++next_id );

        if( popfrm_ndl.empty() ){
                System.out.println( "you can't knit that ... " );
                return(KNIT_ERR); //ERR
        }
        current_sts.set_side( (Eye)(popfrm_ndl.pop()) );
        Eye tmp_above = new Eye();
        current_sts.set_hook( tmp_above );
        pushto_ndl.push( current_sts.hook );

        //now, advance the pointer...
        pre_sts = current_sts;
        current_sts = current_sts.side;
        current_acrs++;

        //DBG System.out.println( "   inc_one2: ");
        current_sts.set_eye( in_sts );
        current_sts.set_sts_inst( StsSym.INC_TWO );
        current_sts.set_id( ++next_id );
        Eye tmp_side = new Eye();
        tmp_above = new Eye();
        current_sts.set_side( tmp_side);
        current_sts.set_hook( tmp_above );
        pushto_ndl.push( current_sts.hook );
        pre_sts = current_sts;
        current_sts = current_sts.side;
        current_acrs++;

        MaxInc++;
        return(0); //OK
}



public int bind_edge(){

        //need binding to distinguish from inconsistant ones
        current_sts.set_eye( StsSym.B );
         current_sts.set_id( ++next_id );
         current_sts.set_side( (Eye)(popfrm_ndl.pop()) );
         pre_sts = current_sts;
         current_sts = current_sts.side;
         return 0;
}


public int end_knit(){
        int err = 0;
        while( popfrm_ndl.empty() ==  false ){
                 err = bind_edge();
         }
```

```
        //bind the last eye
        current_sts.set_eye(StsSym.B);
        current_sts.set_id( ++next_id );
        FIN = true;
        Length--;
        current_row--;
        return err;
}



public String selectr_symbol( int sts_id ){
        if (sts_id == StsSym.K){
                return( "| "); //return( "K ");
        }
        if (sts_id == StsSym.P){
                return( "- "); //return( "P ");
        }
        if (sts_id == StsSym.B){
                return( "B ");
        }
        if (sts_id == StsSym.YO){
                return( "o");
        }
        if (sts_id == StsSym.TGL){
                return( "v");
        }
        if (sts_id == StsSym.S){
                return( "S ");
        }
        return( "x" ); //meaning, not known.
}



public String selectr_inst( int sts_id ){
        if (sts_id == StsSym.INC_TWO){
                return( "(inc)");
        }
        if (sts_id == StsSym.DEC_ONE){
                return( "(dec)");
        }
        if (sts_id == StsSym.K){
                return( "K ");
        }
        if (sts_id == StsSym.P){
                return( "P ");
        }
        if (sts_id == StsSym.B){
                return( "B ");
        }
        if (sts_id == StsSym.YO){
                return( "YO");
        }
        if (sts_id == StsSym.TGL){
                return( "TGL");
        }
        if (sts_id == StsSym.S){
                return( "");
        }
        return( "" ); //meaning, not known.
}
```

```
public void inst_knit(){

	System.out.println( "**" );
	System.out.println( "**" );
	System.out.println( "**" );
	System.out.println( ": "+ myname );
	Eye itr = null;
	itr = left_btm;
	while(true){
		if( itr.side != null ){
		System.out.print( " " + selectr_inst( itr.stitch ) );
		System.out.print( "" + selectr_inst( itr.sts_instruction ) );
		itr =itr.side;
		}
		else if( itr.hook != null ){	//change the row
		System.out.print( " " + selectr_inst( itr.stitch ) );
		System.out.println( ":change the row... " );
		itr = itr.hook;
		}
		else{
		//the last one
		System.out.print( " " + selectr_inst( itr.stitch ) );
		System.out.println( " " );
		break;
		}
	}
	System.out.println( "**" );
	System.out.println( "**" );
	System.out.println( "**" );
}


public void print_knit(){

	System.out.println( "**" );
	System.out.println( "**" );
	System.out.println( "**" );
	System.out.println( ":begin the knit " );
	System.out.println( ": "+ myname );
	Eye itr = null;
	itr = left_btm;
	while(true){
		if( itr.side != null ){
		System.out.print( " " + selectr_symbol( itr.stitch ) );
		itr =itr.side;
		}
		else if( itr.hook != null ){	//change the row
		System.out.print( " " + selectr_symbol( itr.stitch ) );
		System.out.println( ":change the row... " );
		itr = itr.hook;
		}
		else{
		//the last one
		System.out.print( " " + selectr_symbol( itr.stitch ) );
		System.out.println( " " );
		System.out.println( ":end of this knit " );
		break;
		}
	}
	System.out.println( "**" );
	System.out.println( "Length (total #of row): "+ Length );
	System.out.println( "Acrs (#of eyts of the widest row): "+ Acrs );
```

```
        System.out.println( "#of eyes in last row : "+ last_acrs );
        System.out.println( "#of eye in first : "+ first_acrs );
        System.out.println( "**" );
        System.out.println( "**" );
        System.out.println( "**" );
}


public int get_grid_hight(){
        return Length + 1 + top_mrg ;
}

public int[] createKnitGrid(){

    int width_of_grid = MaxInc +1 +first_acrs +1 +MaxInc;
    int hight_of_grid = Length + 1 + top_mrg ;

    int grid[] = new int[width_of_grid * hight_of_grid];
    int grid_i = 0;
    Eye itr = left_btm;
    int[] row_ptrn;

       //initialization
       for(int i =0; i<width_of_grid * hight_of_grid; i++)
              grid[i] = StsSym.B;

       //bottom pudding
       for(int i =0; i< width_of_grid; i++){
              grid[grid_i] = StsSym.B;
              grid_i++;
       }
       for(int i =0; i< hight_of_grid -2; i++){
              row_ptrn = get_row_ptrn(i+1, width_of_grid);
              for(int j =0; j< width_of_grid; j++){
                     grid[grid_i] = row_ptrn[j];
                     grid_i++;
              }
        }
       for(int i =0; i< width_of_grid; i++){
              grid[grid_i] = StsSym.B;
              grid_i++;
       }
       return grid;
}


public int[] get_row_ptrn( int rowid, int width_of_grid ){

       int[] r = new int[width_of_grid];
       Eye itr = left_btm;
       int[] row_ptrn = get_row_ptrn(rowid);
       int j =0;
       //count real sts's in the row
       int sts_count = 0;
       int i = 0;
       for(i=0; i< row_ptrn.length; i++){
              if( row_ptrn[i] != 0 )
                     sts_count++;
       }
       int pud = (width_of_grid - sts_count )/2;
       //System.out.println( "beginning the new row: pud =" +pud);
       for(i =0; i< pud; i++){
              //System.out.println( "pud " +j +" value "+ StsSym.B);
```

```
            r[j] = StsSym.B; j++;
        }
        for(i =0; i< sts_count; i++){
            //System.out.println( "sts " +j +" value "+ row_ptrn[i]);
            r[j] = row_ptrn[i]; j++;
        }
        for(; j< width_of_grid; j++){
            //System.out.println( "pud " +j +" value "+ StsSym.B);
            r[j] = StsSym.B;
        }
        return r;
}


public int get_Acrs(){
        return Acrs;
}
public int get_Length(){
        return Length;
}
public int get_first_acrs(){
        return first_acrs;
}
public int get_last_acrs(){
        return last_acrs;
}
public int get_current_row(){
        return current_row;
}


public int[] get_row_ptrn( int rowid ){

        //this copy all sts in row BxxxxxxxxxxxxB, inclucive
        int[] r = new int[Acrs+2];
        Eye itr = left_btm;
        int i=1; int j=0;
        while( i <= rowid ){
            if( i == rowid ){
                 r[j] = itr.stitch; j++;
             }
            if( itr.side != null ){
               itr =itr.side;
            }
            else if( itr.hook != null ){  //change the row
               itr = itr.hook; i++;
            }
            else{
               break;
            }
        }
        return r;
}



public int add_one(int sts, int sts_inst, int sts2){

        //parse sts_inst and call apporopriate sts
        if(sts == StsSym.K || sts == StsSym.P ||
               sts == StsSym.YO || sts == StsSym.TGL){
            if(sts_inst == StsSym.INC_ONE){
               return sts; //do nothing. do it later
```

```
            }
            else if(sts_inst == StsSym.INC_TWO){
               inc_one( sts2, sts); return sts;
            }
            else{
               knit_one( sts ); return sts;
            }
        }
        if(sts == StsSym.B ){
            if(sts_inst == StsSym.DEC_ONE){
               dec_one(); return sts;
            }
            else{
               return sts;
               //bind_edge() is take care by end_knit()
            }
        }
        return 0;
}



public Knit copy(){

        Eye itr= left_btm;
        Knit newk = new Knit();

        itr =itr.side;
        //to skip duping the first one.
        //is made when knit is created.

        //its really the error if there is anything but B or K
        while( itr.side != null ){
               if( itr.stitch == StsSym.B &&
                   itr.sts_instruction != StsSym.DEC_ONE )
                      System.out.println(" what? 1 ");
               else
                      newk.cast_one(itr.stitch);
               itr =itr.side;
        }
        newk.rownd_row();
        //end of the first row

        //its really the error if this is null.
        if( itr.hook != null )
               itr = itr.hook;

        int prev_sts = 0;
        while(true){
            if( itr.side != null ){
               prev_sts = newk.add_one(itr.stitch,
                      itr.sts_instruction, prev_sts);
               itr =itr.side;
            }
            else if( itr.hook != null ){
               itr = itr.hook;
               newk.rownd_row();
            }
            else
               break; //fall from the loop
        }
        newk.end_knit();  //beter this way.
        return newk;
```

```
}


public Knit stack_knit(Knit x){

       if (x.get_first_acrs() != last_acrs ){
              //this is error
              return null;
       }

       Knit newk = new Knit();
       Eye itr= left_btm;
       itr =itr.side;
       while( itr.side != null ){
              if( itr.stitch == StsSym.B &&
                  itr.sts_instruction != StsSym.DEC_ONE )
                     System.out.println(" what? 1 ");
              else
                     newk.cast_one(itr.stitch);
              itr =itr.side;
       }
       newk.rownd_row();
       if( itr.hook != null )
              itr = itr.hook;
       int prev_sts = 0;
       while(true){
           if( itr.side != null ){
              prev_sts = newk.add_one(itr.stitch,
                     itr.sts_instruction, prev_sts);
              itr =itr.side;
           }
           else if( itr.hook != null ){
              itr = itr.hook;
              newk.rownd_row();
           }
           else
              break; //fall from the loop
       }
       //newk.rownd_row();

       itr = x.left_btm;
       itr =itr.side;
       prev_sts = 0;
       while(true){
           if( itr.side != null ){
              prev_sts = newk.add_one(itr.stitch,
                     itr.sts_instruction, prev_sts);
              itr =itr.side;
           }
           else if( itr.hook != null ){
              itr = itr.hook;
              newk.rownd_row();
           }
           else
              break; //fall from the loop
       }
       newk.end_knit();
       return newk;
}



public Knit join_knit( Knit x ){
```

```
    //this function create/return a new knit
    //that is dup of self:x left:right
    //institch instruction.

    if (x.get_Length() != Length ){
           //this is error
           return null;
    }
    Knit dup = new Knit();
    Eye itr= left_btm;
    Eye right_itr = x.left_btm;
    int prev_sts =0;

    //first row ,left knit
    itr =itr.side; //skip the v.first sts
    while( itr.side != null ){
           if( itr.stitch == StsSym.B &&
               itr.sts_instruction != StsSym.DEC_ONE )
                  System.out.println(" what? 1 ");
           else
                  dup.cast_one(itr.stitch);
           itr =itr.side;
    }
    if( itr.hook != null )
           itr = itr.hook;

    //first row ,right knit
    right_itr =right_itr.side; //skip the v.first sts
    while( right_itr.side != null ){
           if( right_itr.stitch == StsSym.B &&
               right_itr.sts_instruction != StsSym.DEC_ONE )
                  System.out.println(" what? 1 ");
           else
                  dup.cast_one(right_itr.stitch);
           right_itr = right_itr.side;
    }
    dup.rownd_row();
    if( right_itr.hook != null )
           right_itr = right_itr.hook;
    //end of first rows

while(true){
    //even rows
    prev_sts = 0;
    while( right_itr.side != null ){
           prev_sts = dup.add_one(right_itr.stitch,
                  right_itr.sts_instruction, prev_sts);
           right_itr =right_itr.side;
    }
    if( right_itr.hook != null )
           right_itr = right_itr.hook;
    else
           break; //fall from the loop
    prev_sts = 0;
    while( itr.side != null ){
           prev_sts = dup.add_one(itr.stitch,
                  itr.sts_instruction, prev_sts);
           itr =itr.side;
    }
    dup.rownd_row();
    if( itr.hook != null )
           itr = itr.hook;
```

```
       else
              break; //fall from the loop
       //end even rows

       //odd rows
       prev_sts = 0;
       while( itr.side != null ){
              prev_sts = dup.add_one(itr.stitch,
                     itr.sts_instruction, prev_sts);
              itr =itr.side;
       }
       if( itr.hook != null )
              itr = itr.hook;
       else
              break; //fall from the loop
       prev_sts = 0;
       while( right_itr.side != null ){
              prev_sts = dup.add_one(right_itr.stitch,
                     right_itr.sts_instruction, prev_sts);
              right_itr =right_itr.side;
       }
       dup.rownd_row();
       if( right_itr.hook != null )
              right_itr = right_itr.hook;
       else
              break; //fall from the loop
       //end odd rows
    }
    dup.end_knit();
    return dup;

}


}//end_of_the_class.knit


//Kswing.java
//
//
//
//

import javax.swing.*;
import java.awt.*;

public class Kswing {

    static public int myint = 0;

    protected static ImageIcon createImageIcon(String path,
        String description) {
        java.net.URL imgURL = Kswing.class.getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL, description);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }

    //private static void createAndShowGUI() {
```

```
    public static void createAndShowGUI(int mygrid[], int grid_hight) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("QDP of your Knitting");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container contentPane = frame.getContentPane();

       contentPane.setBackground(Color.white);
        contentPane.setLayout(new SpringLayout());


//begin design

       JLabel stslbl;

ImageIcon knit_kIcon = createImageIcon("sml/k.jpg", "knit_k");
ImageIcon knit_pIcon = createImageIcon("sml/p.jpg", "knit_p");
ImageIcon knit_brIcon = createImageIcon("sml/br.jpg", "knit_br");
ImageIcon knit_br_bIcon =createImageIcon("sml/br_b.jpg","knit_br_b");
ImageIcon knit_xxIcon =createImageIcon("sml/xx.jpg","knit_xx");
ImageIcon knit_c_decIcon = createImageIcon("sml/c_dec.jpg", "knit_c_dec");
ImageIcon knit_l_d_decIcon = createImageIcon("sml/l_d_dec.jpg", "knit_l_d_dec");
ImageIcon knit_r_d_decIcon = createImageIcon("sml/r_d_dec.jpg", "knit_r_d_dec");
ImageIcon knit_l_decIcon = createImageIcon("sml/l_dec.jpg", "knit_l_dec");
ImageIcon knit_r_decIcon = createImageIcon("sml/r_dec.jpg", "knit_r_dec");
ImageIcon knit_l_incIcon = createImageIcon("sml/l_inc.jpg", "knit_l_inc");
ImageIcon knit_r_incIcon = createImageIcon("sml/r_inc.jpg", "knit_r_inc");
ImageIcon knit_tglIcon = createImageIcon("sml/tgl.jpg", "knit_tgl");
ImageIcon knit_yoIcon = createImageIcon("sml/yo.jpg", "knit_yo");

       for(int i=0; i< mygrid.length; i++){
              stslbl = new JLabel("");
              if(mygrid[i] == StsSym.K) //1
                     stslbl.setIcon( knit_kIcon );
              else if(mygrid[i] == StsSym.P) //2
                     stslbl.setIcon( knit_pIcon );
              else if(mygrid[i] == StsSym.B) //3
                     stslbl.setIcon( knit_br_bIcon );
              else if(mygrid[i] == StsSym.R_DEC)
                     stslbl.setIcon( knit_brIcon );
              else if(mygrid[i] == StsSym.YO)
                     stslbl.setIcon( knit_yoIcon );
              else if(mygrid[i] == StsSym.TGL)
                     stslbl.setIcon( knit_tglIcon );
              else
                     stslbl.setIcon( knit_brIcon );
              contentPane.add(stslbl);
       }
       int comp_total = contentPane.getComponentCount();
        //SpringUtilities.makeCompactGrid(contentPane,
             //hight,width,0,0,0,0);
        SpringUtilities.makeCompactGrid(contentPane,
             grid_hight, comp_total/grid_hight, 0,0,0,0);
//endofdesign

        frame.pack();
        frame.setVisible(true);
    }

       /**********************
    public static void main(String[] args) {
       javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                //kpswing.createAndShowGUI();
```

```
                createAndShowGUI();
            } });
    }
       *************************/
}
```

Appendix B. Credit, and Notes on Software Development Environment

KP language was developed with J2SE ver1.4.2 SDK at Redhat Linux 7.2 (Enigma, Kernel 2.4.7-10) for i686, and Mac OS 10.3 (Darwin 7.0.0) on iBook with default PowerPC 750.

Antlr ver 2.7.4 was used to generate KP grammar and tree parser.

Many a code snippet for tree parser, code generation, and semantic checking was modified from the examples at Antlr's web site, as well as the impressive array of projects from past PLT students. Mx language from PLT spring 2003 is especially helpful.

Spring layout utilities for Knit's graphical display were downloaded from Sun Microsystem's Swing tutorial website.

The KP source codes ware written mostly with vi, or vim, editor, written by Bram Moolenaar and others. Tcsh, my shell, is written by Bill Joy and whole a lot of others after him.