Project Report

By: Pratham Ravindra Nagpure
Roll number : 112001054
Submitted on: 6th May 2023

---

1. File structure description

- ***include/AbsSynTree.h :*** This file contains all the structs and functions required for creating the syntax tree and writing the code. The prominent structs and functions are mentioned here:
  - `struct Node` : This structure contains a union which has all the types of the nodes in it, and an attribute type which takes care which type of node is in the union. Having a single struct for node helps in the construction of the tree.
  - `struct Node* createNode(type,args)` : This function creates the node of the given type and initializes it with the args.
  - `void checkCode()` : Checks the semantic errors of the input code.
  - `void generateAST()` : Prints the abstract syntax tree.
  - `void generateCode()` : Prints the equivalent C code for the input program.
  - `void red()` : Sets the terminal color to red, used for printing errors
  - `void reset()` : Sets the terminal color back to normal
- ***src/AbsSynTree.c :*** This file contains the definitions for the functions declared in ***include/AbsSynTree.h***
- ***compiler.l :*** The *lex* file for the compiler which identifies the tokens of the language.

- ***compiler.y :*** The *yacc* file for the compiler which contains the context free grammar of the language and builds the abstract syntax tree.
- ***symbolTable.h :*** This file provides the following methods for the symbol table as follows
  - `findlst()` : Finds variable in local symbol table which is a **linked list.**
  - `insertlst()` : Inserts variable name with value in the local symbol table.
  - `freelst()`: Empties the local symbol table.
  - `findgst()` : Finds variable in global symbol table which is a **linked list.**
  - `insertgst()` : Inserts variable name with value in the global symbol table.
  - `findfst()` : Finds variable in function symbol table which is a **linked list.**
  - `insertfst()` : Inserts variable name with value in the function symbol table.
- ***symbolTable.c :*** This file contains the definitions of functions present in ***symbolTable.h***
- ***Makefile :*** Makes the compiler executable.

2. Features

All the features expected by the instructor are implemented :

+ Global declaration of arrays, function and variables
+ Local declaration of variables
+ Integer and boolean data types
+ `for and while` loops
+ `if - then - else` conditions
+ Arithmetic operators such as `+, -, *, /, %`
+ Relational operators such as `<, >, ! ==, > =, < =`
+ Assignment statements
+ Write and read Statements
+ Main function

+ No limit in the length of variable names and number of variables.
+ Nested conditions are allowed
+ Logical operators such as AND, OR, NOT

Additional features implemented by me:

+ Static Multidimensional arrays

3. Semantic Errors

These are the semantic error checks implemented by me.

1. **Incorrect number of arguments in functions**:

If the number of arguments in the declaration of the function does not match with the number of arguments in the function call or definition then the compiler catches the error and mentions the **line number** in which it occurred as shown below.

Function declaration with 2 arguments
In testcases/test2

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

Function called here with only one argument

```
15      sum(1);
```

Error given by the compiler

```
error: too less arguments for function 'sum' near line number 16
```

Similar reply will be for the more arguments case.

2. **Undeclared variables**:

If a variable is used without declaring then the compiler will catch that error and give the error mentioning the **line number**
In testcases/test3

```
5   decl enddecl
6   begin
7       temp=1;
```

Here temp is not declared but still it is used
Error given by the compiler

```
error: Use of undeclared identifier 'temp' near line number 8
```

3. **Same variable declared multiple times**:

If a variable is declared multiple times with the same name the compiler will catch the error and mentions the **line number** of the **repeated** and the **old declarations**.
In testcases/test3

```
6    decl
7    integer temp;
8    boolean temp;
9    enddecl
```

```
error: identifier 'temp' is redeclared near line number 8 which was
previously declared near line number 7
```

4. **Array dimensions of declaration and access do not match :**

   The compiler will give the error saying whether there are too many or too few dimensions and mention the **line number** where error occurred.

In testcases/test4

Array declaration with 2 dimensions

```
2    integer a[10][23];
```

Array access with 3 dimensions

```
8        a[1][3][5]=4;
```

Error given by the compiler

```
error: Too many dimensions for array refrence 'a' near line number 8
```

5. **Not a function but called as one:**

   If some variable is called like a function then the compiler will give the error mentioning the **line number**.

In testcases/test5

Here a is a variable

```
2    integer a;
```

But in the code it is called like a function.

```
8        a();
```

The compiler gives this error.

```
error: 'a' is not a function near line number 8
```