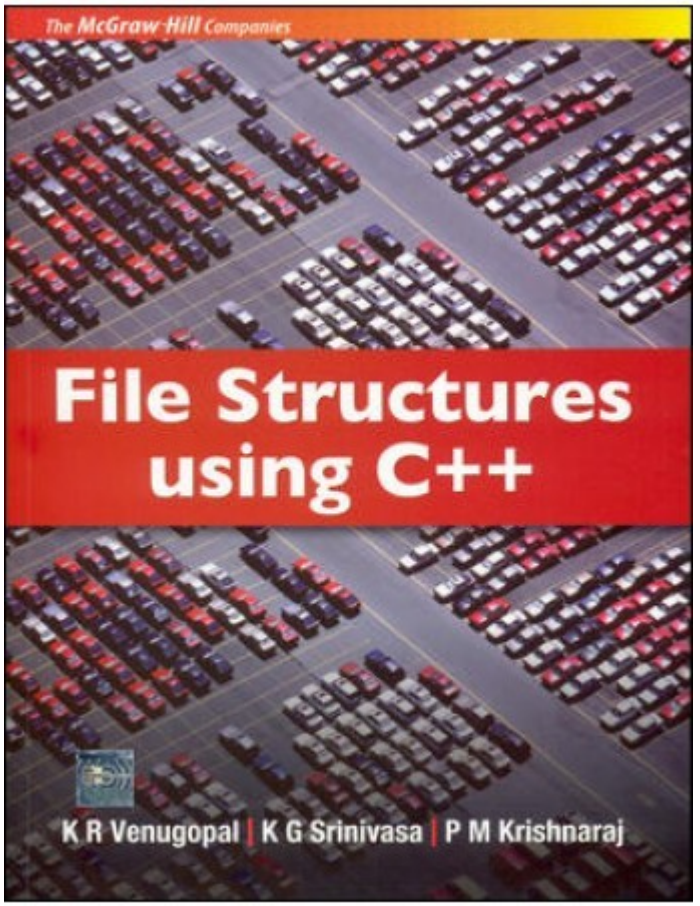


Click here for the book site

Program 1

Write a C++ program to read series of names, one per line, from standard input and write these names spelled in reverse order to the standard output using I/O redirection and pipes.

Repeat the exercise using an input file specified by the user instead of the standard input and using an output file specified by the user instead of the standard output.

Program 1

```
cout<<"How many names you want to enter ?";
cin>>count;
while(count--)
{
  cout<<"\nName:";
  cin>>name;
  rev.erase();
  for (i=name.length();i>=0;i--)
        rev+=name[i];
  cout<<"\nReverse:"<<rev;
}
```

Input data

Reverse the input

Output

Program 1

- Normal Usage
  - ./a.out
- With Input File
  - ./a.out < inputfile.txt

This should have the count also

- With Output File
  - ./a.out > outputfile.txt
- With Both Input File and Output File
  - ./a.out < inputfile.txt >outputfile.txt

Program 2

Write a C++ program to read and write student objects with fixed-length records and the fields delimited by "|".

Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.

Class Definition

```
class student
{
  public:
      string USN;
      string Name;
      string Branch;
      int Semester;
      string buffer;

      void pack();
      void unpack();
      int search(string);
      void modify(string);
      void read_data();
      void write_to_file();
      int delete_from_file(string);
}
```

Main Program

```
case 1: cout<<"\n\nhow many records to insert?\n";
        cin>>count;
        for (i=0;i<count;i++)
        {
            cout<<"Data\n";
            s1.read_data();
            s1.pack();
        }
        s1.write_to_file();
        break;
case 2: cout <<"\n\nEnter the USN to delete";
        cin>>key;
        i=s1.delete_from_file(key);
        break;
case 3: cout <<"\n\nEnter the USN to modify";
        cin>>key;
        s1.modify(key);
        break;
case 4: cout <<"\n\nEnter the USN to search";
        cin>>key;
        i=s1.search(key);
        break;
default: cout<<"\n\nWrong Choice";
```

Writing multiple records to file

void student::read_data()

```
cout<<"\nUsn:";
cin>>USN;
cout<<"\nName:";
cin>>Name;
cout<<"\nBranch:";
cin>>Branch;
cout<<"\nSemster:";
cin>>Semester;
```

void student::pack()

```
string sem,temp;
stringstream out;
out << Semester;
sem = out.str();
```

Converting integer ' Semester to string 'sem'

```
temp.erase();
```

```
temp+=USN+'|'+Name+'|'+Branch+'|'+sem;
```

Pack the fields

```
for(;temp.size()<100;) temp+='$';
```

Stuff it till reaches fixed size (100)

```
buffer+=temp+'\n';
```

Add the stuffed record to buffer

void student::write_to_file()

```
fstream file;
file.open("1.txt",ios::out|ios::app);
file<<buffer;
file.close();
```

void student::unpack()

```
string sem;
int ch=1,i=0;
USN.erase();
while (buffer[i]!='|')
  USN+=buffer[i++];

Name.erase();
i++;
while (buffer[i]!='|')
  Name+=buffer[i++];

Branch.erase();
i++;
while (buffer[i]!='|')
  Branch+=buffer[i++];

sem.erase();
i++;
while (buffer[i]!='$')
  sem+=buffer[i++];
istringstream out(sem);
out>>Semester;
```

Converting string 'sem to integer ' Semester

int student::search(string key)

```
ifstream file;
int flag=0, pos=0;
file.open("1.txt",ios::in);
while (! file.eof())
{
        buffer.erase();
        getline(file,buffer);  Read ONE record from FILE
        unpack();
        if (key==USN)
        {
                pos=file.tellg();    The location of the record in FILE
                flag=1;
        }
}
file.close();
if (flag) cout<<"Found the USN. The record is  "<<buffer;
else cout<<"Not Found ";
return pos;
```

int student::delete_from_file(string key)

```
fstream file;
char del_mark='*';
int pos=0,flag=0;

pos=search(key);
if (pos){
        file.open("1.txt");
        pos-=101;
        file.seekp(pos,ios::beg);
        file.put(del_mark);
        flag=1;
        }
file.close();
if (!flag) return 0;
else return 1;
```

Rewind to begining of the record

void student::modify(string key)

```
int choice;
```

```
if (delete_from_file(key)){
```

Delete old Record

```
cout<<"\n What to modify?";
cin>>choice;
switch(choice)
{
        case 1: cout<<"\nUSN:"; cin>>USN; break;
        case 2:   cout<<"\nName:";cin>>Name;break;
        case 3:   cout<<"\nBranch:";cin>>Branch;break;
        case 4:   cout<<"\nSemster:";cin>>Semester;break;
        default: cout <<"Wrong Choice";
}
```

Read the new Record

```
buffer.erase();
```

```
pack();
write_to_file();
```

Insert new Record

Program 3

Write a C++ program to read and write student objects with Variable - Length records using any suitable recordstructure.

Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.

Import these from Program2

Class Definition

Main Program

void student::read_data()

void student::write_to_file()

void student::unpack()

int student::search(string key)

void student::modify(string key)

void student::pack()

Fixed Length Record

```
string sem,temp;
stringstream out;
out << Semester;
sem = out.str();
temp.erase();
temp+=USN+'|'+Name+'|'+Branch+'|'+sem;
for(;temp.size()<100;) temp+='$';
buffer+=temp+'\n';
```

Variable Length Record

```
string sem;
stringstream out;
out << Semester;
sem = out.str();
buffer+=USN+'|'+Name+'|'+Branch+'|'+sem+'$'+'\n';
```

int student::delete_from_file(string key)

Fixed Length Record

```
fstream file;
char del_mark='*',t;
int pos,flag=0;
pos=search(key);
if (pos){
      file.open("1.txt");
      pos-=101;
      file.seekp(pos,ios::beg);
      file.put(del_mark);
      flag=1;
      }
file.close();
if (!flag) return 0;
else return 1;
```

Variable Length Record

```
fstream file;
char del_mark='*',t;
int pos,flag=0;
pos=search(key);
if (pos){
      file.open("1.txt");
      pos-=2;
      t=file.get();
      while (t!='$' && pos!=0)
      { pos--; file.seekp(pos,ios::beg);t=file.get();}
      if (pos!=0) file.seekp(pos+2,ios::beg);
      else file.seekp(pos,ios::beg);
      file.put(del_mark);
      flag=1;
      }
file.close();
if (!flag) return 0; else return 1;
```

Program 12

Write a C++ program to reclaim the free space resulting from the deletion of records using linked lists.

Class Declaration

```
class student
{
  public:
        string USN;
        string Name;
        string Branch;
        int Semester;
        string buffer;
        int avail[10];
        int top;

        void initialize();
        void read_data();
        void pack();
        void write_to_file();
        void unpack();
        int search(string);
        int delete_from_file(string);
        void modify(string);
};
```

int student::delete_from_file(string key)

```
fstream file;
char del_mark='*',t;
int pos,flag=0;
pos=search(key);
if (pos){
        file.open("1.txt");
        pos-=101; //skip the $$$$$$ and \n characters
        file.seekp(pos,ios::beg);
        file.put(del_mark);
        flag=1;
        avail[++top]=pos;
        cout<<"\n\nThe position of deleted record is:"<<pos;
        }

file.close();
buffer.empty();
if (!flag) return 0;
else return 1;
```

Add the address to the avail list

void student::write_to_file()

```
fstream file;
int pos;
file.open("1.txt");
pos = avail[top];
if (pos){
        file.seekp(pos,ios::beg);
        file<<buffer;
        top--;
        cout<<"\n\nReusing Deleted Space at position:"<<pos;
        }
else file<<buffer;
file.close();
```

Pop the avail list and get the position to insert

Decrement the avail list

Program 4

Write a C++ program to write student objects with Variable - Length records using any suitable field structure and to read from this file a student record using RRN.

Class Declaration

Total number of Records in the file

Import from Program 3

```
class student
{
  public:
      string USN;
      string Name;
      string Branch;
      int Semester;

      string buffer;

      int count;

      int rrn_list[100];

      void read_data();
      void pack();
      void write_to_file();
      void unpack();

      void create_rrn();
      void search_by_rrn(int);
}
```

Main Program

```
switch (choice)
    {
        case 1: cout<<"Data\n";
            s1.read_data();
            s1.pack();
            s1.write_to_file();
            break;
        case 2: cout <<"\n\nEnter the RRN";
            cin>>rrn;
            s1.search_by_rrn(rrn);
            break;
        case 3: return 0;
        default:cout<<"\n\nWrong Choice";
    }
```

void student::create_rrn()

```
ifstream file;
int pos;
count=-1;
file.open("1.txt",ios::in);
while (!file.eof())
{
        pos=file.tellg();
        buffer.erase();
        getline(file,buffer);
        rrn_list[++count]=pos;
}
file.close();
```

void student::search_by_rrn(int rrn)

```
fstream file;
if (rrn>count) cout<<"\n Not Found";
else{
    buffer.erase();
    file.open("1.txt");
    pos=rrn_list[rrn];
    file.seekp(pos,ios::beg);
    getline(file,buffer);
    cout<<"\n"<<buffer<<"\n" ;
    }
```

Program 5

Write a C++ program to implement simple index on primary key for a file of student objects. Implement add ( ),search ( ), delete ( ) using the index.

Class Declaration

```
class primary_index
{
  public:
      string USN_list[100];
      int Address_list[100];
      int count;

      void create_primary_index();
      void insert();
      void remove(string);
      void search(string);
      int search_primary_index(string);
      string extract_USN(string);
      void sort_primary_index();
}
```

Main Program

```
case 1: cout<<"Enter the Student details";
        i1.insert();
        break;
 case 2: cout <<"\n\nEnter the USN to search";
        cin>>key;
        i1.search(key);
        break;
 case 3:cout <<"\n\nEnter the USN to delete";
        cin>>key;
        i1.remove(key);
        break;
 case 4: return 0;
 default:cout<<"\n\nWrong Choice";
```

create_primary_index()

```
fstream file;
    int pos;
    string buffer,USN;
    count=-1;
    file.open("1.txt",ios::in);
    while (!file.eof())
    {
        pos=file.tellg();
        buffer.erase();
        getline(file,buffer);
        if (buffer.empty()) break;
        USN=extract_USN(buffer);
        USN_list[++count]=USN;
        Address_list[count]=pos;
    }
    file.close();
    sort_primary_index();
```

Read ONE Record

Insert into Index

Sort Index???

string primary_index::extract_USN(string buffer)

```
string USN;
int i=0;
USN.erase();
while (buffer[i]!='|')
    USN+=buffer[i++];
return USN;
```

void primary_index::sort_primary_index()

```
for (int i=0;i<=count;i++)
      {
        for (int j=i+1;j<=count;j++)
        {
            if (USN_list[i]>USN_list[j])
            {
              temp_USN=USN_list[i];
              USN_list[i]=USN_list[j];
              USN_list[j]=temp_USN;

              temp_Address=Address_list[i];
              Address_list[i]=Address_list[j];
              Address_list[j]=temp_Address;
      }}}
```

void primary_index::insert()

```
cin>>USN; cin>>Name; cin>Branch;cin>>Semester;
stringstream out;  out << Semester; sem = out.str();
```

Read student data

```
buffer.erase();
buffer=USN+'|'+Name+'|'+Branch+'|'+sem+'$'+'\n';
```

Pack

```
file.open("1.txt",ios::out|ios::app);
pos=file.tellp();
file<<buffer;
file.close();
```

Insert into file

```
USN_list[++count]=USN;
Address_list[count]=pos;
```

Add to Index

```
sort_primary_index();
```

Sort Index

void primary_index::search(string key)

```
int pos=0,address;
string buffer;
fstream file;
buffer.erase();

pos=search_primary_index(key);

if (pos){
      file.open("1.txt");
      address=Address_list[pos];
      file.seekp(address,ios::beg);
      getline(file,buffer);
      cout<<"Found Record:"<<buffer;
      file.close();
      }
else cout<<"\nNot Found";
```

int primary_index:: search_primary_index(string key)

```
int low=0,high=count,mid=0,flag=0,pos;
while (low<=high)
{
      mid=(low+high)/2;
      if (USN_list[mid]==key) {flag=1; break;}
      if (USN_list[mid]>key) high=mid-1;
      if (USN_list[mid]<key) low=mid+1;
}
if (flag) return mid;
else return 0;
```

void primary_index ::remove(string key)

Search for the required record

Delete the record from file

Delete the record from Index list

```
int pos=0,address,i;
char del_ch='*';
fstream file;

pos=search_primary_index(key);

if (pos){
      address=Address_list[pos];
      file.open("1.txt");
      file.seekp(address,ios::beg);
      file.put(del_ch);
      file.close();

      cout<<"\nRecord Deleted:  ";

      for (i=pos;i<count;i++)
      {
            USN_list[i]=USN_list[i+1];
            Address_list[i]=Address_list[i+1];
      }
      count--;
}
else cout<<"\nNot Found";
```

Program 6

Write a C++ program to implement index on secondary key, the name, for a file of student objects. Implement add( ), search( ), delete( ) using the secondary index.

Class Declaration

```
class secondary_index
{
  public:
      string Name_list[100];
      int Address_list[100];
      int count;

      void create_index();
      void insert();
      void remove(string);
      void delete_from_file(int);
      void search(string);

      int search_index(string);
      void read_from_file(int);
      string extract_Name(string);
      void sort_index();
};
```

Main Program

```
case 1: cout<<"Enter Student Details\n";
        i1.insert();
        break;

case 2: cout <<"\n\nEnter the name to search ";
        cin>>key;
        i1.search(key);
        break;

case 3:cout <<"\n\nEnter the name to delete";
        cin>>key;
        i1.remove(key);
        break;

case 4: return 0;

default:cout<<"\n\nWrong Choice";
```

void secondary_index::create_index()

```
fstream file;
int pos;
string buffer,Name;
count=-1;
file.open("1.txt",ios::in);
while (!file.eof())
{
        pos=file.tellg();
        buffer.erase();
        getline(file,buffer);
        if (buffer.empty()) break;
        Name=extract_Name(buffer);
        Name_list[++count]=Name;
        Address_list[count]=pos;
}
file.close();
sort_index();
```

string secondary_index::extract_Name(string buffer)

```
string USN,Name;
int i=0;
USN.erase();
while (buffer[i]!='|')
  USN+=buffer[i++];

Name.erase();
i++;
while (buffer[i]!='|')
  Name+=buffer[i++];

return Name;
```

void secondary_index::sort_index()

```
for (int i=0;i<=count;i++)
{
  for (int j=i+1;j<=count;j++)
  {
      if (Name_list[i]>Name_list[j])
      {
        temp_Name=Name_list[i];
        Name_list[i]=Name_list[j];
        Name_list[j]=temp_Name;

        temp_Address=Address_list[i];
        Address_list[i]=Address_list[j];
        Address_list[j]=temp_Address;
}}}
```

void secondary_index::insert()

```
cin>>USN; cin>>Name; cin>Branch;cin>>Semester;
stringstream out;  out << Semester; sem = out.str();

buffer.erase();
buffer=USN+'|'+Name+'|'+Branch+'|'+sem+'$'+'\n';

file.open("1.txt",ios::out|ios::app);
pos=file.tellp();
file<<buffer;
file.close();

Name_list[++count]=Name;
Address_list[count]=pos;

sort_primary_index();
```

void secondary_index:: search(string key)

```
pos=search_index(key);

if (pos){
        read_from_file(pos);
        t=pos;

        while (Name_list[++t]==key)
                read_from_file(t);
        t=pos;

        while (Name_list[--t]==key)
                read_from_file(t);
        }
        else cout<<"\nNot Found";
```

```
while (low<=high)
        {
                mid=(low+high)/2;
                if (Name_list[mid]==key) {flag=1; break;}
                if (Name_list[mid]>key) high=mid-1;
                if (Name_list[mid]<key) low=mid+1;
        }
        if (flag) return mid;
        else return 0;
```

```
address=Address_list[pos];
file.open("1.txt");
file.seekp(address,ios::beg);
getline(file,buffer);
file.close();
cout<<"\nFound the record:  "<<buffer;
```

void secondary_index::

remove(string key)

```
pos=search_index(key);
if (pos){
      read_from_file(pos);
      cout<<"\n Delete?";
      cin>>choice;
      if (choice) delete_from_file(pos);
      t=pos;
      while (Name_list[++t]==key){
            read_from_file(t);
            cout<<"\n Delete?";
            cin>>choice;
            if (choice) delete_from_file(t);
            }t=pos;
      while (Name_list[--t]==key){
            read_from_file(t);
            cout<<"\n Delete?";
            cin>>choice;
            if (choice) delete_from_file(t);
            }}
else cout<<"\n Not Found";
```

```
fstream file;
      file.open("1.txt");
      address=Address_list[pos];
      file.seekp(address,ios::beg);
      getline(file,buffer);
      cout<<"\nFound the record:  "<<buffer;
      file.close();
```

```
file.open("1.txt");
      address=Address_list[pos];
      file.seekp(address,ios::beg);
      file.put(del_ch);
      cout<<"\nRecord Deleted:  ";

      for (i=pos;i<count;i++)
            {
                  Name_list[i]=Name_list[i+1];
                  Address_list[i]=Address_list[i+1];
            }
      count--;
```

Program 7

Write a C++ program to read two lists of names and then match the names in the two lists using Cosequential Match based on a single loop. Output the names common to both the lists.

Class Declaration

```
class coseq
{
public:
        string list1[100], list2[100];
        int count1, count2;

        void load_list();
        void sort_list();
        void match();
};
```

Main Program

```
int main()
{
        coseq c1;
        c1.load_list();
        c1.sort_list();
        c1.match();
        return 0;
}
```

void coseq::load_list()

```
fstream file;
string name;
count1=-1;count2=-1;
file.open("name1.txt");

while (!file.eof()){
        name.erase();
        getline(file,name);
        list1[++count1]=name;
        }
file.close();

file.open("name2.txt");

while (!file.eof()){
        name.erase();
        getline(file,name);
        list2[++count2]=name;
        }

file.close();
```

void coseq::sort_list()

```
int i,j;
string temp;
for (i=0;i<=count1;i++)
{
  for (j=i+1;j<=count1;j++)
  {
       if (list1[i]>list1[j])
       {
         temp=list1[i];
         list1[i]=list1[j];
         list1[j]=temp;
       }}}


for (i=0;i<=count2;i++)
{
  for (j=i+1;j<=count2;j++)
  {
       if (list2[i]>list2[j])
       {
         temp=list2[i];
         list2[i]=list2[j];
         list2[j]=temp;
       }}}
```

void coseq::match()

```
int i=0,j=0;
while (i<=count1 && j<=count2)
{
      if (list1[i]==list2[j]) {cout<<"\n"<<list1[i];i++;j++;}
      if (list1[i] < list2[j]) i++;
      if (list1[i] > list2[j]) j++;
}
```

Program 8

Write a C++ program to read k Lists of names and merge them using k-way merge algorithm with k = 8

Class Declaration

```
class coseq
{
public:
        string list[8][100];
        string outlist[100];
        int count[8];
        int current[8];

        void load_list();
        void read_file(int);
        void sort_list(int);
        void merge();
};
```

Main Program

```
coseq c1;
c1.load_list();
c1.merge();
return 0;
```

void coseq::load_list()

```
for (int i=1;i<=8;i++)
    {
        count[i]=-1;
        read_file(i);

        sort_list(i);

    }
```

```
int i,j;
string temp;
for (i=0;i<=count[k];i++)
{for (j=i+1;j<=count[k];j++)
  {if (list[k][i]>list[k][j])
      {
        temp=list[k][i];
        list[k][i]=list[k][j];
        list[k][j]=temp;
      }}}
```

```
fstream file;
string name;
switch(i){
      case 1: file.open("name1.txt");break;
      case 2: file.open("name2.txt");break;
      case 3: file.open("name3.txt");break;
      case 4: file.open("name4.txt");break;
      case 5: file.open("name5.txt");break;
      case 6: file.open("name6.txt");break;
      case 7: file.open("name7.txt");break;
      case 8: file.open("name8.txt");break;
      }
while (!file.eof()){
      name.erase();
      getline(file,name);
      list[i][++count[i]]=name;
      }
file.close();
```

void coseq::merge()-1

```
string smallest;
int small_list, t=-1,start=1,avail[8],avail_lists=8;
for (int i=1;i<=8;i++) {avail[i]=1;current[i]=1;}

while (avail_lists>1)
{
      if (!avail[start]) {start++; continue;}
      small_list=start;
      smallest=list[start][current[start]];
  for (int i=start+1;i<=7;i++)
  {
      if (!avail[i]) continue;

      if (list[i][current[i]]<smallest)
      {
            smallest=list[i][current[i]];
            small_list=i;
      }
   }
```

Continue until ONE list remains

If current list is empty, ignore it

Consider the current element as smallest

If current element of current list is smaller, swap them

void coseq::merge()-2

```
current[small_list]++;
 if (current[small_list]>count[small_list])
{
       avail[small_list]=0;avail_lists--;
}

 outlist[++t]=smallest;

  for (int j=1;j<=8;j++)
       if (j!=small_list) {
             if (list[j][current[j]]==smallest){
                   current[j]++;}}
}
```

If current smallest element is the last element in the list, make the list unavailable for future use

Send smallest element to outlist

If any list has the element same as smallest element, increment them

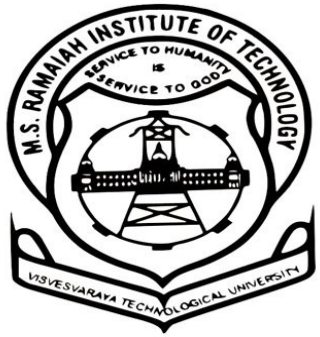
void coseq::merge()-3

```
for (int i=1;i<=8;i++) if (avail[i]) {
        for (int j=current[i];j<=count[i];j++)
              outlist[++t]=list[i][j];}

      cout<<"\nThe Merged List:";

      for (int i=0;i<=t+1;i++) cout<<"\n"<<outlist[i];
```

Send the remaining elements of the lists to outlist

Program 9

Write a C++ program to implement B-Tree for a given set of integers and its operations insert ( ) and search ( ). Display the tree.



Class Definition

```
struct node
{
        int ele[4];
        int child[4];
};
```

```
class btree
{
public:
        node *tree[10][10];
        int count[10];
        int leaf;
        int path[10];

        btree();
        node* create_node();
        void insert(int);
        void main_search(int);
        void display_tree();
        void insert_node(node*,int);
        void search(int);
        int search_node (node*,int);
        int nodefull(node*);
        void split(node*);
};
```

Main Program

```
while(1)
        {
            cout<<"\n\n\nMain Menu\n----------\n1.Insert\n2.Search\n3.Display
                Tree\n4.Exit\n\nEnter your choice:";
            cin>>choice;
            switch(choice)
            {
            case 1:  cout<<"\nEnter the element:";
                     cin>>key;
                     bt.insert(key);
                     break;
            case 2:cout<<"Enter the key:";
                     cin>>key;
                     bt.main_search(key);
                     break;
            case 3: bt.display_tree();
                     break;
            case 4: return 0;
            default: cout<<"\nEnter valid choice";
            }
        }
```

btree::btree()

```
leaf=-1;
for (int i=0;i<10;i++)
{count[i]=-1;path[i]=-1;}
```

node* btree::create_node()

```
node *n;
 n = new node;
 for (int i=0;i<4;i++) {n->ele[i]=-1;n->child[i]=-1;}
 return n;
```

void btree::insert(int key) - 1

```
int n, parent;
node *first_node;
if (leaf==-1)
{
        first_node=create_node();
        tree[0][0]=first_node;
        leaf++;count[0]++;
        first_node->ele[0]=key;
}
```

void btree::insert(int key) - 2

```
else if (leaf==0)
    {
        if (nodefull(tree[0][0]))
        {
            path[leaf]=0;
            split(tree[0][0]);
            insert(key);
        }
        else insert_node(tree[0][0],key);
    }
```

void btree::insert(int key) - 3

```
else{
        search(key);
        n=path[leaf];
        parent=path[leaf-1];

        if ( (nodefull(tree[leaf][n])) )
        {
            split(tree[leaf][n]);
            insert(key);
        }
        else
            insert_node(tree[leaf][n],key);
    }
```

void btree::main_search(int key)

```
int flag=0, i;
node *node1;
search(key);
node1=tree[leaf][path[leaf]];

for (i=0;node1->ele[i]!=-1;i++)
      if (node1->ele[i]==key) {flag=1; break;}

cout<<"\nThe path traversed is: ";
for (i=0;path[i]!=-1;i++)
      cout<<path[i]<<" -> ";

if (flag) cout <<"\nElement Found";
else cout<<"\nNot Found";
```

void btree::display_tree()

```
int i,j,k;
        for (i=0;i<=leaf;i++)
        {
                cout<<"\n\nLevel------ " <<i<<"\n";
                for (j=0;j<=count[i];j++)
                {
                        for (k=0;tree[i][j]->ele[k]!=-1;k++)
                                cout<<" "<<tree[i][j]->ele[k];
                        cout<<"\t";
                }
        }
```

void btree::search(int key)

```
int i,j,temp;
      path[0]=0;                    always start the path from root

      if (leaf){                    search only if there are more than 1 level

            j=0;
            for (i=0;i<leaf;i++)
            {
                  temp=search_node(tree[i][j],key);
                  path[i+1]=temp;
                  j=temp;
            }}
```

int btree::search_node(node *node1, int key)

```
for (int i=0;i<4;i++)
        {
                if (key<=node1->ele[i]) return node1->child[i];
                else if (node1->ele[i+1]==-1) return node1->child[i];
        }
```

int btree::nodefull(node *node1)

```
if (node1->ele[3]!=-1) return 1;

else return 0;
```

void btree::insert_node(node *node1, int key) -1

```
int flag=0, count=-1,i,j, x, y, l;
node *newnode, *parent;
for (i=0;i<4;i++) if (node1->ele[i]!=-1) ++count;
i=0;
while (!flag && node1->ele[i]!=-1)
{
      if (node1->ele[i] > key)
      {
            flag=1;
            for (int j=count;j>=i;j--)
               node1->ele[j+1]=node1->ele[j];
            node1->ele[i]=key;
      }
      i++;
}
```

not considering duplicate entries

void btree::insert_node(node *node1, int key) -2

```
if (!flag)                                  if highest element added at end, update parent
      {
            node1->ele[count+1]=key;
            for (i=leaf-1;i>=0;i--)
            {
                  n1=tree[i][path[i]];
                  for (t=0;n1->ele[t]!=-1;t++);
                  n1->ele[t-1]=key;
      }     }
```

void btree::split(node *oldnode) - 1

```
node *newnode, *parent, *n1, *n2;
int i,j,k,n,t,x,y,pos;

newnode = create_node();                    create new node

newnode->ele[0]=oldnode->ele[2];            copy elements to new node
newnode->ele[1]=oldnode->ele[3];


oldnode->ele[2]=-1;                         delete entries in old node
oldnode->ele[3]=-1;


t=count[leaf];
n=path[leaf];
for (i=t,j=t+1;i>n;i--,j--)                 move the elements in leaf level one place right
      tree[leaf][j]=tree[leaf][i];
```

void btree::split(node *oldnode) - 2

```
tree[leaf][n+1] = newnode;          insert new node to the tree
count[leaf]++;                      increse the count of leaf level nodes

x=leaf;

if (count[leaf]+1==1) t=1; else t=log(count[leaf]+1)/log(2);
                                    how many levels does the tree need?

if (t!=leaf)                        increase the level of the tree
{
      ++leaf;
      count[leaf]=count[x];

      for (i=0;i<=count[leaf];i++)          copy the leaf nodes to the new level
            std::swap(tree[leaf][i],tree[x][i]);
}

for (i=leaf-1;i>=0;i--) count[i]=-1;        empty the tree
```

void btree::split(node *oldnode) - 3

```
for (i=t,j=i-1;i>0;i--,j--)                    create the parent nodes
    {
        for (k=0;k<(count[i]+1)/2;k++)
        {
            n1=tree[i][2*k];
            n2=tree[i][(2*k)+1];

            for (x=0;n1->ele[x]!=-1;x++);     find last element in the nodes
            for (y=0;n2->ele[y]!=-1;y++);

            newnode=create_node();
            count[j]++;
            tree[j][count[j]]=newnode;

            newnode->ele[0]=n1->ele[x-1];       copy them to new parent
            newnode->child[0]=2*k;
            newnode->ele[1]=n2->ele[y-1];
            newnode->child[1]=(2*k)+1;
        }
```

void btree::split(node *oldnode) - 4

```
if (count[i]!=1 && count[i]%2==0)
{
        {
            n2=tree[i][count[i]];
            for (y=0;n2->ele[y]!=-1;y++);
            newnode->ele[2]=n2->ele[y-1];
            newnode->child[2]=count[i];
        }
    }
```

if one node is remaining at leaf copy its highest element to parent

Program 10

Write a C++ program to implement B+ Tree for a given set of integers and its operations insert ( ) and search ( ). Display the tree.

Class Declaration

```
class bptree
{
public:
        node *tree[10][10];
        int count[10];
        int leaf;
        int path[10];
        node *head;

        bptree();
        node* create_node();
        void insert(int);
        void main_search(int);
        void display_tree();
        void insert_node(node*,int);
        void search(int);
        int search_node (node*,int);
        int nodefull(node*);
        void split(node*);
        void display_seqset();
};
```

```
struct node
{
        int ele[4];
        int child[4];
        node *next;
};
```

Main Program

```
while(1)
{
      cout<<"\n\n\nMain Menu\n------------------\n1.Insert\n2.Search\n3.Display
                  Tree\n4.Display Sequence Set\n5.Exit\n\nEnter your choice:";
      cin>>choice;
switch(choice)
{
      case 1:  cout<<"\nEnter the element:";
                  cin>>key;
                  bt.insert(key);
                  break;
      case 2:cout<<"Enter the key:";
                  cin>>key;
                  bt.main_search(key);
                  break;
      case 3: bt.display_tree();
                  break;
      case 4: bt.display_seqset();
                  break;
      case 5: return 0;
      default: cout<<"\nEnter valid choice";
}
```

bptree::bptree()

```
leaf=-1;
for (int i=0;i<10;i++)
{count[i]=-1;path[i]=-1;}
```

node* bptree::create_node()

```
node *n;
n = new node;
for (int i=0;i<4;i++) {n->ele[i]=-1;n->child[i]=-1;}

n->next=NULL;

return n;
```

void bptree::insert(int key)

```
if (leaf==-1)
{
        first_node=create_node();
        tree[0][0]=first_node;
        leaf++;count[0]++;
        first_node->ele[0]=key;

        head=first_node;                    header node of seq set

}
```

void bptree::display_tree()

```
int i,j,k;
for (i=0;i<=leaf;i++)
{
      cout<<"\n\nLevel------ " <<i<<"\n";
      for (j=0;j<=count[i];j++)
      {
            if (i!=leaf) k=1; else k=0;

            for (;tree[i][j]->ele[k]!=-1;k++)
                  cout<<" "<<tree[i][j]->ele[k];
            cout<<"\t";
      }
}
```

print first element only at leaf level

void bptree::display_seqset()

```
node *t;
int k;
t=head;
cout<<"\n\nThe sequence set is:";
while (t)
{
      for (k=0;t->ele[k]!=-1;k++)
            cout<<" "<<t->ele[k];
      cout<<"\t";
      t=t->next;
}
```

int bptree::search_node(node *node1, int key)

```
if (key<=node1->ele[0])
      return  node1->child[0];

for (int i=1;i<4;i++)
{
    if ((key >= node1->ele[i]) && (key < node1->ele[i+1]))
              return node1->child[i];

      else if (node1->ele[i+1]==-1)
              return node1->child[i];

}
```

key less than first element in node

key more than last element in node

void bptree::insert_node(node *node1, int key) - 1

same as btree

```
int flag=0, count=-1,i,j, x, y, l;
node *newnode, *parent;
for (i=0;i<4;i++) if (node1->ele[i]!=-1) ++count;
i=0;
while (!flag && node1->ele[i]!=-1)
{
      if (node1->ele[i] > key)
      {
            flag=1;
            for (int j=count;j>=i;j--)
               node1->ele[j+1]=node1->ele[j];
            node1->ele[i]=key;
      }
      i++;
}
```

not considering duplicate entries

void bptree::insert_node(node *node1, int key) - 2

```
if (!flag)  node1->ele[count+1]=key;
```

highest element added at end

```
if (node1->ele[0]==key)
{

      for (i=leaf-1;i>=0;i--)
      {
            x=path[i+1];
            if (tree[i][path[i]]->ele[x] > key) tree[i][path[i]]->ele[x]=key;
            else insert_node(tree[i][x],key);
}     }
```

new element is the lowest, hence propogate this till root

void bptree::split(node *oldnode) – 1

(taken from btree)

```
node *newnode, *parent, *n1, *n2;
int i,j,k,n,t,x,y,pos;

newnode = create_node();                    create new node

newnode->ele[0]=oldnode->ele[2];            copy elements to new node
newnode->ele[1]=oldnode->ele[3];


oldnode->ele[2]=-1;                         delete entries in old node
oldnode->ele[3]=-1;


t=count[leaf];
n=path[leaf];
for (i=t,j=t+1;i>n;i--,j--)                 move the elements in leaf level one place right
      tree[leaf][j]=tree[leaf][i];
```

additional tasks for B+ Trees

newnode->next=tree[leaf][n]->next; updating the next pointers

tree[leaf][n]->next=newnode;

void bptree::split(node *oldnode) – 2

(taken from btree)

```
tree[leaf][n+1] = newnode;          insert new node to the tree
count[leaf]++;                      increse the count of leaf level nodes

x=leaf;

if (count[leaf]+1==1) t=1; else t=log(count[leaf]+1)/log(2);
                                    how many levels does the tree need?

if (t!=leaf)                        increase the level of the tree
{
      ++leaf;
      count[leaf]=count[x];

      for (i=0;i<=count[leaf];i++)          copy the leaf nodes to the new level
            std::swap(tree[leaf][i],tree[x][i]);
}

for (i=leaf-1;i>=0;i--) count[i]=-1;        empty the tree
```

void bptree::split(node *oldnode) – 3

(taken from btree)

```
for (i=t,j=i-1;i>0;i--,j--)                    create the parent nodes
    {
        for (k=0;k<=count[i]/3;k++)
        {
            n1=tree[i][2*k];
            n2=tree[i][(2*k)+1];

            for (x=0;n1->ele[x]!=-1;x++);      find last element in the nodes
            for (y=0;n2->ele[y]!=-1;y++);

            newnode=create_node();
            count[j]++;
            tree[j][count[j]]=newnode;

            newnode->ele[0]=n1->ele[0];        copy first elements to new parent
            newnode->child[0]=2*k;
            newnode->ele[1]=n2->ele[0];
            newnode->child[1]=(2*k)+1;
        }
```

void btree::split(node *oldnode) – 4

(taken from btree)

```
if (count[i]!=1 && count[i]%2==0)
{
        {
            n2=tree[i][count[i]];

            for (y=0;n2->ele[y]!=-1;y++);

            newnode->ele[2]=n2->ele[0];

            newnode->child[2]=count[i];
        }
    }
```

if one node is remaining at leaf copy its highest element to parent

copy the first element to parent

Program 11

Write a C++ program to store and retrieve student data from file using hashing. Use any collision resolution technique.

Class Declaration

```
class student
{
  public:
        string USN;
        string Name;
        string Branch;
        int Semester;
        string buffer;

        void read_data();
        void pack();
        void write_to_file();
        void unpack(int);
        void search(string);
        int hash(string);

}
```

Main Program

```
while (1){
cout <<"\nMain Menu\n 1.Add \n\n 2.Search \n\n 3.Exit\n\nEnter the choice:";
cin>>choice;
switch (choice)
{
      case 1: cout<<"Data\n";
            s1.read_data();
            s1.pack();
            s1.write_to_file();
            break;

      case 2: cout <<"\n\nEnter the key";
            cin>>key;
            s1.search(key);
            break;

      case 3: return 0;

      default: cout<<"\n\nWrong Choice";
}
```

int student::hash(string key)

```
int student::hash(string key)
{
      int t;
      t =  (((key[7]-48) *100) + ((key[8]-48)*10) + (key[9]-48)) % 9;
      if (t==0) return 9 else return t;
}
```

Key	Hash Function	Return Value
1MS06IS001	001 % 9	1
1MS06IS029	029 % 9	2
1MS06IS135	135 % 9	9

void student::read_data()

```
cout<<"\nUsn:";
cin>>USN;
cout<<"\nName:";
cin>>Name;
cout<<"\nBranch:";
cin>>Branch;
cout<<"\nSemster:";
cin>>Semester;
```

void student::pack()

```
string sem,temp;
stringstream out;
out << Semester;
sem = out.str();
buffer.erase();
temp.erase();
temp+=USN+'|'+Name+'|'+Branch+'|'+sem;
for(;temp.size()<100;) temp+='$';
buffer=temp+'\n';
```

void student::write_to_file() - 1

```
fstream file;
string temp;
int count, pos;

pos=hash(USN);
pos--;
pos=pos*304;

file.open("1.txt");
file.seekp(pos,ios::beg);
getline(file,temp);
file.close();

count=temp[0]-48;
```

Find the hash value

Find the position to insert

How many records are there in this address?

void student::write_to_file() - 2

```
file.open("1.txt");

if (count<0)
{
      file.seekp(pos,ios::beg);
      file.put('1');
      pos=pos+1;
}
else if (count==1)
{
      file.seekp(pos,ios::beg);
      file.put('2');
      pos=pos+102;
}
else if (count==2)
{

      file.seekp(pos,ios::beg);
      file.put('3');
      pos=pos+203;
}
```

Increment the count of records

void student::write_to_file() - 3

```
cout<<"\nInserting at:"<<pos;
file.seekp(pos,ios::beg);
file<<buffer;
file.close();

if (count==3)
      cout<<"\n\nCannot Insert....Overflow";
```

Insert the record


```
string sem;
int ch=1,i=0;
USN.erase();

if (flag==1) i++;          skip the count if there are records in that address

while (buffer[i]!='|')
  USN+=buffer[i++];

Name.erase();
i++;
while (buffer[i]!='|')
  Name+=buffer[i++];

Branch.erase();
i++;
while (buffer[i]!='|')
  Branch+=buffer[i++];

sem.erase();
i++;
while (buffer[i]!='$')
  sem+=buffer[i++];
istringstream out(sem);
out>>Semester;
```

void student::unpack(int flag)

void student::search(string key) -1

```
fstream file;
int flag=0, pos=0, count,i=1;
string temp;

pos=hash(key);
pos--;
pos=pos*304;

file.open("1.txt");
file.seekp(pos,ios::beg);
getline(file,temp);

count=temp[0]-48;
```

Hash and find the required address

Read the count

void student::search(string key) -2

```
file.seekp(pos,ios::beg);

while (i<=count)
{
      buffer.erase();
      getline(file,buffer);
      unpack(i++);
      if (key==USN) flag=1;
}

if (!flag) cout<<"\n\nKey not found:";
else {
      cout<<"\nThe Record details are-";
      cout<<USN<<Name<<Branch<<<<Semester;
    }
file.close();
```

Read each record and compare