CS335A: Milestone 2

Shubham Kumar: 200966
Priyanka Meena: 200731
Maurya Jadav: 200567

Compilation and Execution instructions:

1. make symboltable < filepath { symboltable.txt will be created}
2. make run < filepath { output.txt will be created . This will check semantic error}

Symbol Table Structure and Implementation:

1. Data Structure of Symbol Table:
   vector of struct : vector<symtabentry> symbolTable
   struct symtabentry{
   string lexeme;
   string synCat;
   string dataType;
   string class_id;
   string func_id;
   vector<string> arguements;
   int lineno;
   int scope;
   };
   Functions used for structuring symbol table:
   to insert entries:

   void insert_entry(string _lexeme, string _synCat, string _dataType ,string _class_id,string _func_id, int _lineno,int _scope) ;

```
void insert_entry(string _lexeme, string _synCat, string _dataType ,string _class_id,string _func_id,int _lineno,int _scope) {
    if(!check(_lexeme,_class_id,_func_id,_scope)){
        symtabentry temp;
        temp.lexeme = _lexeme;
        temp.synCat = _synCat;
        temp.dataType = _dataType;
        temp.class_id = _class_id;
        temp.func_id=_func_id;
        temp.lineno = _lineno;
        temp.scope = _scope;
        symbolTable.push_back(temp);
    }
    else{
        cout<<_lexeme<<": redeclaration at line no:"<<yylineno<<endl;exit(30);
    }

}
```

Symantic Analysis and Type Checking Implementation:

1. **Functions to be used:**
   To get Type:
   char * getType(string _lexeme,string _class_id,string _func_id, int _scope);
   Desciption: This function returns the data type of a variable using class id , function id , scope(level)

```
char * getType(string _lexeme,string _class_id,string _func_id, int _scope){
        while(_scope>-1){
                        for(auto i:symbolTable){
                        if((i.lexeme==_lexeme&&i.class_id==_class_id&&i.scope==_scope)) { return strdup(i.dataType.c_str());}
                }
                _scope--;
        }
    cout<<_lexeme<<": this variable is not declared in this scope"<<yylineno<<endl;
    exit(31);

}
```

To check if entry is already available in symbol table or not:
bool check(string _lexeme,string _class_id,string _func_id,int _scope);
Desciption: This function check idf entry has already been inserted or not.

```
bool check(string _lexeme,string _class_id,string _func_id, int _scope){
    for(auto i:symbolTable){
        if((i.lexeme==_lexeme&&i.class_id==_class_id&&i.scope==_scope&&i.func_id==_func_id)) return true;
    }
    return false;
}
```

To check the type match between two:
bool typeCheck(string s1,string s2);
Desciption: This function checks the data type for two strings

```
bool typeCheck(string s1, string s2){
        if(s1==s2){
                return true;
        }
        return false;
}
```

To get and check the return type of the return type of a function:
char * getReturnType(string _class_id,string _func_id);
Desciption: This function return the return type of the function uding function id and classid. It is used to match the the datatype of return statement and return type present in the symbol table .

```
char * getReturnType(string _class_id,string _func_id){
        for(auto i:symbolTable){
        if((i.class_id==_class_id&&i.func_id==_func_id)) { return strdup(i.dataType.c_str());}
    }
    exit(31);

}
```