❖ Course Objectives:

- To understand practical implementation and usage of non linear data structures for solving problems of different domain.
- To strengthen the ability to identify and apply the suitable data structure for the given real-world problems.
- To analyze advanced data structures including hash table, dictionary, trees, graphs, sorting algorithms and file organization.

❖ Course Outcomes:

On completion of the course, learner will be able to–

- CO1: Understand the ADT/libraries, hash tables and dictionary to design algorithms for a specific problem.
- CO2: Choose most appropriate data structures and apply algorithms for graphical solutions of the problems.
- CO3: Apply and analyze non-linear data structures to solve real world complex problems.
- CO4: Apply and analyze algorithm design techniques for indexing, sorting, multi-way searching, file organization and compression.
- CO5: Analyze the efficiency of most appropriate data structure for creating efficient solutions for engineering design situations.

INDEX

	cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Check whether the graph is connected or not. Justify the storage representation used.	
8	Given sequence k = k1 <k2 < … <kn of n sorted keys, with a search probability pi for each key ki. Build the Binary search tree that has the least search cost given the access probability for each key?	
9	A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword.	
10	Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language.	
11	Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.	
12	Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.	

Group A

Experiment No-1

Aim:- Consider telephone book database of N clients Make use of Hash Table Implementation to quickly look up clients telephone number make use two collision handling technique and compare this using number of comparison required to find a set of telephone number.

Prerequisite: C++ Programming

Objective: To understand the use functions for Collision Handling Technique.

Input: N Number of Clients

Theory:

Hashing is a technique that uses fewer key comparisons and searches the element in O(n) time in the worst case and in O(1) time in the average case.
The task is to implement all functions of phone directory:

1. **create_record**
2. **display_record**
3. **delete_record**
4. **search_record**
5. **update_record**

Approach: We are creating a hash table, and inserting records. For deleting, searching, or updating an entity, the client ID is asked and on the basis of equality operator, details are displayed or processed. If the record is not found, then an appropriate message is displayed. Collision is the major problem in the hashing technique. In open addressing (closed hashing), all collisions are resolved in the prime area i.e., the area that contains all of the home addresses. When a collision occurs, the prime area addresses are searched for an open or unoccupied element using linear probing.
Steps for inserting entities in a hash table:
1. If the location is empty, directly insert the entity.
2. If mapped location is occupied then keep probing until an empty slot is found. Once an empty slot is found, insert the entity.

1. **Create Record:** This method takes details from the user like ID, Name and Telephone number and create new record in the hashtable.
2. **Display Record:** This function is created to display all the record of the diary.
3. **Delete Record:** This method takes the key of the record to be deleted. Then, it searches in hash table if record id matches with the key. Then, that record is deleted.
4. **Search Record:** This method takes the key of the record to be searched. Then, it traverses the hash table, if record id matches with the key it displays the record detail.
5. **Update Record:** This method takes the key of the record to be searched. Then, it traverses the hash table, if record id matches with the key then it displays the record detail.

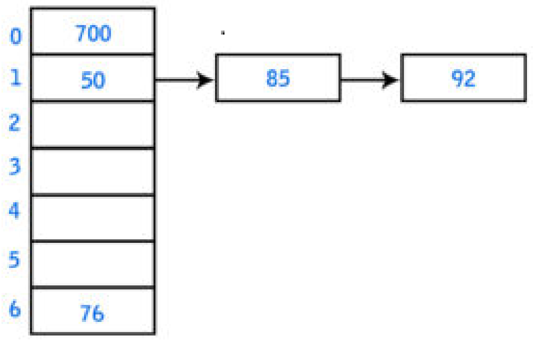
Collision Resolution Techniques:

When one or more hash values compete with a single hash table slot, collisions occur. To resolve this, the next available empty slot is assigned to the current hash value. The most common methods are open addressing, chaining, probabilistic hashing, perfect hashing and coalesced hashing technique.

Let's understand them in more detail:

a) Chaining:

This technique implements a linked list and is the most popular collision resolution techniques. Below is an example of a chaining process.



Here, since one slot has 3 elements – {50, 85, 92}, a linked list is assigned to include the other 2 items {85, 92}. When you use the chaining technique, inserting or deleting of items with the hash table is fairly simple and high performing. Likewise, a chain hash table inherits the pros and cons of a linked list. Alternatively, chaining can use dynamic arrays instead of linked lists.

b) Open Addressing:

This technique depends on space usage and can be done with linear or quadratic probing techniques. As the name says, this technique tries to find an available slot to store the record. It can be done in one of the 3 ways –

- Linear probing – Here, the next probe interval is fixed to 1. It supports best caching but miserably fails at clustering.
- Quadratic probing – the probe distance is calculated based on the quadratic equation. This is considerably a better option as it balances clustering and caching.
- Double hashing – Here, the probing interval is fixed for each record by a second hashing function. This technique has poor cache performance although it does not have any clustering issues.

Below are some of the hashing techniques that can help in resolving collision**.**

c) Probabilistic hashing:

This is memory-based hashing that implements caching. When collision occurs, either the old record is replaced by the new or the new record may be dropped. Although this scenario has a risk of losing data, it is still preferred due to its ease of implementation and high performance.

d) Perfect hashing:

When the slots are uniquely mapped, there is very less chances of collision. However, it can be done where there is a lot of spare memory.

e) Coalesced hashing:

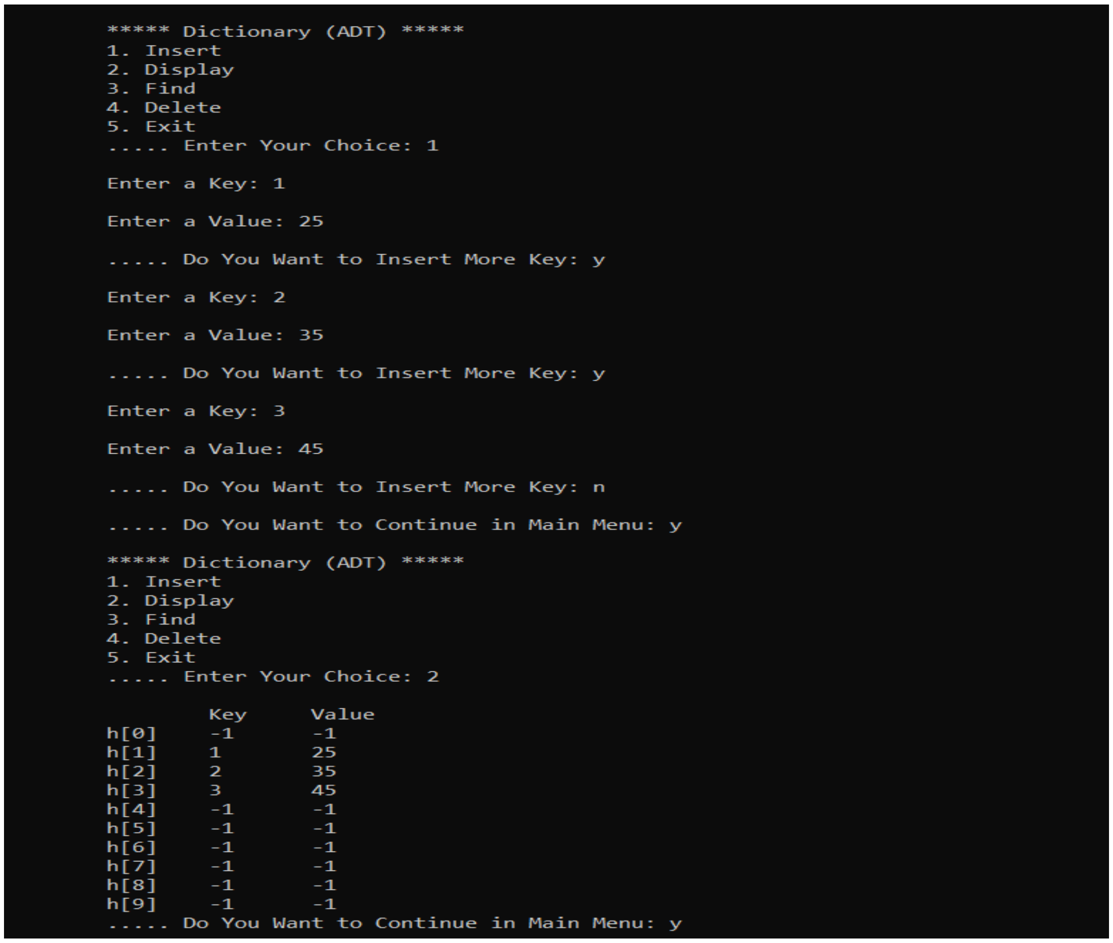
This technique is a combo of open address and chaining methods. A chain of items is stored in the table when there is a collision. The next available table space is used to store the items to prevent collision.

Program:

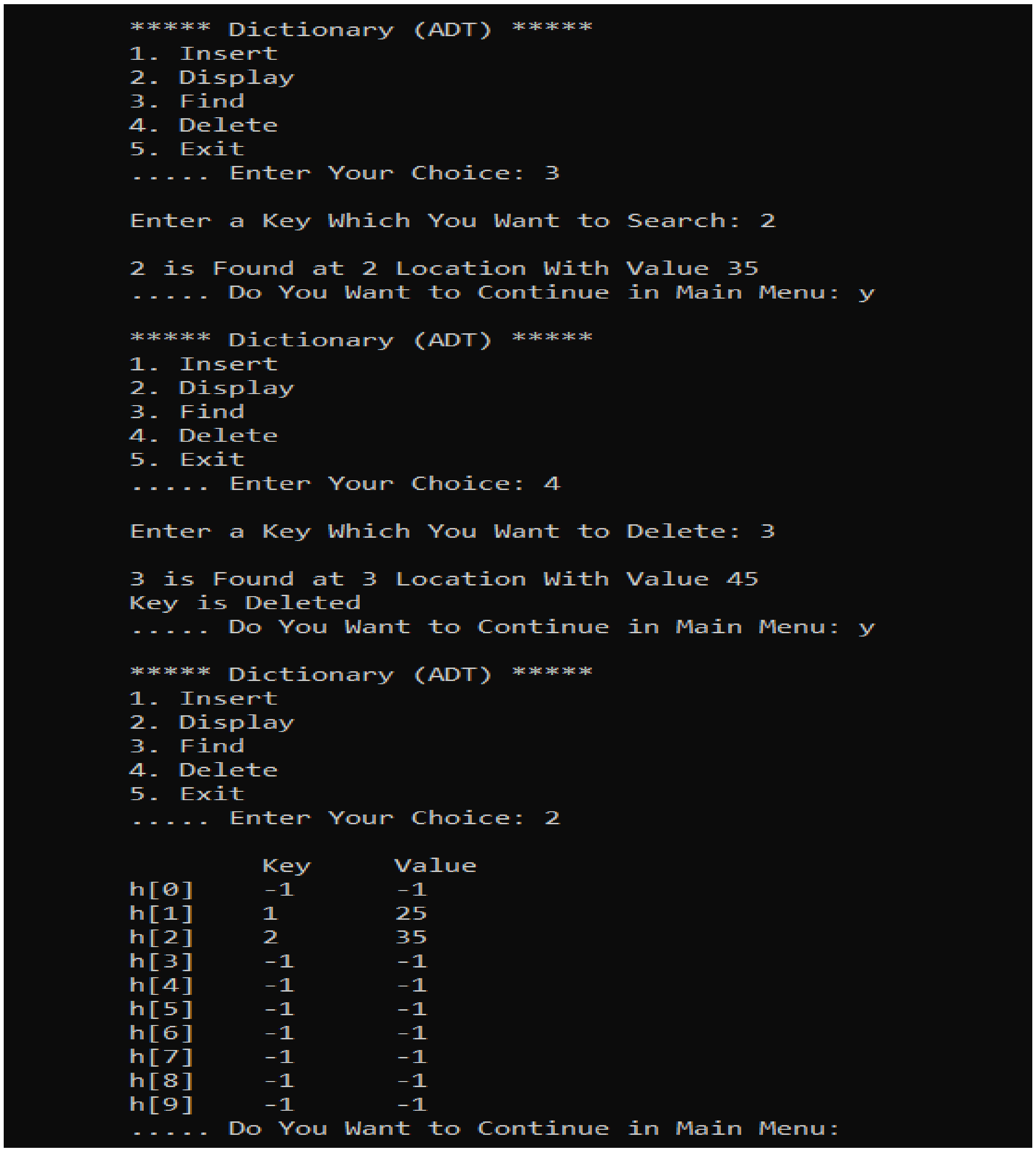
```
#include<iostream>
#include<string.h>
using namespace std;
class HashFunction{
     typedef struct hash{
  int key;
  int value;
 }hash;
 hash h[10];
    public:
 HashFunction();
 void insert();
 void display();
 int find(int);
 void Delete(int);
  };
HashFunction::HashFunction(){
 int i;
 for(i=0;i<10;i++){
  h[i].key=-1;
  h[i].value=-1; } }
void HashFunction::Delete(int k){
 int index=find(k);
 if(index==-1){
  cout<<"\n\tKey Not Found";
   }
 else{
  h[index].key=-1;
  h[index].value=-1;
  cout<<"\n\tKey is Deleted";
   }
  }
int HashFunction::find(int k){
 int i;
 for(i=0;i<10;i++){
  if(h[i].key==k){
   cout<<"\n\t"<<h[i].key<<" is Found at "<<i<<" Location With Value "<<h[i].value;
   return i;  } }
 if(i==10){
  return -1;} }
void HashFunction::display(){
 int i;
 cout<<"\n\t\tKey\tValue";
 for(i=0;i<10;i++){
  cout<<"\n\th["<<i<<"]\t"<<h[i].key<<"\t"<<h[i].value;} }
void HashFunction::insert(){
 char ans;
 int k,v,hi,cnt=0,flag=0,i;
 do{
  if(cnt>=10){
   cout<<"\n\tHash Table is FULL";
   break;}
  cout<<"\n\tEnter a Key: ";
  cin>>k;
  cout<<"\n\tEnter a Value: ";
  cin>>v;
  hi=k%10;// hash function
  if(h[hi].key==-1){
   h[hi].key=k;
   h[hi].value=v;}
      else{
   for(i=hi+1;i<10;i++){
    if(h[i].key==-1){
     h[i].key=k;
     h[i].value=v;
     flag=1;
     break; }}
   for(i=0;i<hi && flag==0;i++){
    if(h[i].key==-1){
     h[i].key=k;
     h[i].value=v;
     break;}  }}
      flag =0;
      cnt++;
      cout<<"\n\t..... Do You Want to Insert More Key: ";
      cin>>ans;
   }while(ans=='y'||ans=='Y'); }
 int main(){
 int ch,k,index;
 char ans;
 HashFunction obj;
 do{
  cout<<"\n\t***** Dictionary (ADT) *****";
  cout<<"\n\t1. Insert\n\t2. Display\n\t3. Find\n\t4. Delete\n\t5. Exit";
  cout<<"\n\t..... Enter Your Choice: ";
  cin>>ch;
  switch(ch){
   case 1:  obj.insert();
     break;
   case 2: obj.display();
```

```
     break;
   case 3: cout<<"\n\tEnter a Key Which You Want to Search: ";
     cin>>k;
     index=obj.find(k);
     if(index==-1){
      cout<<"\n\tKey Not Found"; }
     break;
   case 4: cout<<"\n\tEnter a Key Which You Want to Delete: ";
     cin>>k;
     obj.Delete(k);
     break;
   case 5:
     break; }
  cout<<"\n\t..... Do You Want to Continue in Main Menu: ";
  cin>>ans;
   }while(ans=='y'||ans=='Y');
}
```

Output:



```
***** Dictionary (ADT) *****
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 1

Enter a Key: 1

Enter a Value: 25

..... Do You Want to Insert More Key: y

Enter a Key: 2

Enter a Value: 35

..... Do You Want to Insert More Key: y

Enter a Key: 3

Enter a Value: 45

..... Do You Want to Insert More Key: n

..... Do You Want to Continue in Main Menu: y

***** Dictionary (ADT) *****
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 2

        Key     Value
h[0]    -1      -1
h[1]    1       25
h[2]    2       35
h[3]    3       45
h[4]    -1      -1
h[5]    -1      -1
h[6]    -1      -1
h[7]    -1      -1
h[8]    -1      -1
h[9]    -1      -1
..... Do You Want to Continue in Main Menu: y
```



```
***** Dictionary (ADT) *****
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 3

Enter a Key Which You Want to Search: 2

2 is Found at 2 Location With Value 35
..... Do You Want to Continue in Main Menu: y

***** Dictionary (ADT) *****
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 4

Enter a Key Which You Want to Delete: 3

3 is Found at 3 Location With Value 45
Key is Deleted
..... Do You Want to Continue in Main Menu: y

***** Dictionary (ADT) *****
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 2

        Key     Value
h[0]    -1      -1
h[1]    1       25
h[2]    2       35
h[3]    -1      -1
h[4]    -1      -1
h[5]    -1      -1
h[6]    -1      -1
h[7]    -1      -1
h[8]    -1      -1
h[9]    -1      -1
..... Do You Want to Continue in Main Menu:
```

Experiment No-2

Aim: Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with / without replacement. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique. Standard Operations: Insert(key, value), Find(key), Delete(key)

Prerequisite: C++ Programming

Objective: To understand the use functions for Collision Handling Technique.

Input: Set of (key, Value) Pairs keys mapped Value

Theory:

What is Collision?
Since a hash function gets us a small number for a key which is a big integer or string, there is a possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision and must be handled using some collision handling technique.

How to handle Collisions?
There are mainly two methods to handle collision:

1) Separate Chaining

2) Open Addressing

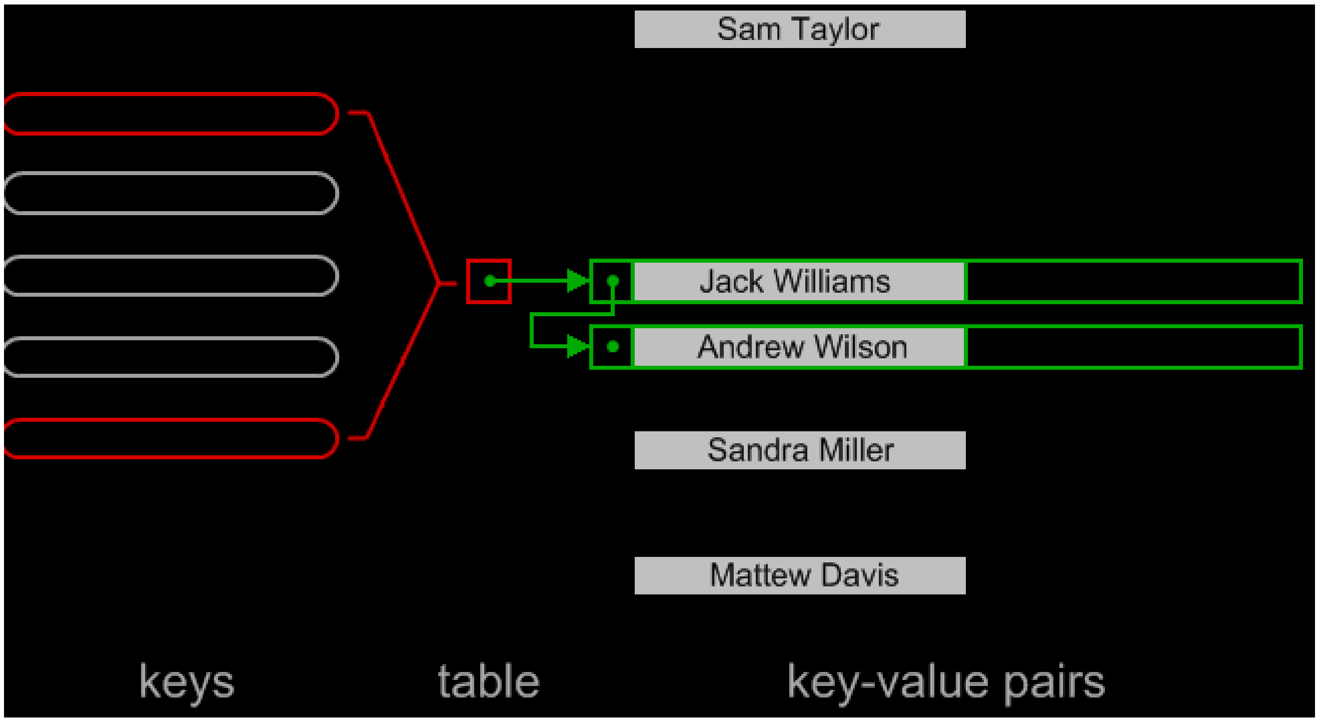
In this article, only separate chaining is discussed. We will be discussing Open addressing in the next post.

Separate Chaining:
The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

Hashing with Chaining

Hashing is a possible way to resolve collisions. Each slot of the array contains a link to a singly-linked list containing key-value pairs with the same hash. New key-value pairs are added to the end of the list. Lookup algorithm searches through the list to find matching key. Initially table slots contain nulls. List is being created, when value with the certain hash is added for the first time.

Chaining illustration



Complexity analysis

Assuming, that hash function distributes hash codes uniformly and table allows dynamic resizing, amortized complexity of insertion, removal and lookup operations is constant. Actual time, taken by those operations linearly depends on table's load factor.

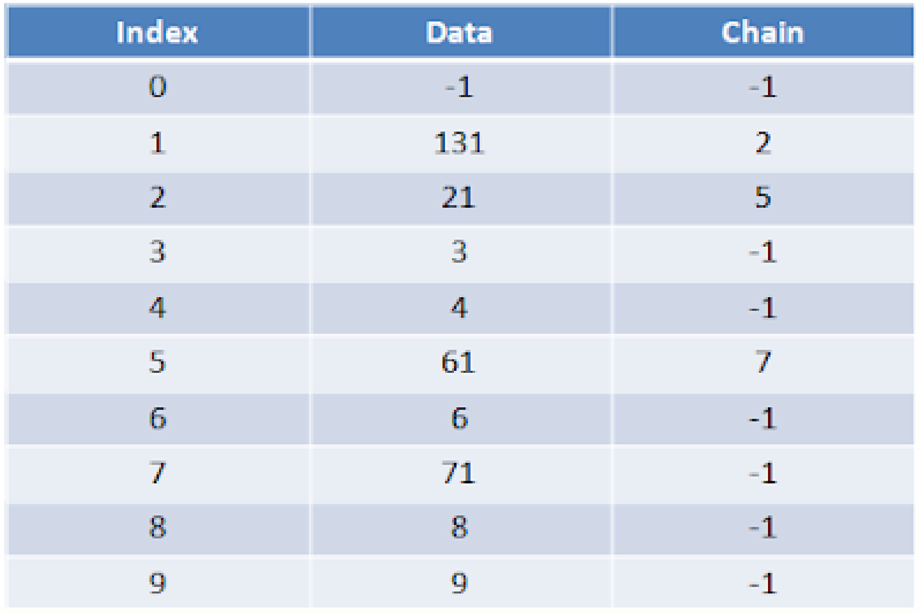
Note. Even substantially overloaded hash table, based on chaining, shows well performance. Assume hash table with 1000 slots storing 100000 items (load factor is 100). It requires a bit more memory (size of the table), than a singly-linked list, but all basic operations will be done about 1000 times faster on average. Draw attention, that computational complexity of both singly-linked list and constant-sized hash table is O(n).

Chaining Without Replacement

chaining is a concept which introduces an additional field with data i.e. chain. A separate chain table is maintained for colliding data. When collision occurs, we store the second colliding data by linear probing method. The address of this colliding data can be stored with the first colliding element in the chain table, without replacement.

For example, consider elements:

131, 3, 4, 21, 61, 6, 71, 8, 9



Index	Data	Chain
0	-1	-1
1	131	2
2	21	5
3	3	-1
4	4	-1
5	61	7
6	6	-1
7	71	-1
8	8	-1
9	9	-1

From the example, you can see that the chain is maintained from the number who demands for location 1. First number 131 comes, we place it at index 1. Next comes 21, but collision occurs so by linear probing we will place 21 at index 2, and chain is maintained by writing 2 in chain table at index 1. Similarly, next comes 61, by linear probing we can place 61 at index 5 and chain will be maintained at index 2. Thus, any element which gives hash key as 1 will be stored by linear probing at empty location but a chain is maintained so that traversing the hash table will be efficient.

Program:

```
#include<iostream>
#include<string.h>
using namespace std;
class HashFunction{
     typedef struct hash{
  int key;
  int value;
 }hash;
 hash h[10];
    public:
 HashFunction();
 void insert();
 void display();
 int find(int);
 void Delete(int);
  };
HashFunction::HashFunction(){
 int i;
 for(i=0;i<10;i++){
  h[i].key=-1;
  h[i].value=-1; } }
void HashFunction::Delete(int k){
 int index=find(k);
 if(index==-1){
  cout<<"\n\tKey Not Found";
```

```
   }
 else{
  h[index].key=-1;
  h[index].value=-1;
  cout<<"\n\tKey is Deleted";
   }
  }
int HashFunction::find(int k){
 int i;
 for(i=0;i<10;i++){
  if(h[i].key==k){
   cout<<"\n\t"<<h[i].key<<" is Found at "<<i<<" Location With Value "<<h[i].value;
   return i;  } }
 if(i==10){
  return -1;} }
void HashFunction::display(){
 int i;
 cout<<"\n\t\tKey\tValue";
 for(i=0;i<10;i++){
  cout<<"\n\th["<<i<<"]\t"<<h[i].key<<"\t"<<h[i].value;} }
void HashFunction::insert(){
 char ans;
 int k,v,hi,cnt=0,flag=0,i;
 do{
  if(cnt>=10){
   cout<<"\n\tHash Table is FULL";
 break;}
cout<<"\n\tEnter a Key: ";
cin>>k;
cout<<"\n\tEnter a Value: ";
cin>>v;
hi=k%10;// hash function
if(h[hi].key==-1){
 h[hi].key=k;
 h[hi].value=v;}
    else{
 for(i=hi+1;i<10;i++){
  if(h[i].key==-1){
   h[i].key=k;
   h[i].value=v;
   flag=1;
   break; }}
 for(i=0;i<hi && flag==0;i++){
  if(h[i].key==-1){
   h[i].key=k;
   h[i].value=v;
   break;}  }}
    flag =0;
    cnt++;
    cout<<"\n\t..... Do You Want to Insert More Key: ";
    cin>>ans;
 }while(ans=='y'||ans=='Y'); }
```

```
int main(){
 int ch,k,index;
 char ans;
 HashFunction obj;
 do{
  cout<<"\n\t***** Dictionary (ADT) *****";
  cout<<"\n\t1. Insert\n\t2. Display\n\t3. Find\n\t4. Delete\n\t5. Exit";
  cout<<"\n\t..... Enter Your Choice: ";
  cin>>ch;
  switch(ch){
   case 1:  obj.insert();
     break;
   case 2: obj.display();
     break;
   case 3: cout<<"\n\tEnter a Key Which You Want to Search: ";
     cin>>k;
     index=obj.find(k);
     if(index==-1){
      cout<<"\n\tKey Not Found"; }
     break;
   case 4: cout<<"\n\tEnter a Key Which You Want to Delete: ";
     cin>>k;
     obj.Delete(k);
     break;
   case 5:
     break; }
  cout<<"\n\t..... Do You Want to Continue in Main Menu: ";
  cin>>ans;
   }while(ans=='y'||ans=='Y');
}
```

Output:

```
***** Dictionary (ADT) *****
        1. Insert
        2. Display
        3. Find
        4. Delete
        5. Exit
        ..... Enter Your Choice: 1
        Enter a Key: 1
        Enter a Value: 45
        ..... Do You Want to Insert More Key: y
        Enter a Key: 2
        Enter a Value: 79
        ..... Do You Want to Insert More Key: y
        Enter a Key: 3
        Enter a Value: 57
        ..... Do You Want to Insert More Key: y
        Enter a Key: 4
        Enter a Value: 98
        ..... Do You Want to Insert More Key: y
        Enter a Key: 5
        Enter a Value: 76
        ..... Do You Want to Insert More Key: y
```

```
Enter a Key: 6

Enter a Value: 32

..... Do You Want to Insert More Key: n

..... Do You Want to Continue in Main Menu: y

***** Dictionary (ADT) *****

1. Insert

2. Display

3. Find

4. Delete

5. Exit

..... Enter Your Choice: 2

      Key   Value

h[0]   -1    -1

h[1]   1     45

h[2]   2     79

h[3]   3     57

h[4]   4     98

h[5]   5     76

h[6]   6     32

h[7]   -1    -1

h[8]   -1    -1

h[9]   -1    -1

..... Do You Want to Continue in Main Menu: y

***** Dictionary (ADT) *****

1. Insert

2. Display

3. Find

4. Delete
```

```
5. Exit

..... Enter Your Choice: 3

Enter a Key Which You Want to Search: 5

5 is Found at 5 Location With Value 76

..... Do You Want to Continue in Main Menu: y

***** Dictionary (ADT) *****

1. Insert

2. Display

3. Find

4. Delete

5. Exit

..... Enter Your Choice: 4

Enter a Key Which You Want to Delete: 1

1 is Found at 1 Location With Value 45

Key is Deleted

..... Do You Want to Continue in Main Menu: y

***** Dictionary (ADT) *****

1. Insert

2. Display

3. Find

4. Delete

5. Exit

..... Enter Your Choice: 5

..... Do You Want to Continue in Main Menu: n
```

Group B

Experiment No-03

Aim: A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

Prerequisite: C++ Programming

Objective: To understand the use Terminology of Tree and Construct a tree

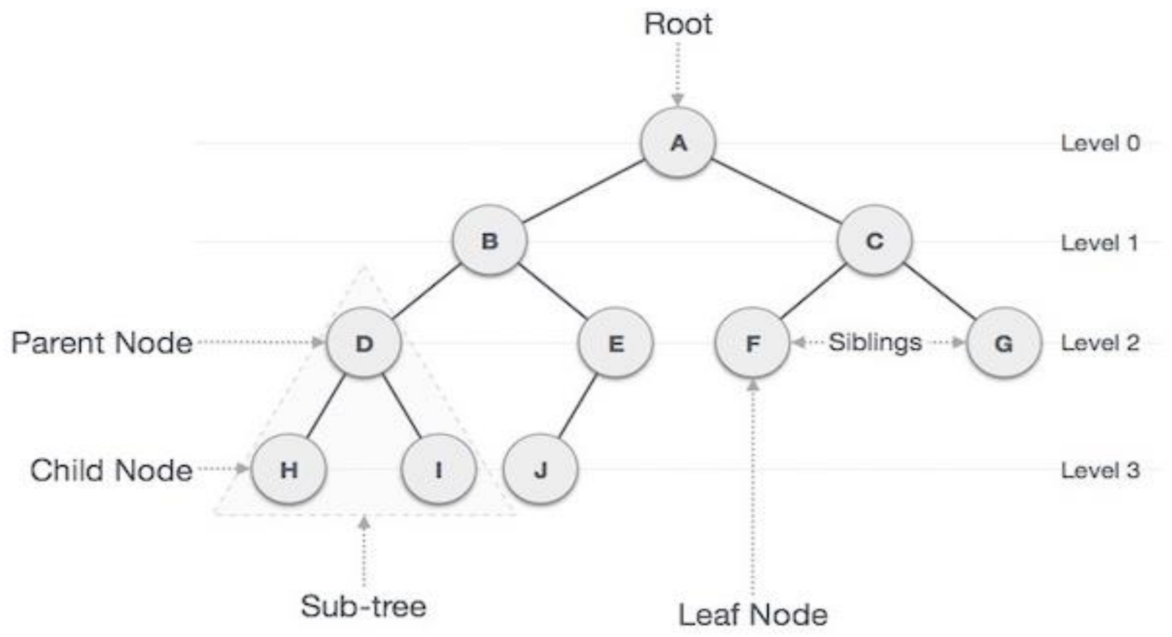
Input: A book consists of chapters, chapters consist of sections and sections consist of subsections.

Theory:

What is Tree

Tree represents the nodes connected by edges. We will discuss binary tree or binary search tree specifically.

Binary Tree is a special datastructure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.



Important Terms

Following are the important terms with respect to tree.

- **Path** − Path refers to the sequence of nodes along the edges of a tree.
- **Root** − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
- **Parent** − Any node except the root node has one edge upward to a node called parent.
- **Child** − The node below a given node connected by its edge downward is called its child node.
- **Leaf** − The node which does not have any child node is called the leaf node.
- **Subtree** − Subtree represents the descendants of a node.
- **Visiting** − Visiting refers to checking the value of a node when control is on the node.
- **Traversing** − Traversing means passing through nodes in a specific order.
- **Levels** − Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
- **keys** − Key represents a value of a node based on which a search operation is to be carried out for a node.

Print Nodes in Top View of Binary Tree

Top view of a binary tree is the set of nodes visible when the tree is viewed from the top. Given a binary tree, print the top view of it. The output nodes can be printed in any order.

A node x is there in output if x is the topmost node at its horizontal distance. Horizontal distance of left child of a node x is equal to horizontal distance of x minus 1, and that of right child is horizontal distance of x plus 1.

```
      1
    /   \
   2     3
  / \   / \
 4   5 6   7
```

Top view of the above binary tree is

4 2 1 3 7

```
      1
    /   \
  2       3
    \
      4
        \
          5
            \
              6
```

Top view of the above binary tree is

2 1 3 6

Program:

```
#include<iostream>

#include<stdlib.h>

#include<string.h>

using namespace std;

struct node{

 char name[20];

 node *next;

 node *down;

 int flag;

};

class Gll{

char ch[20]; int n,i;

 node *head=NULL,*temp=NULL,*t1=NULL,*t2=NULL;
 public:
 node *create();
 void insertb();
 void insertc();
 void inserts();
 void insertss();
 void displayb();
};
node *Gll::create(){
node *p=new(struct node);
 p->next=NULL;
 p->down=NULL;
 p->flag=0;
 cout<<"\n enter the name";
cin>>p->name;
 return p;}
void Gll::insertb() {
 if(head==NULL) {
 t1=create();
 head=t1;}
 else{
 cout<<"\n book exist"; } }
void Gll::insertc(){
 if(head==NULL) {
 cout<<"\n there is no book"; }
 else{
```

```
cout<<"\n how many chapters you want to insert";
cin>>n;
for(i=0;i<n;i++){
t1=create();
if(head->flag==0){
head->down=t1; head->flag=1;
}
else {
temp=head;
temp=temp->down;
while(temp->next!=NULL)
temp=temp->next;
temp->next=t1;}}}}
void Gll::inserts() {
if(head==NULL)  {
cout<<"\n there is no book"; }
else{
cout<<"\n Enter the name of chapter on which you want to enter the section";
cin>>ch;
temp=head;
if(temp->flag==0) {
cout<<"\n their are no chapters on in book";}
else{
temp=temp->down;
while(temp!=NULL){
if(!strcmp(ch,temp->name)) {
```

```
cout<<"\n how many sections you want to enter";
 cin>>n;
 for(i=0;i<n;i++){
 t1=create();
 if(temp->flag==0) {
 temp->down=t1;
 temp->flag=1; cout<<"\n**";
 t2=temp->down;
 }
 else{
 cout<<"\n#####";
 while(t2->next!=NULL){
 t2=t2->next; }
 t2->next=t1; } }
 break; }
 temp=temp->next; } } }
}
void Gll::insertss(){
 if(head==NULL){
 cout<<"\n there is no book"; }
else{
cout<<"\n Enter the name of chapter on which you want to enter the section"; //ask for chapter
 cin>>ch;
temp=head;
 if(temp->flag==0){
 cout<<"\n there are no chapters in book";}
```

```
else {
temp=temp->down;
while(temp!=NULL){
if(!strcmp(ch,temp->name)) {
cout<<"\n enter name of section in which you want to enter the sub section";
cin>>ch;
if(temp->flag==0){
cout<<"\n there are no sections "; }
else{
temp=temp->down;
while(temp!=NULL){
if(!strcmp(ch,temp->name)){
cout<<"\n how many subsections you want to enter";
cin>>n;
for(i=0;i<n;i++){
t1=create();
if(temp->flag==0) {
temp->down=t1;
temp->flag=1; cout<<"\n**";
t2=temp->down; }
else {
cout<<"\n#####";
while(t2->next!=NULL){
t2=t2->next; }
t2->next=t1; } }
break; }
```

```
 temp=temp->next; }} }
 temp=temp->next; }} }}
void Gll::displayb(){
if(head==NULL){
cout<<"\n book not exist";}
 else{
 temp=head;
cout<<"\n NAME OF BOOK: "<<temp->name;
 if(temp->flag==1){
 temp=temp->down;
 while(temp!=NULL){
cout<<"\n\t\tNAME OF CHAPTER: "<<temp->name;
 t1=temp;
 if(t1->flag==1){
 t1=t1->down;
 while(t1!=NULL){
cout<<"\n\t\t\t\tNAME OF SECTION: "<<t1->name;
 t2=t1;
 if(t2->flag==1){
 t2=t2->down;
 while(t2!=NULL){
cout<<"\n\t\t\t\t\t\t NAME OF SUBSECTION: "<<t2->name;
 t2=t2->next;}}
 t1=t1->next;}}
 temp=temp->next;}}}}
int main(){
```

```
Gll g;
int x;
 while(1)  {
cout<<"\n\n enter your choice";
 cout<<"\n 1.insert book";
 cout<<"\n 2.insert chapter";
 cout<<"\n 3.insert section";
 cout<<"\n 4.insert subsection";
 cout<<"\n 5.display book";
 cout<<"\n 6.exit";
 cin>>x;
 switch(x){
case 1: g.insertb();
 break;
 case 2: g.insertc();
 break;
 case 3: g.inserts();
 break;
 case 4: g.insertss();
 break;
 case 5: g.displayb();
 break;
 case 6: exit(0);
 }
 } return 0;
}
```

Output:

```
Enter your choice
 1.Insert book
 2.Insert chapter
 3.Insert section
4.Insert subsection
 5.Display book
 6.Exit
Enter Your Choice:1
Enter the name:DSA
 Enter your choice
1.Insert book
 2.Insert chapter
3.Insert section
 4.Insert subsection
5.Display book
6.Exit
Enter Your Choice:2
 How many chapters you want to insert:2
 Enter the name: hashing
Enter the name:Tree
 Enter your choice
1.Insert book
 2.Insert chapter
 3.Insert section
 4.Insert subsection
 5.Display book
```

```
6.Exit

Enter Your Choice:3

Enter the name of chapter on which you want to enter the section:hashing

How many sections you want to enter:3 Enter the name:collisionresolution

 **

 Enter the name:hashingfunc

 #####

Enter the name:hashingtypes

 #####

Enter your choice

 1.Insert book

 2.Insert chapter

3.Insert section

 4.Insert subsection

 5.Display book

6.Exit

 Enter Your Choice:4

Enter the name of chapter on which you want to enter the section:hashing

 Enter name of section in which you want to enter the sub section:collisionresolution

 How many subsections you want to enter:2

 Enter the name:openHashing

 **

Enter the name:closeHashing

 #####

 Enter your choice

 1.Insert book

2.Insert chapter

 3.Insert section
 4.Insert subsection
 5.Display book
 6.Exit
 Enter Your Choice:5
NAME OF BOOK: DSA
NAME OF CHAPTER: hashing
NAME OF SECTION: collisionresolution
 NAME OF SUBSECTION: openHashing
 NAME OF SUBSECTION: closeHashing
 NAME OF SECTION: hashingfunc
 NAME OF SECTION: hashingtypes
 NAME OF CHAPTER: Tree
Enter your choice
 1.Insert book
 2.Insert chapter
3.Insert section
 4.Insert subsection
5.Display book
 6.Exit
Enter Your Choice:6
```

Experiment No-4

Aim - Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree -
i. Insert new node, ii. Find number of nodes in longest path from root, iii. Minimum data value found in the tree, iv. Change a tree so that the roles of the left and right pointers are swapped at every node, v. Search a value

Prerequisite: C++ Programming

Objective: To understand the use functions for Binary Search Tree

Input: i. Insert new node
ii. Find number of nodes in longest path from root
iii. Minimum data value found in the tree,
iv. Change a tree so that the roles of the left and right pointers are swapped at every node,
v. Search a value

Theory

The following is the definition of Binary Search Tree(BST) according to Wikipedia Binary Search Tree is a n ode-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree. There must be no duplicate nodes.

The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search for a given key.

Searching a key

For searching a value, if we had a sorted array we could have performed a binary search. Let's say we want to search a number in the array what we do in binary search is we first define the complete list as our search space, the number can exist only within the search space. Now we compare the number to be searched or the element to be searched with the mid element of the search space or the median and if the record being searched is lesser we go searching in the left half else we go searching in the right half, in case of equality we have found the element. In binary search we start with **‘n’** elements in search space and then if the mid element is not the element that we are looking for, we reduce the search space to **‘n/2’** and we go on reducing the search space till we either find the record that we are looking for or we get to only one element in Search operation in binary search tree will be very similar. Let’s say we want to search for the number, what we’ll do is we’ll start at the root, and then we will compare the value to be searched with the value of the root if it’s equal we are done with the search if it’s lesser we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are lesser and all the elements in the right subtree are greater. Searching an element in the binary search tree is basically this traversal in which at each step we will go either towards left or right and hence in at each step we discard one of the sub-trees. If the tree is balanced, we call a tree balanced if for all nodes the difference between the heights of left and right subtrees is not greater than one, we will start with a search space of **‘n’**nodes and when we will discard one of the sub-trees we will discard **‘n/2’** nodes so our search space will be reduced to **‘n/2’** and then in the next

Illustration to search 6 in below tree:

1. Start from the root.

2. Compare the searching element with root, if less than root, then recurse for left, else recurse for right.

3. If the element to search is found anywhere, return true, else return false.

Insertion of a key

A new key is always inserted at the leaf. We start searching a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

```
       100                         100
      /   \      Insert 40        /    \
    20     500    --------->        20     500
   /  \                       /  \
  10   30                   10   30
                                  \
                                  40
```

Program:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
class bstnode{
    public:
    int data;
    bstnode *left,*right;
    bstnode(int x){
        data=x;
        left=right=NULL;
    }
};
class bst{
    bstnode*root;
    public:
    bst(){
        root=NULL;
    }
        bstnode*create();
        void insert(int x);
        bstnode*find(int x);
        bstnode*minvalue(bstnode*root);
        bstnode*maxvalue(bstnode*root);
        int longest_path(bstnode*T);
        void display(bstnode*t);
        bstnode*mirror(bstnode*t);
};
bstnode*bst::create(){
    int x,i,n;
    root=NULL;
    cout<<"Enter total number of nodes:";
    cin>>n;
    cout<<"Enter tree value:";
    for(i=0;i<n;i++){
        cin>>x;
        insert(x);
    }
    return(root);
}
void bst::insert(int x){
    bstnode *p,*q,*r;
    r=new bstnode (x);
    if(root==NULL){
        root=r;
        return;
    }
    p=root;
    while(p!=NULL){
        q=p;
        if(x>p->data)
            p=p->right;
        else
            p=p->left;
    }
    if(x>q->data)
        q->right=r;
    else
        q->left=r;
}
bstnode*bst::find(int x){
    while(root!=NULL){
        if(x==root->data)
        return (root);
        if(x>root->data)
        root=root->right;
        else
        root=root->left;
    }
    return NULL;
}
bstnode *bst:: minvalue(bstnode*root){
    while(root->left!=NULL){
        root=root->left;
    }
  cout<<root->data;
}
bstnode *bst:: maxvalue(bstnode*root) {
    while(root->right!=NULL)
     {
        root=root->right;
    }
   cout<<root->data;
 }
int bst::longest_path(bstnode*T){
    int hl,hr;
    if(T==NULL)
    return(0);
    if(T->left==NULL && T->right==NULL)
    return(0);
    hl=longest_path(T->left);
    hr=longest_path(T->right);
    if(hl>hr){
        return(hl+1);
    }
    else{
        return(hr+1);
    }
}
void bst::display(bstnode *t){
    if(t!=NULL){
        display(t->left);
        cout<<"\t"<<t->data;
        display(t->right);
    }
}
bstnode*mirror(bstnode*t)
{
    bstnode*temp;
    if(t!=NULL)
    {
        temp=t->left;
        t->left=mirror(t->right);
        t->right=mirror(temp);
    }
    return(t);
}
int main(){
    int ch,x,i;
    bst b;
    bstnode*p,*q,*root;
    do{
        cout<<"\n1.Create \n2.Find \n3.Find_min
\n4.Find_max\n5.Longest_path\n6.Display\n7.Mirror\n8.Exit";
        cout<<"\nEnter your choice : ";
        cin>>ch;
        switch(ch){
            case 1:
            root=b.create();
            break;
            case 2:
            cout<<"Enter node to be searched ";
            cin>>x;
            p=b.find(x);
            if(p==NULL)
            cout<<"\nNode not found ";
            else
            cout<<"Node found"<<p->data;
            break;
            case 3:
            cout<<"\n The minimum value: ";
            b.minvalue(root);
            break;
            case 4:
            cout<<"\n The maximum value: ";
            b.maxvalue(root);
            break;
            case 5:
            i=b.longest_path(root);
            cout<<" Longest path in tree: "<<i+1;
            break;
            case 6:
            b.display(root);
            break;
            case 7:
            root = mirror(root);
            b.display(root);
            break;
            case 8:
            exit(0);
            break;
        }
    }while(ch!=9);
    return 0;
}
```

Output:

```
1.Create
2.Find
3.Find_min
4.Find_max
5.Longest_path
6.Display
7.Mirror
8.Exit
Enter your choice : 1
Enter total number of nodes:6
Enter tree value:10
43
2
65
45
32
```

```
1.Create
2.Find
3.Find_min
4.Find_max
5.Longest_path
6.Display
7.Mirror
8.Exit
Enter your choice : 2
Enter node to be searched 32
Node found32
1.Create
2.Find
3.Find_min
4.Find_max
5.Longest_path
6.Display
7.Mirror
8.Exit
Enter your choice : 3
 The minimum value: 2
1.Create
2.Find
3.Find_min
4.Find_max
5.Longest_path
6.Display
7.Mirror
8.Exit
Enter your choice : 4
 The maximum value: 65
1.Create
2.Find
3.Find_min
4.Find_max
5.Longest_path
6.Display
7.Mirror
8.Exit
Enter your choice : 5
 Longest path in tree: 4
1.Create
2.Find
3.Find_min
4.Find_max
5.Longest_path
```

```
6.Display
7.Mirror
8.Exit
Enter your choice : 6
        2       10      32      43      45      65
1.Create
2.Find
3.Find_min
4.Find_max
5.Longest_path
6.Display
7.Mirror
8.Exit
Enter your choice : 7
        65      45      43      32      10      2
1.Create
2.Find
3.Find_min
4.Find_max
5.Longest_path
6.Display
7.Mirror
8.Exit
Enter your choice : 8
```

Experiment No-05

Aim: Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm

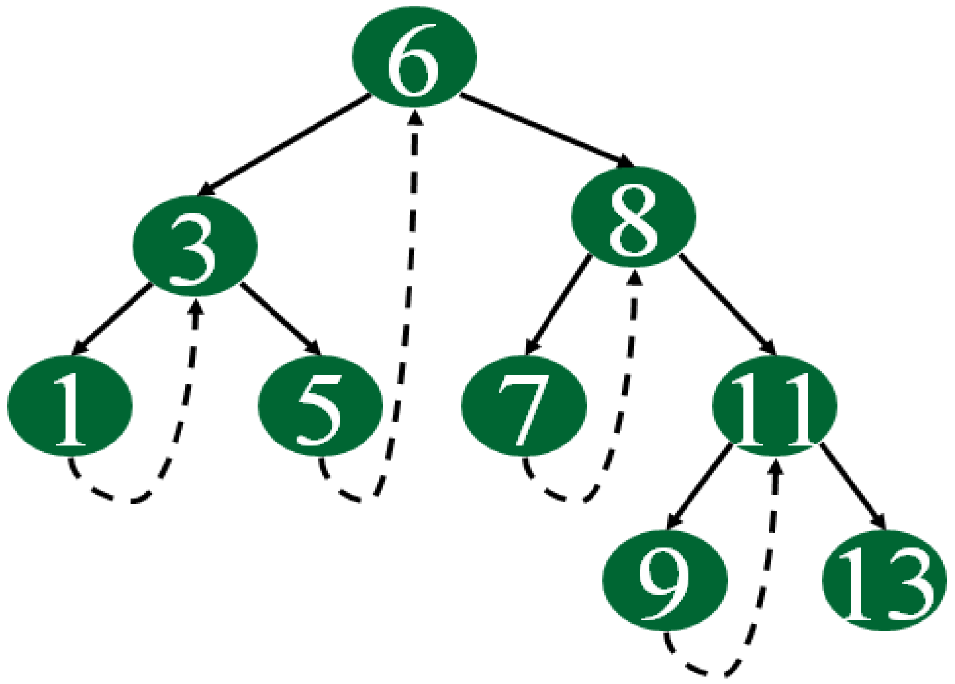
Prerequisite: C++ Programming

Objective: To Convert general tree to Binary tree.

Input: General tree

Convert a Binary Tree to Threaded binary tree:

Idea of Threaded Binary Tree is to make inorder traversal faster and do it without stack and without recursion. In a simple threaded binary tree, the NULL right pointers are used to store inorder successor. Where-ever a right pointer is NULL, it is used to store inorder successor. Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



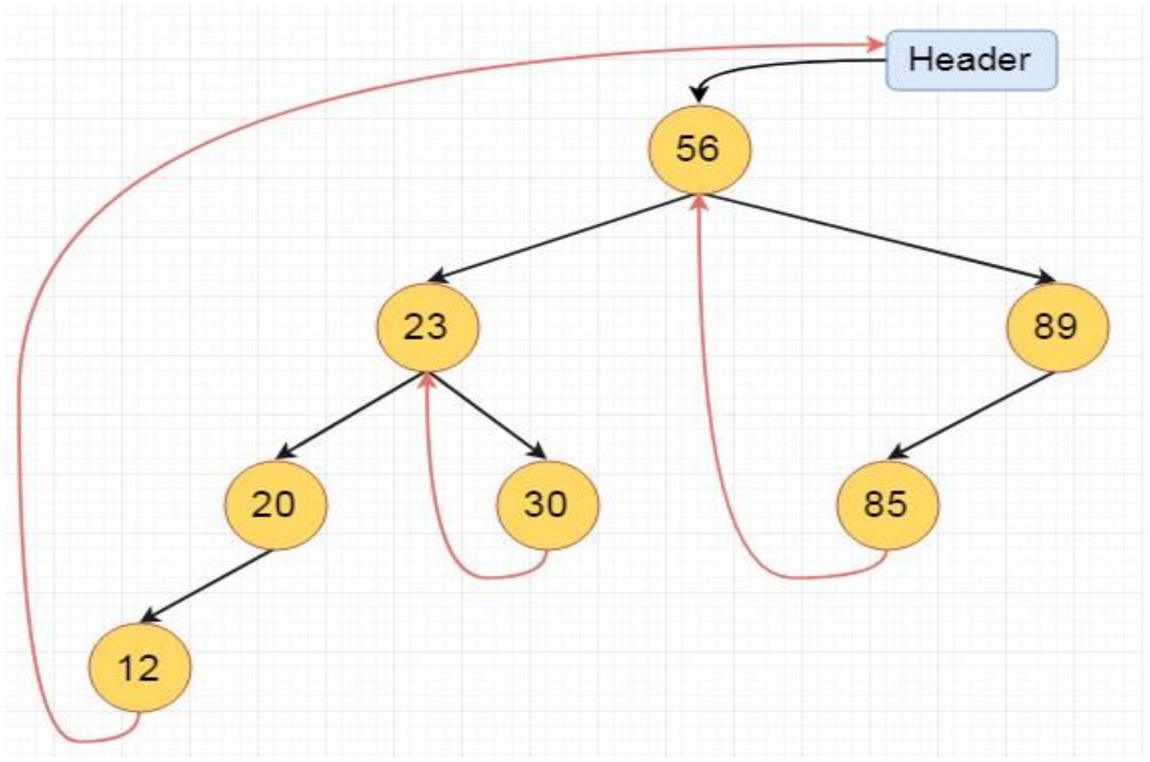
Following is structure of a single-threaded binary tree.

Here we will see the threaded binary tree data structure. We know that the binary tree nodes may have at most two children. But if they have only one children, or no children, the link part in the linked list representation remains null. Using threaded binary tree representation, we can reuse that empty links by making some threads.

If one node has some vacant left or right child area, that will be used as thread. There are two types of threaded binary tree. The single threaded tree or fully threaded binary tree. In single threaded mode, there are another two variations. Left threaded and right threaded.

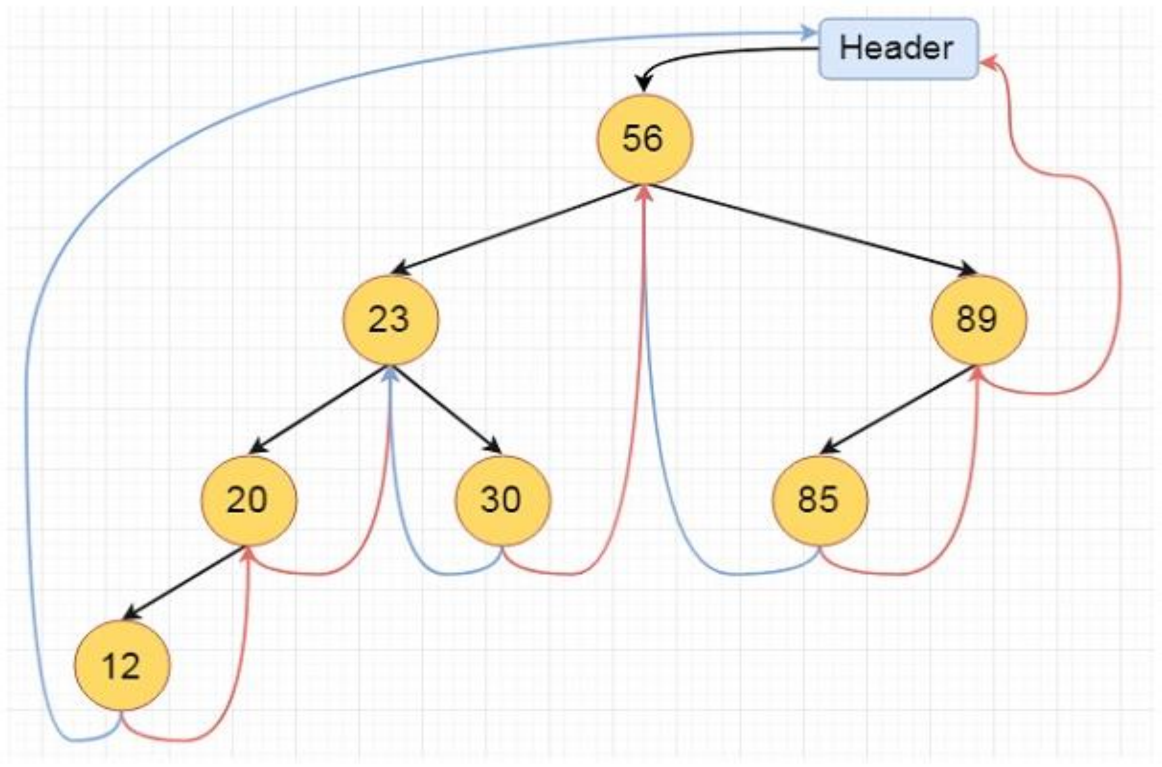
In the left threaded mode if some node has no left child, then the left pointer will point to its inorder predecessor, similarly in the right threaded mode if some node has no right child, then the right pointer will point to its inorder successor. In both cases, if no successor or predecessor is present, then it will point to header node.

For fully threaded binary tree, each node has five fields. Three fields like normal binary tree node, another two fields to store Boolean value to denote whether link of that side is actual link or thread.



These are the examples of left and right threaded tree

This is the fully threaded binary tree



Program:

```
#include <iostream>
#include <stack>
#include<string>
using namespace std;
class BTNODE {
BTNODE * left,*right;
char data;
public:
BTNODE() {
left=right=NULL;
data='\0';}
```

```
friend class ExpressionTree; };
class ExpressionTree{
BTNODE * root;
public:
ExpressionTree() {
root=NULL; }
void create(string);
void Inorder(BTNODE *);
void Preorder(BTNODE *);
void Postorder(BTNODE *);
void Delete_tree(BTNODE *);
friend int main(); };
void ExpressionTree::create(string s) {
stack<BTNODE *> s1;
BTNODE *T1,*T2,*T3;
int i;
for(i=s.length()-1;i>=0;i--) {
if(isalnum(s[i])) {
T1=new BTNODE;
T1->data=s[i];
s1.push(T1); }
else {
T3=s1.top();
s1.pop();
T2=s1.top();
s1.pop();
T1=new BTNODE;
```

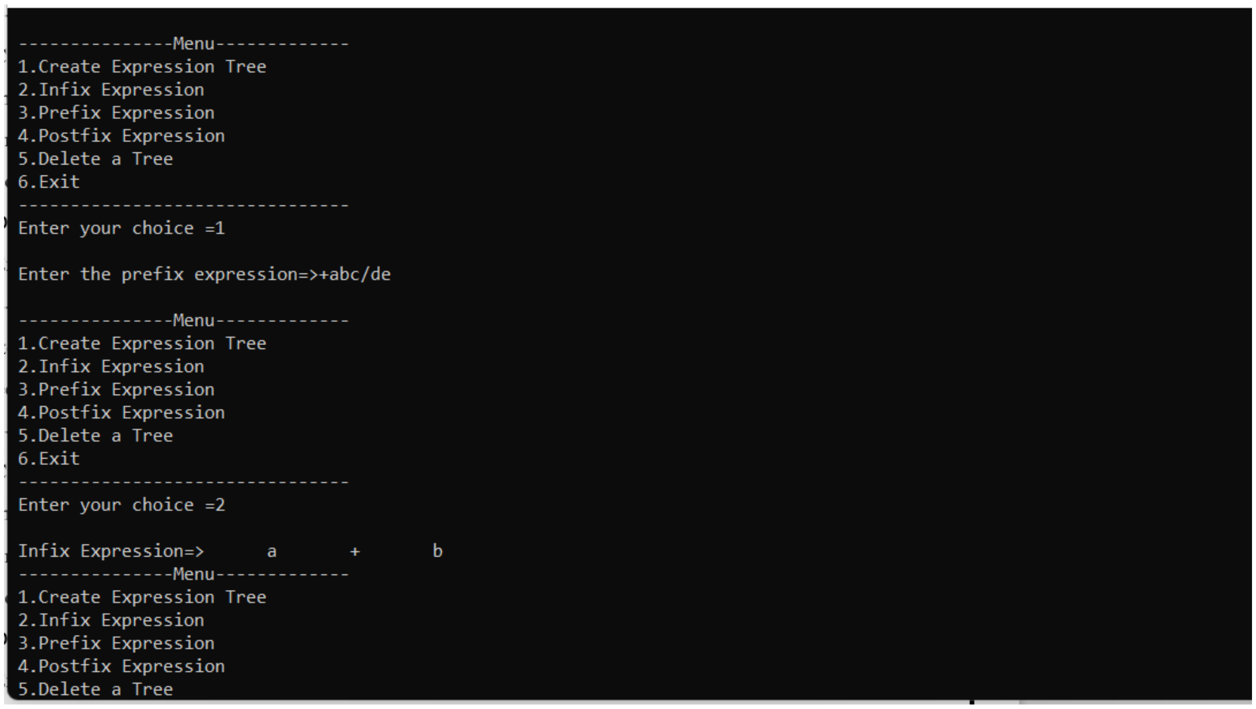
```
T1->data=s[i];
T1->left=T3;
T1->right=T2;
s1.push(T1); } }
root=s1.top();
s1.pop(); }
void ExpressionTree::Inorder(BTNODE *T) {
if(T!=NULL) {
Inorder(T->left);
cout<<"\t "<<T->data;
Inorder(T->right); } }
void ExpressionTree::Preorder(BTNODE *T) {
if(T!=NULL) {
cout<<"\t "<<T->data;
Preorder(T->left);
Preorder(T->right); } }
void ExpressionTree::Postorder(BTNODE *T) {
stack<BTNODE*> s2;
BTNODE * prev=NULL;
do {
while(T!=NULL) {
s2.push(T);
T=T->left; }
T=s2.top();
if(T->right==NULL ||T->right==prev) {
cout<<"\t "<<T->data;
prev=s2.top();
```

```
s2.pop();
T=NULL; }
else
T=T->right;
}while(!s2.empty()); }
void ExpressionTree::Delete_tree(BTNODE *T) {
if(T!=NULL) {
Delete_tree(T->left);
Delete_tree(T->right);
delete T; } }
int main() {
ExpressionTree E;
string s;
int ch;
do {
cout<<"\n ---------------Menu-------------";
cout<<"\n 1.Create Expression Tree";
cout<<"\n 2.Infix Expression";
cout<<"\n 3.Prefix Expression";
cout<<"\n 4.Postfix Expression";
cout<<"\n 5.Delete a Tree";
cout<<"\n 6.Exit";
cout<<"\n --------------------------------";
cout<<"\n Enter your choice =";
cin>>ch;
switch(ch) {
case 1:
```

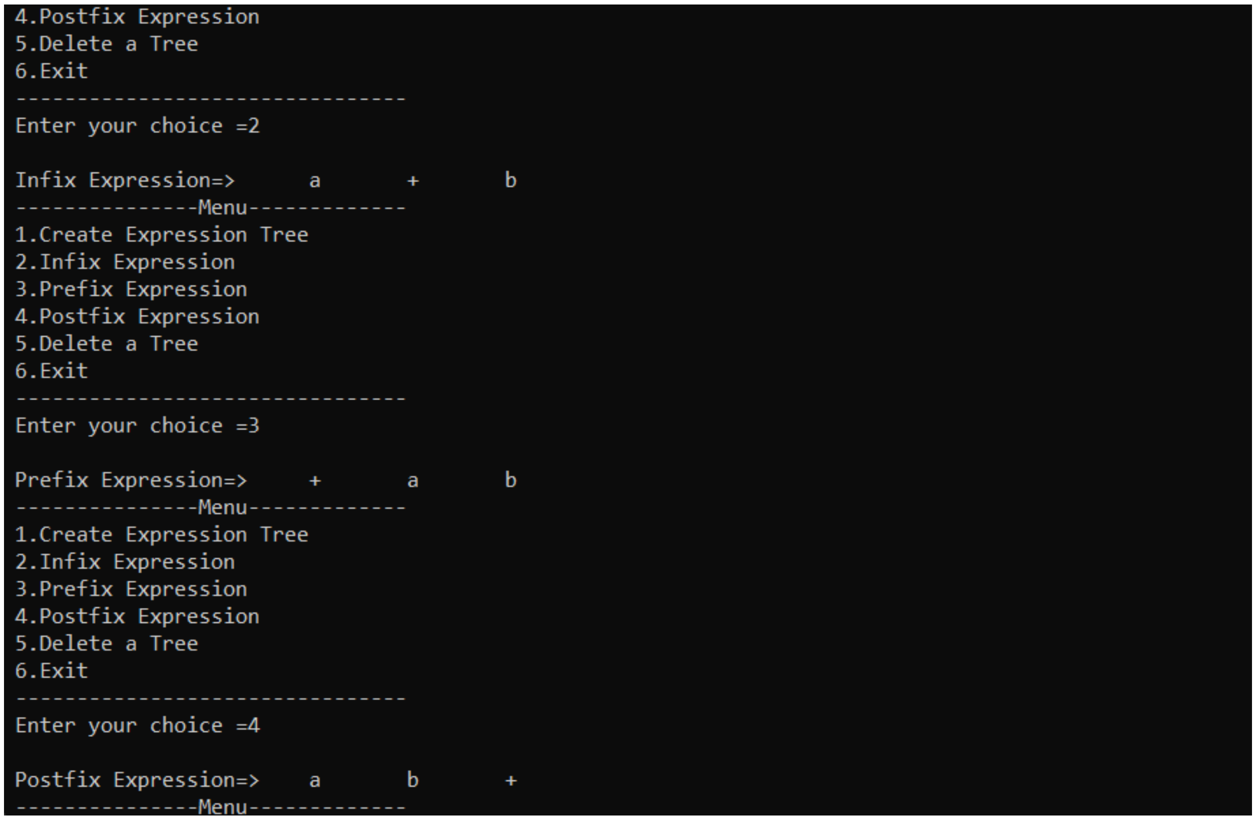
```
cout<<"\n Enter the prefix expression=>";
cin>>s;
E.create(s);
break;
case 2:
if(E.root==NULL)
cout<<"\n Tree is empty";
else {
cout<<"\n Infix Expression=>";
E.Inorder(E.root); }
break;
case 3:
if(E.root==NULL)
cout<<"\n Tree is empty";
else {
cout<<"\n Prefix Expression=>";
E.Preorder(E.root); }
break;
case 4:
if(E.root==NULL)
cout<<"\n Tree is empty";
else {
cout<<"\n Postfix Expression=>";
E.Postorder(E.root); }
break;
case 5:
if(E.root==NULL)
```

```
cout<<"\n Tree is empty";
else {
E.Delete_tree(E.root);
cout<<"\n Tree deleted";
E.root=NULL; }
break;
case 6:
break;
default:
cout<<"\n Invalid choice !"; }
}while(ch != 6);
return 0;
}
```

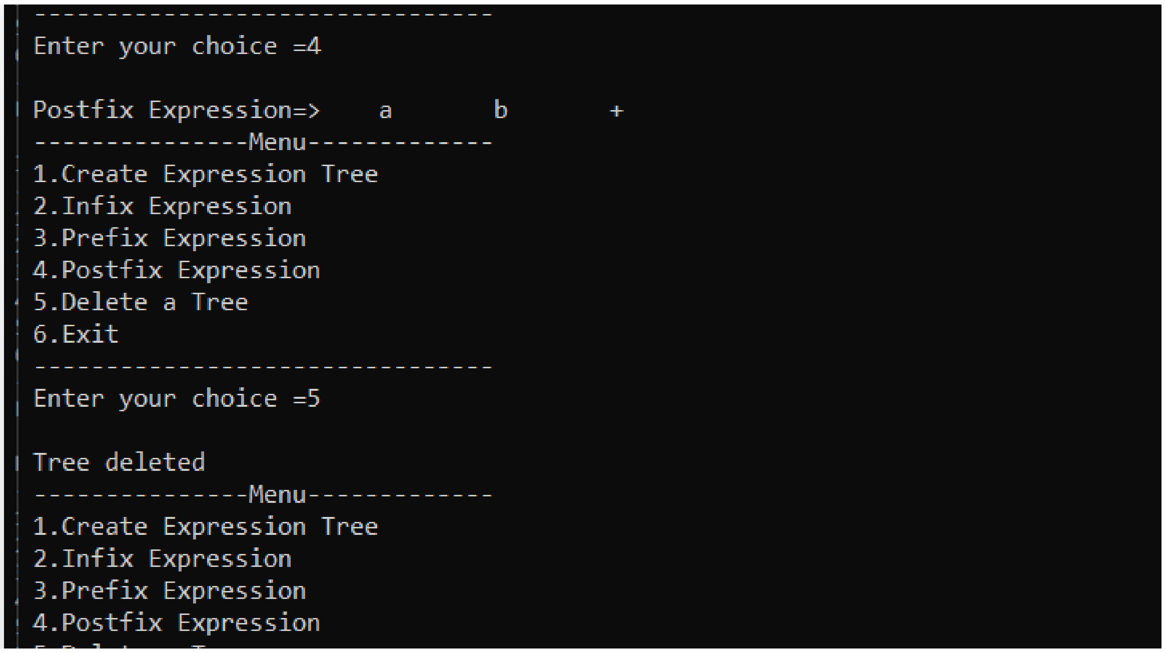
Output:



```
---------------Menu-------------
1.Create Expression Tree
2.Infix Expression
3.Prefix Expression
4.Postfix Expression
5.Delete a Tree
6.Exit
--------------------------------
Enter your choice =1

Enter the prefix expression=>+abc/de

---------------Menu-------------
1.Create Expression Tree
2.Infix Expression
3.Prefix Expression
4.Postfix Expression
5.Delete a Tree
6.Exit
--------------------------------
Enter your choice =2

Infix Expression=>      a       +       b
---------------Menu-------------
1.Create Expression Tree
2.Infix Expression
3.Prefix Expression
4.Postfix Expression
5.Delete a Tree
```



```
4.Postfix Expression
5.Delete a Tree
6.Exit
--------------------------------
Enter your choice =2

Infix Expression=>      a       +       b
---------------Menu-------------
1.Create Expression Tree
2.Infix Expression
3.Prefix Expression
4.Postfix Expression
5.Delete a Tree
6.Exit
--------------------------------
Enter your choice =3

Prefix Expression=>     +       a       b
---------------Menu-------------
1.Create Expression Tree
2.Infix Expression
3.Prefix Expression
4.Postfix Expression
5.Delete a Tree
6.Exit
--------------------------------
Enter your choice =4

Postfix Expression=>    a       b       +
---------------Menu-------------
```



```
--------------------------------
Enter your choice =4

Postfix Expression=>    a       b       +
---------------Menu-------------
1.Create Expression Tree
2.Infix Expression
3.Prefix Expression
4.Postfix Expression
5.Delete a Tree
6.Exit
--------------------------------
Enter your choice =5

Tree deleted
---------------Menu-------------
1.Create Expression Tree
2.Infix Expression
3.Prefix Expression
4.Postfix Expression
```

Group C

Experiment No-06

Aim: Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

Prerequisite: C++ Programming

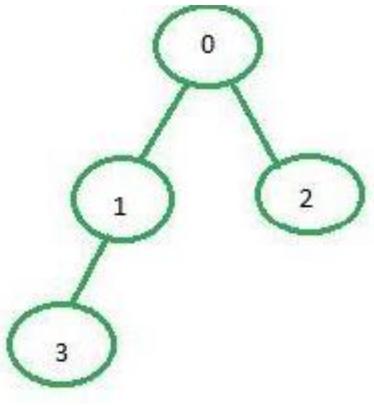
Objective: To understand the use functions for DFS and BFS

Input: Insert the elements

Theory

Breadth First Search (BFS) has been discussed in this article which uses adjacency list for the graph representation. In this article, adjacency matrix will be used to represent the graph.

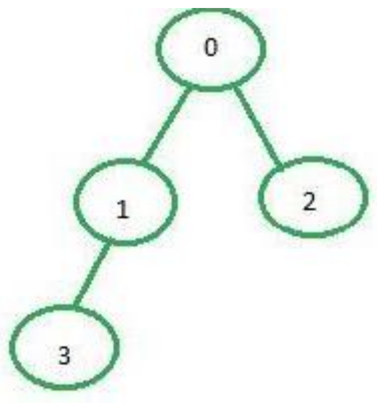
Adjacency matrix representation: In adjacency matrix representation of a graph, the matrix **mat[][]** of size n*n (where n is the number of vertices) will represent the edges of the graph where **mat[i][j] = 1** represents that there is an edge between the vertices **i** and **j** while **mat[i][j] = 0** represents that there is no edge between the vertices **i** and **j**.



Below is the adjacency matrix representation of the graph shown in the above image:

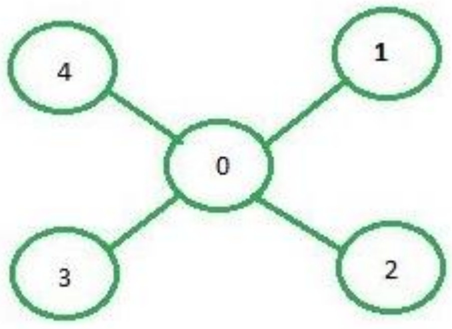
$$\begin{array}{ccccc}
 & 0 & 1 & 2 & 3 \\
0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 0 & 0 \\
3 & 0 & 1 & 0 & 0
\end{array}$$

Input: source = 0



Output: 0 1 2 3

Input: source = 1

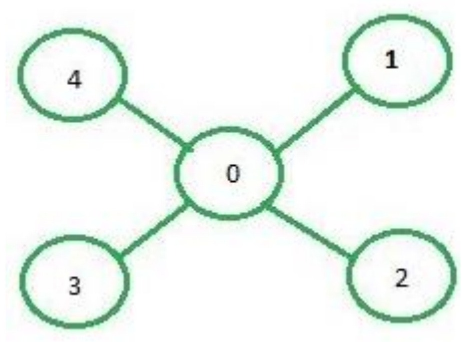


Output:1 0 2 3 4

Depth First Search (DFS)

DFS has been discussed in this article which uses adjacency list for the graph representation. In this article, adjacency matrix will be used to represent the graph.

Adjacency matrix representation: In adjacency matrix representation of a graph, the matrix **mat[][]** of size n*n (where n is the number of vertices) will represent the edges of the graph where **mat[i][j] = 1** represents that there is an edge between the vertices **i** and **j** while **mat[i][i] = 0** represents that there is no edge between the vertices **i** and **j**.

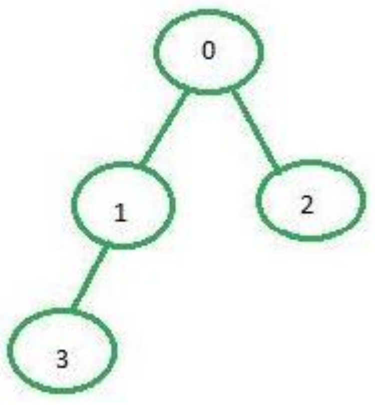


Below is the adjacency matrix representation of the graph shown in the above image:

	0	1	2	3	4
0	0	1	1	1	1
1	1	0	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	1	0	0	0	0

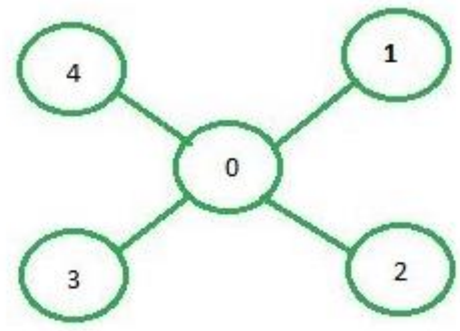
Examples:

Input: source = 0



Output: 0 1 3 2

Input: source = 0



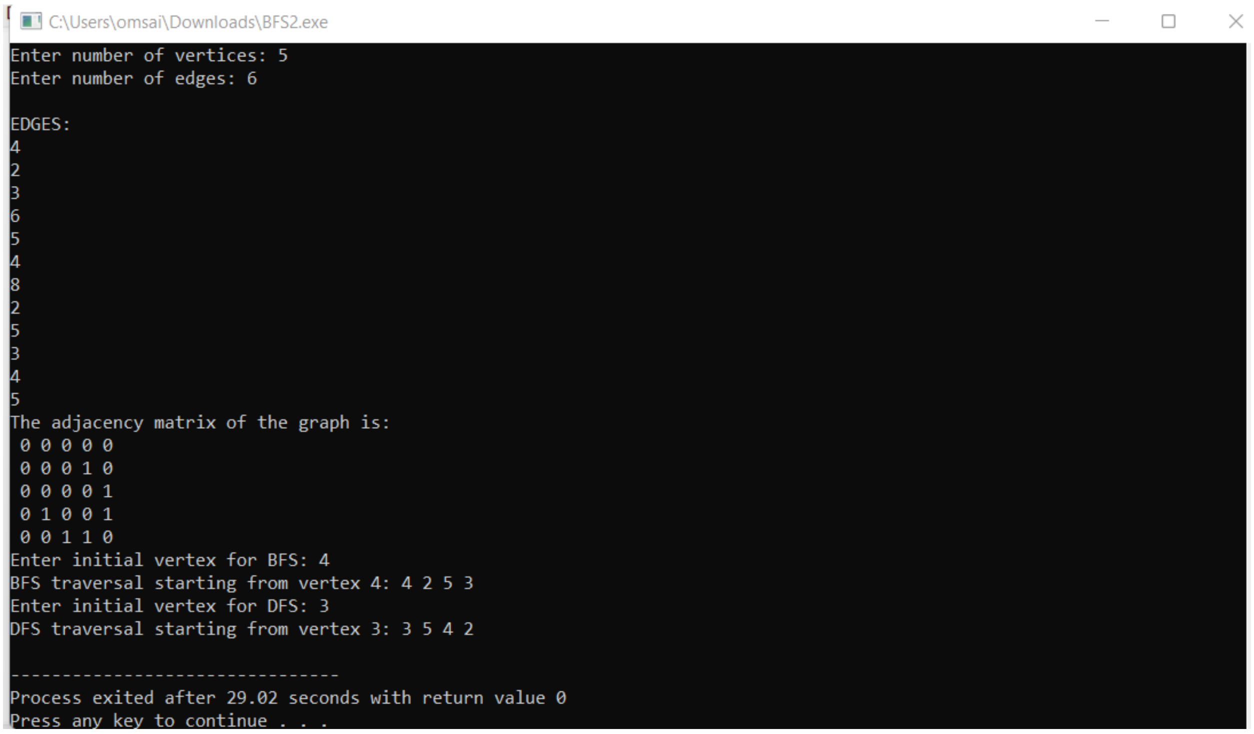
Output: 0 1 2 3 4

Program:

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int cost[10][10], i, j, k, n, qu[10], front = -1, rear = -1, v, visit[10] = {0}, visited[10] = {0};
int stk[10], top = -1, visit1[10] = {0}, visited1[10] = {0};

void bfs() {
    cout << "Enter initial vertex for BFS: ";
    cin >> v;
    cout << "BFS traversal starting from vertex " << v << ": ";
    cout << v << " ";
    visited[v] = 1;
    qu[++rear] = v;
    while (front != rear) {
        v = qu[++front];
        for (j = 1; j <= n; j++) {
            if (cost[v][j] != 0 && visited[j] == 0) {
                cout << j << " ";
                visited[j] = 1;
                qu[++rear] = j;
            }
        }
    }
    cout << endl;
}
void dfs() {
    cout << "Enter initial vertex for DFS: ";
    cin >> v;
    cout << "DFS traversal starting from vertex " << v << ": ";
    cout << v << " ";
    visited1[v] = 1;
    stk[++top] = v;
    while (top != -1) {
        v = stk[top--];
        for (j = 1; j <= n; j++) {
            if (cost[v][j] != 0 && visited1[j] == 0) {
                cout << j << " ";
                visited1[j] = 1;
                stk[++top] = j;
            }
        }
    }
    cout << endl;
}
```

```
int main() {
    int m;
    cout << "Enter number of vertices: ";
    cin >> n;
    cout << "Enter number of edges: ";
    cin >> m;
    cout << "\nEDGES:\n";
    for (k = 1; k <= m; k++) {
        cin >> i >> j;
        cost[i][j] = 1;
        cost[j][i] = 1;
    }
    // Display adjacency matrix
    cout << "The adjacency matrix of the graph is:\n";
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            cout << " " << cost[i][j];
        }
        cout << endl;
    }
    bfs();
    dfs();
    return 0;
}
```

Output:



C:\Users\omsai\Downloads\BFS2.exe

```
Enter number of vertices: 5
Enter number of edges: 6

EDGES:
4
2
3
6
5
4
8
2
5
3
4
5
The adjacency matrix of the graph is:
 0 0 0 0 0
 0 0 0 1 0
 0 0 0 0 1
 0 1 0 0 1
 0 0 1 1 0
Enter initial vertex for BFS: 4
BFS traversal starting from vertex 4: 4 2 5 3
Enter initial vertex for DFS: 3
DFS traversal starting from vertex 3: 3 5 4 2

--------------------------------
Process exited after 29.02 seconds with return value 0
Press any key to continue . . .
```

Experiment No-07

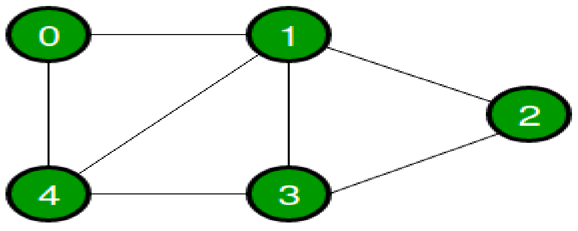
Aim: There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph.
Check whether the graph is connected or not. Justify the storage representation used.

Prerequisite: C++ Programming

Objective: To understand the use functions for adjacency list representation

Input: graph Image and weight.

Theory:
A graph is a data structure that consists of the following two components:
1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not the same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost. Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, and locale. See this for more applications of graph. Following is an example of an undirected graph with 5 vertices.



The following two are the most commonly used representations of a graph.

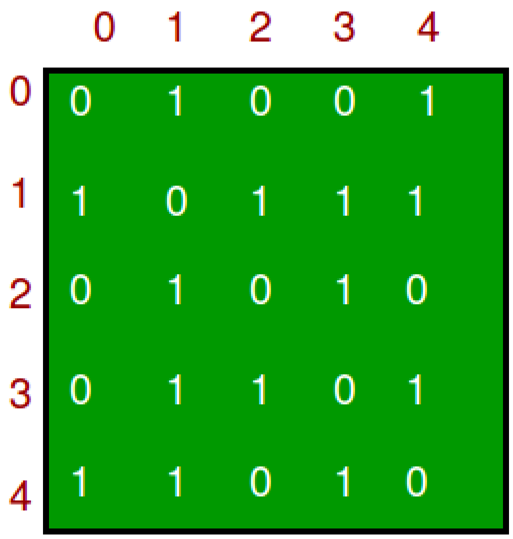
1) **Adjacency Matrix**
2) **Adjacency List**

There are other representations also like, Incidence Matrix and Incidence List. The choice of graph representation is situation-specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

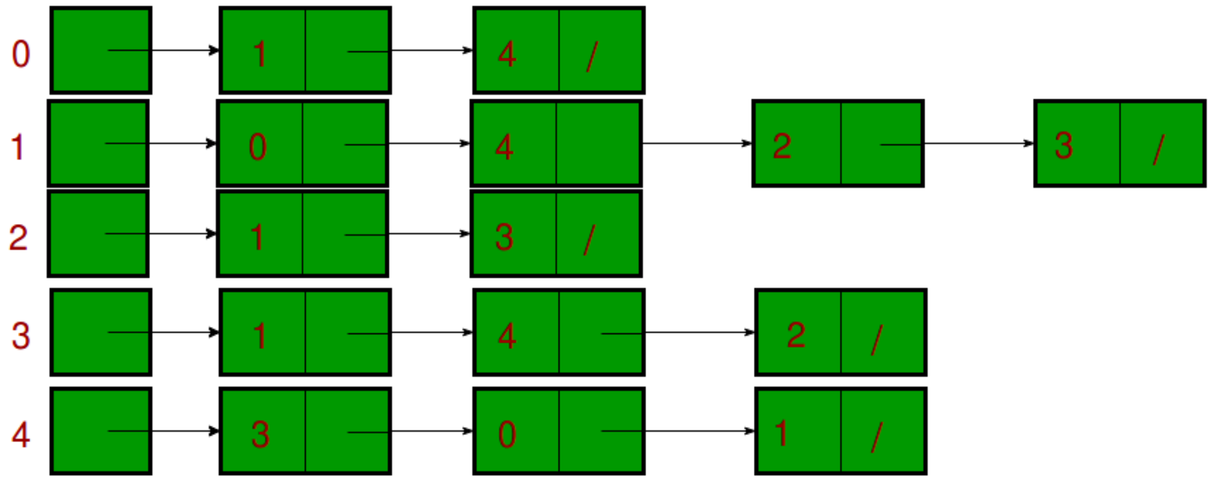
The adjacency matrix for the above example graph is:



$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 2 & 0 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 0 & 1 & 0 \end{array}$$

Adjacency List:

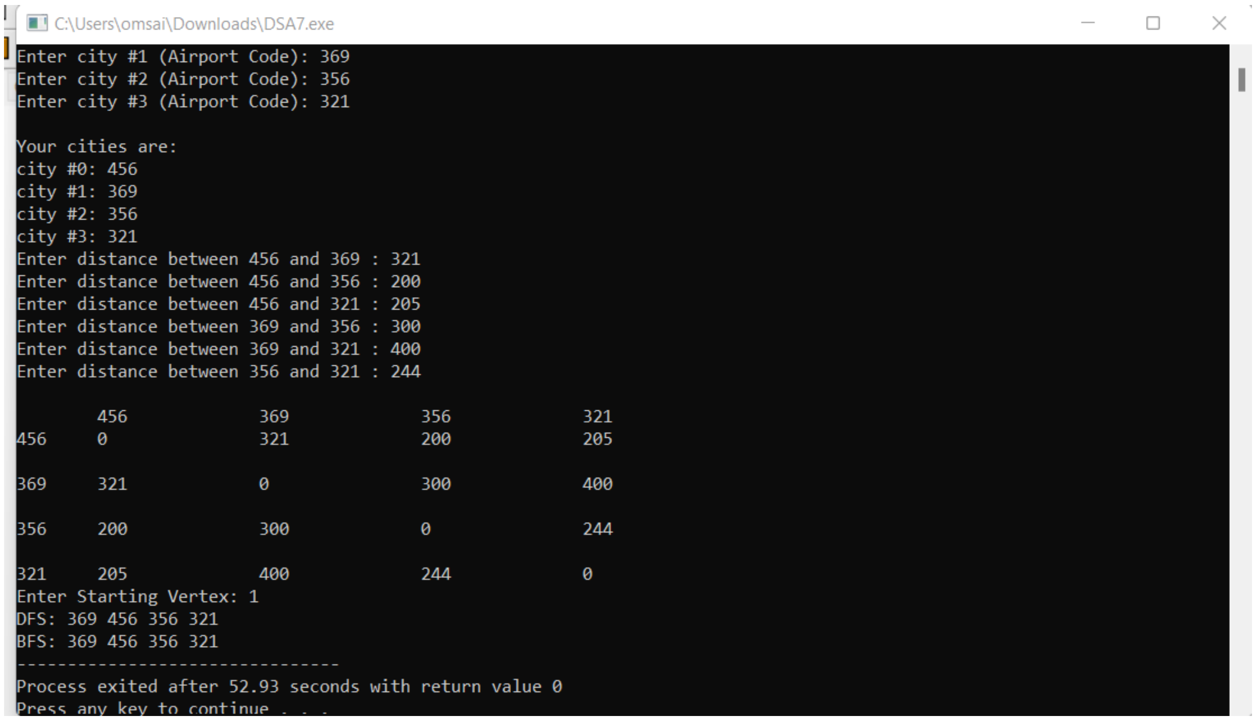
An array of lists is used. The size of the array is equal to the number of vertices. Let the array be an array[]. An entry array[i] represents the list of vertices adjacent to the **i**th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is the adjacency list representation of the above graph.



Program:

```
#include <iostream>
#include <queue>
using namespace std;
int adj_mat[50][50] = {0, 0};
int visited[50] = {0};
void dfs(int s, int n, string arr[])
{
    visited[s] = 1;
    cout << arr[s] << " ";
    for (int i = 0; i < n; i++)
    {
        if (adj_mat[s][i] && !visited[i])
            dfs(i, n, arr);
    }
}
void bfs(int s, int n, string arr[])
{
    bool visited[n];
    for (int i = 0; i < n; i++)
        visited[i] = false;
    int v;
    queue<int> bfsq;
    if (!visited[s])
    {
        cout << arr[s] << " ";
        bfsq.push(s);
        visited[s] = true;
        while (!bfsq.empty())
        {
            v = bfsq.front();
            for (int i = 0; i < n; i++)
            {
                if (adj_mat[v][i] && !visited[i])
                {
                    cout << arr[i] << " ";
                    visited[i] = true;
                    bfsq.push(i);
                }
            }
            bfsq.pop();
        }
    }
}
int main()
{
    cout << "Enter no. of cities: ";
    int n, u;
    cin >> n;
    string cities[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Enter city #" << i << " (Airport Code): ";
        cin >> cities[i];
    }

    cout << "\nYour cities are: " << endl;
    for (int i = 0; i < n; i++)
        cout << "city #" << i << ": " << cities[i] << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            cout << "Enter distance between " << cities[i] << " and " << cities[j] << " : ";
            cin >> adj_mat[i][j];
            adj_mat[j][i] = adj_mat[i][j];
        }
    }
    cout << endl;
    for (int i = 0; i < n; i++)
        cout << "\t" << cities[i] << "\t";
    for (int i = 0; i < n; i++)
    {
        cout << "\n"
             << cities[i];
        for (int j = 0; j < n; j++)
            cout << "\t" << adj_mat[i][j] << "\t";
        cout << endl;
    }
    cout << "Enter Starting Vertex: ";
    cin >> u;
    cout << "DFS: ";
    dfs(u, n, cities);
    cout << endl;
    cout << "BFS: ";
    bfs(u, n, cities);
    return 0;
}
```

Output:



C:\Users\omsai\Downloads\DSA7.exe

```
Enter city #1 (Airport Code): 369
Enter city #2 (Airport Code): 356
Enter city #3 (Airport Code): 321

Your cities are:
city #0: 456
city #1: 369
city #2: 356
city #3: 321
Enter distance between 456 and 369 : 321
Enter distance between 456 and 356 : 200
Enter distance between 456 and 321 : 205
Enter distance between 369 and 356 : 300
Enter distance between 369 and 321 : 400
Enter distance between 356 and 321 : 244

        456             369             356             321
456     0               321             200             205

369     321             0               300             400

356     200             300             0               244

321     205             400             244             0
Enter Starting Vertex: 1
DFS: 369 456 356 321
BFS: 369 456 356 321
--------------------------------
Process exited after 52.93 seconds with return value 0
Press any key to continue . . .
```

Group D

Experiment No-08

Aim: Given sequence k=k1<k2<…<kn of n sorted keys, with a search probability pi for each key ki. Build the Binary search tree that has the least search cost given the access probability for each key?

Prerequisite: C++ Programming

Objective: To understand the use functions for Binary search Tree

Input: BST

Theory:

Optimal Binary Search Tree

Given a sorted array key *[0.. n-1]* of search keys and an array *freq[0.. n-1]* of frequency counts, where *freq[i]* is the number of searches for *keys[i]*. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.
Let us first define the cost of a BST. The cost of a BST node is the level of that node multiplied by its frequency. The level of the root is 1.

Examples:

Input: keys[] = {10, 12}, freq[] = {34, 50}

There can be following two possible BSTs

```
    10              12
     \             /
      12          10
     I             II
```

Frequency of searches of 10 and 12 are 34 and 50 respectively.

The cost of tree I is 34*1 + 50*2 = 134

The cost of tree II is 50*1 + 34*2 = 118

Input: keys[] = {10, 12, 20}, freq[] = {34, 8, 50}

There can be following possible BSTs

```
   10              12               20        10             20
     \           /    \            /            \          /
      12         10    20          12             20         10
        \                         /               /          \
          20                      10              12           12
    I             II             III            IV            V
```

Among all possible BSTs, cost of the fifth BST is minimum.

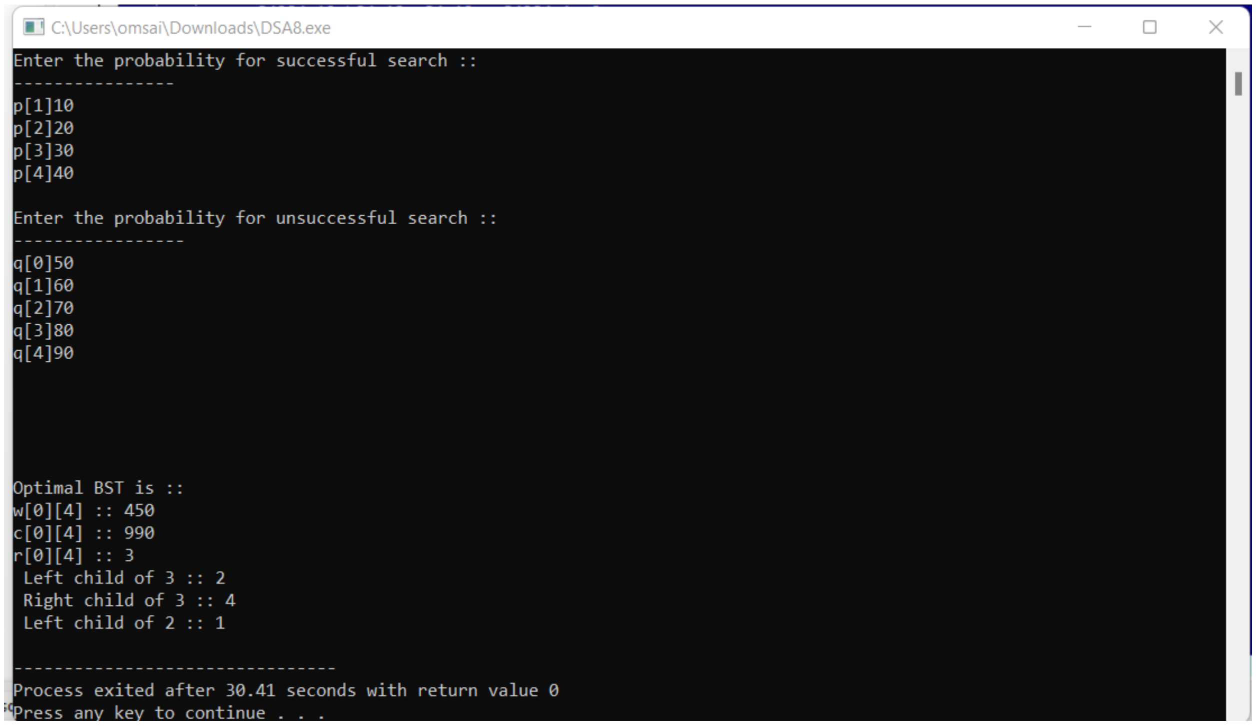
Cost of the fifth BST is 1*50 + 2*34 + 3*8 = 142

Program:

```
#include<iostream>
using namespace std;
void con_obst(void);
void print(int,int);
float a[20],b[20],wt[20][20],c[20][20];
int r[20][20],n;
int main()
  {
        int i;
        cout<<"\n****** PROGRAM FOR OBST ******\n";
        cout<<"\nEnter the no. of nodes : ";
        cin>>n;cout<<"\nEnter the probability for successful search :: ";
        cout<<"\n----------------\n";
        for(i=1;i<=n;i++)
          {
                cout<<"p["<<i<<"]";
                cin>>a[i];
          }
        cout<<"\nEnter the probability for unsuccessful search :: ";
        cout<<"\n-----------------\n";
        for(i=0;i<=n;i++)
         {
                cout<<"q["<<i<<"]";
                cin>>b[i];
         }
        con_obst();
        print(0,n);
        cout<<endl;
}
void con_obst(void)
{       int i,j,k,l,min;
        for(i=0;i<n;i++)
         { //Initialisation
                c[i][i]=0.0;
                r[i][i]=0;
                wt[i][i]=b[i];
                // for j-i=1 can be j=i+1
                wt[i][i+1]=b[i]+b[i+1]+a[i+1];
                c[i][i+1]=b[i]+b[i+1]+a[i+1];
                r[i][i+1]=i+1;
         }
        c[n][n]=0.0;
        r[n][n]=0;
        wt[n][n]=b[n];
        //for j-i=2,3,4....,n
        for(i=2;i<=n;i++)
         {
                for(j=0;j<=n-i;j++)
                 {
                        wt[j][j+i]=b[j+i]+a[j+i]+wt[j][j+i-1];
```

```
                                    c[j][j+i]=9999;
                                    for(l=j+1;l<=j+i;l++)
                                     {
                                                if(c[j][j+i]>(c[j][l-1]+c[l][j+i]))
                                                 {
                                                        c[j][j+i]=c[j][l-1]+c[l][j+i];
                                                        r[j][j+i]=l;
                                                 }
                                     }
                                    c[j][j+i]+=wt[j][j+i];
                         }
                        cout<<endl;
              }
            cout<<"\n\nOptimal BST is :: ";
            cout<<"\nw[0]["<<n<<"] :: "<<wt[0][n];
            cout<<"\nc[0]["<<n<<"] :: "<<c[0][n];
            cout<<"\nr[0]["<<n<<"] :: "<<r[0][n];
 }
void print(int l1,int r1)
 {
            if(l1>=r1)
                        return;
            if(r[l1][r[l1][r1]-1]!=0)
                        cout<<"\n Left child of "<<r[l1][r1]<<" :: "<<r[l1][r[l1][r1]-1];
            if(r[r[l1][r1]][r1]!=0)
                        cout<<"\n Right child of "<<r[l1][r1]<<" :: "<<r[r[l1][r1]][r1];
            print(l1,r[l1][r1]-1);
            print(r[l1][r1],r1);
            return;
}
```

Output:



C:\Users\omsai\Downloads\DSA8.exe

```
Enter the probability for successful search ::
----------------
p[1]10
p[2]20
p[3]30
p[4]40

Enter the probability for unsuccessful search ::
-----------------
q[0]50
q[1]60
q[2]70
q[3]80
q[4]90





Optimal BST is ::
w[0][4] :: 450
c[0][4] :: 990
r[0][4] :: 3
 Left child of 3 :: 2
 Right child of 3 :: 4
 Left child of 2 :: 1

--------------------------------
Process exited after 30.41 seconds with return value 0
Press any key to continue . . .
```

Experiment No-09

Aim: A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword.

Prerequisite: C++ Programming

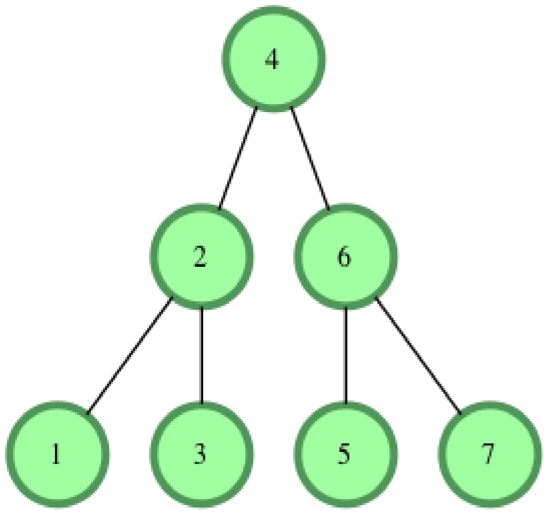
Objective: To understand the use functions for height of balance tree

Input: data elements

Theory

The height of a binary tree is the number of edges between the tree's root and its furthest leaf.

For example, the following binary tree is of height:



Function Description

Complete the *getHeight* or *height* function in the editor. It must return the height of a binary tree as an integer.

getHeight or height has the following parameter(s):

- *root*: a reference to the root of a binary tree.

Note -The Height of binary tree with single node is taken as zero.

Input Format

The first line contains an integer , the number of nodes in the tree.

Next line contains space separated integer where th integer denotes node[i].data.

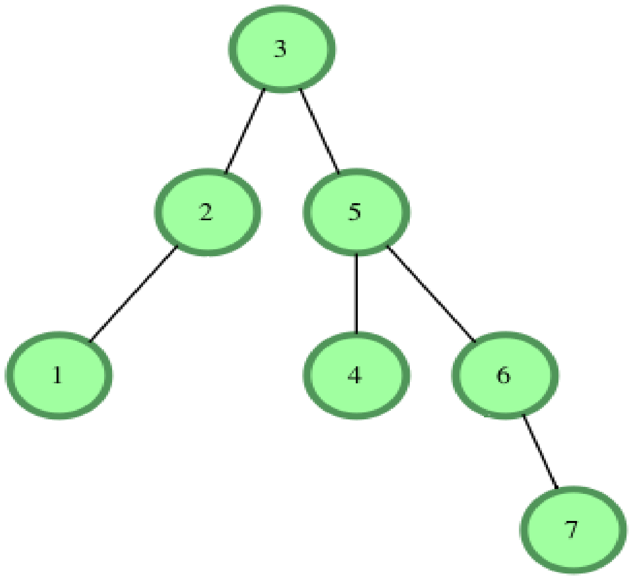
Note: Node values are inserted into a binary search tree before a reference to the tree's root node is passed to your function. In a binary search tree, all nodes on the left branch of a node are less than the node value. All values on the right branch are greater than the node value.

Constraints

Output Format

Your function should return a single integer denoting the height of the binary tree.

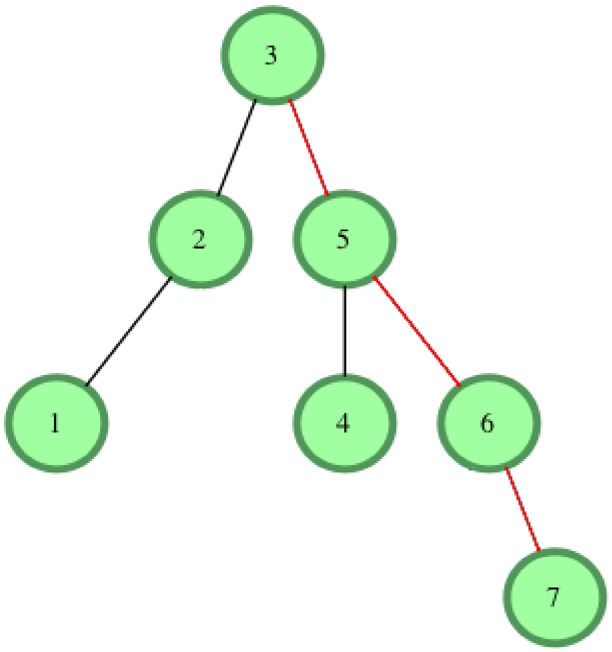
Sample Input



Sample Output: 3

Explanation

The longest root-to-leaf path is shown below:



There are nodes in this path that are connected by edges, meaning our binary tree's .

Program:

```
#include<iostream>

#include<string.h>

using namespace std;

class dict

{

   dict *root,*node,*left,*right,*tree1;

   string s1,s2;

   int flag,flag1,flag2,flag3,cmp;

public:

   dict()
```

```
    {
        flag=0,flag1=0,flag2=0,flag3=0,cmp=0;
        root=NULL;
    }
    void input();
    void create_root(dict*,dict*);
    void check_same(dict*,dict*);
    void input_display();
    void display(dict*);
    void input_remove();
    dict* remove(dict*,string);
    dict* findmin(dict*);
    void input_find();
    dict* find(dict*,string);
    void input_update();
    dict* update(dict*,string);

};
  void dict::input()
  {
      node=new dict;
      cout<<"\nEnter the keyword:\n";
      cin>>node->s1;
      cout<<"Enter the meaning of the keyword:\n";
      cin.ignore();
      getline(cin,node->s2);
      create_root(root,node);
  }
    void dict::create_root(dict *tree,dict *node1)
    {
        int i=0,result;
        char a[20],b[20];
        if(root==NULL)
        {
            root=new dict;
            root=node1;
            root->left=NULL;
            root->right=NULL;
            cout<<"\nRoot node created successfully"<<endl;
            return;
        }
        for(i=0;node1->s1[i]!='\0';i++)
        {
         a[i]=node1->s1[i];
        }
        for(i=0;tree->s1[i]!='\0';i++)
        {
         b[i]=tree->s1[i];
        }
        result=strcmp(b,a);
```

```
check_same(tree,node1);
if(flag==1)
   {
      cout<<"The word you entered already exists.\n";
       flag=0;
   }
   else
   {
if(result>0)
{
   if(tree->left!=NULL)
   {
      create_root(tree->left,node1);
   }
   else
   {
      tree->left=node1;
      (tree->left)->left=NULL;
         (tree->left)->right=NULL;
      cout<<"Node added to left of "<<tree->s1<<"\n";
      return;
   }
   }
   else if(result<0)
   {
```

```
            if(tree->right!=NULL)
            {
                create_root(tree->right,node1);
            }
            else
            {
                tree->right=node1;
                (tree->right)->left=NULL;
                (tree->right)->right=NULL;
                cout<<"Node added to right of "<<tree->s1<<"\n";
                return;
            }
           }
                }
        }
void dict::check_same(dict *tree,dict *node1)
{
 if(tree->s1==node1->s1)
 {
  flag=1;
  return;
 }
 else if(tree->s1>node1->s1)
     {
  if(tree->left!=NULL)
 {
      check_same(tree->left,node1);
 }
     }
    else if(tree->s1<node1->s1)
     {
      if(tree->right!=NULL)
      {
      check_same(tree->right,node1);
      }
     }
}


  void dict::input_display()
  {
   if(root!=NULL)
   {
       cout<<"The words entered in the dictionary are:\n\n";
       display(root);
   }
   else
   {
         cout<<"\nThere are no words in the dictionary.\n";
   }
  }
```

```
   void dict::display(dict *tree)
   {
        if(tree->left==NULL&&tree->right==NULL)
        {
         cout<<tree->s1<<" = "<<tree->s2<<"\n\n";
        }
        else
        {
          if(tree->left!=NULL)
          {
           display(tree->left);
          }
          cout<<tree->s1<<" = "<<tree->s2<<"\n\n";
          if(tree->right!=NULL)
          {
           display(tree->right);
          }
        }
   }
void dict::input_remove()
{
 char t;
 if(root!=NULL)
 {
   cout<<"\nEnter a keyword to be deleted:\n";
   cin>>s1;
   remove(root,s1);
   if(flag1==0)
   {
      cout<<"\nThe word '"<<s1<<"' has been deleted.\n";
   }
   flag1=0;
 }
 else
 {
  cout<<"\nThere are no words in the dictionary.\n";
 }
}
  dict* dict::remove(dict *tree,string s3)
  {
   dict *temp;
      if(tree==NULL)
      {
       cout<<"\nWord not found.\n";
       flag1=1;
       return tree;
      }
      else if(tree->s1>s3)
      {
```

```
 tree->left=remove(tree->left,s3);
 return tree;
}
else if(tree->s1<s3)
 {
 tree->right=remove(tree->right,s3);
 return tree;
}
else
{
 if(tree->left==NULL&&tree->right==NULL)
 {
  delete tree;
  tree=NULL;
 }
 else if(tree->left==NULL)
 {
  temp=tree;
  tree=tree->right;
  delete temp;
 }
 else if(tree->right==NULL)
 {
  temp=tree;
  tree=tree->left;
```

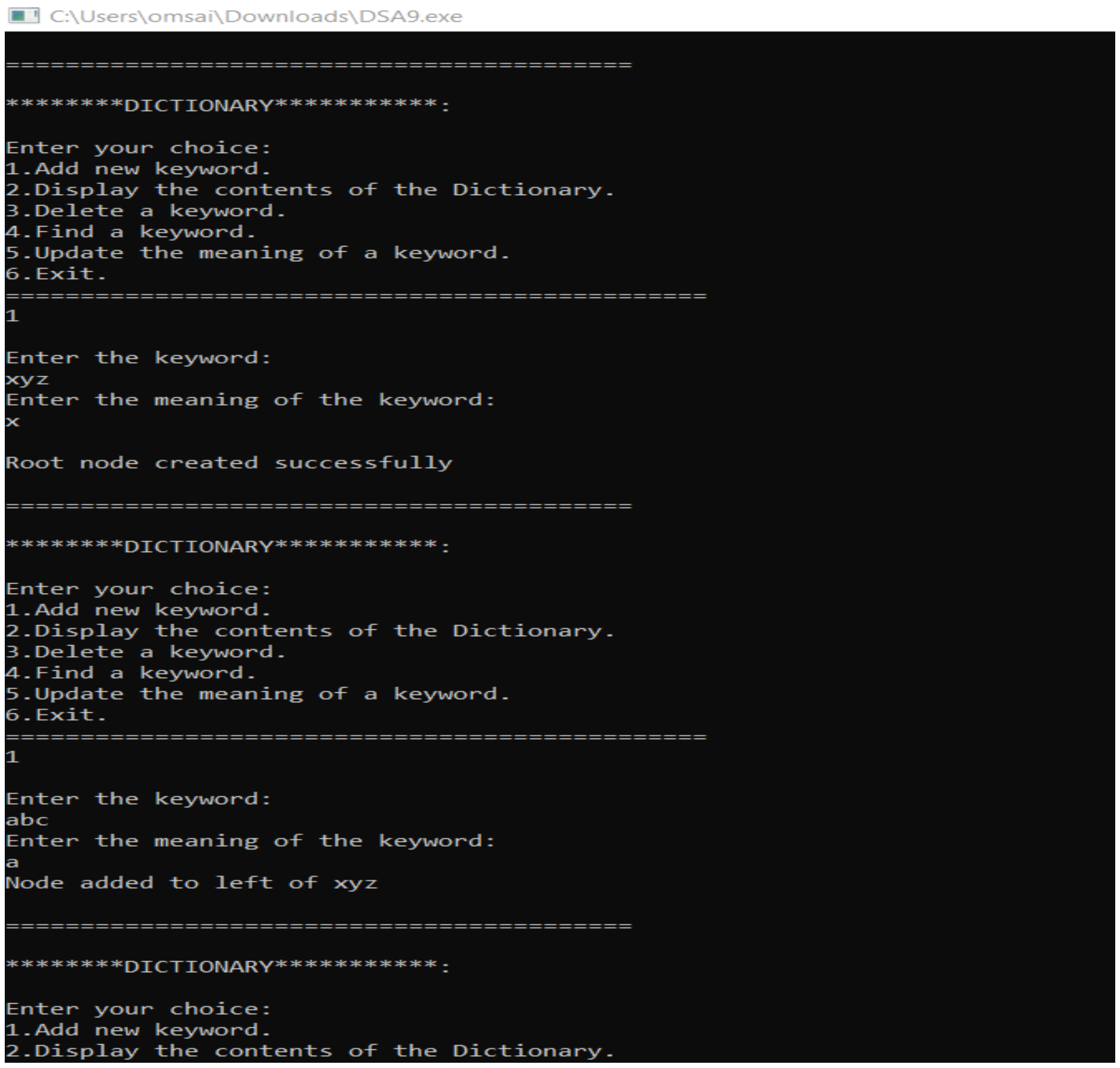
```
      delete temp;
     }
     else
     {
      temp=findmin(tree->right);
      tree=temp;
      tree->right=remove(tree->right,temp->s1);
     }
    }
    return tree;
}
  dict* dict::findmin(dict *tree)
  {
   while(tree->left!=NULL)
   {
    tree=tree->left;
   }
   return tree;
  }
void dict::input_find()
{
 flag2=0,cmp=0;
 if(root!=NULL)
 {
 cout<<"\nEnter the keyword to be searched:\n";
 cin>>s1;
    find(root,s1);
    if(flag2==0)
    {
  cout<<"Number of comparisons needed: "<<cmp<<"\n";
  cmp=0;
    }
 }
 else
 {
  cout<<"\nThere are no words in the dictionary.\n";
 }
}
  dict* dict::find(dict *tree,string s3)
  {
   if(tree==NULL)
   {
    cout<<"\nWord not found.\n";
    flag2=1;
    flag3=1;
    cmp=0;
   }
   else
   {
    if(tree->s1==s3)
```

```
      {
       cmp++;
       cout<<"\nWord found.\n";
       cout<<tree->s1<<": "<<tree->s2<<"\n";
       tree1=tree;
       return tree;
      }
      else if(tree->s1>s3)
      {
       cmp++;
       find(tree->left,s3);
      }
      else if(tree->s1<s3)
      {
       cmp++;
       find(tree->right,s3);
      }
     }
     return tree;
                     }
void dict::input_update()
{
 if(root!=NULL)
 {
 cout<<"\nEnter the keyword to be updated:\n";
```

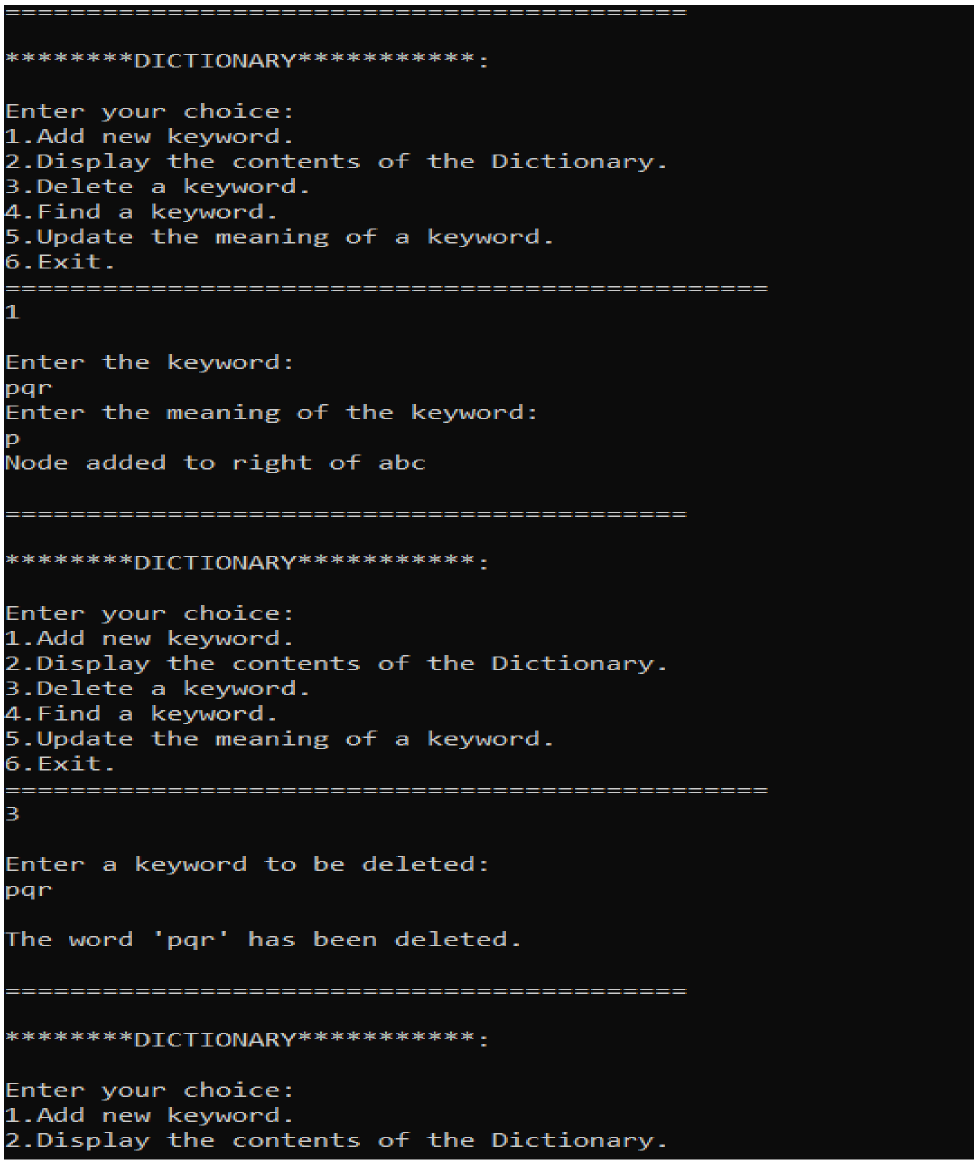
```
cin>>s1;
   update(root,s1);
 }
else
 {
  cout<<"\nThere are no words in the dictionary.\n";
 }
}
  dict* dict::update(dict *tree,string s3)
  {
   flag3=0;
   find(tree,s3);
   if(flag3==0)
   {
      cout<<"\nEnter the updated meaning of the keyword:\n";
      cin.ignore();
      getline(cin,tree1->s2);
      cout<<"\nThe meaning of '"<<s3<<"' has been updated.\n";
   }
      return tree;
  }
    int main()
      {
        int ch;
        dict d;
```

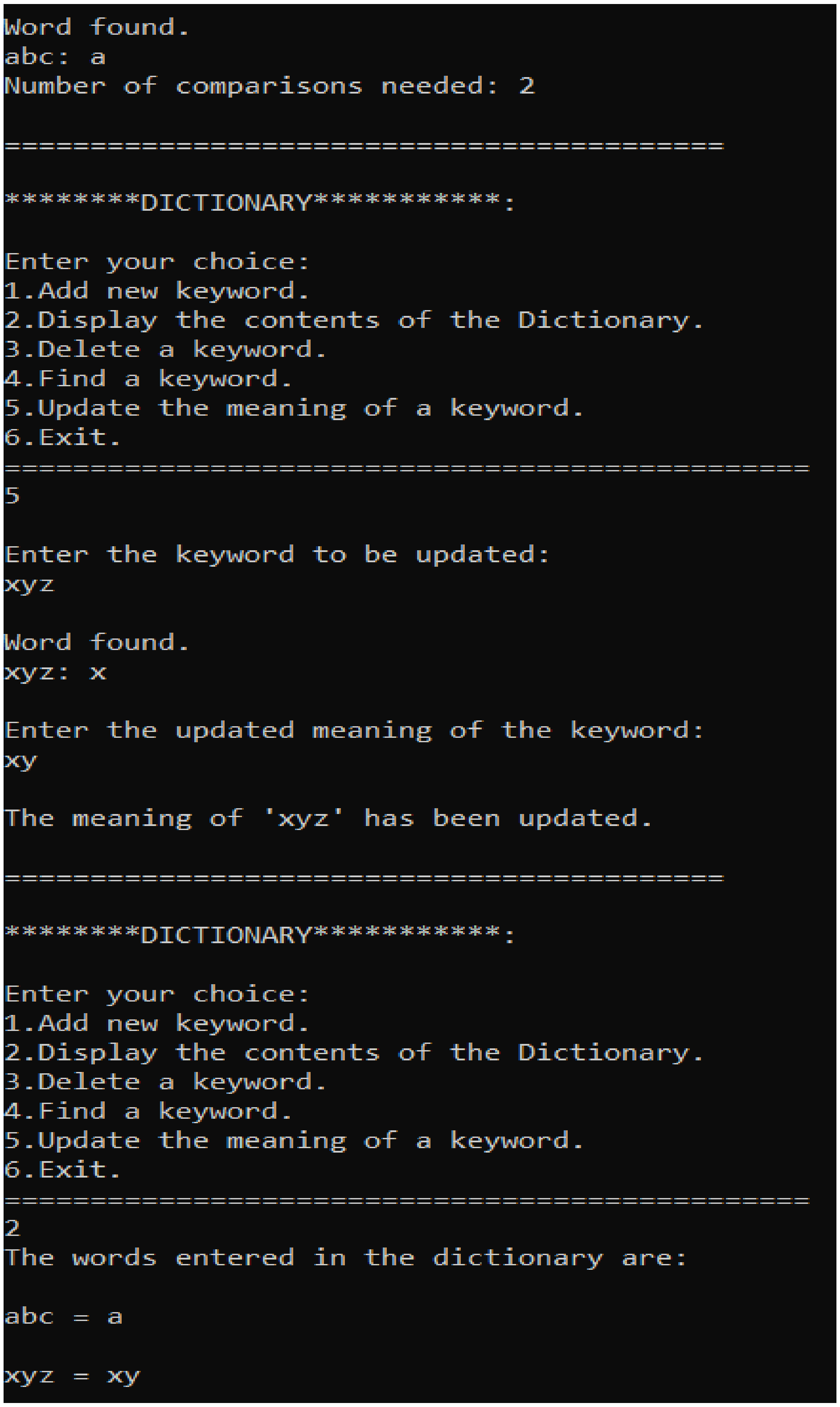
```
do
{
cout<<"\n===========================================\n"
  "\n********DICTIONARY***********:\n"
  "\nEnter your choice:\n"
    "1.Add new keyword.\n"
    "2.Display the contents of the Dictionary.\n"
  "3.Delete a keyword.\n"
  "4.Find a keyword.\n"
  "5.Update the meaning of a keyword.\n"
  "6.Exit.\n"
  "=================================================\n";
cin>>ch;
switch(ch)
{
   case 1:d.input();
          break;
   case 2:d.input_display();
        break;
   case 3:d.input_remove();
          break;
   case 4:d.input_find();
          break;
   case 5:d.input_update();
        break;
        default:cout<<"\nPlease enter a valid option!\n";
            break;
      }
    }while(ch!=6);
   return 0;
}
```

Output:



```
C:\Users\omsai\Downloads\DSA9.exe
==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
===============================================
1

Enter the keyword:
xyz
Enter the meaning of the keyword:
x

Root node created successfully

==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
===============================================
1

Enter the keyword:
abc
Enter the meaning of the keyword:
a
Node added to left of xyz

==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
```


```
==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
===============================================
1

Enter the keyword:
pqr
Enter the meaning of the keyword:
p
Node added to right of abc

==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
===============================================
3

Enter a keyword to be deleted:
pqr

The word 'pqr' has been deleted.

==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
```



```
Word found.
abc: a
Number of comparisons needed: 2

==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
===============================================
5

Enter the keyword to be updated:
xyz

Word found.
xyz: x

Enter the updated meaning of the keyword:
xy

The meaning of 'xyz' has been updated.

==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
===============================================
2
The words entered in the dictionary are:

abc = a

xyz = xy
```

Group E

Experiment No-10

Aim: Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language

Prerequisite: Java

Objectives:
1. To understand concept of heap in data structure.
2. To understand concept & features of java language.

Input: Number Of Array list

Theory:

Heap Sort:
A Binary Heap is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

Why array based representation for Binary Heap?
Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index I, the left child can be calculated by 2 * I + 1 and right child by 2 * I + 2 (assuming the indexing starts at 0).

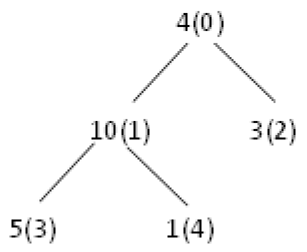
Heap Sort Algorithm for sorting in increasing order:
1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps until size of heap is greater than 1.

How to build the heap?

Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order.

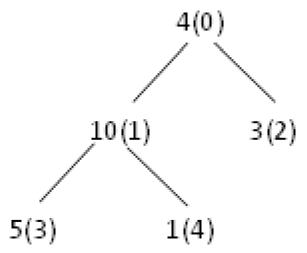
Lets understand with the help of an example:

Input data: 4, 10, 3, 5, 1



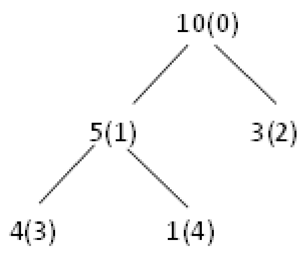
The numbers in bracket represent the indices in the array representation of data.

Applying heapify procedure to index 1:



Applying heap if y procedure to index 0:

The heapify procedure calls itself recursively to build heap in top down manner.



Algorithm:

STEP 1: Logically, think the given array as Complete Binary Tree,

STEP 2: For sorting the array in ascending order, check whether the tree is satisfying Max-heap property at each node, (For descending order, Check whether the tree is satisfying Min-heap property) Here we will be sorting in Ascending order,

STEP 3: If the tree is satisfying Max-heap property, then largest item is stored at the root of the heap. (At this point we have found the largest element in array, Now if we place this element at the end(nth position) of the array then 1 item in array is at proper place.)

STEP 4: We will remove the largest element from the heap and put at its proper place(nth position) in array.

STEP 5: After removing the largest element, which element will take its place? We will put last element of the heap at the vacant place. After placing the last element at the root, The new tree formed may or may not satisfy max-heap property. So, If it is not satisfying max-heap property then first task is to make changes to the tree, So that it satisfies max-heap property.

(Heapify process: The process of making changes to tree so that it satisfies max-heap property is called heapify)

STEP 6: When tree satisfies max-heap property, again largest item is stored at the root of the heap. We will remove the largest element from the heap and put at its proper place(n-1 position) in array.

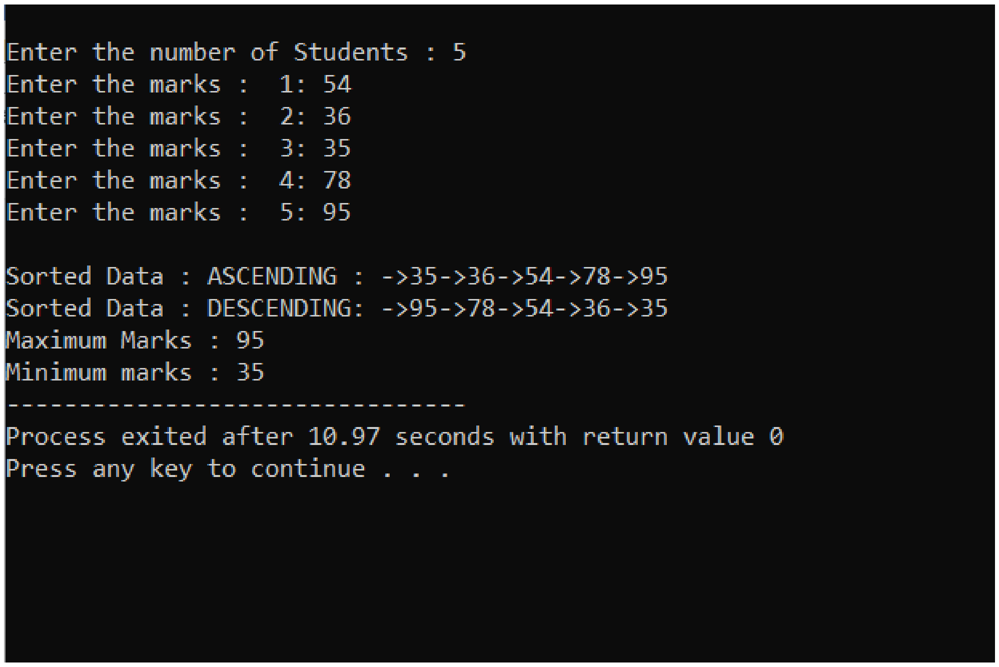
STEP 7: Repeat step 3 until size of array is 1 (At this point all elements are sorted.)

Program:

```
#include<iostream>
using namespace std;
void MaxHeapify(int a[], int i, int n)
{
 int j, temp;
 temp = a[i];
 j = 2*i;
  while (j <= n)
 {
  if (j < n && a[j+1] > a[j])
  j = j+1;
  if (temp > a[j])
   break;
  else if (temp <= a[j])
  {
   a[j/2] = a[j];
   j = 2*j;
  }
 }
 a[j/2] = temp;
 return;
}
void MinHeapify(int a[], int i, int n)
{
 int j, temp;
 temp = a[i];
 j = 2*i;
  while (j <= n)
 {
  if (j < n && a[j+1] < a[j])
  j = j+1;
  if (temp < a[j])
   break;
  else if (temp >= a[j])
  {
   a[j/2] = a[j];
   j = 2*j;
  }
 }
 a[j/2] = temp;
 return;
}
void MaxHeapSort(int a[], int n)
{
 int i, temp;
 for (i = n; i >= 2; i--)
 {
  temp = a[i];
  a[i] = a[1];
  a[1] = temp;
  MaxHeapify(a, 1, i - 1);
 }
}
void MinHeapSort(int a[], int n)
{
 int i, temp;
 for (i = n; i >= 2; i--)
 {
  temp = a[i];
  a[i] = a[1];
  a[1] = temp;

  MinHeapify(a, 1, i - 1);
 }
}
```

```
void Build_MaxHeap(int a[], int n)
{
 int i;
 for(i = n/2; i >= 1; i--)
  MaxHeapify(a, i, n);
}
void Build_MinHeap(int a[], int n)
{
 int i;
 for(i = n/2; i >= 1; i--)
  MinHeapify(a, i, n);
}
int main()
{
 int n, i;
 cout<<"\nEnter the number of Students : ";
 cin>>n;
 n++;
 int arr[n];
 for(i = 1; i < n; i++)
 {
  cout<<"Enter the marks :  "<<i<<": ";
  cin>>arr[i];
 }
 Build_MaxHeap(arr, n-1);
 MaxHeapSort(arr, n-1);
 int max,min;
 cout<<"\nSorted Data : ASCENDING : ";


 for (i = 1; i < n; i++)
  cout<<"->"<<arr[i];
 min=arr[1];
 Build_MinHeap(arr, n-1);
   MinHeapSort(arr, n-1);
   cout<<"\nSorted Data : DESCENDING: ";
 max=arr[1];
   for (i = 1; i < n; i++)
     cout<<"->"<<arr[i];
   cout<<"\nMaximum Marks : "<<max<<"\nMinimum marks : "<<min;
 return 0;
}
```

Output:



```
Enter the number of Students : 5
Enter the marks :  1: 54
Enter the marks :  2: 36
Enter the marks :  3: 35
Enter the marks :  4: 78
Enter the marks :  5: 95

Sorted Data : ASCENDING : ->35->36->54->78->95
Sorted Data : DESCENDING: ->95->78->54->36->35
Maximum Marks : 95
Minimum marks : 35
--------------------------------
Process exited after 10.97 seconds with return value 0
Press any key to continue . . .
```

Group-F

Experiment No-11

Aim: Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

Prerequisite: C++ Programming

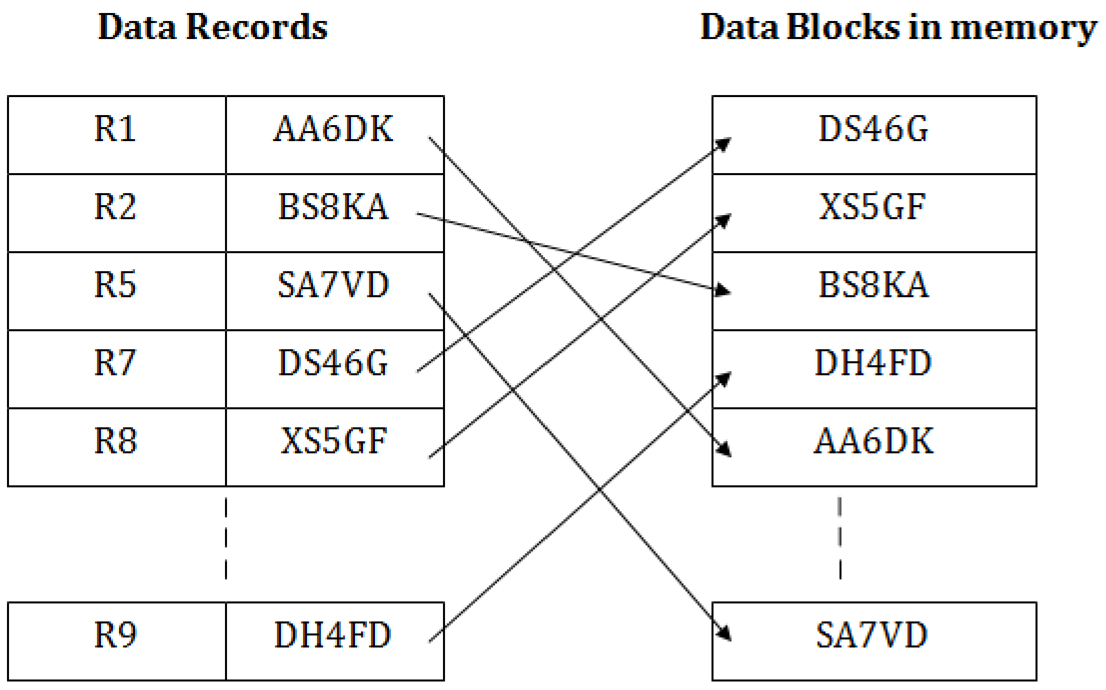
Objective: To understand the use index sequential file to

Input: A text file

Theory: Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.

If any record has to be



retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Process of ISAM:

- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

Cons of ISAM

- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

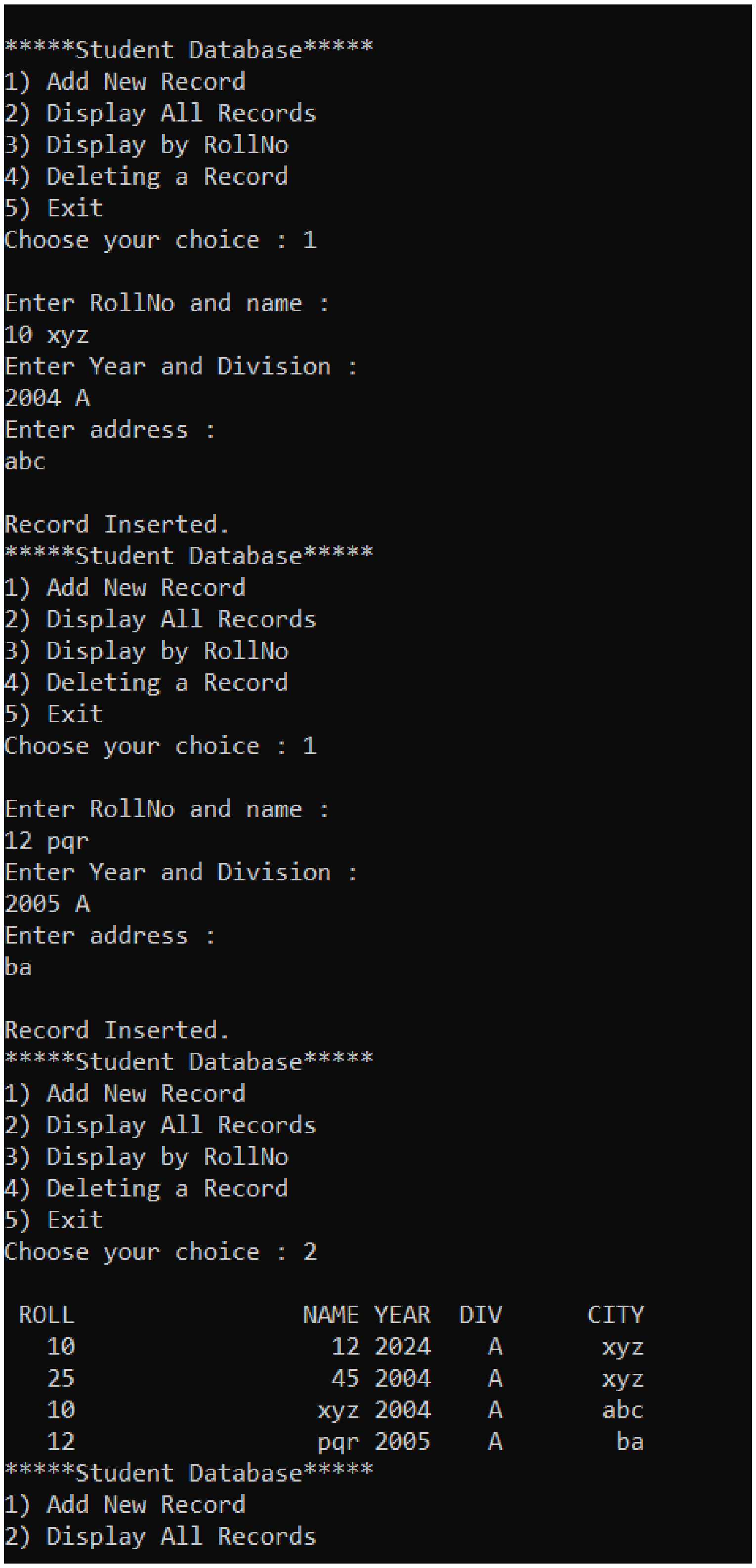
Program:

```
#include <iostream>
#include<fstream>
#include<cstring>
#include<iomanip>
using namespace std;
const int MAX=20;
class Student
{
 int rollno;
 char name[20],city[20];
 char div;
 int year;
public:
 Student()
{
  strcpy(name,"");
  strcpy(city,"");
  rollno=year=div=0;
}
 Student(int rollno,char name[MAX],int year,char div,char city[MAX])
 {
  strcpy(this->name,name);
  strcpy(this->city,city);
  this->rollno=rollno;
  this->year=year;
  this->div=div;
 }
 int getRollNo()
 {
  return rollno;
 }
 void displayRecord()
 {

cout<<endl<<setw(5)<<rollno<<setw(20)<<name<<setw(5)<<year<<setw(5)<<div<<setw(10)<
<city;
 }
};
//==========File Operations ===========
class FileOperations
```

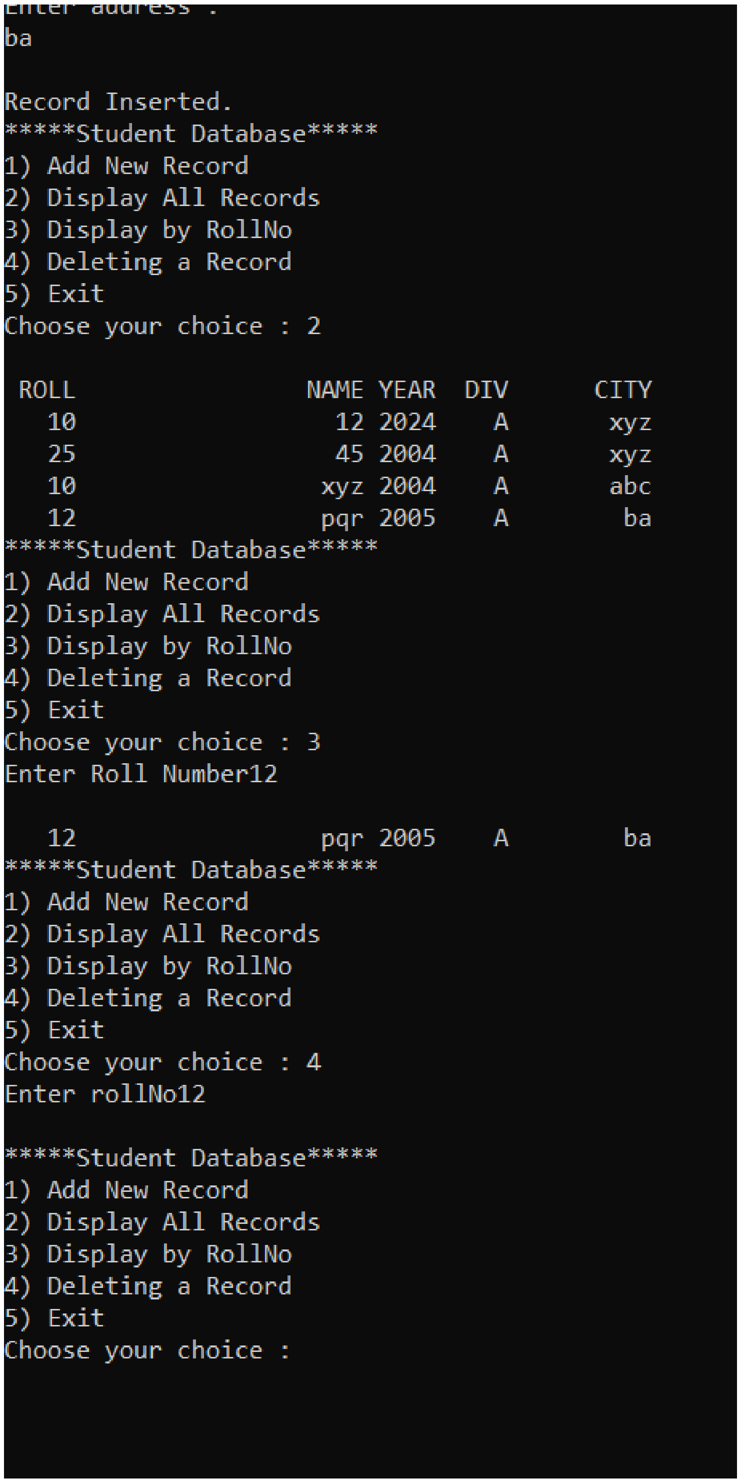
```
{
 fstream file;
public:
 FileOperations(char* filename)
{
file.open(filename,ios::in|ios::out|ios::ate|ios::binary);
}
 void insertRecord(int rollno, char name[MAX],int year, char div,char city[MAX])
 {
  Student s1(rollno,name,year,div,city);
  file.seekp(0,ios::end);
  file.write((char *)&s1,sizeof(Student));
  file.clear();
 }
 void displayAll()
 {
  Student s1;
  file.seekg(0,ios::beg);
  while(file.read((char *)&s1, sizeof(Student)))
  {
   s1.displayRecord();
  }
  file.clear();
 }
 void displayRecord(int rollNo)
 {
  Student s1;
  file.seekg(0,ios::beg);
  bool flag=false;
  while(file.read((char*)&s1,sizeof(Student)))
  {
   if(s1.getRollNo()==rollNo)
   {
    s1.displayRecord();
    flag=true;
    break;
   }
  }
  if(flag==false)
  {
   cout<<"\nRecord of "<<rollNo<<"is not present.";
  }
  file.clear();
 }
 void deleteRecord(int rollno)
 {
  ofstream outFile("new.dat",ios::binary);
  file.seekg(0,ios::beg);
  bool flag=false;
  Student s1;
  while(file.read((char *)&s1, sizeof(Student)))
  {
   if(s1.getRollNo()==rollno)
   {
    flag=true;
    continue;
   }
   outFile.write((char *)&s1, sizeof(Student));
  }
  if(!flag)
  {
   cout<<"\nRecord of "<<rollno<<" is not present.";
  }
  file.close();
  outFile.close();
  remove("student.dat");
  rename("new.dat","student.dat");
  file.open("student.dat",ios::in|ios::out|ios::ate|ios::binary);
 }
 ~FileOperations()
 {
  file.close();
  cout<<"\nFile Closed.";
 }
 };
 int main() {
 ofstream newFile("student.dat",ios::app|ios::binary);
  newFile.close();
  FileOperations file((char*)"student.dat");
     int rollNo,year,choice=0;
     char div;
    char name[MAX],address[MAX];
    while(choice!=5)
    {
      //clrscr();
      cout<<"\n*****Student Database*****\n";
      cout<<"1) Add New Record\n";
      cout<<"2) Display All Records\n";
      cout<<"3) Display by RollNo\n";
      cout<<"4) Deleting a Record\n";
      cout<<"5) Exit\n";
      cout<<"Choose your choice : ";
      cin>>choice;
      switch(choice)
      {
        case 1 : //New Record
         cout<<endl<<"Enter RollNo and name : \n";
         cin>>rollNo>>name;
         cout<<"Enter Year and Division : \n";
         cin>>year>>div;
         cout<<"Enter address : \n";
         cin>>address;
         file.insertRecord(rollNo,name,year,div,address);
         cout<<"\nRecord Inserted.";
         break;
        case 2 :

 cout<<endl<<setw(5)<<"ROLL"<<setw(20)<<"NAME"<<setw(5)<<"YEAR"<<setw(5)<<"DI
 V"<<setw(10)<<"CITY";
```

```
              file.displayAll();
              break;
            case 3 :
              cout<<"Enter Roll Number";
              cin>>rollNo;
               file.displayRecord(rollNo);
              break;
            case 4:
              cout<<"Enter rollNo";
              cin>>rollNo;
              file.deleteRecord(rollNo);
              break;
           case 5 :break;
        }
    }
 return 0;
}
```

Output:



```
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 1

Enter RollNo and name :
10 xyz
Enter Year and Division :
2004 A
Enter address :
abc

Record Inserted.
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 1

Enter RollNo and name :
12 pqr
Enter Year and Division :
2005 A
Enter address :
ba

Record Inserted.
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 2

 ROLL                NAME YEAR  DIV      CITY
   10                  12 2024    A       xyz
   25                  45 2004    A       xyz
   10                 xyz 2004    A       abc
   12                 pqr 2005    A        ba
*****Student Database*****
1) Add New Record
2) Display All Records
```



```
Enter address :
ba

Record Inserted.
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 2

 ROLL                NAME YEAR  DIV      CITY
   10                  12 2024    A       xyz
   25                  45 2004    A       xyz
   10                 xyz 2004    A       abc
   12                 pqr 2005    A        ba
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 3
Enter Roll Number12

   12                 pqr 2005    A        ba
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 4
Enter rollNo12

*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice :
```

Experiment NO-12

Aim: Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

Prerequisite: C++

Objective: To understand the files and its operation

Input: File

Theory :

Attributes of a File

Following are some of the attributes of a file :

- **Name** . It is the only information which is in human-readable form.
- **Identifier**. The file is identified by a unique tag(number) within file system.
- **Type**. It is needed for systems that support different types of files.
- **Location**. Pointer to file location on device.
- **Size**. The current size of the file.
- **Protection**. This controls and assigns the power of reading, writing, executing.
- **Time, date, and user identification**. This is the data for protection, security, and usage monitoring.

File Access Methods

The way that files are accessed and read into memory is determined by Access methods. Usually a single access method is supported by systems while there are OS's that support multiple access methods.

1. Sequential Access

- Data is accessed one record right after another is an order.
- Read command cause a pointer to be moved ahead by one.
- Write command allocate space for the record and move the pointer to the new End Of File.
- Such a method is reasonable for tape.

2. Direct Access

- This method is useful for disks.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on which blocks are read/written, it can be dobe in any order.
- User now says "read n" rather than "read next".
- "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

3. Indexed Sequential Access

- It is built on top of Sequential access.
- It uses an Index to control the pointer while accessing files.

What is a Directory?

Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

Following is the information maintained in a directory :

- **Name** : The name visible to user.
- **Type** : Type of the directory.
- **Location** : Device and location on the device where the file header is located.
- **Size** : Number of bytes/words/blocks in the file.
- **Position** : Current next-read/next-write pointers.
- **Protection** : Access control on read/write/execute/delete.

- **Usage** : Time of creation, access, modification etc.

A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities. From user's perspective a file is the smallest allotment of logical secondary storage.

Attributes	Types	Operations
Name	Doc	Create
Type	Exe	Open
Size	Jpg	Read
Creation Data	Xis	Write
Author	C	Append
Last Modified	Java	Truncate
Protection	class	Delete
		Close

File type	Usual extension	Function
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, java, pas, asm, a	Source code in various languages

File type	Usual extension	Function
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word Processor	wp, tex, rrf, doc	Various word processor formats
Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information

FILE DIRECTORIES:

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Information contained in a device directory are:

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner id
- Protection information

Operation performed on directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

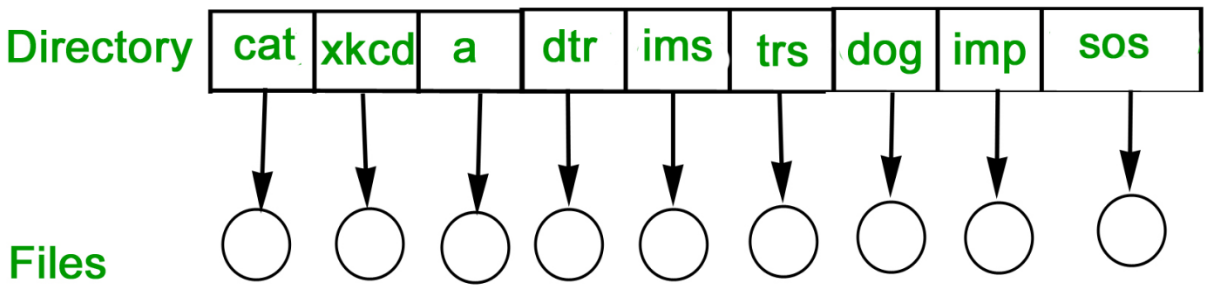
Advantages of maintaining directories are:

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

SINGLE-LEVEL DIRECTORY

In this a single directory is maintained for all the users.

- **Naming problem:** Users cannot have same name for two files.
- **Grouping problem:** Users cannot group files according to their need.

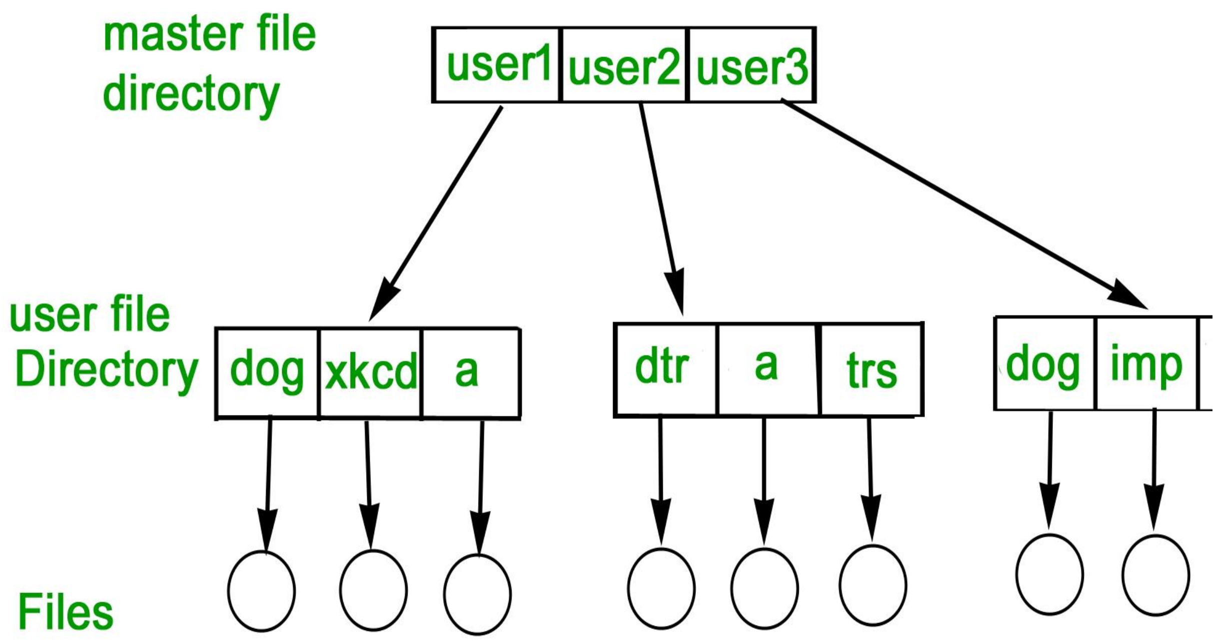


TWO-LEVEL DIRECTORY

In this separate directories for each user is maintained.

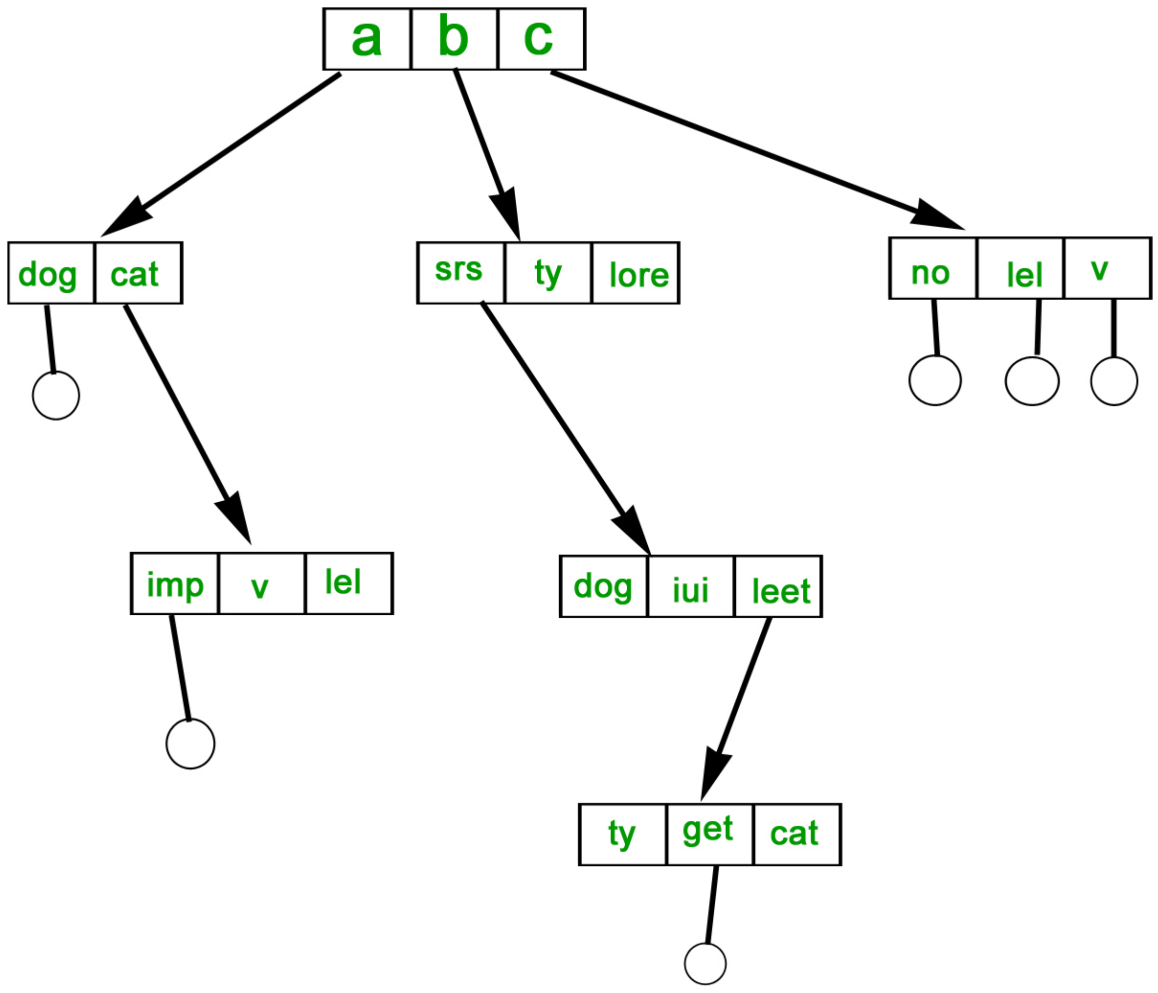
- Path name: Due to two levels there is a path name for every file to locate that file.

- Now,we can have same file name for different user.
- Searching is efficient in this method.



TREE-STRUCTURED DIRECTORY :

Directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file

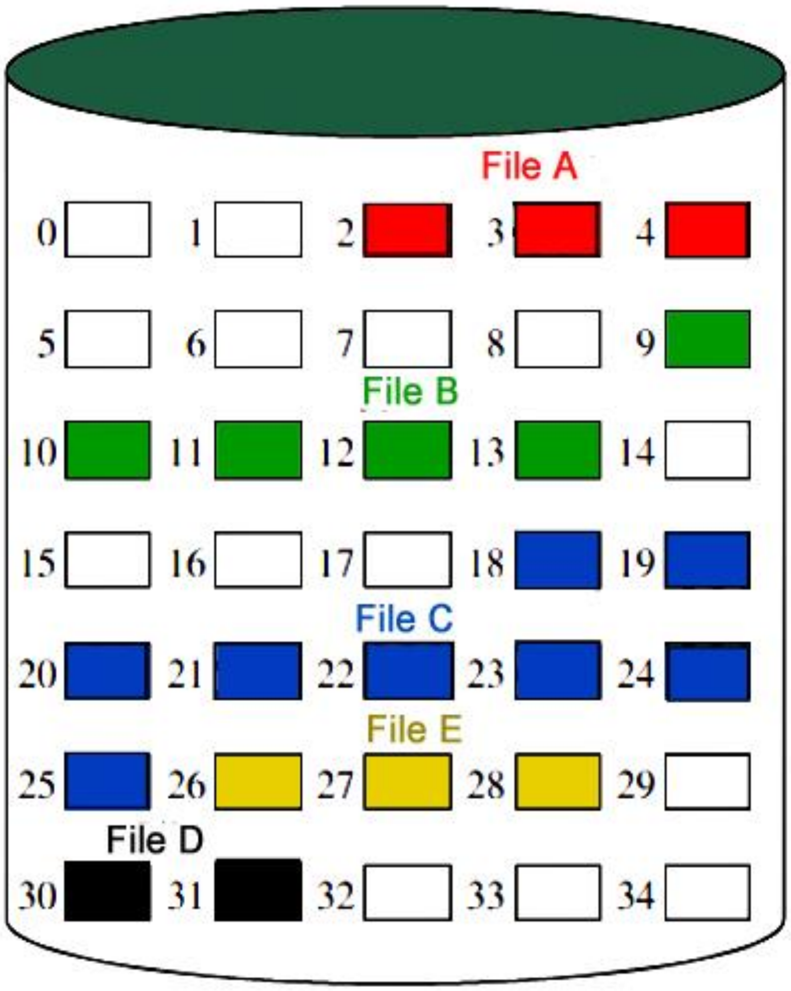


FILE ALLOCATION METHODS

1. **Continuous Allocation:**

   A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing.

   It is also easy to retrieve a single block. For example, if a file starts at block b, and the ith block of the file is wanted, its location on secondary storage is simply b+i-1.



File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Disadvantage

- External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length. Compaction algorithm will be necessary to free up additional space on disk.
- Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

2. **Linked Allocation (Non-contiguous allocation):**

   Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. Increase in file size is always possible if free disk block is available. There is no external fragmentation because only one block at a time is needed but there can be internal fragmentation but it exists only in the last disk block of file.

   Disadvantage:

- Internal fragmentation exists in last disk block of file.
- There is an overhead of maintaining the pointer in every disk block.
- If the pointer of any disk block is lost, the file will be truncated.
- It supports only the sequencial access of files.

3. **Indexed Allocation:**

   It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to the file. Allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality. This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.

Disk Free Space Management

Just as the space that is allocated to files must be managed ,so the space that is not currently allocated to any file must be managed. To perform any of the file allocation techniques,it is necessary to know what blocks on the disk are available. Thus we need a disk allocation table in addition to a file allocation table.The following are the approaches used for free space management.

1. **Bit Tables** : This method uses a vector containing one bit for each block on the disk. Each entry for a 0 corresponds to a free block and each 1 corresponds to a block in use. For example: 00011010111100110001
   In this vector every bit correspond to a particular block and 0 implies that, that particular block is free and 1 implies that the block is already occupied. A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks. Thus, a bit table works well with any of the file allocation methods. Another advantage is that it is as small as possible.
2. **Free Block List** : In this method, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved block of the disk.

Program:

```
#include
<bits/stdc++.h> #define
max 20 using
namespace std; struct
employee { string name;
long int code; string
designation;
int sal;
int age;
};
int num; void
showMenu();
employee emp[max], tempemp[max],
sortemp[max], sortemp1[max];
void build()
{
cout << "Build The Table\n"; cout << "Maximum
Entries can be "<< max << "\n"; cout << "Enter the
number of "<< "Entries required"; cin >> num; if
(num > 20) {
cout << "Maximum number of "  << "Entries are 20\n";
num = 20;
}
```

```
cout << "Enter the following data:\n";
for (int i = 0; i < num;
i++) { cout << "Name ";
cin >> emp[i].name; cout
<< "Employee ID "; cin
>> emp[i].code; cout <<
"Designation "; cin >>
emp[i].designation; cout
<< "Salary "; cin >>
emp[i].sal; cout << "Age
";
cin >> emp[i].age;
}
showMenu();
}
void insert()
{
if (num < max) {
int i = num;
num++;
cout << "Enter the information "<< "of the Employee\n";
cout << "Name ";
cin >> emp[i].name;
cout << "Employee ID ";
cin >> emp[i].code; cout
<< "Designation "; cin
>> emp[i].designation;
cout << "Salary "; cin >>
emp[i].sal; cout << "Age
";
cin >> emp[i].age;
}
else {
cout << "Employee Table Full\n";
}
showMenu();
}
void deleteIndex(int i)
{
for (int j = i; j < num - 1; j++) {
emp[j].name = emp[j + 1].name;
emp[j].code = emp[j + 1].code;
emp[j].designation= emp[j + 1].designation;
emp[j].sal = emp[j + 1].sal;
emp[j].age = emp[j + 1].age;
}
```

```
return;
}
void deleteRecord()
{
cout << "Enter the Employee ID
" << "to Delete Record"; int
code; cin >> code; for (int i = 0;
i < num; i++) { if (emp[i].code
== code) { deleteIndex(i); num-
; break;
}
}
showMenu();
}
void displayRecord()
{
cout << "Enter the
Employee" << " ID to
Display Record"; int code;
cin >> code; for (int i = 0; i <
num; i++) if (emp[i].code ==
code) { cout << "Name " <<
emp[i].name << "\n"; cout <<
"Employee ID " <<
emp[i].code << "\n"; cout <<
"Designation " <<
emp[i].designation << "\n";
cout << "Salary " <<
emp[i].sal << "\n"; cout <<
"Age " << emp[i].age <<
"\n";
break;
}
}
showMenu();
}
void showMenu()
{
cout << "-----"<< " Employee"<< " Information"<< "-------------------------\n\n";
cout << "Available Options:\n\n";
cout << "1.Build Table        \n";
cout << "2.Insert New Entry    \n";
cout << "3.Delete Entry        \n";
cout << "4.Display a Record     \n";
cout << "5.Exit                \n";
cout << "Enter Options: ";
```

```
int option; cin
>> option; if
(option == 1) {
build();
}
else if (option == 2) {
insert();
}
else if (option == 3) {
deleteRecord();
}
else if (option == 4) {
displayRecord();
}
else if (option == 5) {
return;
}
else {
cout << "Expected Options"  << " are 1/2/3/4/5";
showMenu(); }
}
int main()
{
showMenu();
return 0;
}
```

Output-

```
----- Employee Information------------------------
Available Options:
1.Build Table
2.Insert New Entry
3.Delete Entry
4.Display a Record
5.Exit
Enter Options: 1

Build The Table
Maximum Entries can be 20 Enter the
number of Entries required2 Enter the
following data:
Name kishori
Employee ID 1234
Designation testing
Salary 22000
```

```
Age 20

Name shreya
Employee ID 5678
Designation programming
Salary 25000
Age 23  ----- Employee Information------------------------

Available Options:

1.Build Table
2.Insert New Entry
3.Delete Entry
4.Display a Record
5.Exit
Enter Options: 2
Enter the information of the Employee
Name dhanu
Employee ID 1909
Designation networking
Salary 40000
Age 29  ----- Employee Information------------------------

Available Options:

1.Build Table
2.Insert New Entry
3.Delete Entry
4.Display a Record
5.Exit
Enter Options: 3



Enter the Employee ID to Delete Record1909  ----- Employee Information------------------------

Available Options:

1.Build Table
2.Insert New Entry
3.Delete Entry
4.Display a Record
5.Exit
Enter Options: 4
Enter the Employee ID to Display Record1234
Name kishori
```

```
Employee ID 1234
Designation testing
Salary 22000
Age 20  ----- Employee Information-------------------------

Available Options:

1.Build Table
2.Insert New Entry
3.Delete Entry
4.Display a Record
5.Exit
Enter Options: 5
=== Code Execution Successful ===
```

Mini project

Experiment No-13

(Mini Project)

Aim: Design a mini project to implement Snake and Ladders Game using Python.

Prerequisite: Python

Objective: To understand the use Python

Input: Snake and Ladders and create box

Theory:

Yes, I know you all have played the Snake Game and definitely, you never wanted to lose. As kids, we all loved looking for cheats in order to never see the "Game Over" message but as techies, I know you would want to make this 'Snake' dance to your beats. This is what I will be showing you all in this article on.

Before moving on, let's have a quick look at all the sub-bits that build the Snake Game in Python: Installing Pygame

1. Create the Screen
2. Create the Snake
3. Moving the Snake
4. Game Over when Snake hits the boundaries
5. Adding the Food
6. Increasing the Length of the Snake
7. Displaying the Score

Installing Pygame:

The first thing you will need to do in order to create games using Pygame is to install it on your systems. To do that, you can simply use the following command:

pip install pygame

Once that is done, just import Pygame and start off with your game development. Before moving on, take a look at the Pygame functions that have been used in this Snake Game along with their descriptions.

Function	Description
init()	Initializes all of the imported Pygame modules (returns a tuple indicating success and failure of initializations)
display.set_mode()	Takes a tuple or a list as its parameter to create a surface (tuple preferred)
update()	Updates the screen
quit()	Used to uninitialize everything
set_caption()	Will set the caption text on the top of the display screen
event.get()	Returns list of all events
Surface.fill()	Will fill the surface with a solid color
time.Clock()	Helps track time time
font.SysFont()	Will create a Pygame font from the System font resources

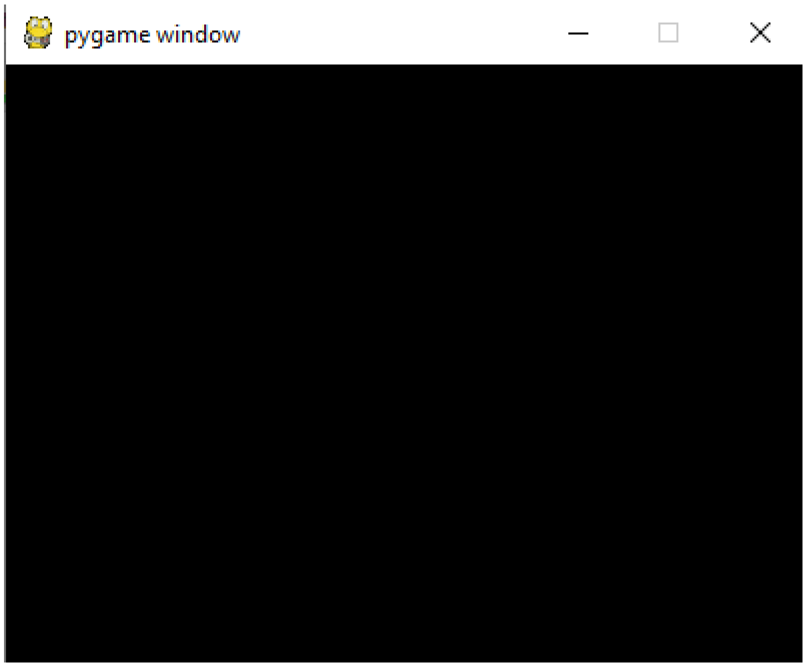
Create the Screen:

To create the screen using Pygame, you will need to make use of the *display.set_mode()* function. Also, you will have to make use of the *init()* and the *quit()* methods to initialize and uninitialize everything at the start and the end of the code. The *update()* method is used to update any changes made to the screen. There is another method i.e *flip()* that works similarly to the update() function. The difference is that the update() method updates only the changes that are made (however, if no parameters are passed, updates the complete screen) but the flip() method redoes the complete screen again.

CODE:

```
import pygame
pygame.init()
dis=pygame.display.set_mode((400,300))
pygame.display.update()
pygame.quit()
quit()
```

OUTPUT:

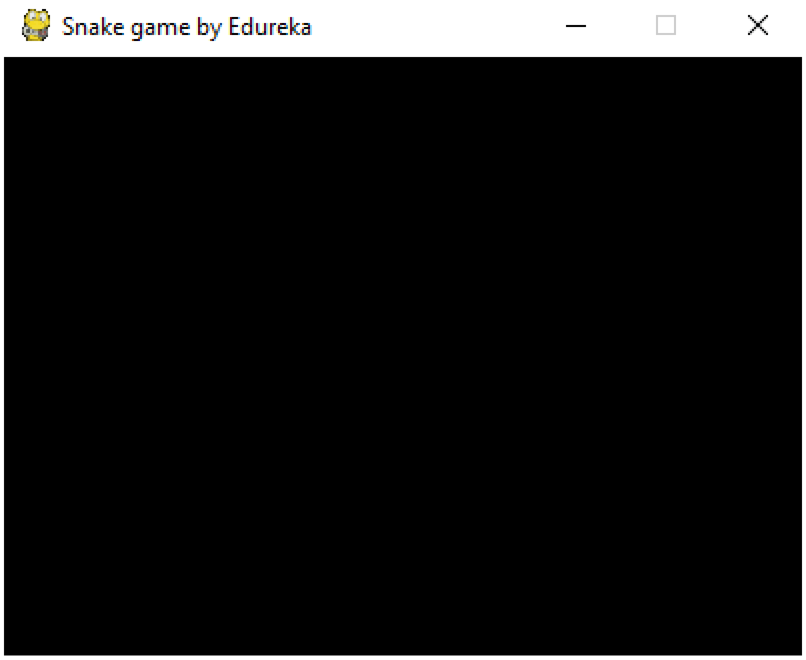


But when you run this code, the screen will appear, but it will immediately close as well. To fix that, you should make use of a game loop using the while loop before I actually quit the game as follows:

```
while not game_over:

for event in pygame.event.get():

print(event)

#prints out all the actions that take place on the screen

pygame.quit()

quit()
```

When you run this code, you will see that the screen that you saw earlier does not quit and also, it returns all the actions that take place over it. I have done that using the *event.get()* function. Also, I have named the screen as “Snake Game by Edureka” using the *display.set_caption()* function.

OUTPUT:



Now, you have a screen to play your Snake Game, but when you try to click on the close button, the screen does not close. This is because you have not specified that your screen should exit when you hit that close button. To do that, Pygame provides an event called "QUIT" and it should be used as follows:

```
import pygame

pygame.init()

dis=pygame.display.set_mode((400,300))

pygame.display.update()

pygame.display.set_caption('Snake game by Edureka')

game_over=False

while not game_over:

    for event in pygame.event.get():

        if event.type==pygame.QUIT:
```

```
        game_over=True
pygame.quit()
quit()
```

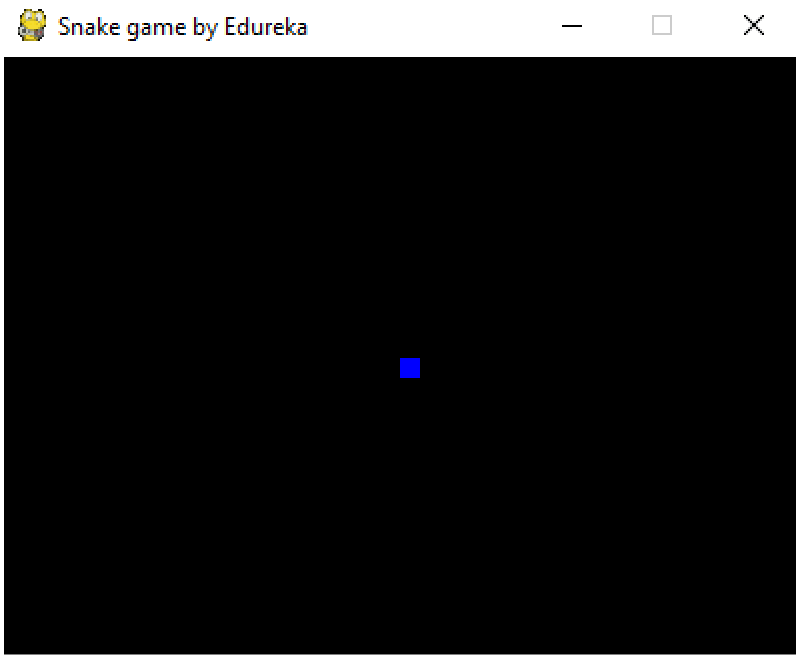
So now your screen is all set. The next part is to draw our snake on the screen which is covered in the following topic.

Create the Snake:

To create the snake, I will first initialize a few color variables in order to color the snake, food, screen, etc. The color scheme used in Pygame is RGB i.e “Red Green Blue”. In case you set all these to 0’s, the color will be black and all 255’s will be white. So our snake will actually be a rectangle. To draw rectangles in Pygame, you can make use of a function called *draw.rect()* which will help yo draw the rectangle with the desired color and size.

```
import pygame
pygame.init()
dis=pygame.display.set_mode((400,300))

pygame.display.set_caption('Snake game by Edureka')

blue=(0,0,255)
red=(255,0,0)

game_over=False
while not game_over:
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            game_over=True
    pygame.draw.rect(dis,blue,[200,150,10,10])
    pygame.display.update()
pygame.quit()
quit()
```

OUTPUT:



As you can see, the snakehead is created as a blue rectangle. The next step is to get your snake moving.

Moving the Snake:

Next

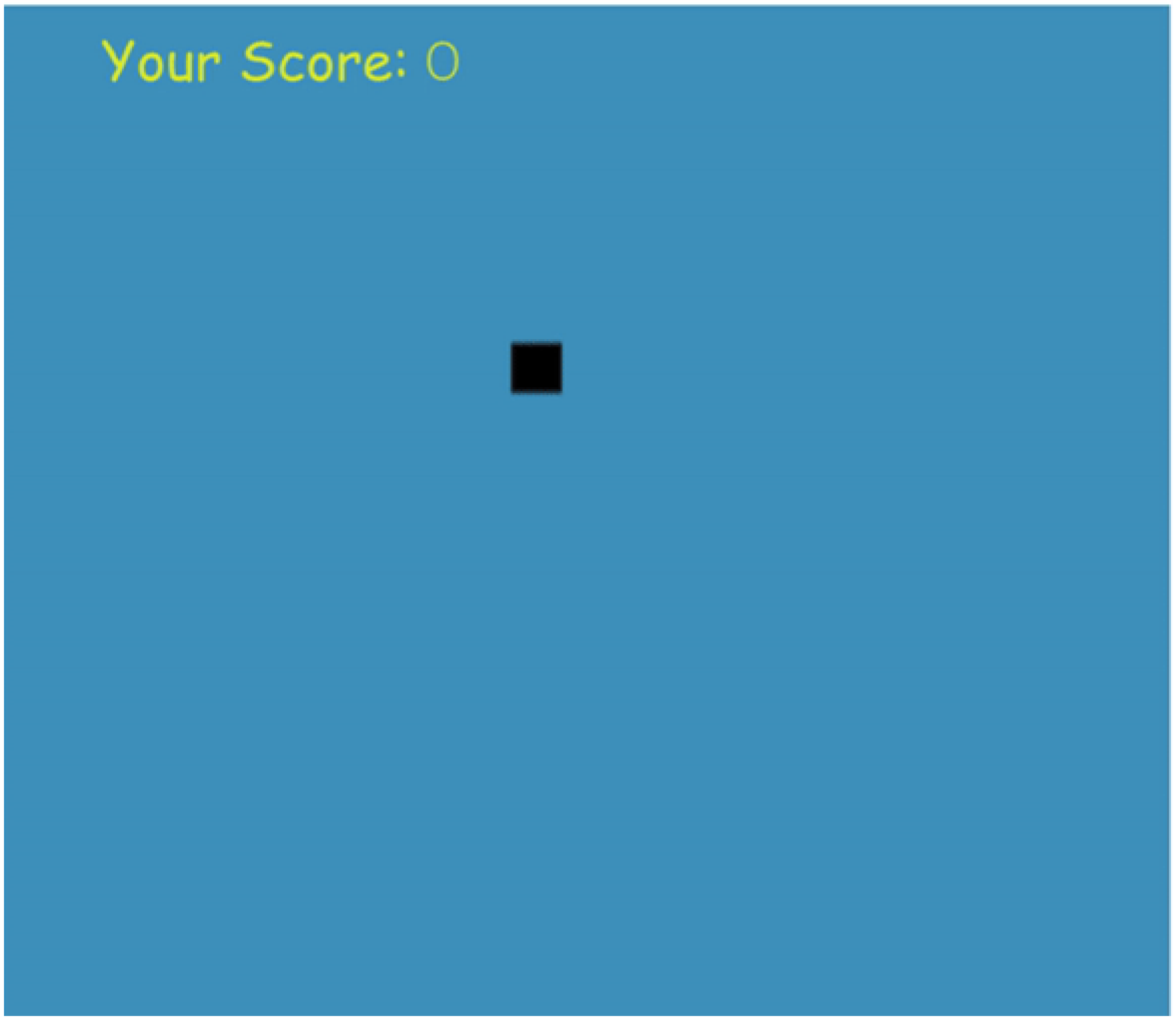
To move the snake, you will need to use the key events present in the KEYDOWN class of Pygame. The events that are used over here are, K_UP, K_DOWN, K_LEFT, and K_RIGHT to make the snake move up, down, left and right respectively. Also, the display screen is changed from the default black to white using the *fill()* method.

I have created new variables *x1_change* and *y1_change* in order to hold the updating values of the x and y coordinates.

```
import pygame

pygame.init()

white = (255, 255, 255)
black = (0, 0, 0)
red = (255, 0, 0)
```

```
dis = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Snake Game by Edureka')

game_over = False

x1 = 300
y1 = 300

x1_change = 0
y1_change = 0

clock = pygame.time.Clock()

while not game_over:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_over = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                x1_change = -10
                y1_change = 0
            elif event.key == pygame.K_RIGHT:
                x1_change = 10
                y1_change = 0
            elif event.key == pygame.K_UP:
                y1_change = -10
                x1_change = 0
            elif event.key == pygame.K_DOWN:
                y1_change = 10
                x1_change = 0

    x1 += x1_change
    y1 += y1_change
    dis.fill(white)
    pygame.draw.rect(dis, black, [x1, y1, 10, 10])

    pygame.display.update()

    clock.tick(30)

pygame.quit()
quit()
```

OUTPUT:



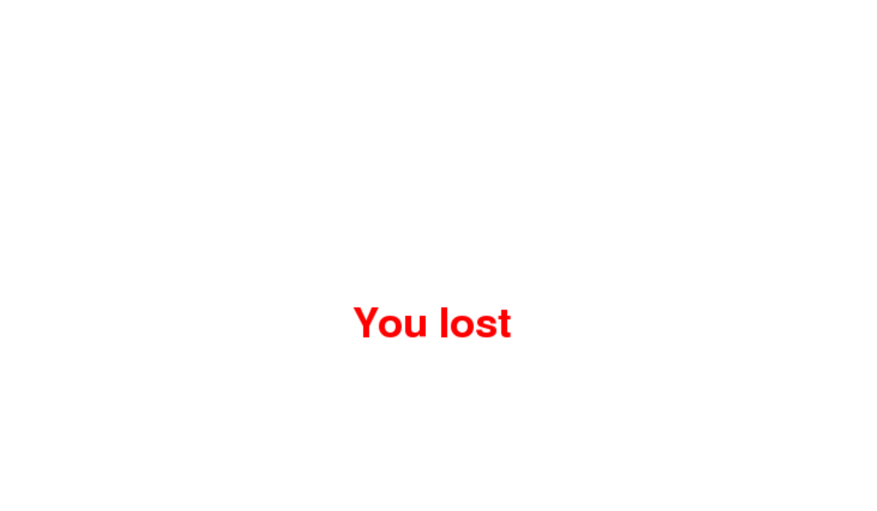
Game Over when Snake hits the boundaries:

In this snake game, if the player hits the boundaries of the screen, then he loses. To specify that, I have made use of an 'if' statement that defines the limits for the x and y coordinates of the snake to be less than or equal to that of the screen. Also, make a not over here that I have removed the hardcodes and used variables instead so that it becomes easy in case you want to make any changes to the game later on.

```
import pygame
import time
pygame.init()

white = (255, 255, 255)
```

```
black = (0, 0, 0)
red = (255, 0, 0)

dis_width = 800
dis_height  = 600
dis = pygame.display.set_mode((dis_width, dis_width))
pygame.display.set_caption('Snake Game by Edureka')

game_over = False

x1 = dis_width/2
y1 = dis_height/2

snake_block=10

x1_change = 0
y1_change = 0

clock = pygame.time.Clock()
snake_speed=30

font_style = pygame.font.SysFont(None, 50)

def message(msg,color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width/2, dis_height/2])

while not game_over:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_over = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                x1_change = -snake_block
                y1_change = 0
            elif event.key == pygame.K_RIGHT:
                x1_change = snake_block
                y1_change = 0
            elif event.key == pygame.K_UP:
                y1_change = -snake_block
                x1_change = 0
            elif event.key == pygame.K_DOWN:
                y1_change = snake_block
                x1_change = 0

    if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 0:
        game_over = True

    x1 += x1_change
    y1 += y1_change
    dis.fill(white)
    pygame.draw.rect(dis, black, [x1, y1, snake_block, snake_block])

    pygame.display.update()

    clock.tick(snake_speed)

message("You lost",red)
pygame.display.update()
time.sleep(2)

pygame.quit()
quit()
```

OUTPUT:



```
You lost
```

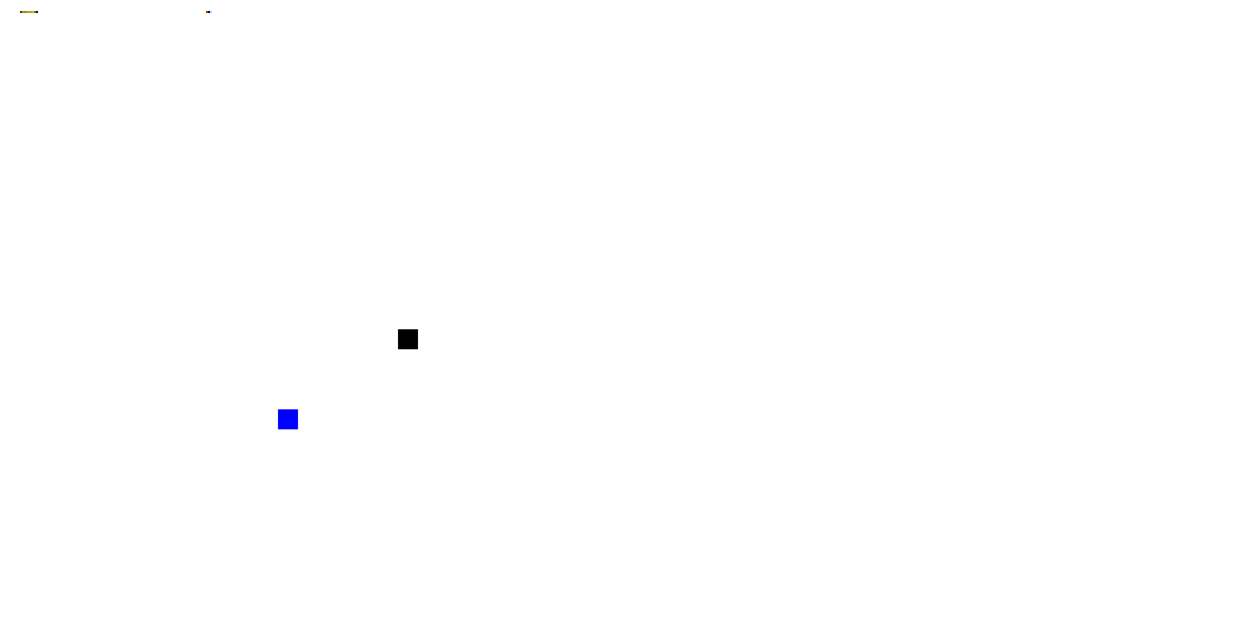
Adding the Food:

Here, I will be adding some food for the snake and when the snake crosses over that food, I will have a message saying “Yummy!!”. Also, I will be making a small change wherein I will include the options to quit the game or to play again when the player loses.

```
import pygame
import time
import random

pygame.init()

white = (255, 255, 255)
```

```
black = (0, 0, 0)
red = (255, 0, 0)
blue = (0, 0, 255)

dis_width = 800
dis_height = 600

dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Snake Game by Edureka')

clock = pygame.time.Clock()

snake_block = 10
snake_speed = 30

font_style = pygame.font.SysFont(None, 30)


def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width/3, dis_height/3])


def gameLoop():  # creating a function
    game_over = False
    game_close = False

    x1 = dis_width / 2
    y1 = dis_height / 2

    x1_change = 0
    y1_change = 0

   foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
   foody = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0

    while not game_over:

        while game_close == True:
            dis.fill(white)
            message("You Lost! Press Q-Quit or C-Play Again", red)
            pygame.display.update()

            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        game_over = True
                        game_close = False
                    if event.key == pygame.K_c:
                        gameLoop()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                game_over = True
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    x1_change = -snake_block
                    y1_change = 0
                elif event.key == pygame.K_RIGHT:
                    x1_change = snake_block
                    y1_change = 0
                elif event.key == pygame.K_UP:
                    y1_change = -snake_block
                    x1_change = 0
                elif event.key == pygame.K_DOWN:
                    y1_change = snake_block
                    x1_change = 0

        if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 0:
            game_close = True

        x1 += x1_change
        y1 += y1_change
        dis.fill(white)
    pygame.draw.rect(dis, blue, [foodx, foody, snake_block, snake_block])
        pygame.draw.rect(dis, black, [x1, y1, snake_block, snake_block])
        pygame.display.update()

        if x1 == foodx and y1 == foody:
            print("Yummy!!")
        clock.tick(snake_speed)

    pygame.quit()
    quit()


gameLoop()
```

OUTPUT:



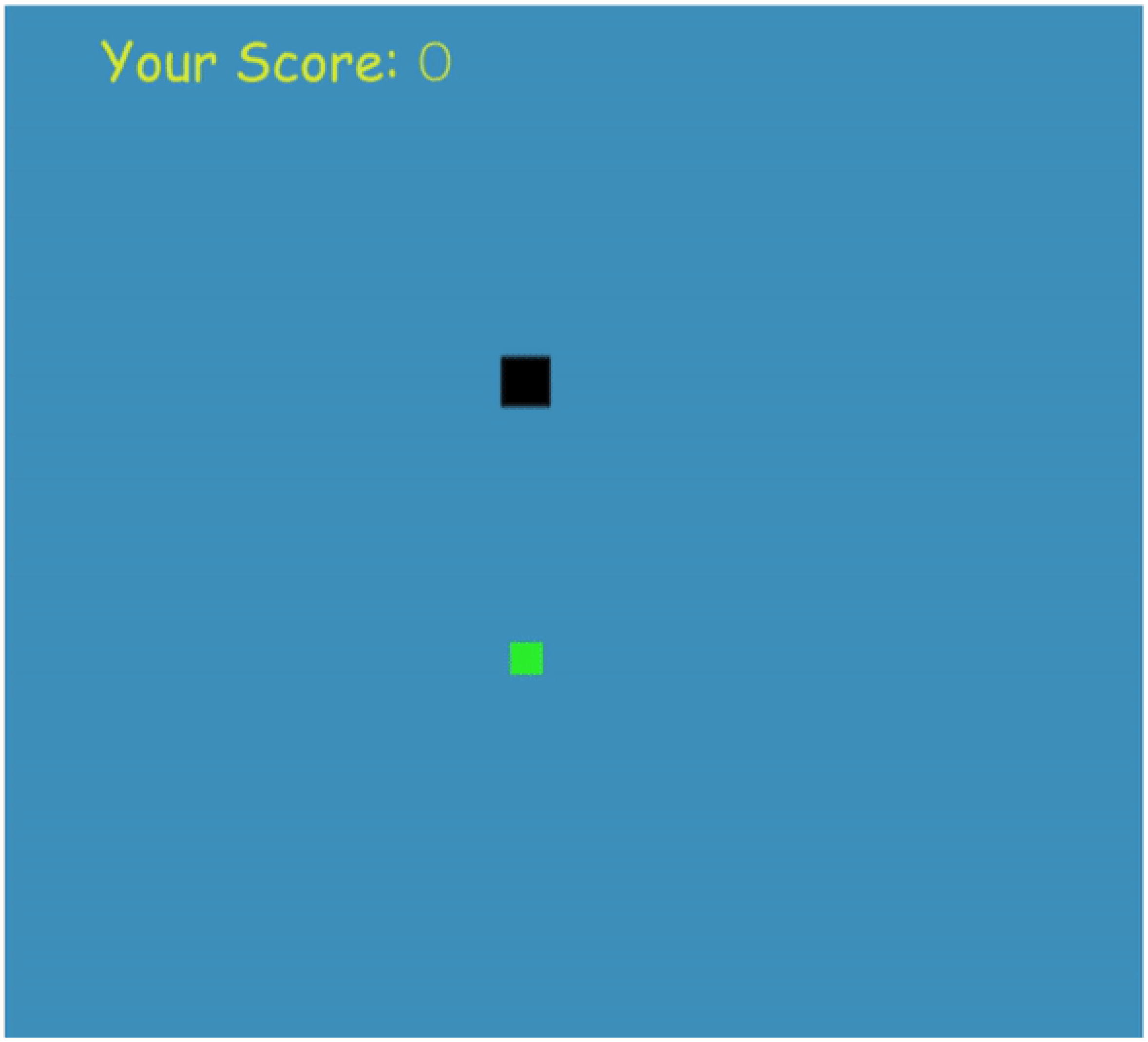
Increasing the Length of the Snake:

The following code will increase the size of our sake when it eats the food. Also, if the snake collides with his own body, the game is over and you ill see a message as "You Lost! Press Q-Quit or C-Play Again". The length of the snake is basically contained in a list and the initial size that is specified in the following code is one block.

```
import pygame
import time
import random

pygame.init()

white = (255, 255, 255)
yellow = (255, 255, 102)
black = (0, 0, 0)
red = (213, 50, 80)
green = (0, 255, 0)
blue = (50, 153, 213)

dis_width = 600
dis_height = 400

dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Snake Game by Edureka')
```

```
 clock = pygame.time.Clock()

snake_block = 10
snake_speed = 15

font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)

def our_snake(snake_block, snake_list):
    for x in snake_list
 pygame.draw.rect(dis, black, [x[0], x[1], snake_block, snake_block])


def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])


def gameLoop():
    game_over = False
    game_close = False

    x1 = dis_width / 2
    y1 = dis_height / 2

    x1_change = 0
    y1_change = 0

    snake_List = []
    Length_of_snake = 1
foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) *
 foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0

    while not game_over:

        while game_close == True:
            dis.fill(blue)
            message("You Lost! Press C-Play Again or Q-Quit", red)

            pygame.display.update()

            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        game_over = True
                        game_close = False
                if event.key == pygame.K_c:
                    gameLoop()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                game_over = True
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    x1_change = -snake_block
                    y1_change = 0
                elif event.key == pygame.K_RIGHT:
                    x1_change = snake_block
                    y1_change = 0
                elif event.key == pygame.K_UP:
                    y1_change = -snake_block
                    x1_change = 0
                elif event.key == pygame.K_DOWN:
                    y1_change = snake_block
                    x1_change = 0

        if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 0:
            game_close = True
        x1 += x1_change
        y1 += y1_change
        dis.fill(blue)
        pygame.draw.rect(dis, green, [foodx, foody, snake_block, snake_block])
        snake_Head = []
        snake_Head.append(x1)
        snake_Head.append(y1)
        snake_List.append(snake_Head)
        if len(snake_List) > Length_of_snake:
            del snake_List[0]

        for x in snake_List[:-1]:
            if x == snake_Head:
                game_close = True

        our_snake(snake_block, snake_List)


        pygame.display.update()

        if x1 == foodx and y1 == foody:
       foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
      foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0
            Length_of_snake += 1
            clock.tick(snake_speed)

        pygame.quit()
        quit()


    gameLoop()
```

OUTPUT:

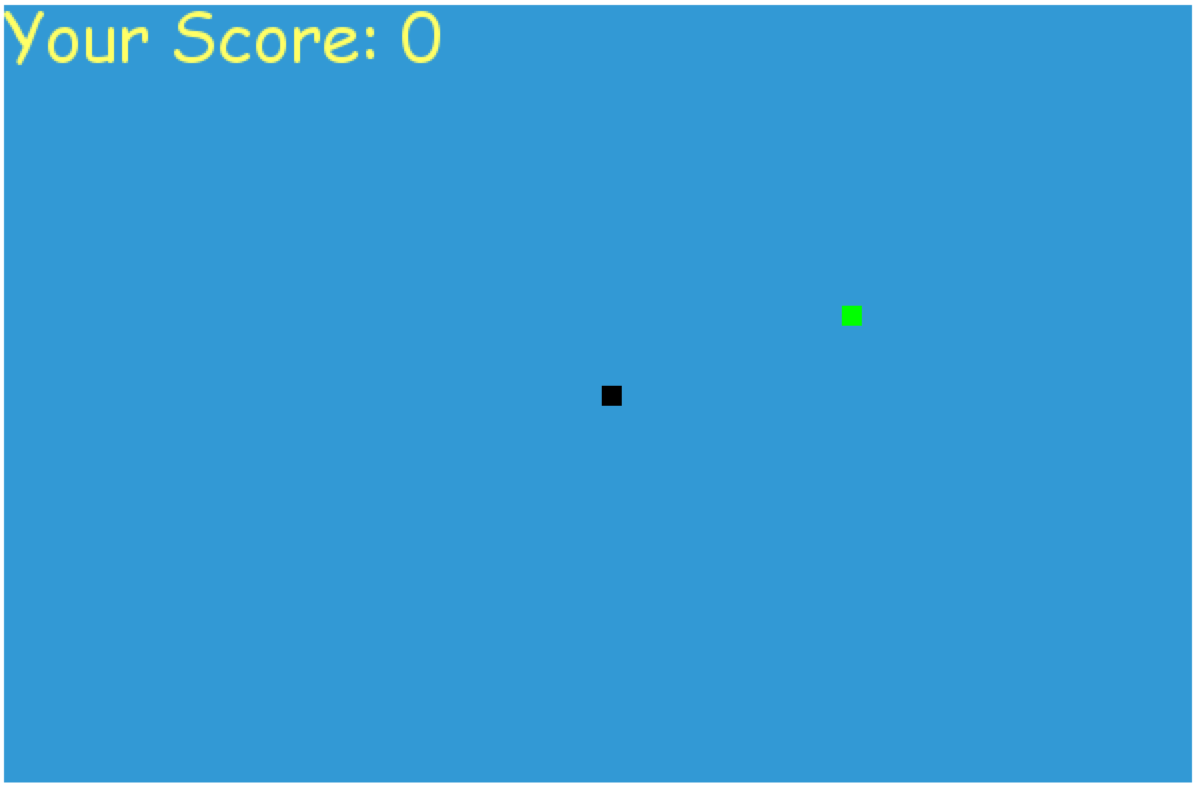


Displaying the Score:

Last but definitely not the least, you will need to display the score of the player. To do this, I have created a new function as “Your_score”. This function will display the length of the snake subtracted by 1 because that is the initial size of the snake.

```
import pygame
import time
import random

pygame.init()

white = (255, 255, 255)
yellow = (255, 255, 102)
black = (0, 0, 0)
red = (213, 50, 80)
green = (0, 255, 0)
blue = (50, 153, 213)

dis_width = 600
dis_height = 400

dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Snake Game by Edureka')

clock = pygame.time.Clock()

snake_block = 10
snake_speed = 15

font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)


def Your_score(score):
    value = score_font.render("Your Score: " + str(score), True, yellow)
    dis.blit(value, [0, 0])



def our_snake(snake_block, snake_list):
    for x in snake_list:
        pygame.draw.rect(dis, black, [x[0], x[1], snake_block, snake_block])


def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])


def gameLoop():
    game_over = False
    game_close = False

    x1 = dis_width / 2
    y1 = dis_height / 2

    x1_change = 0
    y1_change = 0

    snake_List = []
    Length_of_snake = 1

    foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
    foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0

    while not game_over:

        while game_close == True:
            dis.fill(blue)
            message("You Lost! Press C-Play Again or Q-Quit", red)
            Your_score(Length_of_snake - 1)
            pygame.display.update()

            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        game_over = True
                        game_close = False
                    if event.key == pygame.K_c:
                        gameLoop()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                game_over = True
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    x1_change = -snake_block
                    y1_change = 0
                elif event.key == pygame.K_RIGHT:
                    x1_change = snake_block
                    y1_change = 0
                elif event.key == pygame.K_UP:
                    y1_change = -snake_block
                    x1_change = 0
                elif event.key == pygame.K_DOWN:
                    y1_change = snake_block
                    x1_change = 0

        if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 0:
            game_close = True
        x1 += x1_change
        y1 += y1_change
        dis.fill(blue)
        pygame.draw.rect(dis, green, [foodx, foody, snake_block, snake_block])
        snake_Head = []
        snake_Head.append(x1)
        snake_Head.append(y1)
        snake_List.append(snake_Head)
        if len(snake_List) > Length_of_snake:
            del snake_List[0]

        for x in snake_List[:-1]:
            if x == snake_Head:
                game_close = True

        our_snake(snake_block, snake_List)
        Your_score(Length_of_snake - 1)

        pygame.display.update()

        if x1 == foodx and y1 == foody:
            foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
            foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0
            Length_of_snake += 1

        clock.tick(snake_speed)

    pygame.quit()
    quit()


gameLoop()
```

OUTPUT:



```
Your Score: 0
```