Compiler Construction

BPDC

(Lab - 07)

1 Extending the Symbol Table Implementation

In this lab assignment, we are supposed to extend our symbol table implementation (in C) that we already have, to incorporate the constraints of block structured languages. To be specific, we have to re-implement *void insert_to_table*(*char var*[30], *int type*, *bool isArray*) (additional field *isArray* is to deal with arrays) and *int lookup_in_table*(*char var*[30]) functions so as to maintain a chain of symbol tables corresponding to a sequence of nested blocks (refer Section 2.7 in the book). Further, you are expected to exploit hashing too (you could refer to any of those standard online implementations, preferably array based, to keep it simple). You could assume your input programs not to contain blocks with more than 20 variables defined within, but the number of nested blocks (and hence the length of symbol table chain) should be unbounded.

Your symbol table entry would have the following structure with at most 20 entries in one symbol table (*var_list*[20])

```
struct symbol_table{
char var_name[30];
int type;
bool isArray;//is True if this is an array and false otherwise
(additional field to handle arrays)
}var_list[20];
```

You could implement your symbol table as a singly/doubly linked linked list, where a function *create_new_table*() would always create a new table and attach it to the active one whenever there is an opening curly brace in the input. Similarly, a function *delete_table*() should delete the memory allocated to the currently active symbol table as soon as it sees a closing brace. For the time being, you could use your own main function with a template similar to the following to test your implementations. Even a menu driven main function, that would let the user to decide whether to create, delete, insert or lookup, is also fine. For testing, we would be having our own main function and would be plugging in your modules in to the same.

```
int main()
{
    int type;
    create_new_table();/*'{' seen Block B0*/
    insert_to_table("var1",0,1);/*var1 is of type 0(int) and is an array*/
    insert_to_table("var2",1,0);/*var1 is of type 1(char) and is not an array*/
    create_new_table();//'{' seen (new nested block) Block B1
    insert_to_table("var3",0,1);/*var3 is of type 0(int) and is an array*/
    insert_to_table("var4",1,0);/*var4 is of type 1(char) and is not an array*/
```

```
    if((type=lookup_in_table("var3"))!=-1)/*we are not making use of isArray
    information for the time being*/
        printf("variable var3 is of type %d",type);
    delete_table();/*a closing brace (Free up memory allotted for B1)*/
   if((type=lookup_in_table("var3"))!=-1)/*we are not making use of isArray
   information for the time being*/
        printf("variable var3 is of type %d",type);
    else
            printf("undeclared");/*should display undeclared since "var3"
            been declared inside B1*/
     delete_table();/*seen a closing brace
     (Free up memory allotted for B0)*/
}
```

Once you have all these four functions implemented, we could smoothly integrate the same to your compiler under construction.