progL Programing Language Manual

PLC, April 2019 - Assignment 2

GIT Repository

https://github.com/akraradets/PLC_progL

Group Members

st120224 Mr. Akraradet Sinsamersuk
st120534 Ms. Ruangtiwa Kimsungnoen
st120474 Mr. Nawarathnage Lakmal Deshapriya

Background

progL is a programming language which is similar to JAVA. And it supports 3 data types which include '**bool**, **int**, **string**', and progL has following main programming language elements,

- Variable Declaration and Definition
- Run Basic Mathematical Operations such as Plus, Subtraction, Multiplication, and Division
- Run Basic Conditional Operations such as, greater than, less than, equals, etc;
- Conditional Statement (IF-THEN-EISE)
- While Loop
- Function Call

A simplified version of progL was first implemented in Python programing language to design the overall architecture of progL and the final version was developed with JAVA LEX and CUP libraries. In our early attempts in Python, the code was much more simple, short and easy to understand than JAVA which makes it easy to test ideas. Briefly, the main 3 components of the progL structure that we tested in Python environment was as follows,

1. Define lexical rules
2. Define grammar rules as function and pass commands to a pass trees
3. Run and control the flow of pass tree generated from a grammar in a recursive manner to get the output

In the case of Python, PLY library was used (PLY.LEX - lexical analyzer, YACC.YACC - syntax analyzer).

Basic Rules of progL

1. The end of a statement must be used with '**;**'.
2. Types of variable:
    - **bool** for boolean
    - **int** for integer
    - **String** for character or string
3. Names of variable cannot be int, bool, string, if, else, while, null, true, false, function, return, print, =, (,), {, }, +, -, *, /, &, ||, <, >, !.

How to Evaluate Code in progL

After running code, you will get the following user interface. Insert to code in the Input box and click on Run to get the results

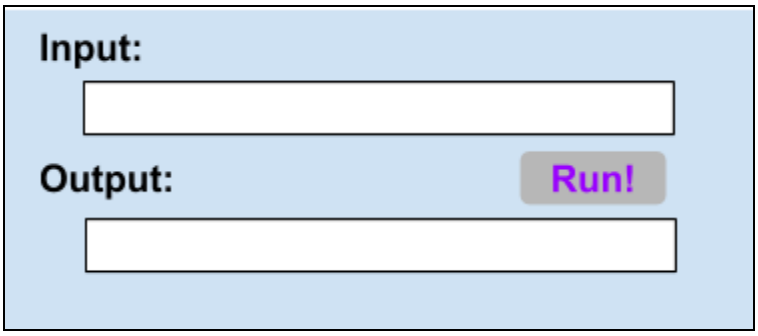

Figure1: UI for using progL.

In the input field, users can type the statement like below to get the output by clicking the Run button.

```
int x = 1; int y;
y = 5 + x;
if(x == y) { print( y ); }
else { print( y = 0 ); };
```

Programing Elements progL

1. Variables

It can be **A-Z, a-z, 0-9, _.**

- For **int** type, the variable should be
  `int x; int Y = 20+1;`.
- For **bool** type, the variable should be
  `bool z = true;`.
- For **string** type, the variable should be in " ", *e.g.*
  `string test_1 = "Test";` .

2. Operations

We have 4 operations as follow,

- ' **+** ' to plus the expression. The **plus** accepts **int** and **string** type,
  e.g. for **string,** this can combine them.
  `string y = "Hello"; print (y + "" + "World");`
  This will print `Hello World.`
- ' **-** ' to minus the expression. The minus accept only **int** type , *e.g.*
  `int x = 10; int y = x-10; print (y-10);`
  This will print `-10` .
- ' ***** ' to time the expression. Only **int** is accepted , *e.g.*
  `int x = 5; if (x*1 == 6) { print (x); }; print(" 'if' is not working");`
  The output will print `'if' is not working` .
- ' **/** ' to divide the expression, *e.g.*

```
int x = 1;
function g(int x){ print(x); };
g(x*50/10);
```

The output will print `5.`

3. Conditions

For only **bool** type,

- && - And Operation - check the result of first-condition and second-condition like the normal AND condition. The result is shown in the following table.

Condition (AND)	Result
True && True	**True**
True && False	**False**
False && True	**False**
False && False	**False**

- || - OR Operation - check the result of first-condition and second-condition like the normal **OR** condition. The result is shown in following table.

Condition (OR)	Result
True || True	**True**
True || False	**True**
False || True	**True**
False || False	**False**

- ! - NOT Operation - negate condition value, ***e.g.***

  `bool x = true; bool y = !x;` value of **y** is false.

- For only **int** Data Type.
  - **' == '**

    to check that the value of first-condition and second-condition are **equal**, *e.g.*

```
int x = 5; int y = 5; if (x == y) { print "x is equal with y";};
```

    It will print `x is equal with y` because the result of condition is true.
  - **' < '**

    to check that the value of first-condition is **lesser** than second-condition, *e.g.*

```
int x = 50; int y = 10; if (x < y) { print"x is lesser than y";};
```

    It will not print `x is lesser than y` because the result of condition is false.
  - **' > '**

    to check that the value of first-condition is **greater** than second-condition, *e.g.*

```
int x = 50; int y = 50; if (x > y) { print "x is greater than y";};
```

    It will not print `x is greater than y` because the result of condition is false.
  - **' >= '**

    to check that the value of first-condition is **equal** or **greater** than second-condition, *e.g.*

```
int x = 50; int y = 50;
if (x >= y) { print "x is equal or greater than y";};
```

    It will print `x is equal or greater than y` because the result of condition is true
  - **' <= '**

    to check that the value of first-condition is **equal** or **lesser** than second-condition, *e.g.*

```
int x = 10; int y = 50; if (x >= y){ print "x is equal or lesser than y";};
```

It will print `x is equal or lesser than y` because the result of condition is true.

4. Conditional Statement-

progL is supporting Two types of conditional statements which are “if” and “if-then-else”

- **‘ if ’**

Syntax of **if :**

```
if (condition) { 1st-statement; 2nd-statement; ... n-statement; };
```

It is used to check the result of condition before doing the statement in **if,** *e.g.*

```
int x = 10; int y = 50;
if (x <= y) { print "x is equal or lesser than y";};
```

Before `print` function, it will check result from `(x <= y)` first, and the result of this is `x is equal or lesser than y` .

- **‘ if-else ’**

Syntax of **if-else :**

```
if (condition) { 1st-statement; 2nd-statement; ... n-statement; }
else { 1st-statement; 2nd-statement; ... n-statement; };
```

It is used to check the result of condition before doing the statements. The **if-else** has 2 directions to choose. First, the condition is checked, if the result of the condition is true, the statements in **if** will be done. Otherwise, the statements in **else** will be done, *e.g.*

```
bool x = true; bool y = !x;
if ( x && y ) { print("The result is true!");}
else { print("The result is false!"); };
```

Before both of `print` functions, it will check result from `(x && y)` first. If the result is **true**, `print` function in **if** will be done, and the result is `The result is true!`. On the other hand, `print` function in **else** will be done, and the result is `The result is false!`.

5. While Loop -

- **‘ while ’**

Syntax of **while :**

```
while ( condition ) { 1st-statement; 2nd-statement; ... n-statement; };
```

It is used to be a loop (*like normal while loop*). The **while** will check result of condition first. If the result is **true**, the statements in while loop is done, after that the condition will be checked again. If the result still to be true, all statements will be done again, then check the condition. This loop will do until the result is **false**, *e.g.*

```
int x = 0;
while ( x+2 > -1 ) { print(x); x = x-1; };
```

The result from this is 0, -1, -2 because **x** is evaluated like

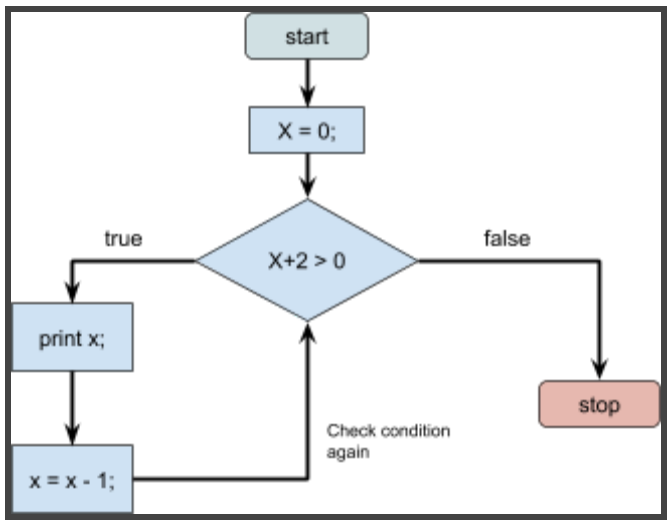

Figure 2 : The work of while loop.

This loop ends when (x (that is -2 in 3th checking-time) + 2) is equal with 0 .

6. Function call:

- **Abstract**
  progL's function uses **Copying Mechanism** and **Dynamic Binding**.

- **Declaration**
  The example of declaring a function along with execution result is the following.

```
int x =10;
function f(int x){
    function a(int x){ print("aF"); };
    x = 30;
    a();
    print(x); };
function g(int x){
    function a(int x){ print("aG"); };
    x = 20;
    f();          // output is aF, 30.
    a();          // output is aG.
```

```
    print(x); }; // output is 20.
g(); print(x);   // after print output from g(), 10 will be printed.
```

- **Invoking**
  When the function is invoked, these are the step of what is happening.
  1. Create a new runtime environment and push it to the memory stack.
  2. If the parameter is needed, declare a variable of parameter name and its type.
  3. If an argument is passed, assign the value of the passed argument to the parameter.
  4. Run the statements.
  5. When all of the statements are executed, destroy the created environment.

- **Default Parameter**

```
function f(int x = 20){
      print(x);
};
f(10); // print 10
f();   // print 20
```

- **Return**

```
int x;
function f(){
    return 1+1;
};
x = f();
print(x); // print 2
```

Memory and Environment -

In progL, the Memory which contains a stack of environment. The environment will be created and push into the stack every time a function is invoked (refer to *Programing Elements progL - 6. function call/invoking* section).

Every time the progL is initiated, it will create the first environment and push into the stack to use as the environment for the main section.

An Environment consists of a hashtable for variable and a hashtable for a function call, which means that a variable and a function can share the same name. This also means that declaring both variables and functions inside the function environment is possible.