**Chapter 1**

# Introduction

The tower of Hanoi is based on an ancient mathematical puzzle in which there are three vertical posts, one of which has a number of round donut-like disks threaded over it, stacked in descending size order, the largest disk on the bottom ranging to the smallest on the top.

The object of the puzzle I to move the entire stack of disks to another of the three posts while moving only one disk has to be moved with every other turn, while each successively lager disk is moved a proportionally fewer amount of times.

The disks correspond to the media sets of tape rotation scheme and the moves to the backup jobs themselves.

In information technology, a backup, or the process of backing up, refers to copying and archiving of computer data so it may be used to restore the original after a data loss event.

Backups have two distinct purposes is to recover data after its loss, be it by data deletion or corruption. The secondary purpose of backups is to recover data from an earlier time, according to a user-defined data retention policy.

Regular backups are vital to protect ourselves against failures of the software, the hardware, or even of the users. There is always a conflict between the need to have frequent backups and the cost of keeping such backups for a long time. The Tower of Hanoi pattern is a useful compromise. The name comes from the children's game where you start with three pegs on a piece of board. The object is to move a pile of discs from one peg to another by moving one disc at a time and never putting a larger disc on a smaller one.

The code is implemented using object oriented language called c++.

The following topics are used in the system module.

* Polymorphism
* Object
* Classes
  1. **Motivation of the project**

The Tower of Hanoi rotation scheme lets you keep several current copies of data, several week-old copies, and a few month- or year-old copies.

Each tape set is used a different number of times. When a new tape set is added, it is slated to be reused every other rotation. Older tape sets are used every fourth rotation, every eighth rotation, and so on. You can perform a tape-set rotation daily or weekly. For example, if you have five weekly tape sets labeled A, B, C, D, and E, your tape rotation would look like this: A B A C A B A D A B A C A B A E (each letter represents a week of backups).

Any backup strategy starts with a concept of data repository. The backup data needs to be stored, and probably should be organized to a degree. The organization could be as simple as a sheet of paper with a list of all backup media ( castes ) and the dates they were produced .A more sophisticated setup could include a computerized index, catalog, or relational data base. Different approaches have different advantages. part of model is the backup rotation scheme .

**Incremental**

An incremental style repository aims to make it more feasible to store backups from more points in time by organizing the data into increments of change between points in time.

**Differential**

Each differential backup saves the data that has changed since the last full backup. It has advantage that only a maximum of two data sets are needed to restore the data.

**1.2Problem Statement**

“To design and implement Back up Rotation System to reduce tapes requirement compared with the GFS Scheme”

**Chapter 2**

# System Requirements

**2.1Hardware and Software Requirement**

* **Hardware System Configuration**:

Processor - Intel Core i5

Speed - 1.8 GHz

RAM - 256 MB (min)

Hard Disk - 10 GB

* **Software System Configuration:**

Operating System -windows

Programming Language - C++

Compiler - minggw Compiler

**Chapter 3**

**System Design**

**3.1System Module**

Step 1: Start

Step 2: Enter help to get list of commands

While (true)

Accept command

If (cmd==“back up”)

Enter address of folder to back up

1.Ob.backup (str);

Get number of days

Else if(backup==8)

Backups are already done

Step 3: If(cmd==“restore”)

Enter the day to backup and restore

If(num>=ob.get number of days() )

It is not yet backed up

1.ob.restore(num);

Step 4: if(cmd==“exit”)

Process done

Else

If(cmd==“help”)

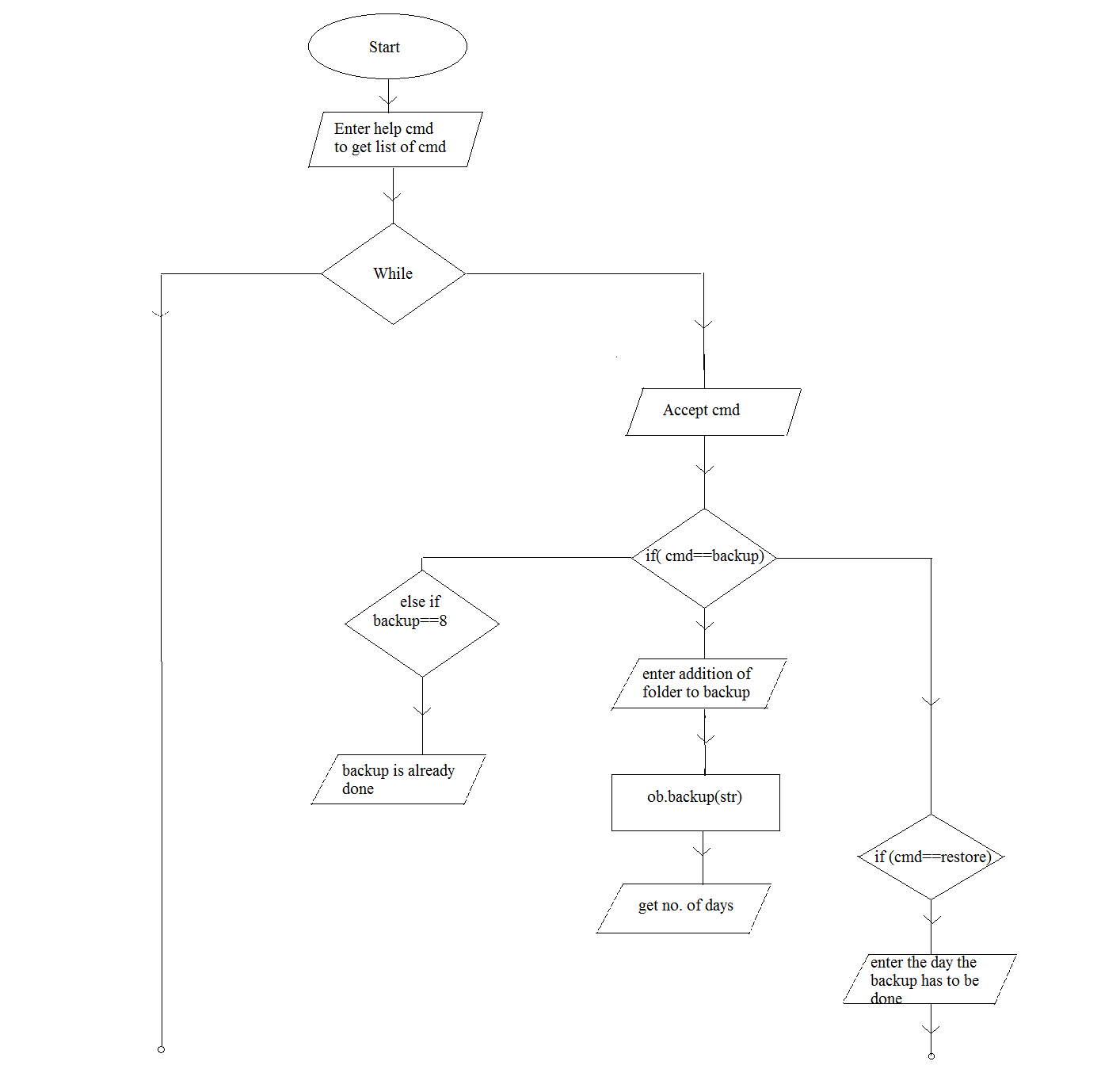
1.ob.help();

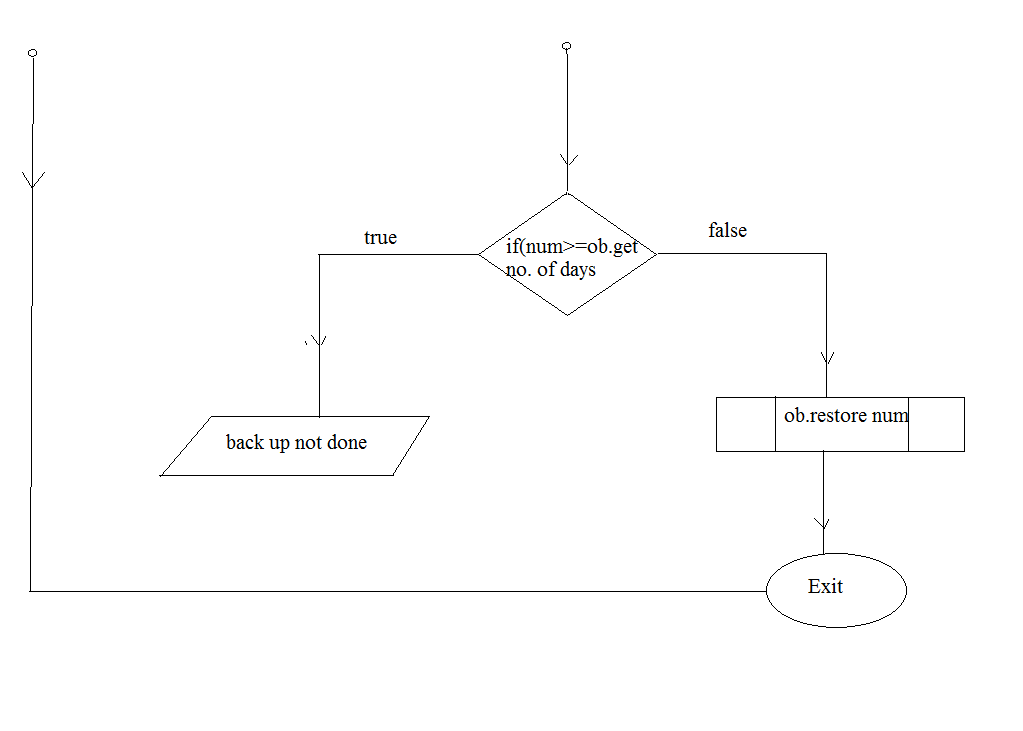
Else

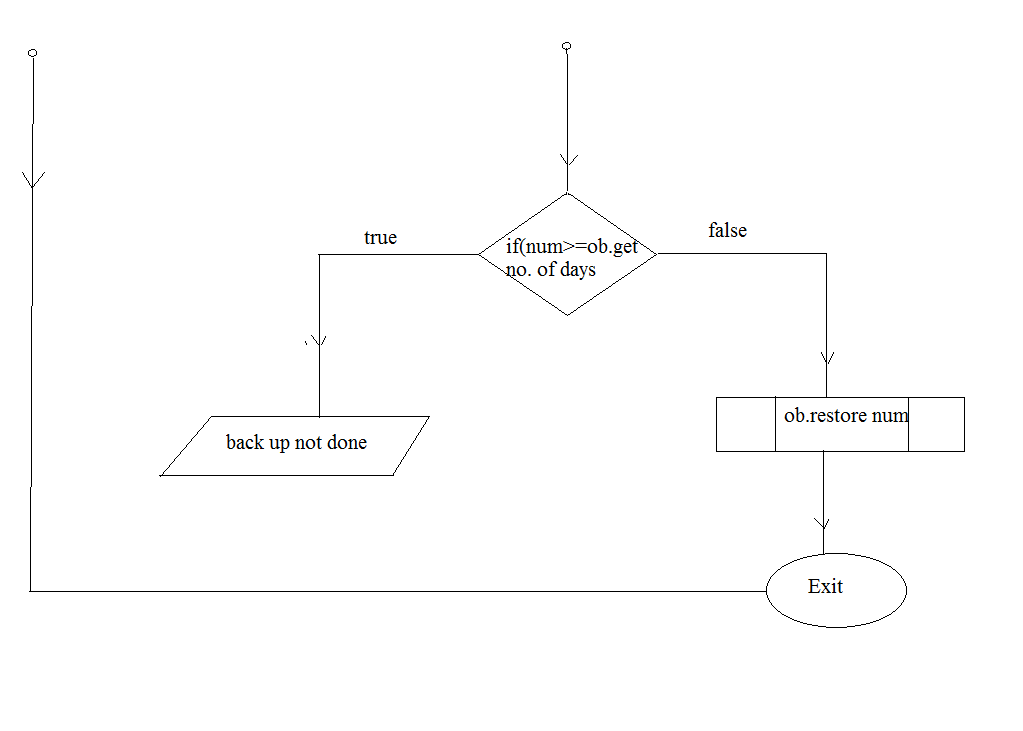
Invalid command

Step 5: Stop

**3.2Architecture**







**3.3 SYSTEM MODULE (CODE)**

include<windows.h>

#include <bits/stdc++.h>

using namespace std;

class Backup {

private:

string path;

intnumber\_of\_day;

const string BACKUP\_STORE = "C:\\BACKUP";

const string RESTORE\_PATH = "C:\\RESTORE";

string str[9] = {"#", "1", "12", "13", "134", "15", "156", "17", "178"};

vector<string>last\_full\_backup;

vector<string>last\_backup;

vector<string>Backup\_names[10];

public:

Backup();

vector<string>get\_filenames(string folder);

boolIspathexist(string path);

boolcreate\_path(string path);

boolcpy\_files(string src, string dest);

voidfull\_backup(string src, string dest);

boolsearch\_vector(vector <string>vec, string str);

voidincremental\_backup(string src, string dest);

voiddifferential\_backup(string src, string dest);

void backup(string path);

void Restore(intNumber\_of\_day);

intget\_number\_of\_day();

void help();

};

Backup :: Backup()

{

number\_of\_day = 1;

create\_path(RESTORE\_PATH);

}

vector<string> Backup :: get\_filenames(string folder)

{

vector<string> names;

stringsearch\_path = folder + "/\*.\*";

WIN32\_FIND\_DATA fd;

HANDLE hFind= ::FindFirstFile(search\_path.c\_str(), &fd);

if(hFind != INVALID\_HANDLE\_VALUE) {

do {

// read all (real) files in current folder

// , delete '!' read other 2 default folder . and ..

if(! (fd.dwFileAttributes& FILE\_ATTRIBUTE\_DIRECTORY) ) {

names.push\_back(fd.cFileName);

}

}while(::FindNextFile(hFind, &fd));

::FindClose(hFind);

}

return names;

}

bool Backup :: Ispathexist (string path)

{

const DWORD attributes = GetFileAttributes(path.c\_str());

if (INVALID\_FILE\_ATTRIBUTES != attributes)

{

return true;

}

return false;

}

bool Backup :: create\_path(string path)

{

stringcmd = "mkdir ";

system((cmd + path).c\_str());

if(!Ispathexist(path))

{

return false;

}

return true;

}

bool Backup :: cpy\_files(string src, string dest)

{

stringcmd = "xcopy ";

system((cmd + src + " " + dest).c\_str());

}

void Backup :: full\_backup(string src, string dest)

{

cpy\_files(src, dest);

last\_full\_backup.empty ();

last\_backup.empty();

Backup\_names[number\_of\_day].empty();

last\_full\_backup = get\_filenames(src);

last\_backup = get\_filenames(src);

Backup\_names[number\_of\_day] = get\_filenames(src);

}

bool Backup :: search\_vector(vector <string>vec, string str)

{

for(inti = 0; i<vec.size(); i++)

{

if(str == vec[i]) return true;

}

return false;

}

void Backup :: incremental\_backup(string src, string dest)

{

Backup\_names[number\_of\_day].empty();

vector<string>to\_check;

to\_check = get\_filenames(src);

for(inti = 0; i<to\_check.size(); i++)

{

if(!search\_vector(last\_backup, to\_check[i]))

{

cpy\_files((src + "\\" + to\_check[i]), dest);

Backup\_names[number\_of\_day].push\_back(to\_check[i]);

}

}

}

void Backup :: differential\_backup(string src, string dest)

{

Backup\_names[number\_of\_day].empty();

last\_backup.empty();

vector<string>to\_check;

to\_check = get\_filenames(src);

for(inti = 0; i<to\_check.size(); i++)

{

if(!search\_vector(last\_full\_backup, to\_check[i]))

{

cpy\_files((src + "\\" + to\_check[i]), dest);

Backup\_names[number\_of\_day].push\_back(to\_check[i]);

last\_backup.push\_back(to\_check[i]);

}

}

}

void Backup :: backup(string path)

{

if(Ispathexist(path))

{

if(!Ispathexist(BACKUP\_STORE))

{

if(!create\_path(BACKUP\_STORE))

{

cout<< "Something went wrong! Try running command prompt as Administrator\n";

return;

}

}

stringnew\_file = "";

new\_file += char('0' + number\_of\_day);

stringnew\_path = BACKUP\_STORE + "\\" + new\_file;

if(!create\_path(new\_path))

{

cout<< "Something went wrong! Try running command prompt as Administrator\n";

return;

}

switch(number\_of\_day)

{

case 1 : full\_backup(path, new\_path);

break;

case 2 :

case 4 :

case 6 :

case 8 : incremental\_backup(path, new\_path);

break;

case 3 :

case 5 :

case 7 : differential\_backup(path, new\_path);

break;

}

number\_of\_day++;

if(number\_of\_day> 8) number\_of\_day -= 8;

}

}

void Backup :: Restore(intNumber\_of\_day)

{

string s = str[Number\_of\_day];

stringdest = "";

dest += RESTORE\_PATH;

dest += "\\";

dest += char('0' + Number\_of\_day);

if(!Ispathexist(dest))

{

if(!create\_path(dest))

{

cout<< "Something went wrong" <<endl;

return ;

}

}

for(inti = 0; i<s.size(); i++)

{

char c = s[i];

stringsrc = "";

src += BACKUP\_STORE + "\\";

src += c;

cpy\_files(src, dest);

}

}

int Backup :: get\_number\_of\_day()

{

returnnumber\_of\_day;

}

void Backup :: help()

{

cout<< "Use" <<endl;

cout<< "backup for Backing up" <<endl;

cout<< "restore for restoring" <<endl;

cout<< "exit - to exit" <<endl;

}

int main()

{

Backup ob;

int backed = 0;

stringcmd;

cout<< "Type help to get list of commands" <<endl;

while(true)

{

cin>>cmd;

if(cmd == "backup")

{

if(backed == 8)

{

cout<< "All 8 Backups have been used. Restart the program to get 8 chances again." <<endl;

continue;

}

stringstr;

cout<< "Enter the address of folder that contains file to be backed(Format : C:\\\\Folder\\\\File)" <<endl;

cin>>str;

backed = ob.get\_number\_of\_day();

ob.backup(str);

} else

if(cmd == "restore")

{

intnum;

cout<< "Enter the day whosw backup to be restored" <<endl;

cin>>num;

if(num>= ob.get\_number\_of\_day())

{

cout<< "It is yet not backed up" <<endl;

continue;

}

ob.Restore(num);

} else

if(cmd == "exit")

{

cout<< "Process done" <<endl;

break;

}else

if(cmd == "help")

{

ob.help();

}

else

{

cout<< "Invalid Command" <<endl;

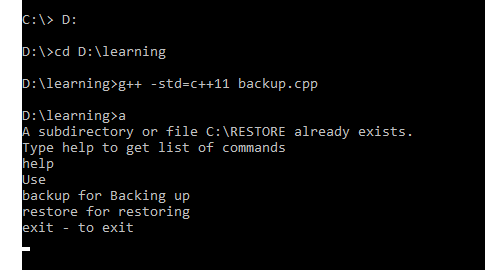
}

}

return 0;

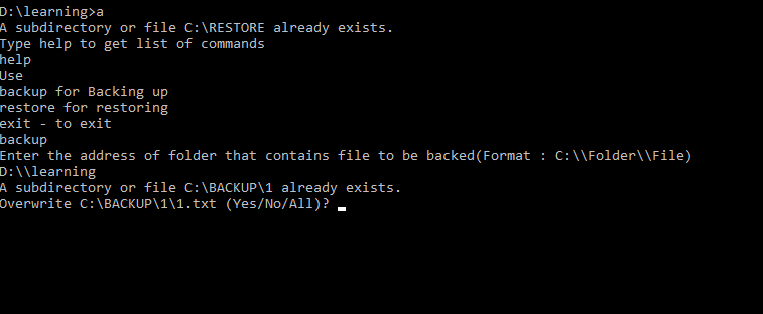
}

**3.3.1 OUTPUT**

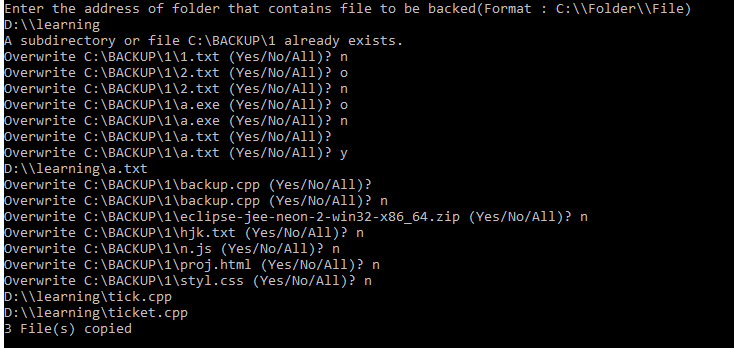


**Snapshot 1:** Navigating the backup cpp file location using g++ -std=c++11backup.cpp command.

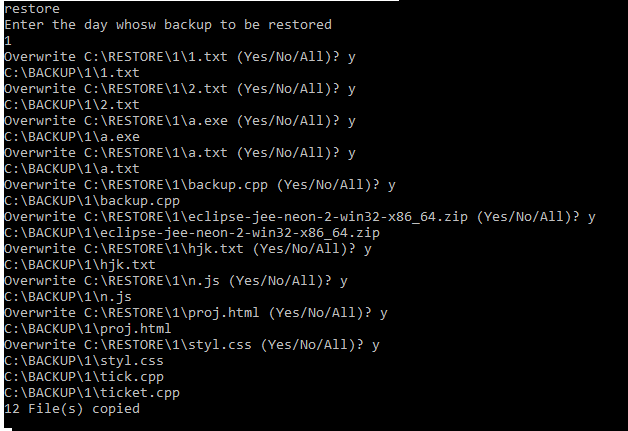
Type help to get list of commands



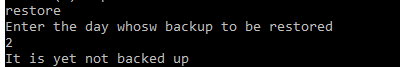
**Snapshot 2:** Backup command is entered. Enter the address of that folder that contains a file to backup in the above mentioned manner.



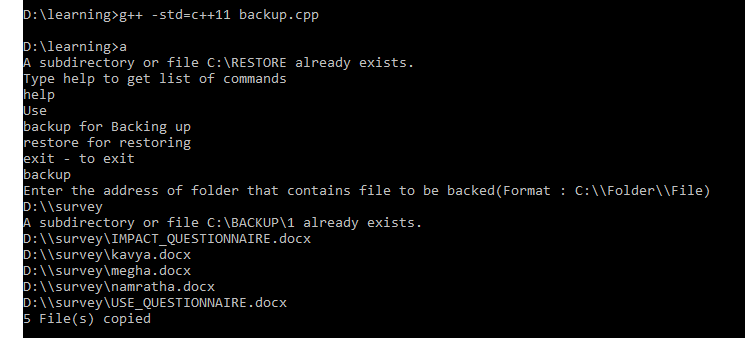
**Snapshot 3:** If the files are already backed up, then it asks permission to overwrite. If we give yes(y) it overwrites and 13 files are copied



**Snapshot 4:** Restore command is entered. The day 1 is entered to restore. If the files are already restored it ask permission to overwrite to restore the files y is entered the files are restored.

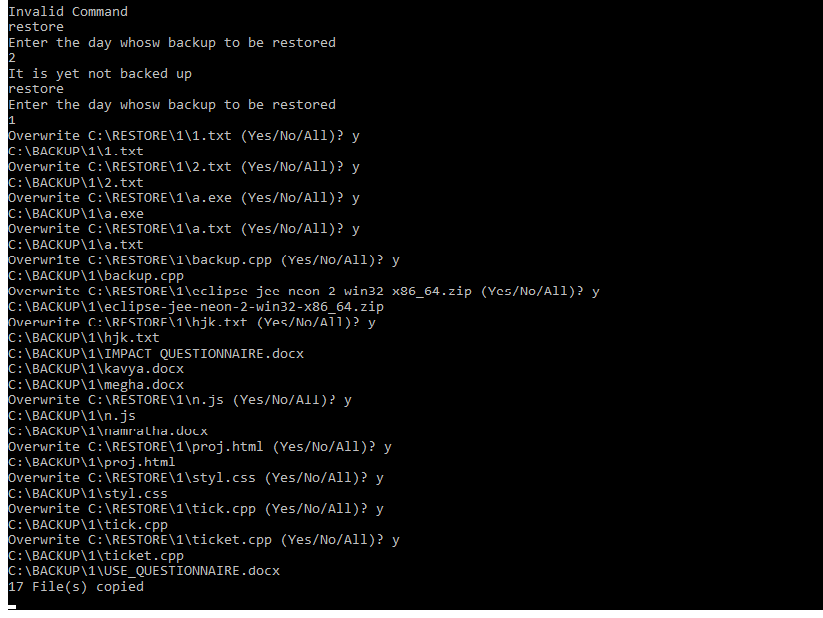


**Snapshot 5:** Restore command is enter It asks the day to restore the backup files. The day 2 is taken as input. The day 2 files are not backed up yet hence it displays error message.



**Snapshot 6:** Navigating the backup cpp file location using g++ -std=c++11backup.cpp command.

Type help to get list of commands .Enter the address of that folder that contains a file to backup in the above mentioned manner. New address is entered to backup



**Snapshot 7**: If the files are already backed up, then it asks permission to overwrite. If we give yes(y) it overwrites and 13 files are copied.



**Snapshot 8:** Exit command is entered. It displays success message that the process is done.

**Chapter 4**

**Results and Discussion**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Days of the cycle** | | | | | | | |
| **Set** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **A** |  | **A** |  | **A** |  | **A** |  |
|  | **B** |  |  |  | **B** |  |  |
|  |  |  | **C** |  |  |  |  |
|  |  |  |  |  |  |  | **D** |

The three-disc solution above is a string of 7 characters which of three different letters. You could use it as a schedule to give you a month of backups by using five tapes labelled "A" to "C".

A. One day old

B. Two days old

C. Four days old

D. Eight days old.

The tower of Hanoi is another tape rotation system that like (grandfather-father-son) was adapted from mainframe use.

Use of this system is far less prevalent, however, because although it arguably provides more comprehensive protection utilizing fewer tapes.

While the products that offer the tower of Hanoi , most notably palindrome’s network archivist, leave very little for the administrator to do in order to set up and use it, most people prefer to have a stronger grasp of the concepts involved before they rely on it for their backups.

**CONCLUSION**

**REFERENCES**

The following sources were very helpful during the completion of project:

* <https://en.wikipedia.org/wiki/Backup_rotation_scheme>
* <https://www.alvechurchdata.co.uk/hints-and-tips/softhanoi.html>