Compiler Design Project
Phase-2

Language "X"

By-
Sourabh Rohit (201551065)
Mehak Piplani (201551072)
Ujwal Tewari (201551085)
Neelansh Sahai (201551086)

Contents-

Intro to the Language-

Our language namely X is an fiscal combination of a predominant part of C grammar and syntax and some parts and function definition of python while at the same time differs from each of their syntax in many places to simplify writing programs.

The language works in two parts which are namely-

1) Prettify- Proper indentation of the code.
2) Compile and execution.

The language serves to indent the given code as per norms given by language X which mostly works on brackets and looping statements along with every variable declaration and initialization in separate lines.
After the indentation has been done the compilation followed by execution of the code given and supplied to the user takes place.

Basic Idea-

To list out the features of X:

a. **Keywords** -

if, for, funk, end, else, break, return, continue, while, do, int, float, char, string, boolean.

b. **Recursion** -

in the functions is recognized.

c. **Declaration of Variable**-

The type of the variable can be mentioned while declaration.

d. **Arrays are supported**-

Also, the default values can be initialized at the time of declaration of an array.

Things the language doesn't support:-

a. **Pointers** are not supported by this language.
b. **Dynamic Memory allocation** not supported.

Compile and Run-

_____

On getting compiled our compiler will first of all **“Prettify”** or in basic terms shall indent the code properly and rearrange it as per norms and required indentation.
We use braces as our means of indentation.
The compiler checks for braces and multi variable declarations in the same lines and the accordingly indents them. The indenation works for nested loops and iterations also.
Initialisations and declaration the have been supplied by the user in single line or in haphazard manner are properly indented in a new file or in the same file as per the choice of the user.
After the the new files is created , it’s used for further compilation and execution by the compiler.
The code in the new file is checked for proper grammar and errors and accordingly reports back after the execution ends.

Grammar for our language-

The starting variable for our language is CODE

<CODE> ⇒ <DECLARATION>
| <DECLARATION> <CODE>

<DECLARATION> ⇒ <FUNCTION_DEFINITION>
| <VARIABLE_DECLARATION>
| <STATEMENTS>

<FUNCTION_DEFINITION>
⇒ funk <TYPE> 'function_name' '('
<PARAMETER>')' '{' NEWLINE
<STATEMENTS>
'}'

<PARAMETER> ⇒ <id_list>
| ϵ

<id_list> ⇒ <TYPE> id
| <TYPE> id ',' <id_list>

<VARIABLE_DECLARATION>
⇒ <TYPE> <DECLARATION_LIST>
| <DECLARATION_LIST>

<VARIABLE_DECLARATION_STATEMENT>

⇒ <VARIABLE_DECLARATION> ‘,’

<DECLARATION_LIST> ⇒ <VARIABLE>
| <VARIABLE> ‘,’<DECLARATION_LIST>

<VARIABLE> ⇒ Assignment_expression
| <EXPRESSION>
| id[size]
| id
| id[size][size]

<TYPE> ⇒ int
| float
| char
| string
| Boolean
| long

<EXPRESSION> ⇒ <EXPRESSION> <Opr> <EXPRESSION>
| (<EXPRESSION>)
|< Relational_statement>
| <function_call>
| id
| size
|<UNARY_OPERATION>

<identifier> ⇒ id //This refers to the variable names
| <STRING> // this is for all kinds of strings
|<CONSTANT> // this is for constant values like in C language
|<const> // these are basically all types of numbers and size includes only Positive integers

<STATEMENTS> ⇒ <LOOPING_STATEMENTS>
| <CONDITIONAL_STATEMENTS>
| <JUMP>
| <STATEMENTS> <STATEMENTS>
| <EXPRESSION_STATEMENT>
|
<VARIABLE_DECLARATION_STATEMENT>
|<Input_statement>
|<Output_statement>
|< function_call> ';' NEWLINE

<Assignment_expression> ⇒ id = <EXPRESSION>

<EXPRESSION_STATEMENT> ⇒ <EXPRESSION> ';' NEWLINE

|<Assignment_expression>';' NEWLINE

<function_call> ⇒ function_name (pass_parameter);

<pass_parameter> ⇒ <identifier>
| <identifier> ','< pass_parameter>

<CONDITIONAL_STATEMENTS> ⇒ if '(' <EXPRESSION> ')' ‘{‘ NEWLINE

<STATEMENTS> ‘}’ NEWLINE

| if '(' <EXPRESSION> ')' ’{‘ NEWLINE
<STATEMENTS> ‘}’ NEWLINE
else ‘{’ NEWLINE
<STATEMENTS> ‘}' NEWLINE

| if '(' <EXPRESSION> ')' ‘{‘ NEWLINE
<STATEMENTS> ‘}’NEWLINE
elseif '(' <EXPRESSION> ')' ’
{‘ NEWLINE
<STATEMENTS> ’}’NEWLINE
else ‘{‘ NEWLINE
<STATEMENTS> ‘}’NEWLINE

<LOOPING_STATEMENTS> ⇒ while ‘(‘ <EXPRESSION>’)’ ‘{‘ NEWLINE
<STATEMENTS>
‘}’NEWLINE

| for (<VARIABLE_DECLARATION> ;
< EXPRESSION>;
<EXPRESSION>)
‘{‘NEWLINE
<STATEMENTS> ‘}’ NEWLINE

<JUMP> ⇒ continue ; NEWLINE
|break; NEWLINE
|return ; NEWLINE
|return <EXPRESSION>; NEWLINE

<UNARY_OPERATION> ⇒ <identifier> < unary_operator>

<Opr> ⇒ +
| *
| /
| %
| -

<unary_operator> ⇒ ++
| --

<relational_opr> ⇒ >=
| <=
| >
| <
| !=
| ==
| && (... Logical AND)
| || (...Logical OR)

<Relational_statement> ⇒ <EXPRESSION> <Relational_list>
| <function_call>< Relational_list>

<Relational_list> ⇒
<relational_opr> <EXPRESSION>
| <relational_opr>< function_call>

| <relational_opr>< Relational_list>

<Input_statement> ⇒ ‘scan’ ‘(’ <pass_parameter> ‘)
‘;’NEWLINE

<Output_statement> ⇒ ‘print’ ‘(‘ <EXPRESSION> ‘)’
‘;’NEWLINE

Coding and Implementation-

The Graphical User interface of "Language X" goes as visible in the following snapshots-

The initial screen offering the choices.

1) **Prettify-**

On having entered the following code-

```
#include<stdio.h>
#include<string.h>
void main()
{
char c[100];
scanf("%s",&c);//taking input of the string
int a=10;int b=10; if(a>b){a=9;}
int flag=0;
c[strlen(c)+1]=' ';//adding space to the string for string termination and eo-string checking
//printf("%s \n",c);
int i=0,j=0,k=0;
        for(i=0;i<strlen(c);i++)
    {
        char ch = c[i];//checking if string begins with if or not
         if(ch=='i' && c[i+1]=='f')
           {
                printf("< 1, 1 > \n");
                flag=1;//if if found then flag set to 1
                i++;
                continue;
           }
        if(flag==1)//if begins with if,process is continued,checked for flag condition
         {
                //checking for the symbolic conditions
           if(ch == ';')
               printf(";      < 4, 4 > \n");
           else if(ch == '(')
               printf("(      < 5, 0 > \n");
           else if(ch == ')')
               printf(")      < 5, 1 > \n");
               else if(ch == '{')
               printf("{      < 6, 0 > \n");
         else if(ch == '}')
               printf("}      < 6, 1 > \n");
         else if(ch == '!' && c[i+1] == '=')
               {printf("!=    < 330, 330 > \n");i++;}
         else if(ch == '!')
               printf("!      < 33, 33 > \n");
         else if(ch == '<' && c[i+1] == '=')
               { printf("<=   < 600, 600 > \n");i++;}
         else if(ch == '>' && c[i+1] == '=')
               {printf(">=    < 620, 620 > \n");i++;}
         else if(ch == '<')
               printf("<      < 62, 62 > \n");
         else if(ch == '>')
               printf(">      < 60, 60 > \n");
         else if(ch == '=' && c[i+1] == '=')
               {printf("==    < 610, 610 > \n");i++;}
         else if(ch == '=')
```

```
                printf("=      < 61, 61 > \n");
          else if(ch>='A' && ch<='z')
            {
            //checking for alpha numeric constants
char rh = c[i];//character for alpha-NUMERIC
                        for(k=i;k<strlen(c);k++)
                        {rh = c[k];
                               //for multi-character-variable/numbers

if(rh=='<'||rh=='>'||rh=='='||rh==';'||rh=='!'||rh=='('||rh==')'||rh=='{'||rh=='}')
                        {i=k-1;break;}
                        else{
                        printf("%c",rh);
                        }
                        }
                 printf("      < 2, #%d > \n", rand());//generating random
            }
          }
     }
}
```

After going through Prettify we have the following output-

```
#include<stdio.h>
#include<string.h>
void main()
{
    char c[100];
    scanf("%s",&c);
   //taking  input of the string
   int a=10;
   int b=10;
    if(a>b)
   {
            a=9;

   }
    int flag=0;
    c[strlen(c)+1]='  ';
   //adding space to the string for string termination and eo-string checking
   //printf("%s  \n",c);

   int i=0,j=0,k=0;
    for(i=0;i<strlen(c);i++)
   {
             char ch = c[i];
            //checking  if string begins with if or not
            if(ch=='i'  && c[i+1]=='f')
```

```
{
                printf("<  1, 1 > \n");

               flag=1;
               //if  if found then flag set to 1
               i++;
                continue;

}
 if(flag==1)//if  begins with if,process is continued,checked for flag condition

{
                //checking  for the symbolic conditions
               if(ch == ';')
               printf(";                    < 4, 4 > \n");

               else if(ch == '(')
               printf("(                    < 5, 0 > \n");

               else if(ch == ')')
               printf(")                    < 5, 1 > \n");

               else if(ch == '{')
               printf("{                    < 6, 0 > \n");

               else if(ch == '}')
               printf("}                    < 6, 1 > \n");

               else if(ch == '!'  && c[i+1] == '=')

               {
                              printf("!=                   < 330, 330 > \n");
                              i++;

               }

               else if(ch == '!')
               printf("!                    < 33, 33 > \n");

               else if(ch == '<'  && c[i+1] == '=')

               {
                               printf("<=                   < 600, 600 > \n");
                              i++;

               }

               else if(ch == '>'  && c[i+1] == '=')

               {
                              printf(">=                   < 620, 620 > \n");
                              i++;

               }

               else if(ch == '<')
```

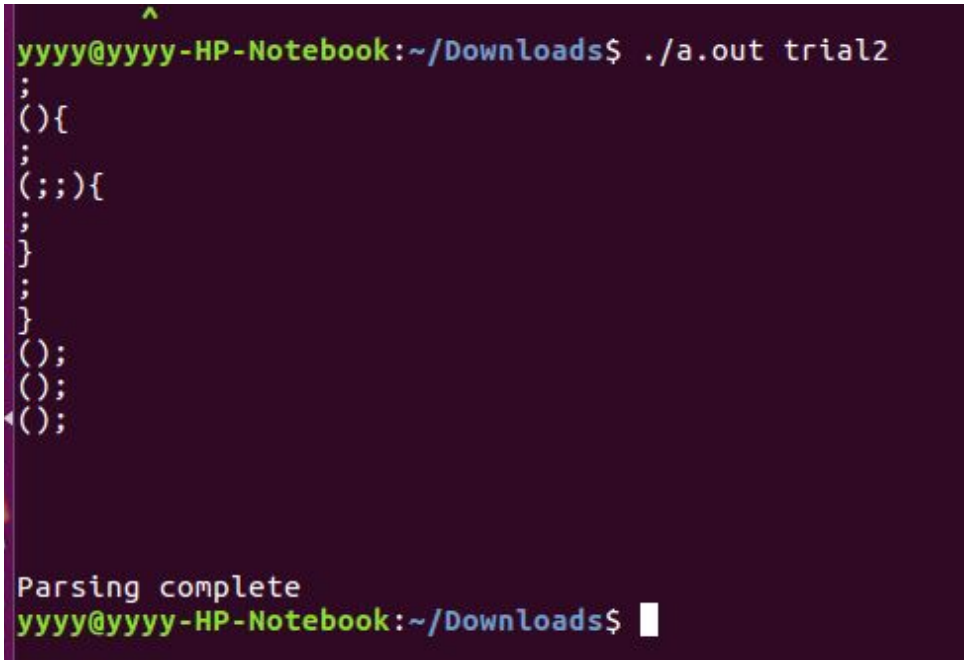
```
                printf("<                < 62, 62 > \n");

                else if(ch == '>')
                printf(">                < 60, 60 > \n");

                else if(ch == '='  && c[i+1] == '=')

                {
                        printf("==              < 610, 610 > \n");
                        i++;

                }

                else if(ch == '=')
                printf("=                < 61, 61 > \n");

                else if(ch>='A'  && ch<='z')

                {
                         //checking  for alpha numeric constants
                        char rh = c[i];
                        //character  for alpha-NUMERIC
                        for(k=i;k<strlen(c);k++)
                        {
                                rh = c[k];
                                 //for  multi-character-variable/numbers
                                if(rh=='<'||rh=='>'||rh=='='||rh==';'||rh=='!'||rh=='('||rh==')'||rh=='{'||rh=='}')

                                {
                                        i=k-1;
                                        break;

                                }
                                 else
                                {
                                         printf("%c",rh);


                                }

                        }
                         printf("    < 2, #%d > \n", rand());
                        //generating random

                }

        }

   }

}
```

Example Codes-

1) **Finding the factorial of a number-**

```
int  a = 10;
funk int factorial(int a){
   int q=1;
   for ( int i=1; i <= a; i ++){
            q=q*i;
      }
  return q;
  }
 int s = factorial(a);
 print(s);
```

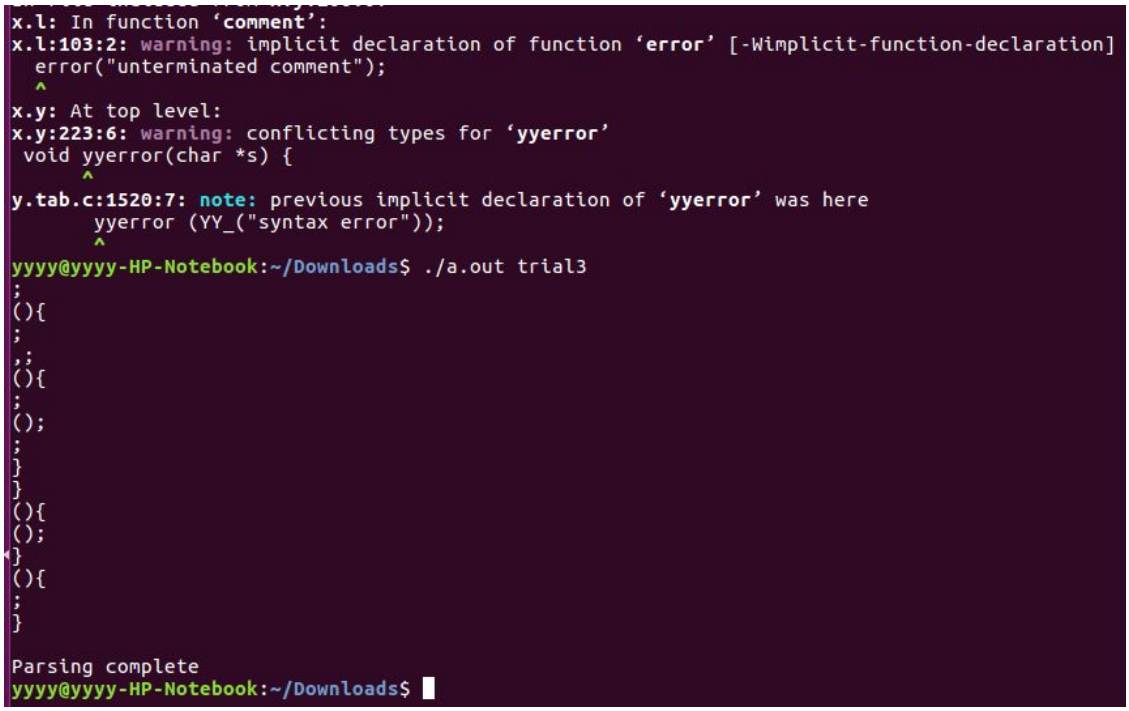
OUTPUT:



2) Finding Armstrong number-

```
 int  a = 10;
funk int armstrong(int a){
  int m=a;
     Int d=0,s=0;
     while(m>0){
                    d=m%10;
                    s=s+cube(d)
                    m=m/10;
                      }
     }
     if(s==a){
                    print(a);
            }

     funk int cube(m){
                    return (m*m*m);
     }
```

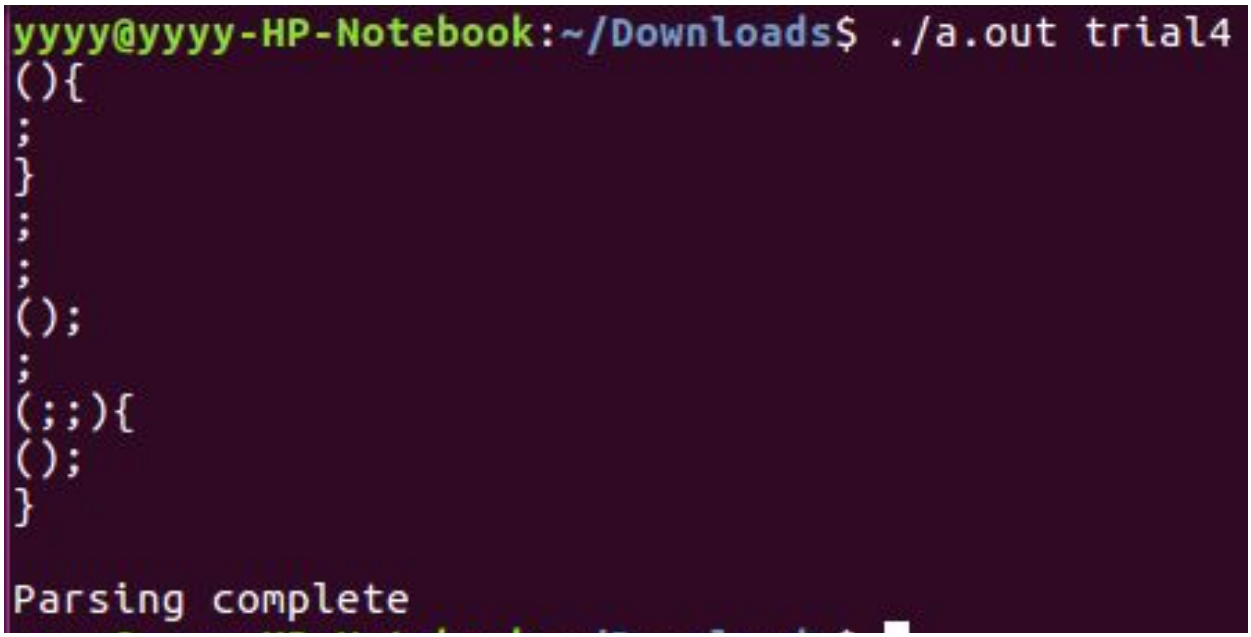
OUTPUT:



3) Finding the sum of cubes for number range one to n-

```
funk int cube(int a) {
                return a*a*a;
}
int b = 1;
int n;
scan(n);
int sum =0;
for(int i=0; i<=n; i++) {
                        sum = sum + cube(i);
}
print(sum);
```

OUTPUT:


yyyy@yyyy-HP-Notebook:~/Downloads$ ./a.out trial4
(){
;
}
;
;
();
;
(;;){
();
}
Parsing complete

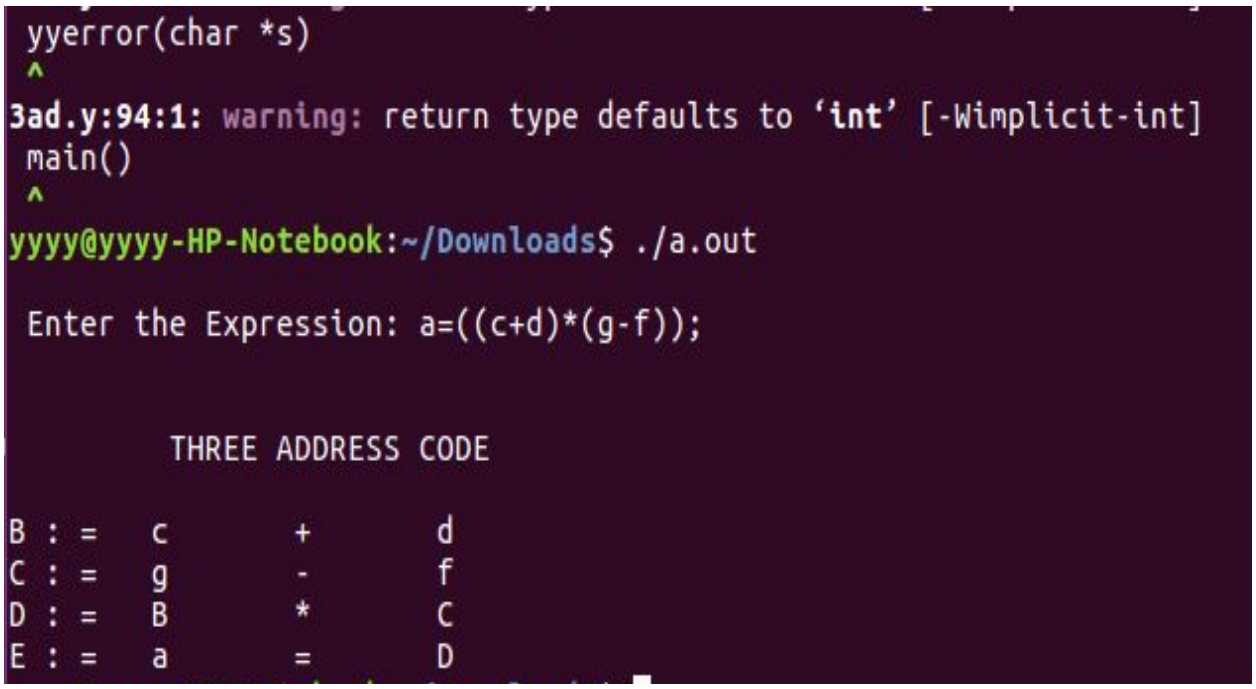
_____

_____

_____

ADDRESS CODE FOR ARITHMETIC OPERATIONS-

We have implemented the three address code for a normal arithmetic operation keeping in mind the precedence.
This has been implemented in lex and yacc as a separate code using the concept like that of symbol table and using a structure in c to store all the parameters of a arithmetic expression.



ADDRESS CODE FOR IF ELSE-

We have also tried to implement the three address code for a normal if else statement.
This has been implemented in lex and yacc as a separate code using the concept like that of symbol table and using a structure in C to store all the parameters of a arithmetic expression and then to give the GOTO statements function of name label1 , label2 and label3 have been used.

```
yyyy@yyyy-HP-Notebook:~/Downloads$ ./a.out

 Enter the Expression: if(c<d) then f+9; else a=c;


         THREE ADDRESS CODE

B : =   c       <       d
if not B  GOTO L
C : =   f       +       9
GOTO L1
L :
D : =   a       =       c
L1 :
```