Compiler Design Lab

Lab Assignment - 3

Group 3

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler.
Symbol table is used to store the names of all entities in a structured form at one place, verify if a variable has been declared, determine the scope of a name, etc.
A symbol table is either linear or hash, a symbol table can be implemented in one of the following ways:

- Linear (sorted or unsorted) list
- Binary Search Tree
- Hash table
- Other ways

2 Ways for symbol table implementation :

Implementation by Hash Table -

- In a hashing scheme two tables are maintained – a hash table and symbol table and is the most commonly used method to implement symbol tables..
- A hash table is an array with index range: 0 to tablesize – 1.These entries are pointers pointing to names of symbol tables.
- To search for a name we use a hash function that will result in any integer between 0 to table size.
- Insertion and lookup can be made very fast.
- Advantage is quick search is possible and disadvantage is that hashing is complicated to implement.

Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
```

```
#define SIZE 20
struct symtab
{
char *data;
int key;
};
struct symtab* hashArray[SIZE];
struct symtab* dummyItem;
struct symtab* item;
int hashCode(int key) {
return key % SIZE;
}
struct symtab search(char key) {

int hashIndex = 1;
while(hashArray[hashIndex] != NULL) {
if(hashArray[hashIndex]->data == key)
return hashArray[hashIndex];
++hashIndex;
hashIndex %= SIZE;
}
return 0;
}
void insert(int key,char* data) {
struct symtab item = (struct symtab) malloc(sizeof(struct symtab));
item->data = data;
item->key = key;
struct symtab temp = (struct symtab) malloc(sizeof(struct symtab));
temp = search(data);
if (temp == NULL){
int hashIndex = key;
while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1) {
++hashIndex;
hashIndex %= SIZE;
}
```

```
hashArray[hashIndex] = item;
}
}
void display() {
int i = 0;
printf("\n\n");
printf("info    id\n");
printf("-----------------------\n");
for(i = 0; i<SIZE; i++) {
if(hashArray[i] != NULL)
printf("\n %d  %s",hashArray[i]->key,hashArray[i]->data);
}
printf("\n");
}
int main() {
char id[20][20];
int i = 0;
printf("Start inputing the identifiers\n");
printf("Enter 0 to display\n\n");
while (i < 20){
scanf("%s",id[i]);
if (!strcmp(id[i],"0")){
break;
}
insert(i,id[i]);
i++;
}
display();
}
```

Implementation by Linked List -

- This implementation is using a linked list. A link field is added to each record.
- Searching for names is done in order pointed by the link of the link field.

- A pointer “First” is maintained to point to the first record of the symbol table.
- Insertion is fast, but lookup is slow for large tables on average

Code:

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<string.h>
#define null 0
int size = 0;
void insert ();
void display ();
int search(char lab[]);
struct symbtab
{
char label[10];
char info[10];
struct symbtab *next;
};
struct symbtab *first, *last;
void main ()
{
int op;
int y;
char la[10];
do
{
printf ("0.INSERT ELEMENTS\n");
printf ("1.DISPLAY TABLE\n");
printf("2.EXIT PROGRAM\n");
printf ("\nEnter your Option: ");
scanf ("%d", &op);
switch (op){
case 0:
```

```
insert ();
break;
case 1:
display ();
break;
case 2:
exit (0);
}
}
while (op < 2);
getch ();
}

void insert () {
int n;
char l[10];
printf ("Enter the Identifier:\n");
scanf ("%s", l);
n = search (l);
if (n == 1)
printf("The label is already in the symbol table. Duplicate cant be inserted\n");
else{
struct symbtab *p;
p = malloc (sizeof (struct symbtab));
strcpy (p->label, l);
printf ("Enter the info:\n");
scanf ("%s", &p->info);
p->next = null;
if (size == 0){
first = p;
last = p;
}
else{
last->next = p;
last = p;
```

```
}
size++;
}
}
void display (){
int i;
struct symbtab *p;
p = first;
printf ("Id\t\t\tInfo\n");
for (i = 0; i < size; i++){
printf ("%s\t\t\t%s\n", p->label, p->info);
p = p->next;
}
}
int search (char lab[]){
int i, flag = 0;
struct symbtab *p;
p = first;
for (i = 0; i < size; i++){
if (strcmp (p->label, lab) == 0)
flag = 1;
p = p->next;
}
return flag;
}
```