FORMAL LANGUAGES AND COMPILERS PROJECT
Report

Coppe Antonio - Fiore Matteo - Sciortino Giulio

General Explanation and Project Structure

In this project we implemented a scientific calculator with many advanced functions.

The goal of our project is to perform, besides the classical operations(+,-,*,/), advanced mathematical functions in the simplest way possible; such as sigma function, sieve of eratosthenes, prime factorisation and many others. These types of functions have never been implemented in a calculator and since many of them are very useful in many fields such as algorithmic complexity, number theory, linear algebra etc. we decided to include them.

For the general structure of the project we used the definition 'expression' to describe inputs. Expression is type *struct* composed of a *struct* num in which the input types will be specified, i.e. int and float and a char type. Our program gives the opportunity to the user to declare variables of the previously mentioned types. Declared variables are saved in symbol tables, a linked list where we store the corresponding variable names and values. The symbol table has the following structure:

```
struct symbolTable
{
    char *name;
    struct Number v;
    char type;
    struct symbolTable *next;

};
```

As previously stated, each variable can be distinguished as "int" or "float". In this way, we are able to perform different computations according to the selected type.
Additionally, we defined a %union. Union identifies the yacc value stack as the union of the various types of values desired.
The union is composed of:

- char* lex; → used as a name for the variable
- struct expr; (see above explanation)
- int intVal; → to recognize input integer (the token integer has as attribute intVal)
- float floatVal; → to recognize input float (the token float has as attribute floatVal)

In the snapshot below, you can see how it is implemented in the yacc file.

```
%union {
    char* lex;
    struct Expr
        {
        struct Number v;
        char type;
    } expr;
    int intVal;
    float floatVal;

}
```

Type Checking

Some operations have some constraints in terms of data types. Therefore, we implemented a method called typeConsensus, which takes two chars as inputs (thus specifying the type of the input i.e. int and float) and returns a char. In this method we verify and change the behaviour of the functions according to the data type we pass as inputs. More generally, if both variables will be integers the operations will be done only on integers, otherwise if one input is a float or both inputs are floats we will return a float value. Additionally, if one input is an int and the other is a float a custom message is printed warning the user that the operations will be done with two different data types. In case none of the inputs are neither integers nor floats an error is printed.

```
//this function is used to find the type of the result of an operation
char typeConsensus(char type1, char type2){
    //when inputs are of type int, result will be int
    if(type1==type2 && type2=='i'){
        return 'i';
    }
    //same with float
    if(type1==type2 && type2=='f'){
        return 'f';
    }
    //if types are different, a choice has to be made. We decided to return a float with a warning message
    //So when a user writes ,for example, 7+3.0, result will be treated as a float (and a proper warning is printed)
    else if(type1!=type2 && (type1=='f'||type1=='i')&&(type2=='f'||type2=='i')){
        printf("\nWarning: you are operating with two different types. Result will be cast to float.\n\n");
        return 'f';
    }
    //if it's not possible to find a consensus, then 'e' is returned (stands for error)
    //this happens when the type of an input is not defined
    return 'e';
}
```

This method is called whenever we declare an operation in the yacc file. This helps us to modify (i.e. use casting) the data types of the function inputs or outputs according to its needs.

Take this snapshot as an example:

```
                                    else  $$.v.i = (int) eratosthenes($3.v.i); }
| BIN '(' expression ',' expression ')'     {$$.type=typeConsensus($3.type,$5.type);
                                     if($$.type=='f')printf("Please insert an integer and not a float for the binomial function \n");
                                    else  $$.v.i = (int) binomial($3.v.i,$5.v.i); }
```

In this case, we are analysing the case of the Binomial function. This function accepts integers only, therefore we call the method typeChecking and if any of those parameters is a float ($$.type == ‘f’) an error is printed. Otherwise the binomial function is called with two integers parameter and the result will be an integer too.

Installation

To run the compiler we have to access the “megacalculator” folder. Now there are two possible ways to interact with the compiler:

1) The user can execute a make file, called correctRun.mk. To do so, type in the terminal the following command:

   make -f correctRun.mk

   This will output the correctOutput.txt file in the folder, with the results of all the computations present in correctInput.txt.

   If the user also wants to see how our megacalculator responds to errors in the input; it is possible to run another make file, called wrongRun.mk.

   In the terminal type the following command:

   make -f wrongRun.mk.

   This will print in the wrongOutput.txt file in the folder, the errors and/or operations results generated by the input file.

2) To run the project via terminal the user must execute the following commands(macOS, linux):

1. lex lex.l
2. yacc -vd yacc.y
3. gcc -g y.tab.c -ly -ll -lm
4. ./a.out

Then the user can execute the multiple operations that our megacalulator offers.

GRAMMAR

Scope: startProgram

List of Terminals/ Tokens:
{ID, POW (“POW”), LOG(“LOG”), FIB(“FIB”), FACT(“FACT”) ,SUM(“+”), PROD(“*”), ABS(“ABS”), PI(“PI”), SMALLER(“<”), GREATER(“>”), EQUAL(“==”),DIFFERENT(“!=”), SMALLEREQUAL(“<=”), GREATEREQUAL(“>=”), RAND(“RAND”), SIGMA(“SIGMA”), ERA(“ERA”), PRIME(“PRIME”), PRIMF(“PRIMF”), GCD(“GCD”), AVG(“AVG”), CEIL(“CEIL”), FLOOR(“FLOOR”), FOR(“FOR”), WHILE(“WHILE”), IF(“IF”), ELSE(“ELSE”), OR(“OR”), INC(“++”), DEC(“--”), AND(“AND”), EXIT(“EXIT”), BIN(“BIN”))}

List of non-terminals
startProgram, op, block, cond, scond, nid, logop, relop, expression

Productions

startProgram	-> op | ε
op	-> expression | block | ε
block	-> WHILE (COND) op | IF (COND) op | IF (COND) op ELSE op | ε
cond	-> scond | scop logop cond | ε
scond	-> nid relop nid
nid	-> ID | REAL
logop	-> AND | OR
relop	-> DIFFERENT | EQUAL | SMALLEREQUAL | SMALLER| GREATER | GREATEREQUAL
expression	-> (expression) | PI | ID | REAL | INTEGER | expression + expression | expression - expression | expression * expression | expression / expression | expression ^ expression | - expression | ID INC | ID DEC | expression INC | expression INC | expression DEC | ABS (expression) | FIB (expression) | SIGMA(expression,expression)| GCD(expression,expression) | AVG(expression,expression) | LOGN (expression) | CEIL (expression) | FLOOR (expression) | ERA (expression)| BIN(expression,expression)| BIN (expression,expression) | RAND(expression,expression,expression)| PRIME(expression) | PRIMF(expression)| ROOT(expression)

INPUT DESCRIPTION

Megacalculator Abilities

a=n	Declaration of a variable
a++	Increment of a variable
a--	Decrement of a variable
n = n	Check if variables are equal

POW	Returns power of a number
ROOT	Returns square root of a number
LOGN	Returns logarithm of a number in base e
FIB	Calculate fibonacci sequence of a number
!	Returns factorial of a number
+	Returns sum of two numbers
*	Returns product of two numbers
ABS	Returns the absolute value of a number
RAND	Returns n random integers between two numbers
SIGMA	Returns the sigma function of a number
ERA	Returns the sieve of eratosthenes
PRIMF	Returns the prime factors of a number
GCD	Calculate greatest common divisor of two numbers
AVG	Returns average between two values
CEIL	Returns ceil of a number
FLOOR	Returns floor of a number
FOR	For loop
WHILE	While loop
IF	If statement
ELSE	Else statement
OR	Or condition
AND	And condition
BIN	Returns binomial coefficient
EXIT	Exits from the program

Examples

Declare a variable: a = 3

Decrement a variable: a--

Check equals: 6 == 6

Other mathematical functions:

LOGN(10)

ROOT(100)

The power of a number can be written as 2 ^ 3 **and also** POWER(2,3)

POWER10(4)

Calculate the factorial: 3!

BINARY(7)

DECIMAL(111)

While loop:

c=7

WHILE (5 > 2) c--

Exit the application: EXIT