A Practical Activity Report For
Data Structures and Algorithms (UCS406)

Submitted By: **Vivek Arora**

101715178
(ENC 8)

Submitted To:

Dr. Sanjay Sharma

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING &TECHNOLOGY, (DEEMED TO BEUNIVERSITY),
PATIALA, PUNJAB

ASSIGNMENT 10

QUESTION 1:

Write a program to covert an infix to postfix expression for a postfix expression using stack .

a+b*c-d/e

```
#include <stdio.h>
#include <stdlib.h>
#include<strings.h>
struct Node
{
char data;
struct Node *next;
}*top=NULL;
void push(char x)
{
struct Node *t;
t=(struct Node*)malloc(sizeof(struct Node));
if(t==NULL)
printf(&quot;stack is full\n&quot;);
else
{
t-&gt;data=x;
t-&gt;next=top;
top=t;
}
}
char pop()
{
struct Node *t;
char x=-1;

if(top==NULL)
printf(&quot;Stack is Empty\n&quot;);
else
{
t=top;
top=top-&gt;next;
x=t-&gt;data;
free(t);
}
```

```
return x;
}
void Display()
{
struct Node *p;
p=top;
while(p!=NULL)
{
printf(&quot;%d &quot;,p-&gt;data);
p=p-&gt;next;
}
printf(&quot;\n&quot;);
}
int isBalanced(char *exp)
{
int i;
for(i=0;exp[i]!=&#39;\0&#39;;i++)
{
if(exp[i]==&#39;(&#39;)
push(exp[i]);

else if(exp[i]==&#39;)&#39;)
{
if(top==NULL)
return 0;
pop();
}
}
if(top==NULL)
return 1;
else
return 0;
}
int pre(char x)
{
if(x==&#39;+&#39; || x==&#39;-&#39;)
return 1;
else if(x==&#39;*&#39; || x==&#39;/&#39;)
return 2;
return 0;
}
int isOperand(char x)
{
```

```
if(x==&#39;+&#39; || x==&#39;-&#39; || x==&#39;*&#39; || x==&#39;/&#39;)
return 0;
else
return 1;
}
char * InToPost(char *infix)
{

int i=0,j=0;
char *postfix;
int len=strlen(infix);
postfix=(char *)malloc((len+2)*sizeof(char));
while(infix[i]!=&#39;\0&#39;)
{
if(isOperand(infix[i]))
postfix[j++]=infix[i++];
else
{
if(pre(infix[i])&gt;pre(top-&gt;data))
push(infix[i++]);
else
{
postfix[j++]=pop();
}
}
}
while(top!=NULL)
postfix[j++]=pop();
postfix[j]=&#39;\0&#39;;
return postfix;
}
int main()
{
char *infix=&quot;a+b*c-d/e&quot;;
push(&#39;#&#39;);
char *postfix=InToPost(infix);
printf(&quot;%s &quot;,postfix);

return 0;
}
```

QUESTION 2:

Write a program to implement the Hash Table and chaining to avoid collisions and perform the following functions:
1. Insert a key
2. Search a key
3. Delete a key

```
#include<iostream>
#include<cstdlib>
#include<string>
#include<cstdio>
using namespace std;
const int T_S = 200;
class HashTableEntry {
   public:
      int k;
      int v;
      HashTableEntry(int k, int v) {
         this->k= k;
         this->v = v;
      }
};
class HashMapTable {
   private:
      HashTableEntry **t;
   public:
      HashMapTable() {
         t = new HashTableEntry * [T_S];
         for (int i = 0; i< T_S; i++) {
            t[i] = NULL;
         }
      }
      int HashFunc(int k) {
         return k % T_S;
      }
      void Insert(int k, int v) {
         int h = HashFunc(k);
         while (t[h] != NULL && t[h]->k != k) {
            h = HashFunc(h + 1);
         }
         if (t[h] != NULL)
            delete t[h];
         t[h] = new HashTableEntry(k, v);
      }
      int SearchKey(int k) {
         int h = HashFunc(k);
         while (t[h] != NULL && t[h]->k != k) {
            h = HashFunc(h + 1);
         }
         if (t[h] == NULL)
            return -1;
         else
            return t[h]->v;
      }
      void Remove(int k) {
         int h = HashFunc(k);
         while (t[h] != NULL) {
            if (t[h]->k == k)
               break;
            h = HashFunc(h + 1);
         }
         if (t[h] == NULL) {
            cout<<"No Element found at key "<<k<<endl;
            return;
         } else {
            delete t[h];
         }
         cout<<"Element Deleted"<<endl;
      }
      ~HashMapTable() {
         for (int i = 0; i < T_S; i++) {
            if (t[i] != NULL)
               delete t[i];
               delete[] t;
         }
      }
};
int main() {
   HashMapTable hash;
   int k, v;
   int c;
   while (1) {
      cout<<"1.Insert element into the table"<<endl;
      cout<<"2.Search element from the key"<<endl;
      cout<<"3.Delete element at a key"<<endl;
      cout<<"4.Exit"<<endl;
```

```
    cout<<"Enter your choice: ";
    cin>>c;
    switch(c) {
      case 1:
        cout<<"Enter element to be inserted: ";
        cin>>v;
        cout<<"Enter key at which element to be inserted: ";
        cin>>k;
        hash.Insert(k, v);
      break;
      case 2:
        cout<<"Enter key of the element to be searched: ";
        cin>>k;
        if (hash.SearchKey(k) == -1) {
          cout<<"No element found at key "<<k<<endl;
          continue;
        } else {
          cout<<"Element at key "<<k<<" : ";
          cout<<hash.SearchKey(k)<<endl;
        }
      break;
      case 3:
        cout<<"Enter key of the element to be deleted: ";
        cin>>k;
        hash.Remove(k);
      break;
      case 4:
        exit(1);
      default:
        cout<<"\nEnter correct option\n";
    }
  }
  return 0;
}
```

QUESTION 3: Write a program to demonstrate Linear Probing for avoiding collision in the hash table.

```
#include&lt;iostream&gt;
#define SIZE 10
using namespace std;
int hash(int key)
```

```
{return key%SIZE;}
int probe(int h[],int key)
{int index=hash(key);
int i=0;
while(h[(index+i)%SIZE]!=0)
i++;
return (index+i)%SIZE;
}
void insert(int h[],int key)
{int index=hash(key);
if(h[index]!=0)
index=probe(h,key);
h[index]=key;

}
int search(int h[],int key)
{int i=0;
int index=hash(key);
while (h[(index+i)%SIZE]!=key)
i++;
return (index+i)%SIZE;
}

int main()
{int h[10]={0}; //hash table size
insert(h,34);
insert(h,65);
insert(h,54);
cout&lt;&lt;search(h,34)&lt;&lt;endl;
cout&lt;&lt;search(h,65)&lt;&lt;endl;
cout&lt;&lt;search(h,54)&lt;&lt;endl;
}
```

QUESTION 4: Write a program to demonstrate Quadratic Probing for avoiding collision in the hash table.

```
#include&lt;iostream&gt;
#define SIZE 10                              //macros
using namespace std;
```

```
int hash(int key)

{return key%SIZE;}

int prob(int h[],int key)
{int index=hash(key);
int i=0;
while(h[(index+(i*i))%SIZE]!=0)
i++;
return (index+(i*i));
}

void insert(int h[],int key)
{int index=hash(key);
if(h[index]!=0)
index=prob(h,key);
h[index]=key;
}

int search(int h[],int key)
{int index=hash(key);
int i=0;
while(h[(index+(i*i))%SIZE]!=key)
i++;
return (index+(i*i));
}

int main()
{int h[10]={0}; //hash table size
insert(h,34);

insert(h,65);
insert(h,54);
cout&lt;&lt;search(h,34)&lt;&lt;endl;
cout&lt;&lt;search(h,65)&lt;&lt;endl;
cout&lt;&lt;search(h,54)&lt;&lt;endl;
}
```

QUESTION 5: Write a program to demonstrate Double Hashing for avoiding collision in the hash table.

```
#include&lt;iostream&gt;
#define SIZE 10
using namespace std;

int hash(int key)
{return key%SIZE;}

int prob(int h[],int key)
{int index=hash(key);
int i=0;
while(h[(index+(i*(7-(key%7))))%SIZE]!=0) //7 is closest prime no less than 10
i++;

return (index+(i*(7-(key%7))));
}

void insert(int h[],int key)
{int index=hash(key);
if(h[index]!=0)
index=prob(h,key);
h[index]=key;
}

int search(int h[],int key)
{int index=hash(key);
int i=0;
while(h[(index+(i*(7-(key%7))))%SIZE]!=key)
i++;
return (index+(i*(7-(key%7))));
}
int main()
{int h[10]={0}; //hash table size
insert(h,34);
insert(h,65);
insert(h,54);
cout&lt;&lt;search(h,34)&lt;&lt;endl;
cout&lt;&lt;search(h,65)&lt;&lt;endl;
cout&lt;&lt;search(h,54)&lt;&lt;endl;
}
```