Final Project Report

Thogata Jagadeesh
112001045

Project folder contents :

- **Sample tests(sub folder)**
  - **test1.txt**
  - **test2.txt**
  - **test3.txt**
- **compiler.y (yacc file)**
- **compiler.l (lex file)**
- **declarations.h**
- **definitions.h**
- **parsetree.h**
- **variables.h**
- **input.txt**
- **CHANGELOG.md**
- **Makefile**
- **compiler(executable) generated after compiling the programme.**
- **Output.c (generated code)**
- **Output (.c executable generated after compiling the c programme)**

grammar.y

1) Including required header files
2) Functions declaration
3) tokens declarations.
4) Grammar for the given programme.
   Grammar accepts all type of read write function calls (write x ; write a[i])
   All types of Assignment statements.
   Conditional statements like If else , if ,while (loop)
   All the arithmetic ,logical statements (like a=b+c, a=b[c]+d,...etc)
   For each rule in the grammar I did the corresponding actions like calling the function **makeNode()** to create a syntax tree and perform basic syntax and semantic analysis.

tokens.l

In the tokens.l i didn't change anything much. I just wrote the return tokens statements for each regular expression. like for while returning WHILE .. etc.

Declarations.h

In this file I declared the symbol table pointers and the treenode pointer and the global variables like lineCount,etc.

Definitions.h

In this file I defined the symbol table structure and Tree node structures.

****Symbol Table****

```
struct Symbol {
        char *Name;   //name of the symbol
        int Type, Size, Binding;
//size of the symbol,type of the symbol, Binding of the symbol
        struct Symbol *next, *parent, *ArgList, *Scope;
```

//Doubly Linked list , **next** pointer contains the address of next entry and **parent** pointer contains the address of previous entry. Remaining fields I declared for future works.

```
};
```

****TreeNode****

```
struct Node {
        int value, type;
        struct Node *t1, *t2, *t3;    //node pointers for the child node pointers.
        struct Symbol *g, *h;        // storing the information about the Tree node
        struct Parameters *P;       //this stores the parameter values
};
```

variables.h

This file Contains definition of the following Functions

1) **NewScope ()** This function is for binding the scope of the variable (Not required for this test ,I Included it this Since I used some of my lab2 functions.)
2) Install() this function adds a symbol to the symbol table.
3) install Function() this function creates separate symbol table for the given function ,scope ..ectc
4) Lookup () this function looks up a symbol in the symbol table.
5) CheckPar(),InstallFunctions().

parsetree.h

This file contains the **MakeNode()** function. This Function is very important Since it is used for Syntax Tree printing and **semantic analysis**. like
1.array index out of bounds .
2.usage of variables without declaration..etc.
and the **syntax analysis** like :
1.Usage Of Arithmetic Expression for Logical Operation is prohibited.
2.Assignment of one type of variable to another type of variable is not allowed.

Makefile

- make all --> compiles all the lex and yacc files and all c files
- make run --> for executing the executable named compiler created in the bin folder.
- make clean --> remove all the intermediate files created.

TESTS

Test cases are there in a folder called Sample_tests.
given.txt(contains the test case given by sir) and remaining test cases are prepared by me.

USAGE

This program takes the file "input.txt" internally written and outputs the corresponding AST for the given program and also gives the semantic and syntax errors in the given program .For testing the different inputs please modify only input.txt and make all and make run then the output will be shown in the terminal.

Note :

i changed the test case 1 bin_search sil end is the token in my language in binary search function i changed the variable name.

Example 1.
Factorial_recursive.sil

```
decl
    integer factorial(integer a);
    integer no;
enddecl

integer factorial (integer a) {
    decl
        integer temp;
    enddecl
begin
    if (a == 1) then temp = 1;
    else temp = a*factorial(a-1);
    endif;
    return (temp);
end
}

integer main(){
decl
    integer temp;
enddecl
begin   write("Enter a number\n");
    read(no);
    temp = factorial (no);
    write("Factorial of");
    write(no);
    write("=");
    write(temp);
    write("\n");
    return 0;
end
}
```

```
FUNCALL VAR
FUNCALL STRING
RETURN NUM ENDMAIN
jaga@JagaTJ:~/GIT/112001045-cs3140/Project$ make exec
gcc output.c -o output
jaga@JagaTJ:~/GIT/112001045-cs3140/Project$ make runc
./output
Enter a number
4
Factorial of4
=24
```

Example 2: sum of first 50 natural numbers

Sum_recursive.sil

```
decl
    integer sum(integer a);
enddecl

integer sum (integer a) {
    decl
        integer temp;
    enddecl
begin
    if (a == 1) then temp = 1;
    else temp = a + sum(a-1);
    endif;
    return (temp);
end
}

integer main(){
decl
    integer temp;
enddecl
begin
    temp = sum (50);
    write(temp);
    return 0;
end
}
```

Output:

```
jaga@JagaTJ:~/GIT/112001045-cs3140/Project$ make run
./compiler
decl
        integer sum(integer a);
enddecl

integer sumDECL
INT FUN PARAM INT VAR
ENDDECL
 (integer a) {
        decl
                integer temp;
        enddecl
begiFUNDECL INT VAR PARAM INT VAR
DECL
INT VAR
ENDDECL
n
        if (a == 1) then temp = 1;
        else temp = a + sum(a-1);
        endif;
        return (temp);
end
}

integer mainIF EQ VAR NUM
ASSIGN VAR NUM
ELSE
ASSIGN VAR ADD VAR FUNCALL ARG SUB VAR NUM
ENDIF
RETURN VAR
ENDFUN
(){
decl
        integer temp;
enddecl
begin
        temp = sum FUN INT MAIN
DECL
INT VAR
ENDDECL
(50);
        write(temp);
        return 0;
end
}
ASSIGN VAR FUNCALL ARG NUM
FUNCALL VAR
RETURN NUM ENDMAIN
jaga@JagaTJ:~/GIT/112001045-cs3140/Project$ make exec
gcc output.c -o output
jaga@JagaTJ:~/GIT/112001045-cs3140/Project$ make runc
./output
1275
jaga@JagaTJ:~/GIT/112001045-cs3140/Project$
```