Exercise – 8: Construct hash table for the storage and retrieval of the symbol.

Aim: To manage the symbols in the program and implement the various operations associated with the symbol table.

Theory:

Numerous symbols (identifiers) are used in the declarative part of the statements and their usage are referred in the executable statements. Hence a storage is required for storing and retrieval of the symbols. The logical storage for storing these symbols can be *set* of symbols where *set* refers to the collection of elements without duplication. The set is viewed as a table and hence we call it as symbol table. However, the data structure used for implementing the set (symbol table) varies as it can be a binary search tree or a B-Tree, or a Hash Map,etc. Hashing is the one of the prefered data structure for storing and retrieving the data in an effective manner. The operations involved with the symbol table in compiler perspective are: addtion of symbols, reference of the symbol.

Steps:

1. Read the symbols in the program during the scanning and parsing process.
2. Define the hash and initialize the hash symbol table.
3. Add the symbol in the hash table (addition of the symbol will be done when the symbols are declared)
4. Look for the symbol in the hash table (look up or find the symbol will happen when the symbol is accessed inside the program in the execuatable statement.
5. If any value is to be assigned, assign the value to the symbol in the symbol table.

Modules

Initialization of the symbol Table

Addion of a symbol

Finding of a symbol

Displaying the symbol

Assignment of values to the symbol in the table

Data Structures and Prototype Declaration:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define true 1
#define false 0
#define teste {printf("Test and exit\n");exit(-1);}
#define test {printf("Test\n");}
#define NAMESIZE 300
#define CHAR 1
#define INT 2
#define FLOAT 3
#define LONG 5
#define DOUBLE 4
#define STRING 6
typedef struct Symbol
{
      short int  type;
      short int width;
      char name[40];
```

```
	union
	{
		char cVal;
		int iVal;
		float fVal;
		long lVal;
		double dVal;
	}value;
	char *sVal;
}Symbol;
typedef struct Node
{
	Symbol symbol;
	struct Node *next;
} Node;
typedef struct Node *Position;
typedef Position List;

class HSet
{
private:
	int hsize;
	List *HashTable;
public:
	HSet(int);
	int  addSymbol(Symbol symbol);
	int findHash(char*);
	void dispSymbolTable();
	void dispNode(Node *p);
```

```
	Node *findSymbol(char *name);
	int assignValue(char *name,char *value);
};
```

Implementation of the symbol table manipulation routines

```
HSet::HSet(int size)
{
int ii;
	hsize = size;
	HashTable = (List*)malloc(sizeof(List) * hsize);
	if(HashTable == NULL) {printf("Memory allocation problem\n");exit(-1);}
	for(ii=0;ii<hsize;ii++)
	{
		HashTable[ii]=(Node*)malloc(sizeof(Node));
		if(HashTable[ii] == NULL) { printf("error in memory allocation
for hash table\n");exit(-1);}
		HashTable[ii]->next = NULL;
	}
}
void HSet::dispSymbolTable()
{
List L;
int ii;
	for(ii=0;ii<hsize;ii++)
	{
		L = HashTable[ii];
		while(L->next != NULL)
```

```
            {
                  dispNode(L->next);
                  L = L->next;
            }
      }
}

void HSet::dispNode(Node *p)
{
      printf("%d\t%s\t",p->symbol.type,p->symbol.name);
      switch(p->symbol.type)
      {
            case CHAR:
                  printf("%c ",p->symbol.value.cVal);
                  break;
            case INT:
                  printf("%d ",p->symbol.value.iVal);
                  break;
            case FLOAT:
                  printf("%f ",p->symbol.value.fVal);
                  break;
            case LONG:
                  printf("%ld ",p->symbol.value.lVal);
                  break;
            case DOUBLE:
                  printf("%lf ",p->symbol.value.dVal);
                  break;
            case STRING:
                  printf("%s",p->symbol.sVal);
```

```
                    break;
        }
        printf("\n");
}

Node* HSet::findSymbol(char *name)
{
List L;
int idx;
        idx = findHash(name);
        L = HashTable[idx];
        while(L->next != NULL)
        {
                if(strcmp(name,L->next->symbol.name) == 0)
                        return L->next;
                L = L->next;
        }
        return NULL;
}

int HSet::assignValue(char *name,char value[])
{
        Node *p;
        char *str;
        p=findSymbol(name);
        if(p!=NULL)
        {
                switch(p->symbol.type)
                {
```

```
                case CHAR:
                        p->symbol.value.cVal = *value;
                        break;
                case INT:
                        p->symbol.value.iVal = atoi(value);
                        break;
                case FLOAT:
                        p->symbol.value.fVal = atof(value);
                        break;
                case LONG:
                        p->symbol.value.lVal = atol(value);
                        break;
                case DOUBLE:
                        p->symbol.value.dVal = atof(value);
                        break;
                case STRING:
                        str = (char*)malloc(strlen(value));
                        strcpy(str,value);
                        p->symbol.sVal=str;
                        break;
            }
        }
        else
        {
            printf("%s : Not found\n",name);
            return -1;
        }
        return 1;
}
```

```
int HSet::findHash(char *name)
{
int hashVal = 0;
      while(*name != '\0')
            hashVal += *name++;
      return hashVal % hsize;
}

int HSet::addSymbol(Symbol symbol)
{
Node *p,*sp;
int ret,idx;
char *str;
      idx = findHash(symbol.name);
      p = HashTable[idx];
      while(p->next != NULL)
      {
            ret = strcmp(symbol.name,p->next->symbol.name);
            if(ret == 0)
            {
                  printf("Symbol: %s is redefined\n",symbol.name);
                  return -1;
            }
            p = p->next;
      }
      sp = (Node*)malloc(sizeof(Node));
      if(NULL == sp)
      {
```

```
            printf("Error in memory allocations\n");
            exit(-1);
      }
      sp->symbol.type = symbol.type;
      strcpy(sp->symbol.name,symbol.name);
      sp->next = NULL;
      p->next = sp;
      return 1;
}
```

Using the symbol table manipulation routines from the main program.

```
int main()
{
HSet symbols(50);
Symbol sym;
int i;

      sym.type = FLOAT;
      strcpy(sym.name,"area");
      symbols.addSymbol(sym);
      symbols.assignValue("area","10.0");
      sym.type = INT;
      strcpy(sym.name,"perimeter");
      symbols.addSymbol(sym);
      symbols.assignValue("perimeter","200");
      sym.type = STRING;
      strcpy(sym.name,"College");
```

```
        symbols.addSymbol(sym);
        symbols.assignValue("College","Mepco Schlenk Engineering College");
        sym.type = STRING;
        strcpy(sym.name,"House");
        symbols.addSymbol(sym);
        symbols.assignValue("House","BG1 Staff Quarters");
        symbols.dispSymbolTable();
        return 1;
}
```

Output

3 area 10.000000

6 House BG1 Staff Quarters

2 perimeter 200

6 College Mepco Schlenk Engineering College

Exercises for the students:

Replace the dummy attributes given for the symbol to the actual compiler-based symbol attributes.