CSE 306L - Week 2 Assignment

Name : Nazeer Mastan Basha
Id : AP21110011037

Topic: Implementation of Symbol Table

The symbol table can be implemented in the unordered list if the compiler is used to handle the small amount of data. A Symbol table can be implemented in one of the following techniques:
• Linear (sorted or unsorted) list
• Binary Search Tree
• Hash table

1. Linked List :

1. • This implementation is using a linked list. A link field is added to each record.
2. • Searching for names is done in order pointed by the link of the link field.
3. • A pointer "First" is maintained to point to the first record of the symbol table.
4. • Insertion is fast O(1), but lookup is slow for large tables - O(n) on average

2. Hash Table:

1. • A hash table is an array with an index range: 0 to table size - 1. These entries are pointers pointing to the names of the symbol table.
2. • To search for a name we use a hash function that will result in an integer between 0 to table size - 1.
3. • Insertion and lookup can be made very fast - O(1).
4. • The advantage is that a quick search is possible and the disadvantage is that hashing is complicated to implement.

3. Binary Search Tree:

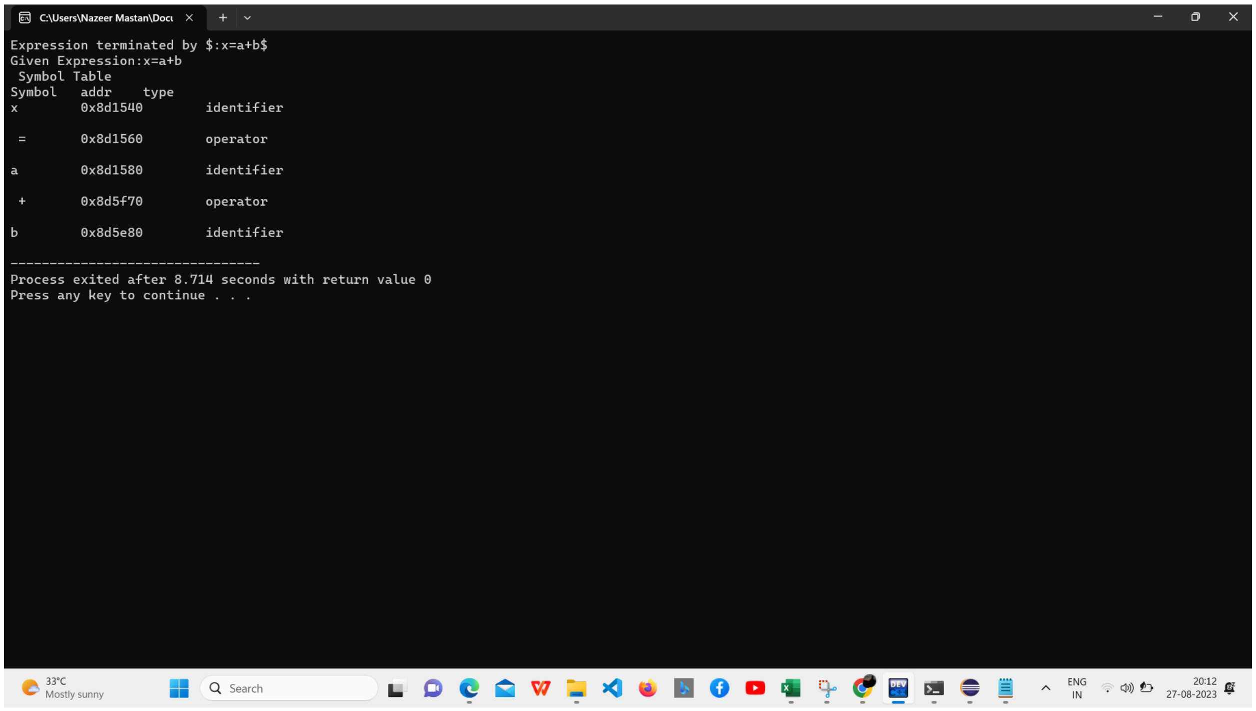
1. • Another approach to implementing a symbol table is to use a binary search tree i.e. we add two link • fields i.e. left and right child.
2. • All names are created as a child of the root node that always follows the property of the binary search tree.
3. • Insertion and lookup are O(log2 n) on average.

Code work

```
#include <iostream>
#include <cctype>
#include <cstdlib>
#include <cstring>
#include <cmath>

int main()
{
    int i = 0, j = 0, x = 0, n;
    void *p, *add[5];
    char ch, srch, b[15], d[15], c;
    std::cout << "Expression terminated by $:";
    while ((c = getchar()) != '$')
    {
        b[i] = c;
        i++;
    }
    n = i - 1;
    std::cout << "Given Expression:";
    i = 0;
    while (i <= n)
    {
        std::cout << b[i];
        i++;
    }
    std::cout << "\n Symbol Table\n";
    std::cout << "Symbol \t addr \t type";
    while (j <= n)
    {
        c = b[j];
        if (isalpha(static_cast<unsigned char>(c)))
        {
            p = malloc(sizeof(c));
            add[x] = p;
            d[x] = c;
            std::cout << "\n" << c << " \t " << p << " \t identifier\n";
            x++;
            j++;
        }
        else
        {
            ch = c;
            if (ch == '+' || ch == '-' || ch == '*' || ch == '=')
            {
                p = malloc(sizeof(ch));
                add[x] = p;
                d[x] = ch;
                std::cout << "\n " << ch << " \t " << p << " \t operator\n";
                x++;
                j++;
            }
        }
    }
    return 0;
}
```

Output:



```
Expression terminated by $:x=a+b$
Given Expression:x=a+b
 Symbol Table
Symbol   addr    type
x        0x8d1540        identifier

 =       0x8d1560        operator

a        0x8d1580        identifier

 +       0x8d5f70        operator

b        0x8d5e80        identifier

--------------------------------
Process exited after 8.714 seconds with return value 0
Press any key to continue . . .
```

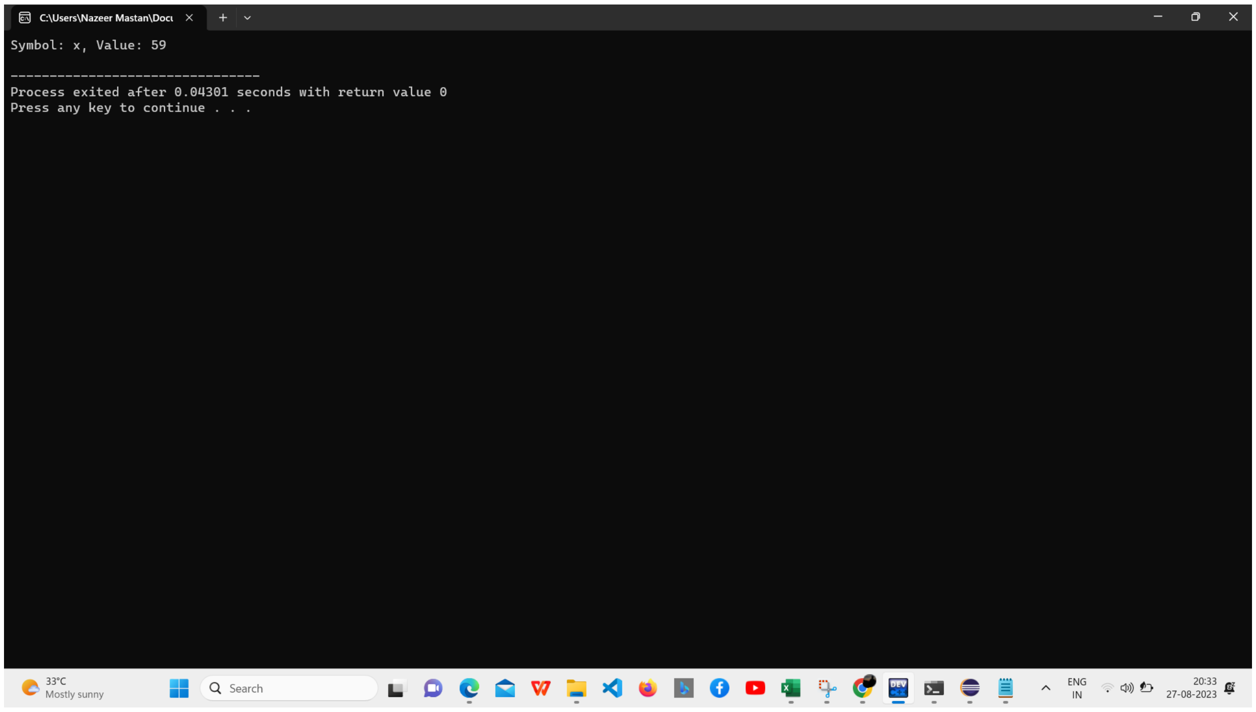
Using The Hash Table

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define HASH_TABLE_SIZE 100
struct SymbolEntry
{
char *name;
int value;
struct SymbolEntry *next;
};
struct SymbolTable
{
struct SymbolEntry *hash_table[HASH_TABLE_SIZE];
};
```

```
unsigned int hash(const char *str)
{
unsigned int hash = 0;
while (*str)
{
hash = (hash << 5) + *str++;
}
return hash %
HASH_TABLE_SIZE;
}
void insert(struct SymbolTable *table, const char *name, int value)
{
unsigned int index = hash(name);
struct SymbolEntry *entry = (struct SymbolEntry *)malloc(sizeof(struct
SymbolEntry));
if (!entry)
{
perror("Memory allocation failed");
exit(EXIT_FAILURE);
}
entry->name = strdup(name);
entry->value = value;
entry->next = table->hash_table[index];
table->hash_table[index] = entry;
}
struct SymbolEntry *search(struct SymbolTable *table, const char
*name)
{
unsigned int index = hash(name);
struct SymbolEntry *entry = table->hash_table[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 < HASH_TABLE_SIZE; i++)
{
symbol_table.hash_table[i] = NULL;
}
insert(&symbol_table, "x", 59);
insert(&symbol_table, "y", 27);
struct SymbolEntry *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 < HASH_TABLE_SIZE; i++)
{
struct SymbolEntry *entry = symbol_table.hash_table[i];
while (entry)
{
struct SymbolEntry *next = entry->next;
free(entry->name);
free(entry);
entry = next;
}
}
return 0;
}
```

Output:



```
Symbol: x, Value: 59

--------------------------------
Process exited after 0.04301 seconds with return value 0
Press any key to continue . . .
```