CS335A: Milestone 4

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

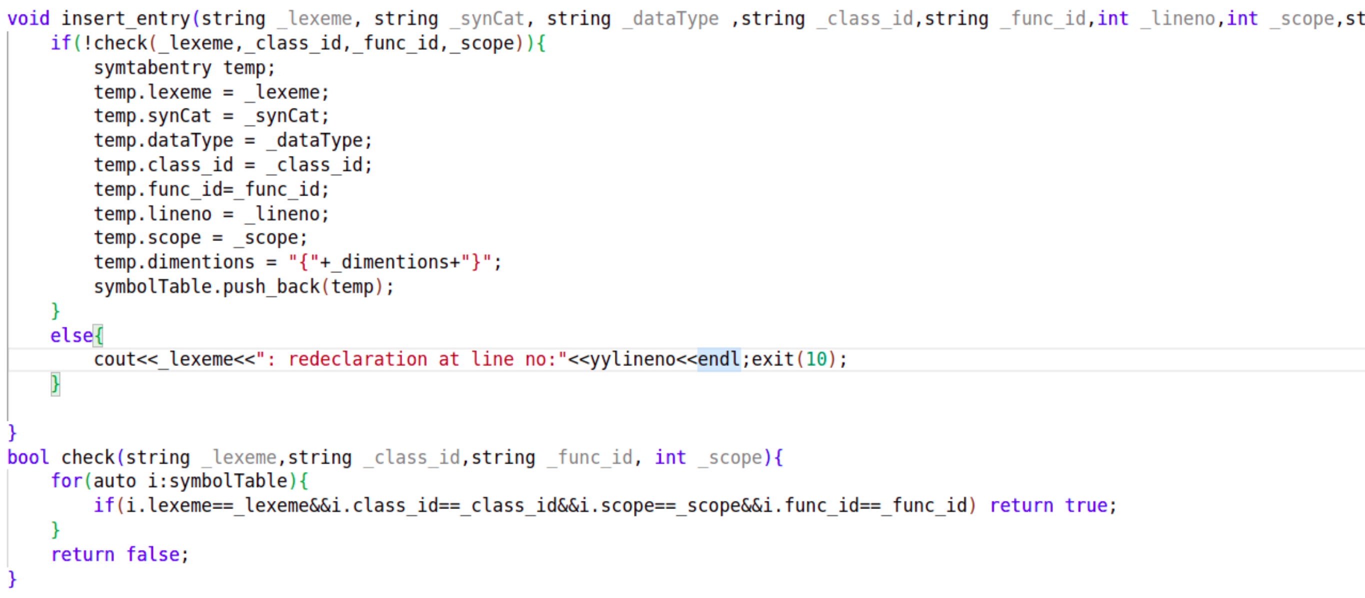
Compilation and Execution instructions:

1. python3 script.py
2. Enter the test full folder path in the prompt

Symbol Table Structure and Implementation:

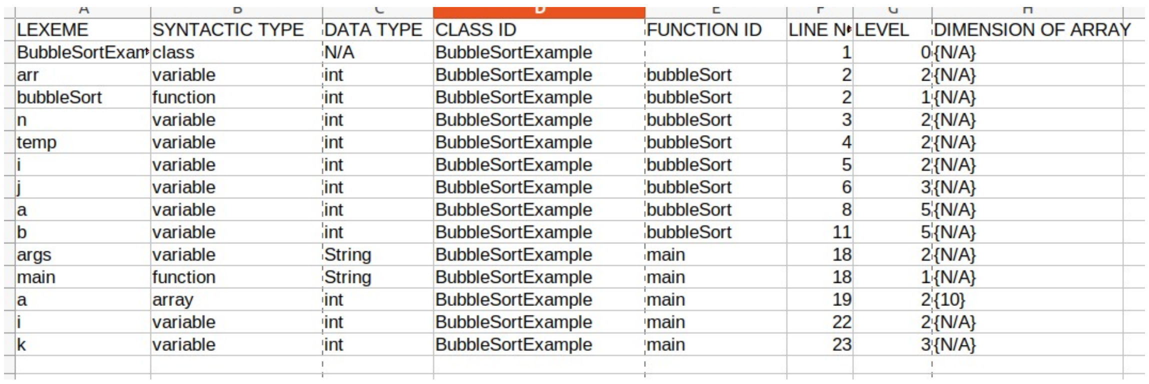
1. Data Structure of Symbol Table:
   vector of struct : vector<symtabentry> symbolTable
   struct symtabentry{
   string lexeme;
   string synCat;
   string dataType;
   string class_id;
   string func_id;
   vector<string> arguements;
   int lineno;
   int scope;
   string dimentions;
   };
   Functions 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) ;



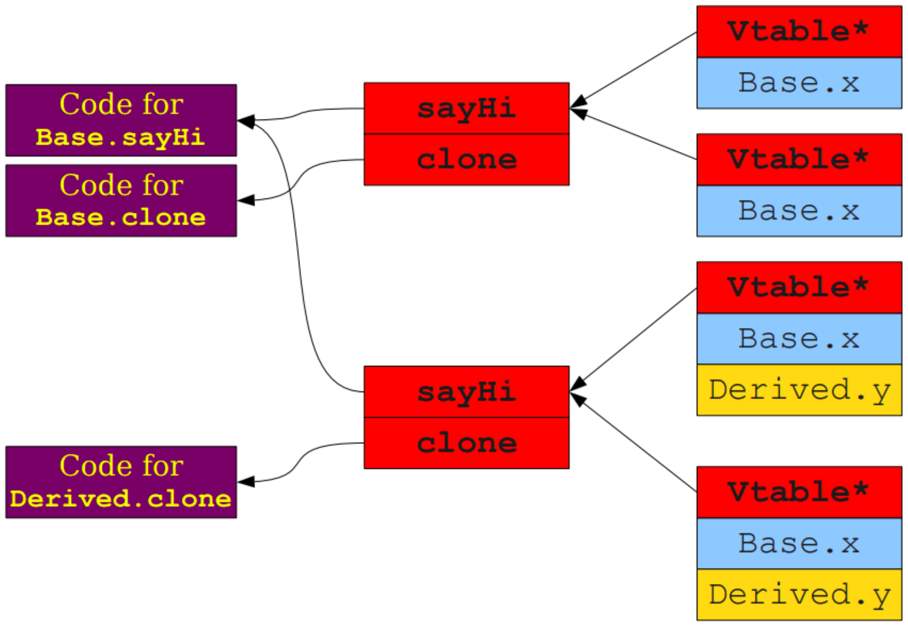
```
void insert_entry(string _lexeme, string _synCat, string _dataType ,string _class_id,string _func_id,int _lineno,int _scope,st
    if(!check(_lexeme,_class_id,_func_id,_scope)){
        symtabentry temp;
        temp.lexeme = _lexeme;
        temp.synCat = _synCat;
        temp.dataType = _dataType;
        temp.class_id = _class_id;
        temp.func_id=_func_id;
        temp.lineno = _lineno;
        temp.scope = _scope;
        temp.dimentions = "{"+_dimentions+"}";
        symbolTable.push_back(temp);
    }
    else{
        cout<<_lexeme<<": redeclaration at line no:"<<yylineno<<endl;exit(10);
    }

}
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;
}
```



LEXEME	SYNTACTIC TYPE	DATA TYPE	CLASS ID	FUNCTION ID	LINE N	LEVEL	DIMENSION OF ARRAY
BubbleSortExam	class	N/A	BubbleSortExample		1	0	{N/A}
arr	variable	int	BubbleSortExample	bubbleSort	2	2	{N/A}
bubbleSort	function	int	BubbleSortExample	bubbleSort	2	1	{N/A}
n	variable	int	BubbleSortExample	bubbleSort	3	2	{N/A}
temp	variable	int	BubbleSortExample	bubbleSort	4	2	{N/A}
i	variable	int	BubbleSortExample	bubbleSort	5	2	{N/A}
j	variable	int	BubbleSortExample	bubbleSort	6	3	{N/A}
a	variable	int	BubbleSortExample	bubbleSort	8	5	{N/A}
b	variable	int	BubbleSortExample	bubbleSort	11	5	{N/A}
args	variable	String	BubbleSortExample	main	18	2	{N/A}
main	function	String	BubbleSortExample	main	18	1	{N/A}
a	array	int	BubbleSortExample	main	19	2	{10}
i	variable	int	BubbleSortExample	main	22	2	{N/A}
k	variable	int	BubbleSortExample	main	23	3	{N/A}

3AC and Runtime support Basically we make a structure Klass which include member variables and a pointer points to its vtable
vtable contains a map which mappes the function withs its activation record
activation record contains stack for parameters, locals ,return value and a pointer named control link which points to the activation record of the caller function.

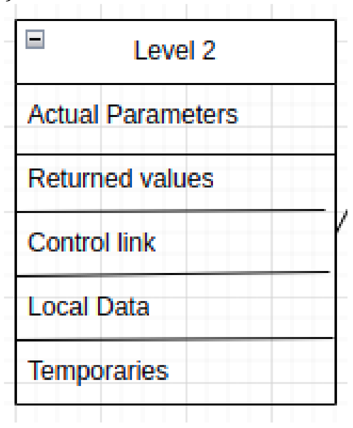


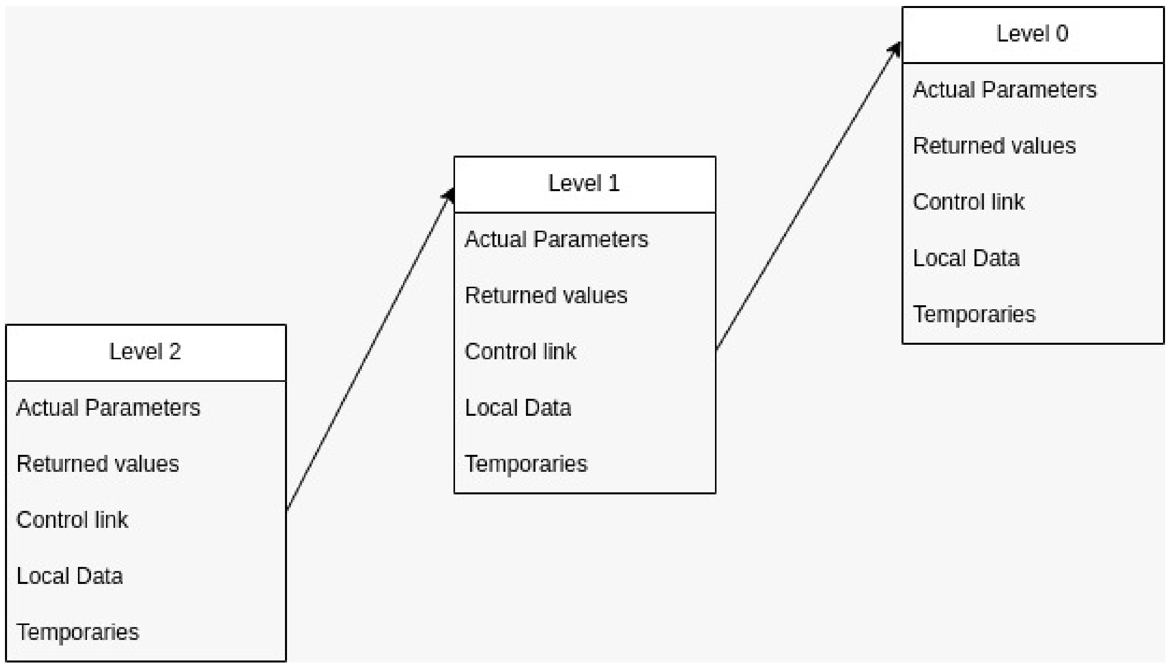
Structure of class:
struct Klass
{ stack < string > local;
vtable _vtable;
};

Structure of Vtables:
struct vtable
{ map <string,act_rec > methods;
};
A virtual function table (or vtable) is an array of pointers to the member function implementations for a particular class.

- To invoke a member function:
- Determine (statically) its index in the vtable.
- Follow the pointer at that index in the object's vtable to the code for the function.
- Invoke that function.

Structure of activation record:
struct act_rec{
stack<string > param;
int storage=0;
act_rec * controlLink;
string returnVal;
stack<string> local;
};





Code Generation

string get_label();
    this function gives the label for the assembly code
void starting_code();
    this function gives the starting code of assembly such begin .data
int is_integer(string sym);
    this function return it is integer or not
void add_op(quad* instr);
    generate the assembly code for addition operator
void sub_op(quad* instr);
    generate the assembly code for subtraction operator
void mul_op(quad* instr);
    generate the assembly code for multiplication operator
void assign_op(quad* instr);
    generate the assembly code for assignment operator
void genCode();
    this is the main function which calls other functions
void initializeRegs();
    initilializes registers
string get_mem_location(string * sym, string* sym2, int idx, int flag);
    get memory location of all type; stack, registers, heap
string getReg(string* sym, string* result, string* sym2, int idx);
    return a register if it empty else save the contents of register and then returns
void findBasicBlocks();
    find the leaders of IR and return address location of leaders
void dfs(int curr, vector<int>&visited, vector<vector<int> >&adj_list);
vector<int> findDeadCode();
    find the dead codes

Contribution Table

Sr no.	Name	Roll No.	Email Id	Contribution
1	Shubham Kumar	200966	shuhamk20@iitk.ac.in	33.3
2	Priyanka Meena	200731	priyankam20@iitk.ac.in	33.3
3	Maurya Jadav	200567	mauryaj20@iitk.ac.in	33.3