SYMBOL TABLE LAB REPORT

S Sabari raj

AP21110011027

CSE P

**Title:**

Symbol Table Implementation

**Description:**

Symbol table is a data structure in designing a compiler or a interpreter created 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 analyse 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.

**Code 1:**

//BST implementation of Symbol table

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Node

{

    char key[100];

    int value;

    char type[10];

    char token[15];

    struct Node\* left;

    struct Node\* right;

};

struct Node\* insert(struct Node\* curr, char\* key, int value,char \*type,char \*token)

{

    if (curr == NULL)

    {

        struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

        strcpy(newNode->key, key);

        newNode->value = value;

        strcpy(newNode->type, type);

        strcpy(newNode->token,token);

        newNode->left = NULL;

        newNode->right = NULL;

        return newNode;

    }

    if (strcmp(key, curr->key) < 0)

    {

        curr->left = insert(curr->left, key, value,type,token);

    }

    else if (strcmp(key, curr->key) > 0)

    {

        curr->right = insert(curr->right, key, value,type,token);

    }

    return curr;

}

int search(struct Node\* curr, char\* key)

{

    if (curr == NULL || strcmp(curr->key, key) == 0)

    {

        return curr != NULL ? curr->value : -1;

    }

    if (strcmp(key, curr->key) < 0)

    {

        return search(curr->left, key);

    }

    return search(curr->right, key);

}

int main()

{

    struct Node\* root = NULL;

    root = insert(root, "x", 8,"int","keyword");

    root = insert(root, "y", 27,"int","keyword");

    root = insert(root,"+",0,"operator","operator");

    printf("%d\n", search(root, "y"));

    printf("%d\n", search(root, "x"));

    return 0;

}

**Output**:

27

8

**Symbol Table record:**

**Symbol Value Type Token**

x 8 int keyword

y 27 int keyword

**Code 2**:

//Hash implementation of symbol table

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 100

struct Symbol

{

    char \*name;

    int value;

    struct Symbol \*next;

};

struct SymbolTable

{

    struct Symbol \*sptr[SIZE];

};

unsigned int hash(const char \*str)

{

    unsigned int hash = 0;

    while(\*str)

    {

        hash = (hash << 5) + \*str++;

    }

    return hash % SIZE;

}

void insert(struct SymbolTable \*table, const char \*name, int value)

{

    unsigned int index = hash(name);

    struct Symbol \*entry = (struct Symbol \*)malloc(sizeof(struct Symbol));

    if (!entry)

    {

        perror("Memory allocation failed");

        exit(EXIT\_FAILURE);

    }

    strcpy(entry->name ,name);

    entry->value = value;

    entry->next = table->sptr[index];

    table->sptr[index] = entry;

}

struct Symbol \*search(struct SymbolTable \*table, const char \*name)

{

    unsigned int index = hash(name);

    struct Symbol \*entry = table->sptr[index];

    while (entry != NULL)

    {

        if (strcmp(entry->name, name) == 0)

        {

            return entry;

        }

        entry = entry->next;

    }

    return NULL;

}

int main()

{

    struct SymbolTable symbol\_table;

    for (int i = 0; i < SIZE; i++)

    {

        symbol\_table.sptr[i] = NULL;

    }

    insert(&symbol\_table, "x", 30);

    insert(&symbol\_table, "y", 8);

    struct Symbol \*entry\_x = search(&symbol\_table, "x");

    if (entry\_x)

    {

        printf("Symbol: %s, Value: %d\n", entry\_x->name, entry\_x->value);

    } else

    {

        printf("Symbol not found.\n");

    }

    for (int i = 0; i < SIZE; i++)

    {

        struct Symbol \*entry = symbol\_table.sptr[i];

        while (entry)

        {

            struct Symbol \*next = entry->next;

            free(entry->name);

            free(entry);

            entry = next;

        }

    }

    return 0;

}

**Output :**  
 30

**Conclusion**:

In summary, symbol tables are essential data structures used in programming to efficiently manage and retrieve information about symbols like variables and functions, aiding scoping, conflict resolution, and accurate compilation or interpretation.