Symbol Table - AP20110010025 - Chinnam Goutham Nischay - CSE A

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various identifiers 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 can be implemented in one of the following ways:

- Linear (sorted or unsorted) list
- Binary Search Tree
- Hash table
- And other ways.

In this lab session, you are required to analyze the various implementations. You need to write code for at least two ways of implementation. Test your code with different test cases.

Submit a report of your analysis and executable code by the end of the session.

Symbol Table is used to store tokens such as Identifiers : Datatype, Variable label and Data value/Attributes in a structured manner so that it could be helpful in the parsing stage.

There are different methods to implement symbol table :

Binary Search Tree :

Binary search trees (BSTs) are data structures that allow for efficient insertion, deletion, and retrieval of data. BSTs are often used to store data in a sorted order, making it easy to search for specific items.

Advantages :

-They are very efficient when it comes to searching for data, as the data is stored in a sorted order.
-They are also easy to update, as the data can be easily inserted or deleted without having to reorganize the entire data structure.

Disadvantages:
-BSTs can become unbalanced if the data is not inserted in the correct order, which can lead to decreased efficiency.

Code :

```
#include<bits/stdc++.h>
using namespace std;

struct node
{
	string datatype,name,value;
	node*left;
	node*right;
};
node*root=NULL;

node*new_node(string datatype,string name,string value)
{
	node*temp=new node;
	temp->datatype=datatype;
	temp->name=name;
	temp->value=value;
	temp->left=NULL;
	temp->right=NULL;
	return temp;
}

void insert(string datatype,string name,string value)
{
	if(root==NULL)
```

```
            root=new_node(datatype,name,value);
        else
        {
            node*temp=root;
            node*current=new node;

            while(temp!=NULL)
            {
                current=temp;
                if(name>temp->name)
                    temp=temp->right;
                else
                    temp=temp->left;
            }
            if(name>current->name)
                current->right=new_node(datatype,name,value);
            else
                current->left=new_node(datatype,name,value);
        }
}

node*search(node*root,string name)
{
        if(root==NULL)
            return NULL;
        else if(root->name==name)
            return root;
        else if(name>root->name)
            return search(root->right,name);
        else
            return search(root->left,name);
}

int main()
{
        insert("int","int1","10");
```

```
        insert("float","fnum","20.0");
        insert("double","dnum","30.0123");
        insert("char","char1","c");
        insert("string","str1","SRM University");

        node*temp=search(root,"str1");
        cout<<temp->datatype<<","<<temp->name<<","<<temp->value;
}
```

Output :

```
> make -s
> ./main
string,str1,SRM University>
```

Hash Table :

A hash table is a data structure that stores data in an associative array format, where the keys are used to access thevalues. A hash table is a good choice for a data structure when you need to store data in a way that allows for fastinsertion and retrieval.

Advantages :
1) Hash tables are very fast for both insertion and lookup.
2) Hash tables use very little memory, because they only store the keys, not the values.
3) Hash tables can be used to implement other data structures, such as queues and stacks.
4) Hash tables can be used to store data in a database.

Disadvantages :
1) Hash Tables can have collisions, where two keys map to the same value. This can be a problem if the keys are not carefully chosen.

2) Hash tables can be expensive to insert into, because the keys need to be hashed and the values need to be stored in an array.

Code :

```
#include <bits/stdc++.h>
using namespace std;

class ST {
  string datatype;
  string key;
  string value;

public:
  ST(string d, string k, string v) {
    datatype = d;
    key = k;
    value = v;
  }
  // function to get key
  string getType() { return datatype; }
  string getKey() { return key; }
  // function to get value
  string getValue() { return value; }
};

// Hashmap
unordered_map<string, ST *> SymbolTable;

void insert(string datatype, string key, string value) {
  if (SymbolTable.find(key) == SymbolTable.end())
    SymbolTable[key] = new ST(datatype, key, value);
}

// Function to get value for a given key
string get(string key) {
```

```
  if (SymbolTable.find(key) == SymbolTable.end())
    return "NULL";
  else
    cout << SymbolTable[key]->getType() << " " <<
SymbolTable[key]->getKey()
         << " " << SymbolTable[key]->getValue();
  return "True";
}
string getKey(string value) {
  unordered_map<string, ST *>::iterator itr;
  for (itr = SymbolTable.begin(); itr != SymbolTable.end(); itr++) {
    if (itr->second->getValue() == value)
      return itr->second->getKey();
  }
}

int main() {
  insert("int", "A", "1");
  insert("char", "char1", "B");
  insert("string", "str1", "SRM University");
  insert("float", "D", "3.14");

  get("str1");
  cout << "\n";
  cout << getKey("3.14") << endl;

  return 0;
}
```

Output :

```
 make -s
 ./main
string str1 SRM University
D
```