CS 315 Theory of Automata
Final Project

Deadline: 28th April 2019 11:55 pm

All of us have experienced a situation as CS majors when we liked or disliked a certain aspect of a given programming language, and asked ourselves “Wouldn't it have been much easier if I could write this statement in this or that way?”. Well, you can’t do much about a language that you did not design.

What if we give you the opportunity to get rid of this limitation? What if we give you the opportunity to design a programming language with features and syntax of your own choice. That language is called YAPL (Yet Another Programming Language). This projects affords each one of you the opportunity to design your own YAPL. Your personal YAPL will be a YAPL with some portion of your name appended to it as prefix or postfix. For example, if your name is Inam, your YAPL can be called as YAPLI -:). Your YAPL should neither match any one of the mainstream programming languages (c, java, python etc...) nor to the YAPL of any other class participant, and must implement the following functionality. Sample test cases of YAPLAH (keep guessing who designed this language) are given in the end. Once you implement your own YAPL, you have to port these test cases of YAPLAH to your language. The TAs will use your provided test cases to evaluate your project.

- Implement a lexer, parser, and interpreter of your YAPL in PLY library (Use ply.lex for tokenizing and ply.yacc for parsing)
- Your code should be able to handle any code which uses functionalities specified in
- the tasks.
- You should implement maximum error handling that a language provides.
- You should upload your code along with the test cases on LMS
- Timely uploading to LMS is YOUR RESPONSIBILITY. Do not wait for the last moment. We will only accept LMS submitted assignments and directly run the test cases on our machine
- Plagiarism will be detected using multiple tools and forwarded directly to the DC

Tasks:

1. Variables (1 pts)
    a. Declaration, assignment, access
    b. Types: int, double, char, strings, bool
    c. Initialisation and declaration of a variable with the name of a pre existing variable should generate error

2. Expressions (2 pts)
    a. Operators: (+ , - , / , * , % , ++, --, nested parentheses)

3. List (2 pts)
    a. Declaration, assignment, access
    b. Types: int, double, char, strings, bool
    c. Access outside the list-size should also give an error

4. If-elseif-else statements (1 pts)
    a. Can be nested
    b. Can use int, double, char, bool and string types for logical expression
    c. Can be an if statement, or just an if-elseif-elseif-...else or if-elseif or an if-else
    d. Operators (<, >, <=,>=, !=, ==)

5. For loops (2 pts)
    a. Can be nested

6. Standard output (1 pts)
    a. Any variable-type should be displayed

7. Recursion (3 pts)
    a. You must handle the ‘scope’ for this b. You’ll need to handle functions for this too.

8. Error Handling (2pts)
    a. Mention the line number
    b. State the reason for the error

9. Structs (4 pts)
    a. Create a struct
    b. Create a variable of type struct
    c. Access the variables declared in the struct

10. Do-while loops (2 pts)
    a. Can be nested

Test Cases of YAPLAH:

Variable Declaration:

```
var a string = "initial";
var b, c int = 1, 2;
var d = true;
var e int;
f := "short";
```

If condition and for loop:

```
if (++i >= 2){
      for (j:=3; j>0; --j){
            i++;
            if (i>4){
                  break;
            }
      }
} else {
      Printl ("tough luck" )//print in one line
}
```

Function:

```
func plus (a int, b int)  int {
      return a+b;
}

func plusPlus(a,b,c int) int {
      return a+b+c;
}

func main(){
      res:= plus(1,2);
      Printl("1+2=", res);
      res=plusPlus(1,2,3);
      Printl("1+2+3 =", res);
}
```

Recursion:

```
func fun(a int, b int) int
{
   if (b == 0)
       return 0;
   if (b % 2 == 0)
       return fun(a+a, b/2);

   return fun(a+a, b/2) + a;
}
```

Struct:

```
struct temp_name{
      var a int;
}
var a_struct temp_name;
a_stuct.a = 5;
Printl(a_struct.a)
```

Expected Output:

5

Do While:

```
do{
      Any Statement
}
while( //Expression which gives a true or false )

var index int = 0;
do{
      Printl(index);
      index++;
}while(index < 5)
```

Expected Output:

```
      0
      1
      2
```

3
4

List:

```
var temp list(int) = [1,2,3,4,5];
Printl(temp)
```

Expected output:

```
1
2
3
4
5
```

list.pop(0) // Removes the head of the list (1 in this case)

list.pop(1) // Removes the tail of the list (5 in this case)

list.push(Number to insert) // Adds the number at the end of the list

list.index(index) // returns value at that index

list.slice(start,end)

Example code:

```
var temp list(int) = [1,2,3,4,5,6];
var temp2 list(int);
temp2 = temp.slice(0,2);
Printl(temp2);
```

Expected output:

```
1
2
3
```