**Name : Omkar Nikhal**

**Class : SE-3**

**Roll no : 21342**

**ASSIGNMENT NO.** 7

|  |  |
| --- | --- |
| Title: | To write a program for implementation of symbol table and perform various operations. |
| Problem Statement: | The symbol table is generated by compiler. From this perspective, the symbol table is a set of name-attribute pairs. In a symbol table for a compiler, the name is an identifier, and the attributes might include an initial value and a list of lines that use the identifier. Perform the following operations on symbol table:   1. Determine if a particular name is in the table 2. Retrieve the attributes of that name 3. Modify the attributes of that name 4. Insert a new name and its attributes 5. Delete a name and its attributes |
| Objective: | 1. To understand concept of symbol table. 2. Why symbol table is needed. |
| Outcome: | 1. Use of symbol table 2. Various methods of implementing symbol table. |
| S/W Packages and H/W Apparatus used: | 1. 64-bit Fedora 17 or latest 64-Bit update of equivalent open source OS 2. Programming tools (64-Bit) and latest open source update of Eclipse Programming framework, TC++, GTK++ |

Theory

Symbol Table:

In computer science, a symbol table is a data structure used by a language translator such as a compiler or interpreter, where each identifier (a.k.a. symbol) in a program's source code is associated with information relating to its declaration or appearance in the source.

A symbol table may only exist during the translation process, or it may be embedded in the output of that process, such as in an ABI object file for later exploitation. For example, it might be used during an interactive debugging session, or as a resource for formatting a diagnostic report during or after execution of a program. And used only in compilers mostly.

Symbol table is used to store information related to various entities like as function name, variable name, objects, classes, interfaces, etc. When identifiers are found, they will be entered into a symbol table, which will hold all relevant information about identifiers. Symbol table is type of data structure that captures scope information. One symbol table for each scope is used. It stores all entities in structured form at one place. By using symbol table, it checks if variable is declared or not. It is also used for syntax checking.

A symbol table is simply a table which can be either linear or a hash table. It maintains an entry for each name in the format as <symbol name, type, attribute>. For example, table has to store information about following variable declaration as static int interest; then it should store the entry such as <interest, int, static> The attribute clause contains the entries related to the name.

There are two types of symbol tables. Static symbol table and Dynamic symbol table.

Static symbol tables are tree tables. They are implemented when symbols are known in advance and no addition and deletion is allowed.

Dynamic symbol tables are used when symbol is not known in advance and insertion and deletion can be done any time.

Symbol Table Implementation Methods:

* Unordered Array Implementations
* Ordered Array Implementations
* Unordered Ordered List
* Binary Search Trees
* Balanced Binary Search Trees
* Hashing

Unordered Array Implementation:

It maintains arrays of keys and values. Instance variables are used to store data. Array *keys[]* holds the keys and *vals[]* holds the values, integer *N* holds the number of entries.

Ordered Array Implementation:

In ordered array implementation, keys are comparable to each other. Ordered array implementation for symbol table used because it provides ordered iteration. Binary search can speed up search.

Ordered or Unordered Linked-list Implementation:

Maintain a linked list with keys and values. Advantage of keeping linked list in order for comparable key, support ordered iterator and cuts search or insert time in half.

Binary Search Tree:

Keep data stored in parent to child format and sorted. Insertion, deletion, etc. operation can be done faster compared to other methods.

Balanced Binary Search Tree:

Space overhead is directly proportional to the number of items in the table. Insertion takes time compared to other methods of implementation.

Hash Table:

We can work faster in hashing methods. Hashing table method is used in most compilers. Complexity is O(1) for hashing table.

Algorithms

HashNode Class Declaration

class HashNode{

public:

int key;

int value;

HashNode \* next;

HashNode(int key, int value){

this->key = key;

this->value = value;

this->next = NULL;

}

};

Insertion

void Insert(int key, int value){

int hash\_val = HashFunc(key);

HashNode \* prev = NULL;

HashNode \* entry = htable[hash\_val];

while(entry != NULL){

prev = entry;

entry = entry->next;

}

if(entry == NULL){

entry = new HashNode(key,value);

if(prev == NULL){

htable[hash\_val] = entry;

}

else{

prev->next = entry;

}

}

else{

entry->value = value;

}

}

Deletion

void Remove(int key){

int hash\_val = HashFunc(key);

HashNode\* entry = htable[hash\_val];

HashNode\* prev = NULL;

if (entry == NULL || entry->key != key){

cout<<"No Element found at key "<<key<<endl;

return;

}

while (entry->next != NULL){

prev = entry;

entry = entry->next;

}

if (prev != NULL){

prev->next = entry->next;

}

delete entry;

cout<<"Element Deleted"<<endl;

}

Search

int Search(int key){

bool flag = false;

int hash\_val = HashFunc(key);

HashNode\* entry = htable[hash\_val];

while (entry != NULL){

if (entry->key == key){

cout<<entry->value<<" ";

flag = true;

}

entry = entry->next;

}

if (!flag)

return -1;

}

Test-Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Input | Output | Result |
| Insert without replacement | |  |  | | --- | --- | | Symbol | Value | | cout | 2 | | cin | temp | | delete | obj | | exit | 0 | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Index | Key | Symbol | Value | Next | | 2 | c | cout | 2 | 3 | | 3 | d | cin | temp | 4 | | 4 | e | delete | obj | 5 | | 5 | f | exit | 0 | -1 | | Pass |
| Insert with replacement | |  |  | | --- | --- | | Symbol | Value | | cout | 2 | | cin | temp | | delete | obj | | exit | 0 | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Index | Key | Symbol | Value | Next | | 2 | c | cout | 2 | 5 | | 3 | d | delete | obj | -1 | | 4 | e | exit | 0 | -1 | | 5 | f | cin | temp | -1 | | Pass |
| Find | Enter symbol: cout  Enter symbol: cerr | Value at symbol “cout” is: 2  Symbol “cerr” not found! | Pass |
| Search Cost | - | Average Search Cost is 1.25 | Pass |
| Delete | Enter symbol: cout  Enter symbol: cout | Deleted symbol: “cout”, value: 2 !  Symbol “cout” not found! | Pass |

Conclusion

After successfully completing this assignment, students have learned implementation of Symbol Table using Hashing and various standard operations of Symbol Table ADT.