

OOPL – ASSIGNMENT 1

Title: Linux Operating System Installation & Configuration.

Objectives:

- 1) To understand how to install Ubuntu Linux Operating System.
- 2) To understand how to Configure Ubuntu Linux Operating System.

Problem Statement:

Install, Configure 64-bit Linux Operating System, study basic architecture, memory system, & basic administration.

Outcomes: Students will be able to install & Configure the Linux Operating System.

Hardware requirements:

Any CPU with Pentium Processor or similar, 2 GB RAM or more, 25 GB Hard Disk or more.

Software requirements:

64 bit Linux Operating System Bootable device

Theory :

In this assignment we will discuss how to install Ubuntu 18.04 on Laptop and desktop with Screenshots.

a. Open Source Operating System

- Definition: Open Source refers to a program or software in which the source code is available to the general public for use & that too free of charge.
- Open source operating system is the OS whose source code are freely available to install onto our systems.
- Example of OSS is: Linux, FreeBSD, Ubuntu, Open Solaris & so on.

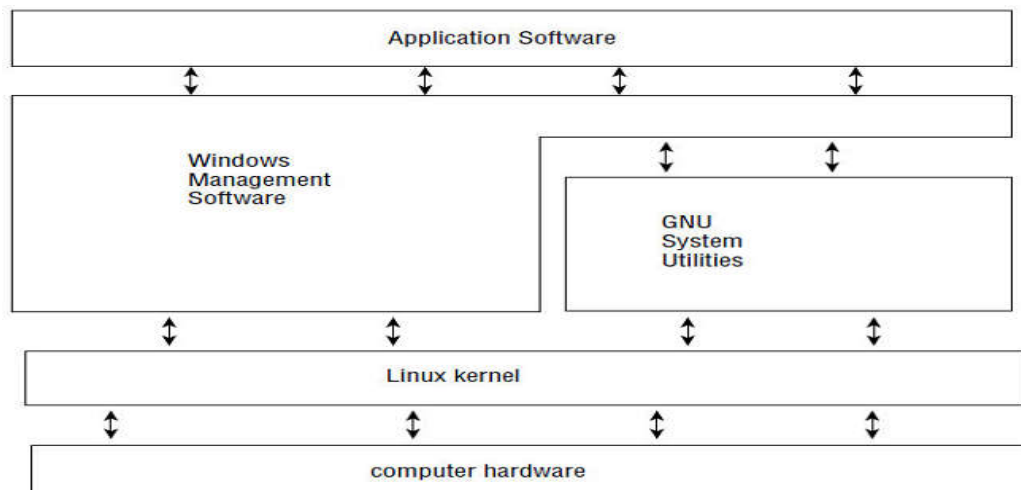
b. Linux:

Linux was first released in 1991 by its author Linus Torvalds at University of Helsinki. Since it grows tremendously in popularity as programmers around the world.

Features of Linux: Multiuser, Multiprocess, Multiplatform, Interoperable, Scalable, Portable, Flexible, Stable, Efficient, Free!

c. Linux System Architecture

4 main parts that makes up a Linux System: Linux Kernel, GNU Utilities, Graphical Desktop Environment & Application Software



d. Ubuntu 18.04 LTS – New Features:

Ubuntu 18.04 LTS has been released with a lot of new features and improvements, some of the important features are listed below:

- Support and Updates of Ubuntu 18.04 LTS for next 5 Years (April 2023)
- New latest and stable Linux Kernel version 4.15
- Installer offer a new option of “minimal Installation”
- Updated LibreOffice 6.0
- The Bionic Beaver supports attractive and beautiful color emojis
- 18.04 LTS also comes with a new Suru icon theme that will make your desktop much more colorful
- Updated GNOME (3.28) desktop environment
- XORG is the new default display server and replaces Wayland
- Fast and improved boot speed
- Along with other major improvements and bug fixes

e. Minimum System Requirement for Ubuntu 18.04 LTS (Desktop)

- 1) 2 GB RAM
- 2) Dual Core Processor (2 GH)
- 3) 25 GB free Hard disk space
- 4) Installer Media (DVD or USB)
- 5) (optional) Internet Connectivity if you are planning to download third party software and updates during the installation

Step-by-Step Guide to Install Ubuntu 18.04 LTS on your Laptop or Desktop

Step 1) Download Ubuntu 18.04 LTS ISO File

Please make sure you have the latest version of Ubuntu 18.04 LTS, If not, please download the ISO file from the link here.

<https://www.ubuntu.com/download/desktop>

Since Ubuntu 18.04 LTS only comes in a 64-bit edition, so you can install it on a system that supports 64-bit architecture.

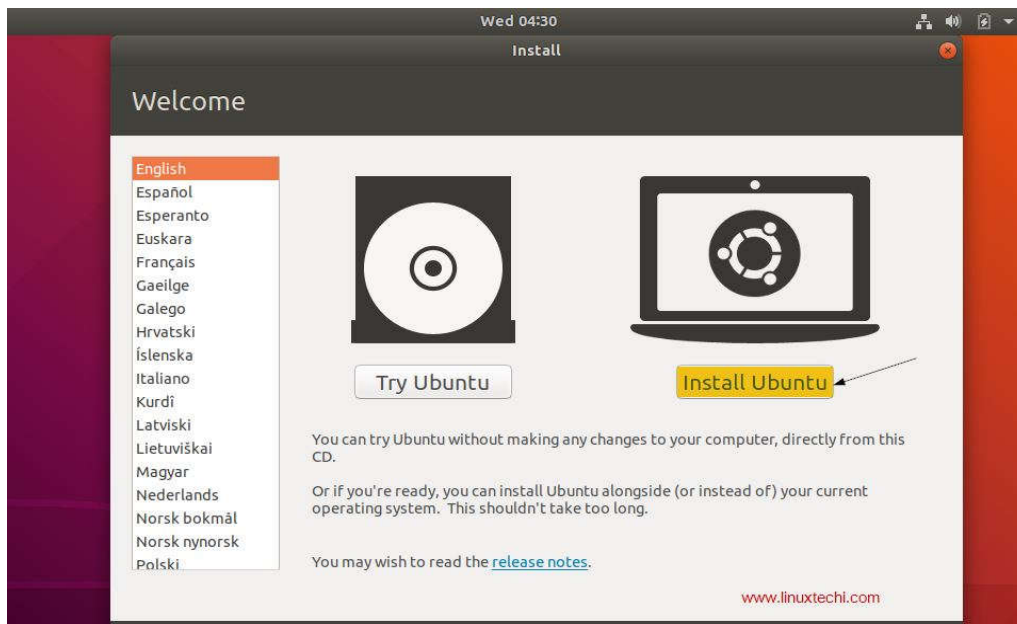
Step 2) Create a Bootable Disk

Once the ISO file is downloaded then next step is to burn the downloaded ISO image into the USB/DVD or flash drive to boot the computer from that drive.

Also make sure you change the boot sequence so that system boots using the bootable CD/DVD or flash drive.

Step 3) Boot from USB/DVD or Flash Drive

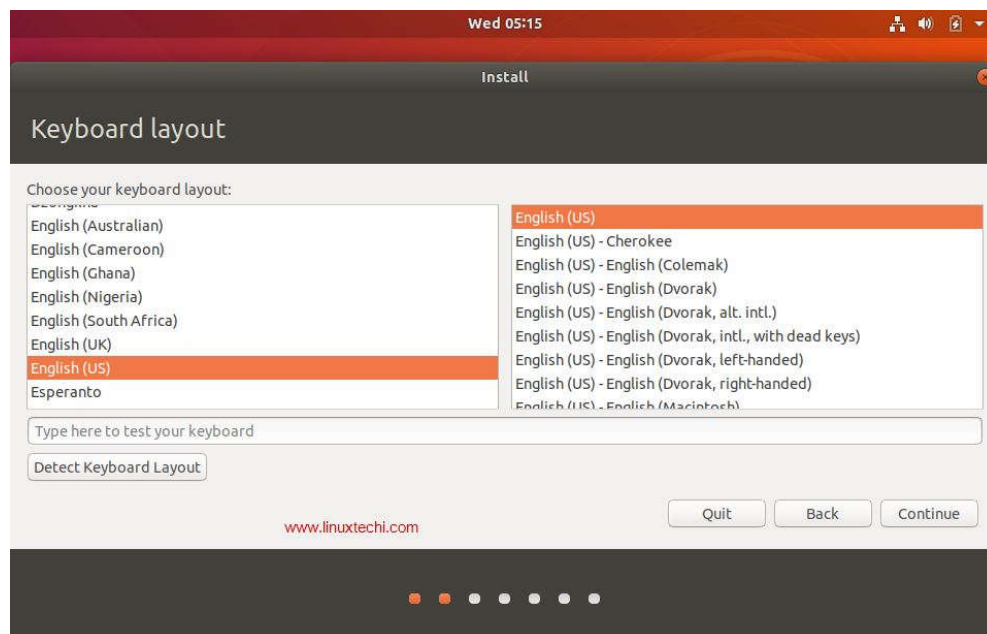
Once the system is booted using the bootable disk, you can see the following screen presented before you with options including “Try Ubuntu” and “Install Ubuntu” as shown in the image below:



Even though when you click “Try Ubuntu” you can have a sneak peek into the 18.04 LTS without installing it in your system, our goal here is to install Ubuntu 18.04 LTS in your system. So click “**Install Ubuntu**” to continue with the installation process.

Step 4) Choose your Keyboard layout

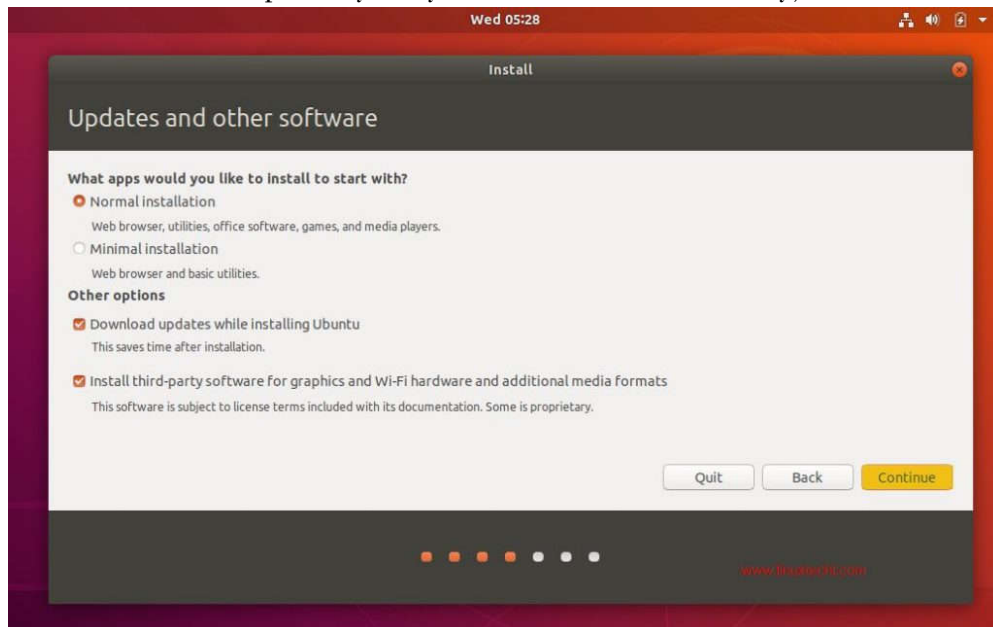
Choose your favorite keyboard layout and click “Continue”. By default, English (US) keyboard is selected and if you want to change, you can change here and click “Continue”,



Step 5) Preparing to Install Ubuntu and other Software

In the next screen, you’ll be provided following beneath options including:

- **Type of Installation:** Normal Installation or Minimal installation, If you want a minimal installation then select second option otherwise go for the Normal Installation. In my case I am doing Normal Installation
- **Download Updates While Installing Ubuntu** (select this option if your system has internet connectivity during installation)
- **Install third party software for graphics and Wi-Fi hardware, MP3 and additional media formats** Select this option if your system has internet connectivity)



click on “Continue” to proceed with installation

Step 6) Select the appropriate Installation Type

Next the installer presents you with the following installation options including:

- Erase Disk and Install Ubuntu
- Encrypt the new Ubuntu installation for security
- Use LVM with the new Ubuntu installation
- Something Else

Where,

Erase Disk and Install Ubuntu – Choose this option if your system is going to have only Ubuntu and erasing anything other than that is not a problem. This ensures a fresh copy of Ubuntu 18.04 LTS is installed in your system.

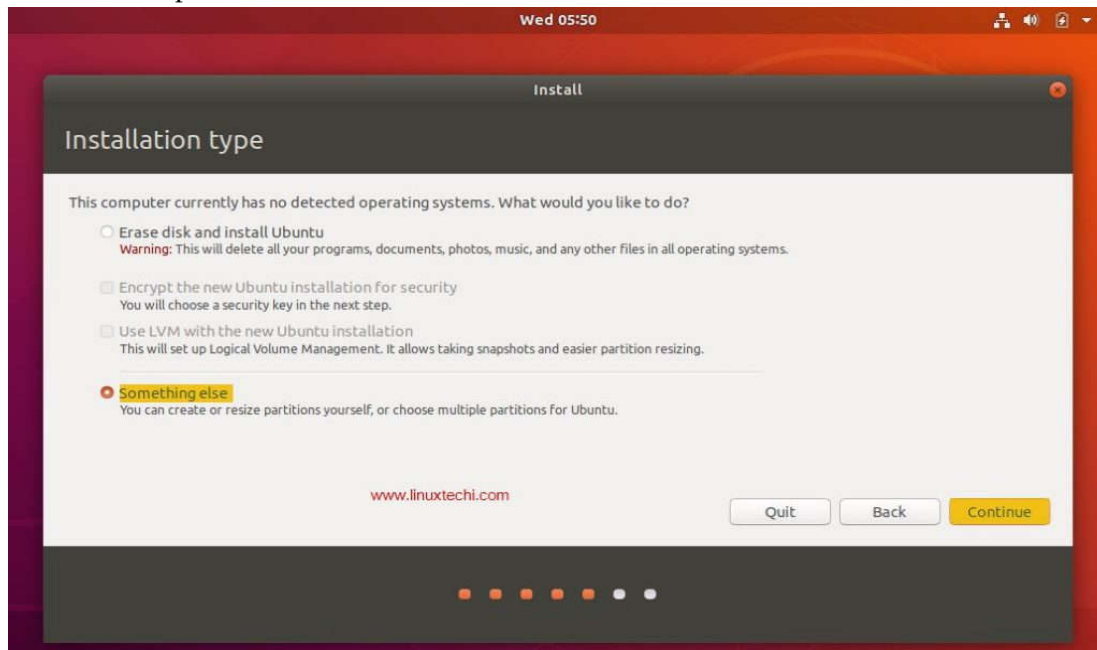
Encrypt the new Ubuntu installation for security – Choose this option if you are looking for extended security for your disks as your disks will be completely encrypted. If you are beginner, then it is better not to worry about this option.

Use LVM with the new Ubuntu installation – Choose this option if you want to use LVM based file systems.

Something Else – Choose this option if you are advanced user and you want to manually create your own partitions and want to install Ubuntu along with existing OS (May be Windows or other Linux Flavor)

In this assignment, we will be creating our custom partitions on a hard disk of 40 GB and the following partitions are to be created:

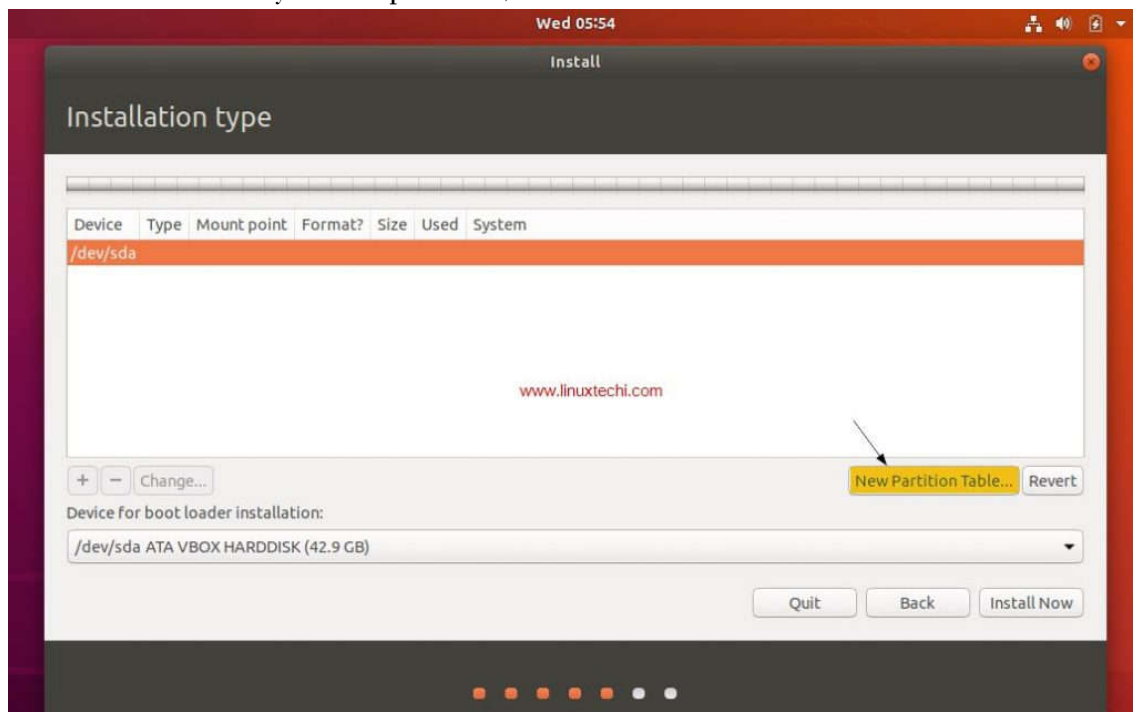
- /boot 1 GB (ext4 files system)
- /home 18 GB (ext4 file system)
- / 12 GB (ext4 file system)
- /var 6 GB (ext4 file system)
- Swap 2 GB

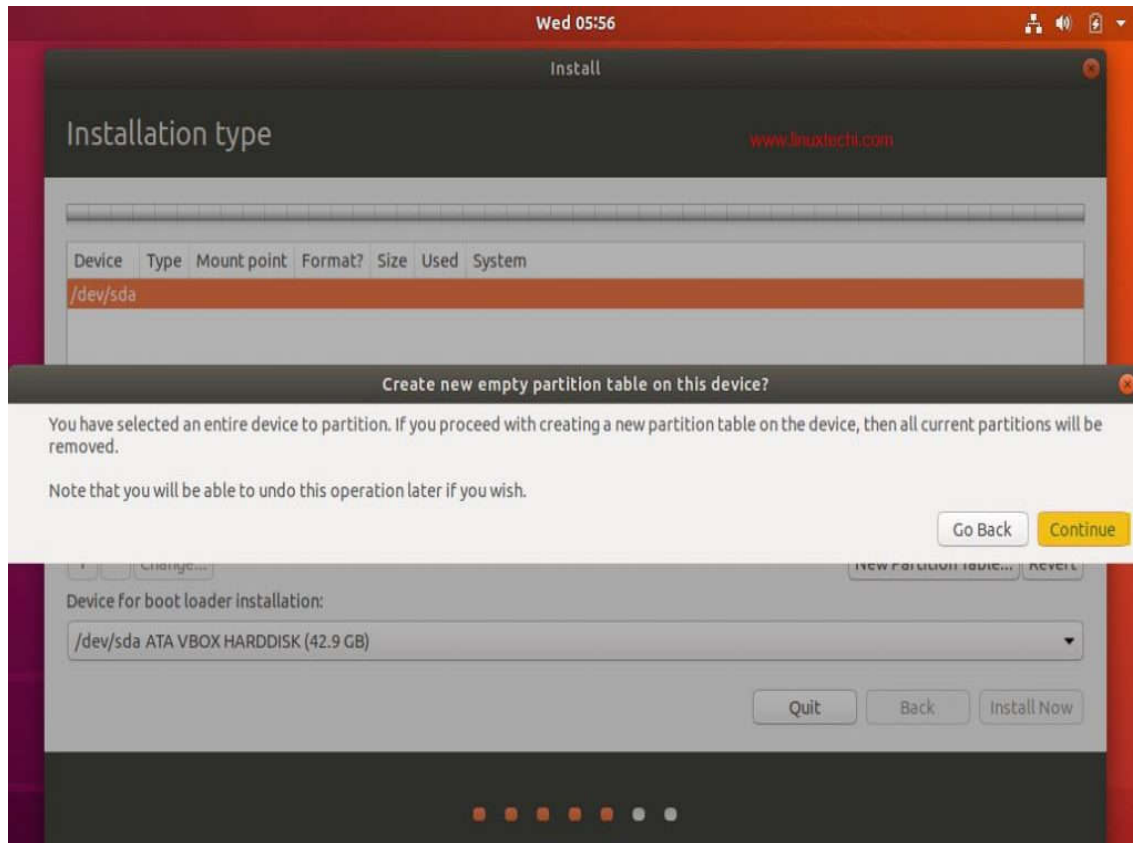


Now, Choose “**Something Else**” and Click on continue

You can see the available disk size for Ubuntu in the next window as shown below:

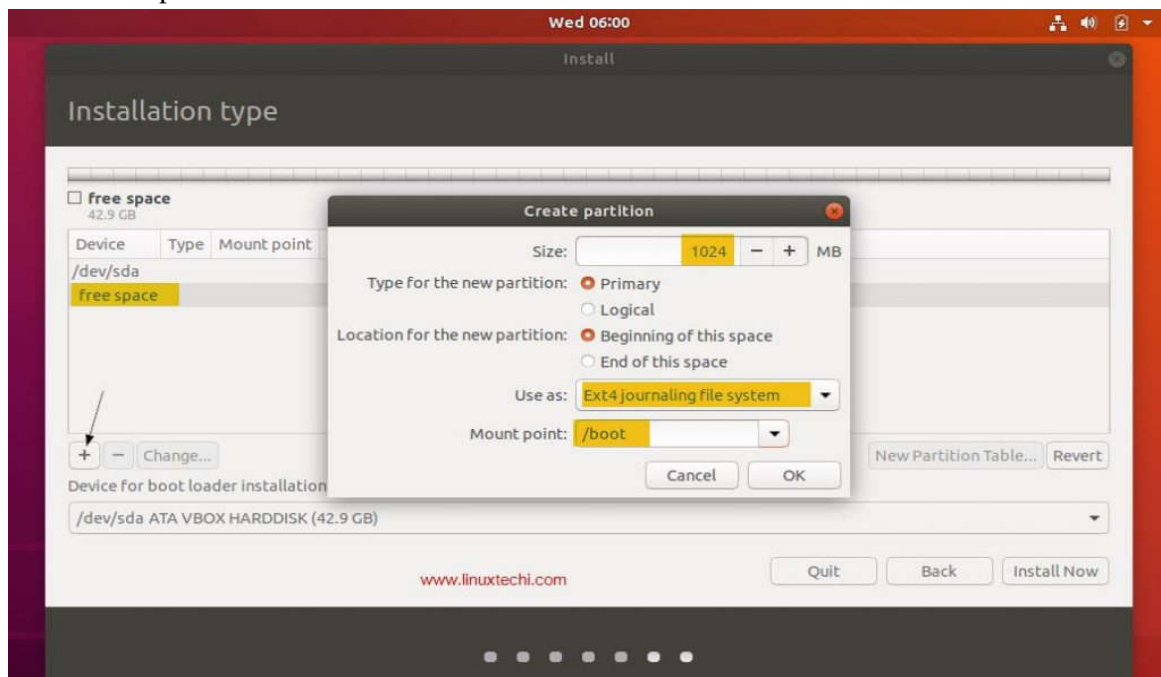
Now in order to create your own partitions, click on “**New Partition Table**”





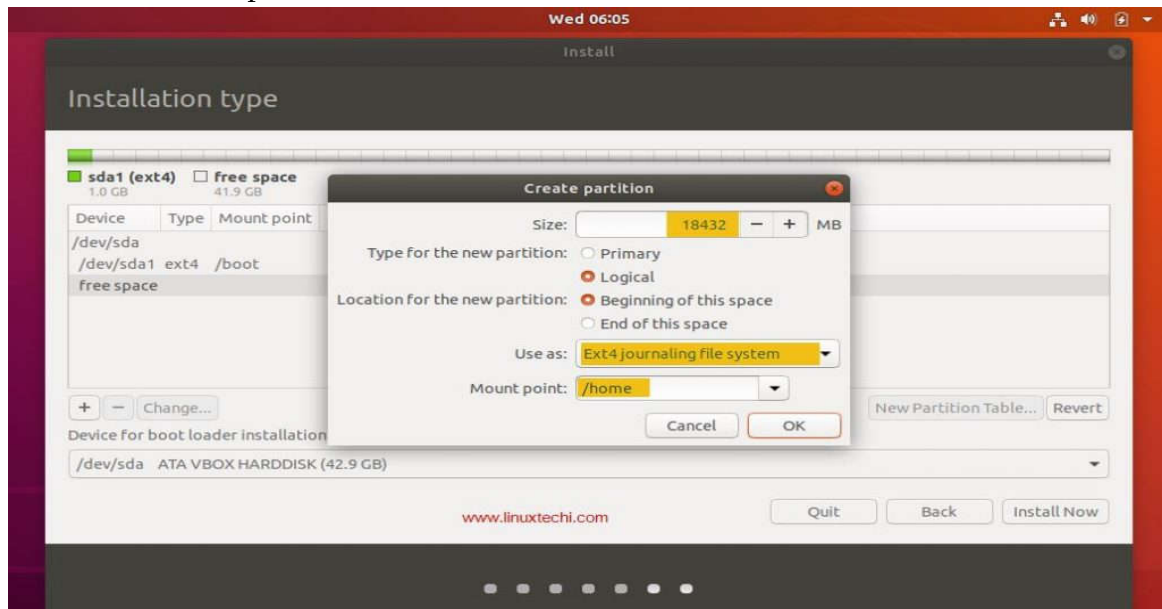
Click on Continue

Create /boot partition of size 1GB, Select the free space and then Click on the “+” symbol to create a new partition



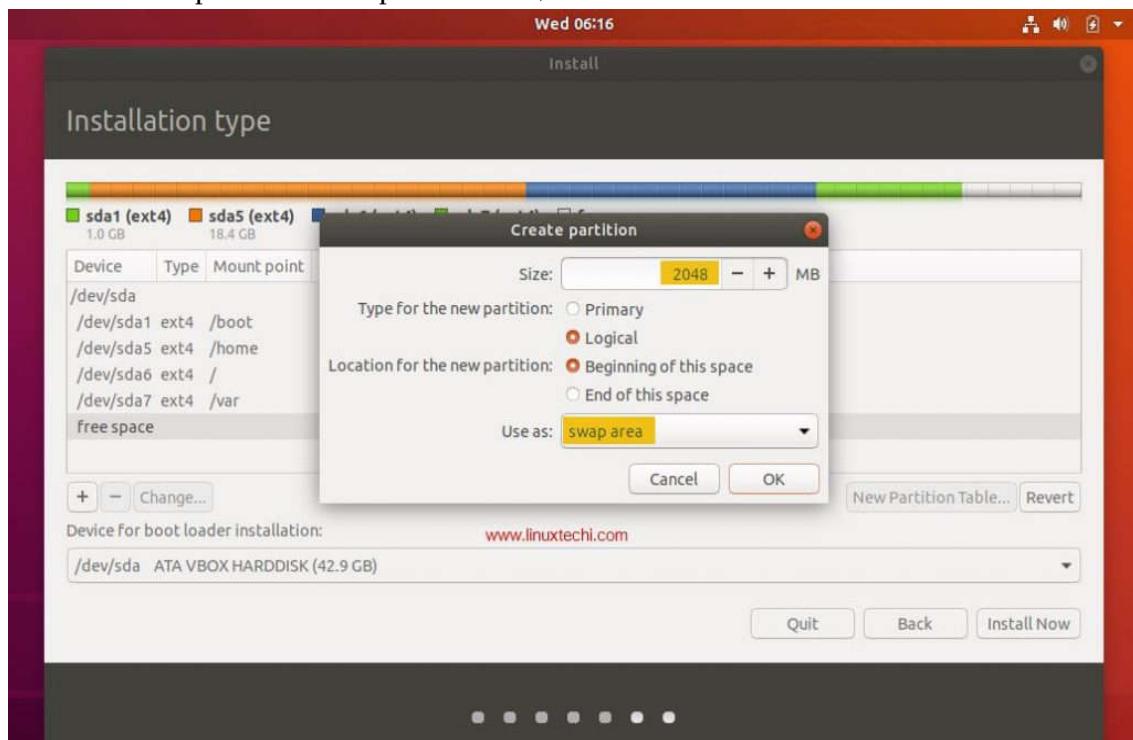
Click on “OK”

Let's create /home partition of size 18 GB,



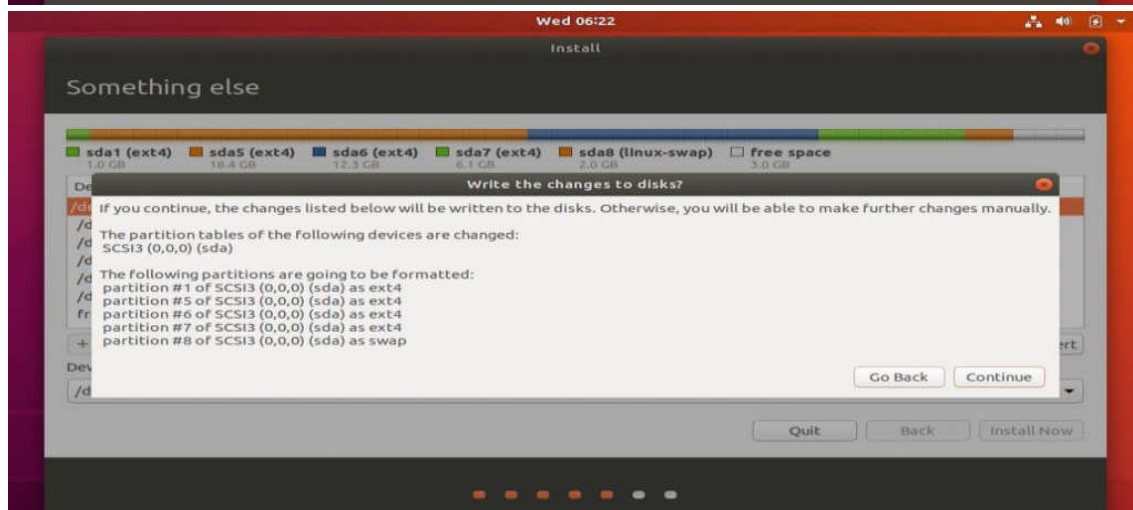
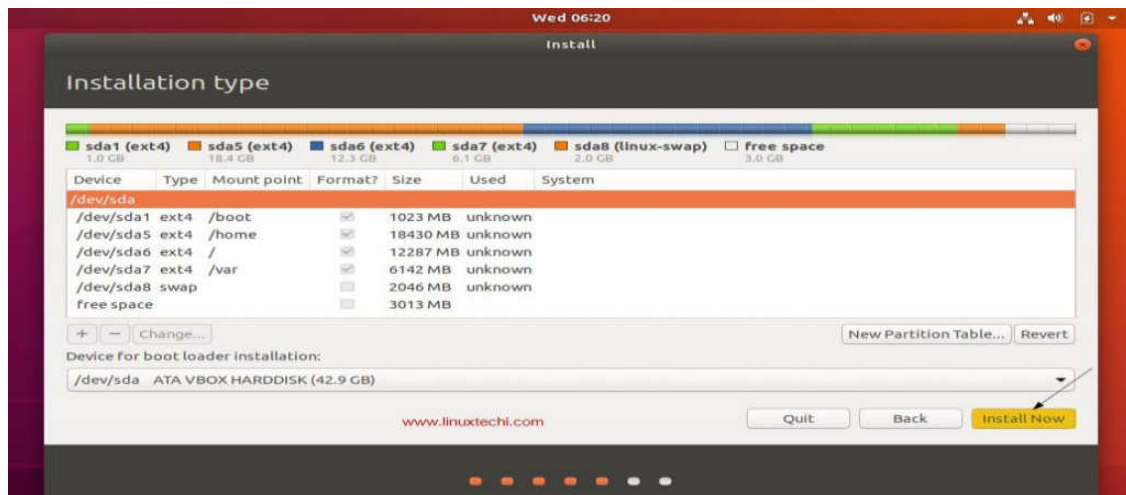
In the same way create / & /var file system of size 12 GB & 6 GB respectively

Now create last partition as swap of size 2 GB,



Click on OK

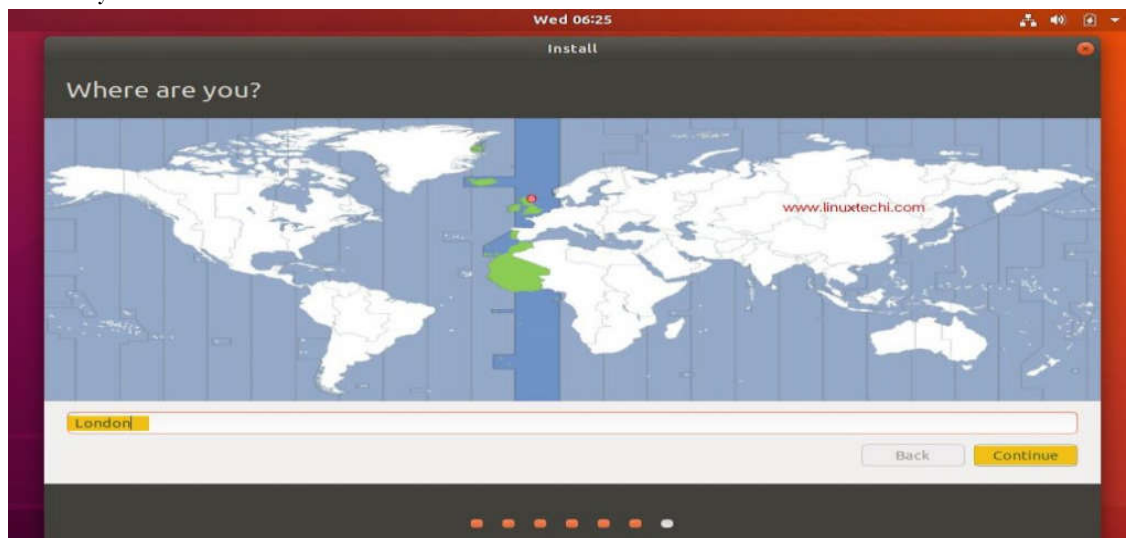
Once you are done with the partition creation task, then click on “**Install Now**” option to proceed with the installation



Now click on “Continue” to write all the changes to the disks

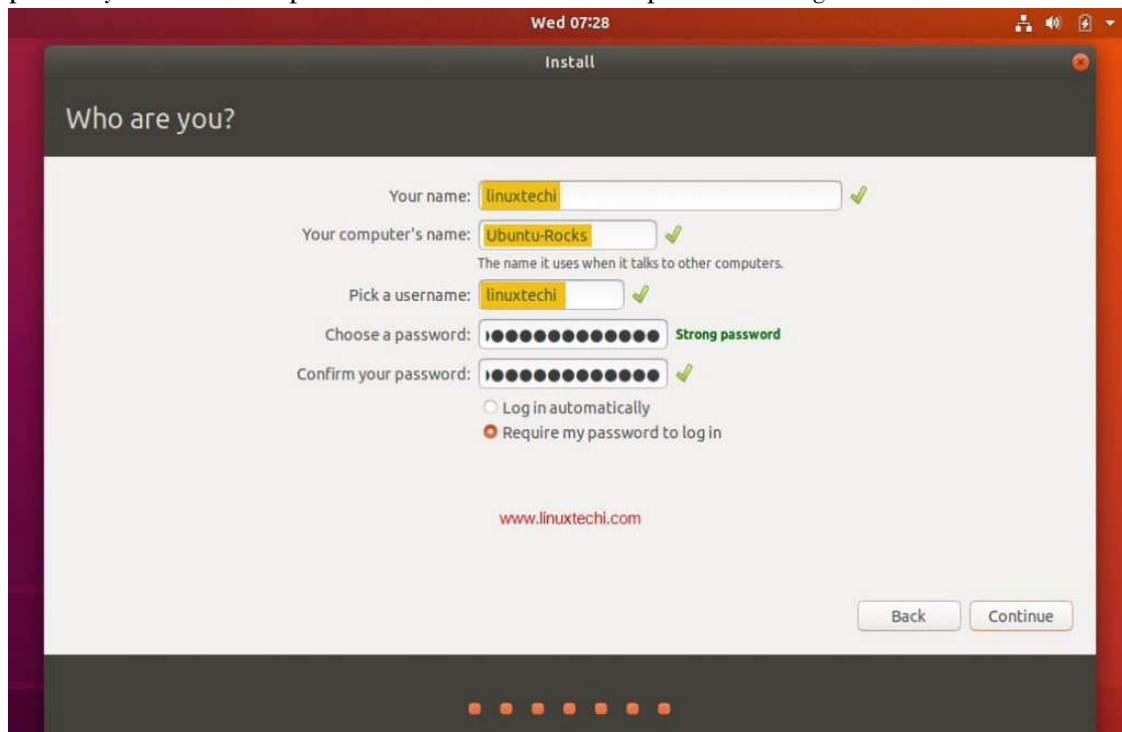
Step 7) Select Your Time zone

Choose your favorite time zone and then click on “Continue”



Step 8) Provide your User Credentials

In the next screen you will be prompted to provide your user credentials. In this screen provide your name, computer name, username and the password to login into Ubuntu 18.04 LTS



The screenshot shows the 'Who are you?' screen in the Ubuntu installer. The window title is 'Install'. The background is a dark red with a faint Ubuntu logo. The form fields are as follows:

- Your name: ✓
- Your computer's name: ✓
The name it uses when it talks to other computers.
- Pick a username: ✓
- Choose a password: Strong password
- Confirm your password: ✓

Below the password fields are two radio buttons:

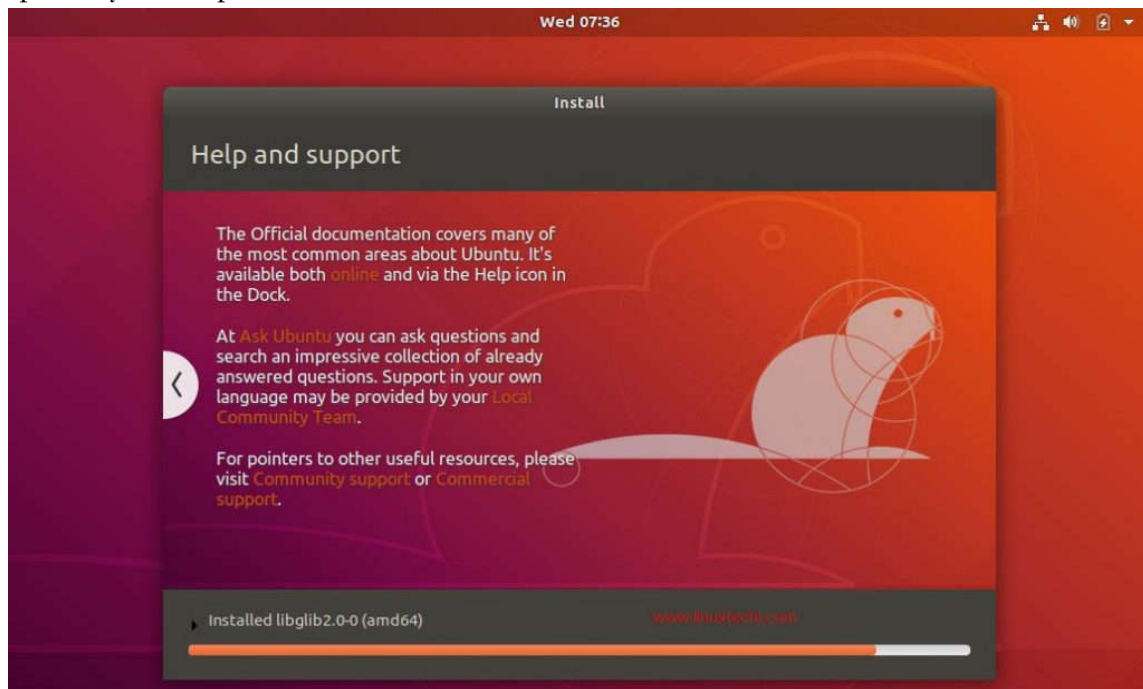
- ☐ Log in automatically
- ☒ Require my password to log in

At the bottom, there is a red URL www.linuxtech1.com and two buttons: 'Back' and 'Continue'. A progress bar at the very bottom shows six steps, with the current step highlighted.

Click “Continue” to begin the installation process.

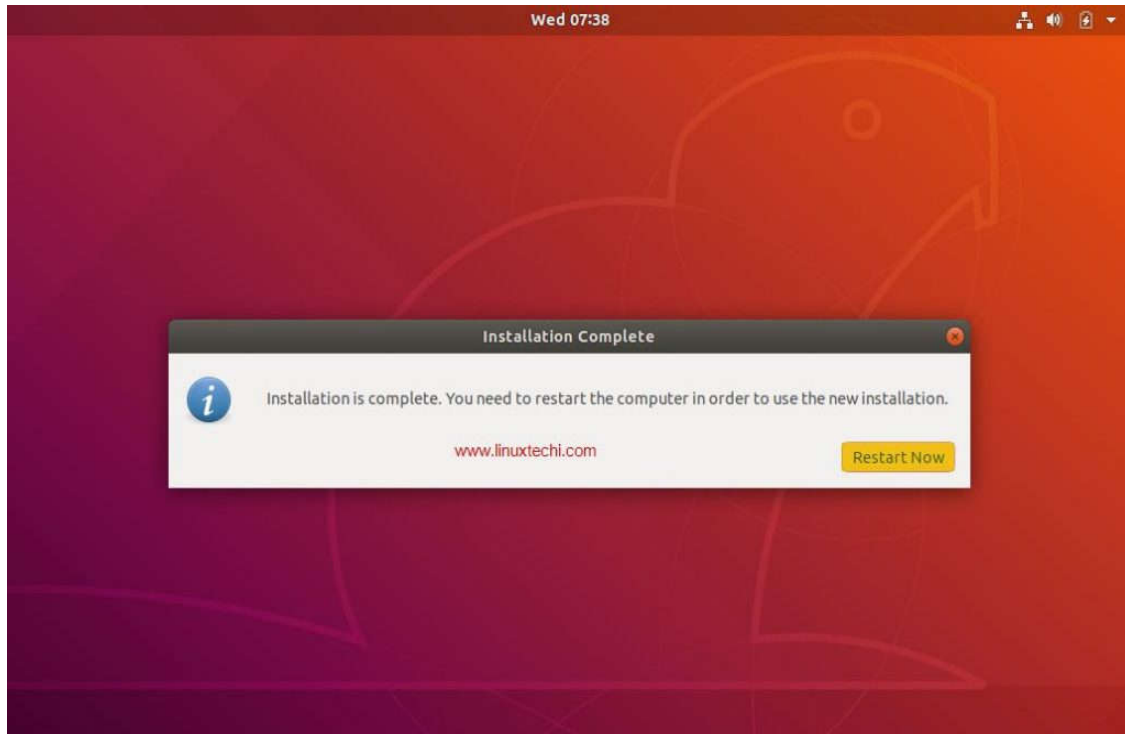
Step 9) Start Installing Ubuntu 18.04 LTS

The installation of Ubuntu 18.04 LTS starts now and will take around 5-10 mins depending on the speed of your computer,



Step 10) Restart Your System

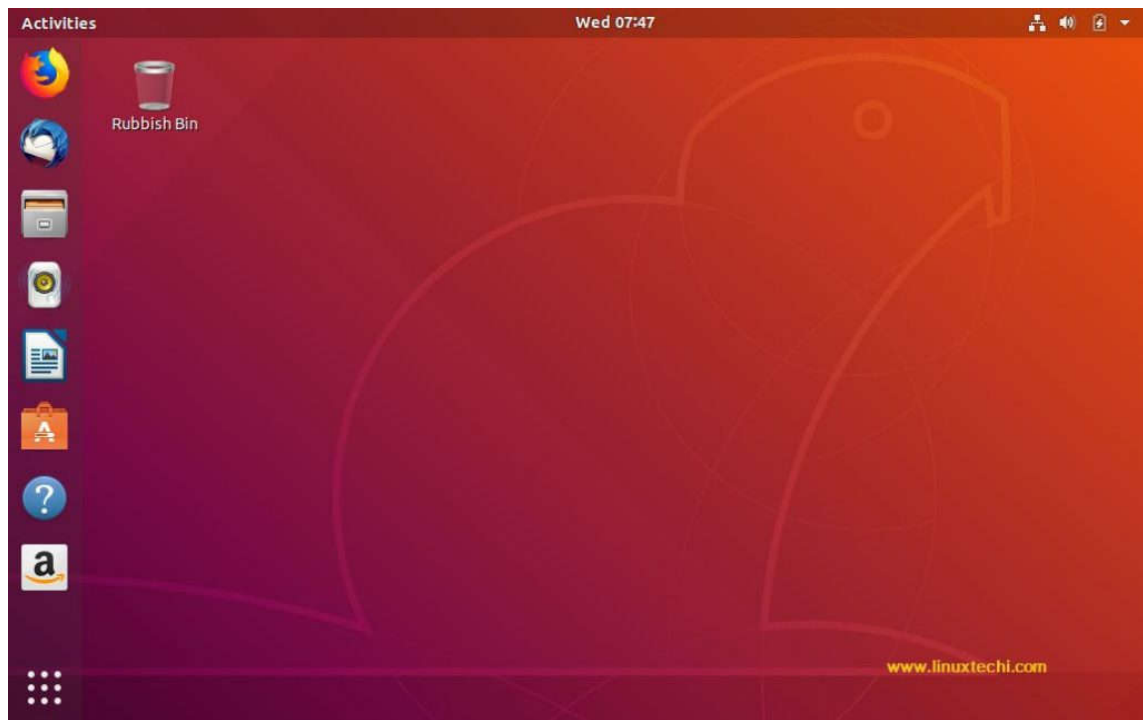
Once the installation is completed, remove the USB/DVD from the drive and Click “Restart Now” to restart your system.



Step:11) Login to Your Ubuntu 18.04 desktop

Once your system has been rebooted after the installation then you will get the beneath login screen, enter the User name and password that you have set during installation (Step 8)





Conclusion

And that concludes our step by step installation guide for Ubuntu 18.04 LTS and it's all up to you now to explore the exciting features of Ubuntu 18.04 LTS and have fun

OOPL – ASSIGNMENT 2

Title: Develop a calculator for arithmetic operators like +,-,*,/,% (use switch statement).

Objectives: To understand use of switch statement for designing calculator program

Problem Statement:

Write a C++ program to create a calculator for an arithmetic operator (+, -, *, /). The program should take two operands from user and performs the operation on those two operands depending upon the operator entered by user. Use a switch statement to select the operation. Finally, display the result.

Outcomes: Students will be able to demonstrate use of switch statement in C++.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64 bit Linux/Windows Operating System, G++ compiler

Theory :

Major factor in the invention of Object-Oriented approach is to remove some of the flaws encountered with the procedural approach. In OOP, data is treated as a critical element and does not allow it to flow freely. It bounds data closely to the functions that operate on it and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects. A major advantage of OOP is code reusability.

Concepts of OOP:

1. Objects
2. Classes
3. Data Abstraction and Encapsulation
4. Inheritance
5. Polymorphism

Objects:

Objects are the basic run-time entities in an object-oriented system. Programming problem is analyzed in terms of objects and nature of communication between them. When a program is executed, objects interact with each other by sending messages. Different objects can also interact with each other without knowing the details of their data or code.

Classes:

A class is a collection of objects of similar type. It is a user defined data type. An object is called variable of class type. Once a class is defined, any number of objects can be created which belong to that class. For example:

```
class student
{
private: // attributes:
char name, DOB;
public: // methods:
total ();
avg();
display();
}
```

Advantages of OOP:

Object-Oriented Programming has the following advantages over conventional approaches:

- 1) OOP provides a clear modular structure for programs which makes it good for defining abstract data types where implementation details are hidden and the unit has a clearly defined interface.
- 2) OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
- 3) OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer. This is particularly useful for developing graphical user interfaces.

Operators:

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provides the following types of operators:

- 1) Arithmetic Operators
- 2) Relational Operators
- 3) Logical Operators
- 4) Bitwise Operators
- 5) Assignment Operators

Arithmetic Operators: There are following arithmetic operators supported by C++ language:

Assume variable A holds 10 and variable B holds 20, then:

Operator Description Example

- 1) + Adds two operands A + B will give 30
- 2) - Subtracts second operand from the first A – B will give -10
- 3) * Multiplies both operands A * B will give 200
- 4) / Divides numerator by de-numerator B / A will give 2
- 5) % Modulus Operator and remainder of after an integer division B % A will give 0
- 6) ++ Increment operator, increases integer value by one A++ will give 11
- 7) -- Decrement operator, decreases integer value by one A– will give 9

Switch Case statement:

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax: The syntax for a switch statement in C++ is as follows:

```
switch(expression){  
    case constant-expression :  
        statement(s);  
        break; //optional  
    case constant-expression :  
        statement(s);  
        break; //optional  
    // you can have any number of case statements.  
    default : //Optional  
        statement(s);  
}
```

The following rules apply to a switch statement:

- 1) The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- 2) You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- 3) The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- 4) When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- 5) When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- 6) Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- 7) A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Algorithm:

Flowchart:

Test Cases:

Test Case 1:

Enter operator : +

Enter two numbers :10 0

Num1=10 Num2=0 Result=10

Do you want to continue : Y

Test Case 2:

Enter operator : /

Enter two numbers :10 0

Num1=10 Num2=0 Result=inf

Do you want to continue : Y

Test Case 3:

Enter operator : -

Enter two numbers :10 -20

Num1=10 Num2= -20 Result= 30

Do you want to continue : Y

Test Case 4:

Enter operator : *

Enter two numbers :2500 2500

Num1=2500 Num2=2500 Result= 6.25e+06

Do you want to continue : Y

Test Case 5:

Enter operator : a

Enter two numbers :10 0

Num1=10 Num2=0 Result=inf

Do you want to continue : Y

Conclusion: Hence, we are able to use arithmetic operators with switch statement.

Program

```
#include<iostream.h>
#include<conio.h>
//using namespace std;
class calculator
{
int num1,num2;
float result;
public:
void getData();
void putData();
void add();
void sub();
void mul();
void div();
void mod();
};
void calculator::getData()
{
cout<<"Enter two numbers";
cin>>num1>>num2;
}
void calculator::putData()
{
cout<<"Num1 = "<<num1<<"\tNum2 = "<<num2<<"\tResult = "<
}
void calculator::add()
{
result=num1+num2;
}
void calculator::sub()
{
result=num1-num2;
}
void calculator::mul()
{
result=num1*num2;
}
void calculator::mod()
{
result=num1%num2;
}
void calculator::div()
{
result=num1/float(num2);
}
```



```

int main()
{
    calculator c;
    char choice,ans;
    do
    {
        clrscr();
        cout<<"\n*****MENU*****\n";
        cout<<"1. +\n2. - \n3. * \n4. /\n5. %\nEnter your operator\n";
        cin>>choice;
        c.getData();
        switch(choice)
        {
            case '+':
                c.add();
                break;
            case '-':
                c.sub();
                break;
            case '*':
                c.mul();
                break;
            case '/':
                c.div();
                break;
            case '%':
                c.mod();
                break;
        }
        c.putData();
        cout<<"\nDo you want to continue(Y/N)";
        cin>>ans;
    }while(ans=='Y' || ans=='y');
    return 0;
}

```

OOPL – ASSIGNMENT 3 (Batch S4)

Title: Demonstrate use of operator overloading for Complex class.

Objectives:

- 1) To understand concept of operator overloading.
- 2) To demonstrate overloading of binary operator, insertion and extraction operator.

Problem Statement:

Implement a class Complex which represents the Complex Number data type. Implement the following operations:

1. Constructor (including a default constructor which creates the complex number 0+0i).
2. Overloaded operator+ to add two complex numbers.
3. Overloaded operator* to multiply two complex numbers.
4. Overloaded << and >> to print and read Complex Numbers.

Outcomes:

- 1) Students will be able to demonstrate use of constructor
- 2) Students will be able to demonstrate binary operator overloading
- 3) Students will be able to demonstrate overloading of insertion and extraction operator using friend function.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements: 64 bit Linux/Windows Operating System, G++ compiler

Theory:

C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

When you call an overloaded function or operator, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution.

Operator overloading is one of the special feature of C++. It also shows the extensibility of C++. C++ permits us to add two variables of user defined types with the same way that is applied with built in type data type. This refers to ability to provide special meaning for existing data type. This mechanism of giving such special meaning to an operator is known as Operator overloading. Operator overloading provides a flexible option for creation of new definition for most of the C++ operators. We can assign additional meaning to all existing C++ operators except following:

- 1) Scope resolution operator (::)
- 2) Size of operator (sizeof)
- 3) Conditional operator (?:)
- 4) Class member access operators (. , .*)

Definition Operator Overloading: To define an additional task to an operator, we must specify what it means in relation to class to which the operator is applied. This is done with the help of a special function, called operator function which describes the task.

```
return type class_name :: operator op( argument list){  
    // Body of the function  
    // The task defined by overloaded operator  
}  
where:  
return type is type of value returned by specified operation.  
op is operator being overloaded.  
operator is a keyword in C++
```

Operator function is no static member function or it may be friend function. A basic difference between them is that friend function will have only one argument for unary and binary operator whereas member function has no argument for unary operators and only one for binary operators. This is because the object used to invoke the member function is passé implicitly and therefore is available for member function. This is not the case with friend functions. Arguments may be passed either by value or by reference.

For defining an additional task to an operator, we must mention what it means in relation to the class to which it (the operator) is applied. The operator function helps us in doing so. The Syntax of declaration of an Operator function is as follows:

```
operator operator_name
```

For example, suppose that we want to declare an Operator function for '='. We can do it as follows:

```
operator =
```

You can overload the assignment operator (=) just as you can other operators and it can be used to create an object just like the copy constructor.

Unary operators overloading in C++: The unary operators operate on a single operand and following are the examples of Unary operators:

- 1) The increment (++) and decrement (--) operators.
- 2) The unary minus (-) operator.
- 3) The logical not (!) operator.

The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--.

Binary operator overloading: In overloading binary operator, a friend function will have two arguments, while a member function will have one argument. Overloading binary operators using friends: Friend functions may be used in the place of member functions for overloading a binary operator. The only difference being that a friend function requires two arguments to be explicitly passed to it while a member function requires only one.

The same complex number program with friend function can be developed as friend complex operator +(complex, complex); and we will define this function as

```
complex operator + (complex a, complex b) {  
    return complex ( c.x + b.x), (a.y + b.y) ;  
}
```

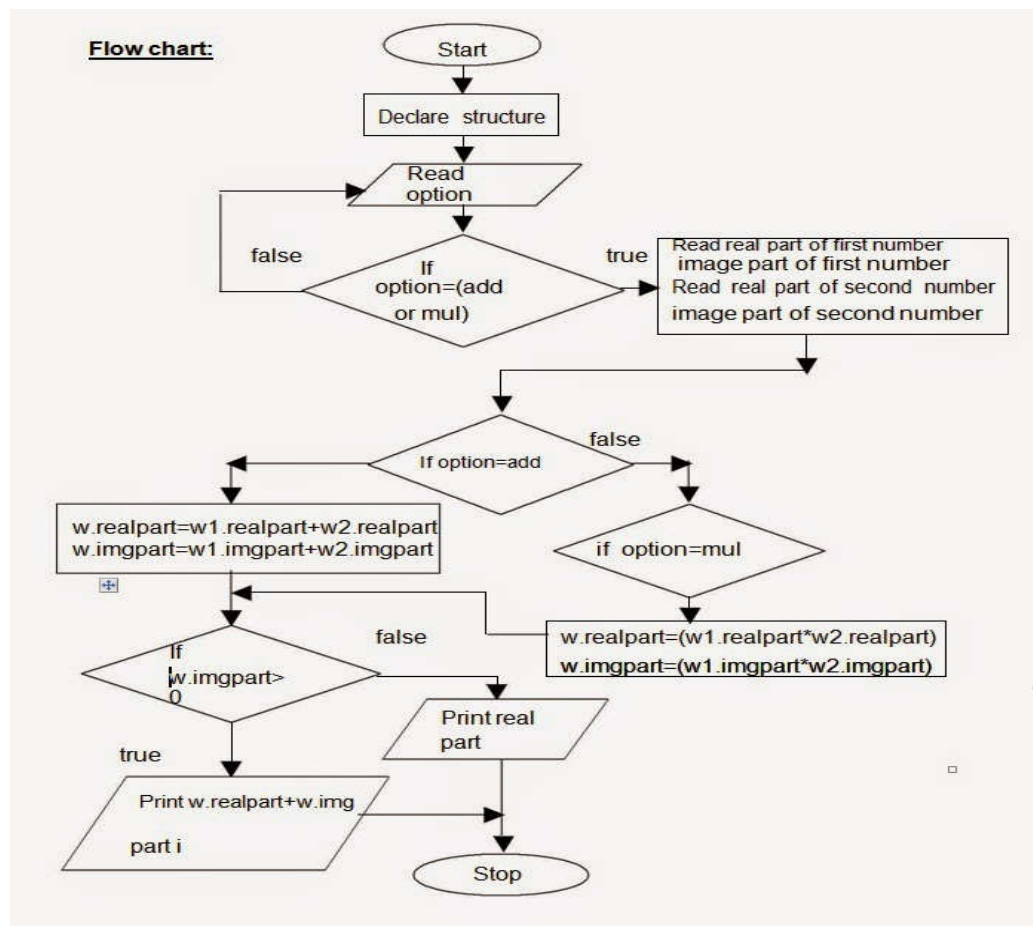
in this case, the statement

c3 = c1 + c2; is equal to c3 = operator + (c1, c2)

Rules for overloading operators:

- 1) Only existing operators can be overloaded. New operators cannot be overloaded.
- 2) The overloaded operator must have at least one operand that is of user defined type.
- 3) We cannot change the basic meaning of an operator. That is to say, We cannot redefine the plus(+) operator to subtract one value from the other.
- 4) Overloaded operators follow the syntax rules of the original operators. They cannot be overridden.
- 5) There are some operators that cannot be overloaded like size of operator(sizeof), membership operator(.), pointer to member operator(*), scope resolution operator(::), conditional operators(?:) etc
- 6) We cannot use “friend” functions to overload certain operators. However, member function can be used to overload them. Friend Functions can not be used with assignment operator(=), function call operator(()), subscripting operator([]), class member access operator(->) etc.
- 7) Unary operators, overloaded by means of a member function, take no explicit arguments and return no explicit values, but, those overloaded by means of a friend function, take one reference argument (the object of the relevant class).
- 8) Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.
- 9) When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.
- 10) Binary arithmetic operators such as +, -, * and / must explicitly return a value. They must not attempt to change their own arguments.

Flowchart:



Conclusion: Hence, we have studied, used and demonstrated use of binary operator overloading and insertion-extraction operator overloading using friend function.

PROGRAM

//Program to demonstrate various operations on Complex class

```
#include<iostream>
```

```
using namespace std;
```

```
class Complex
```

```
{
```

```
    float real,img;
```

```
public:
```

```
//Constructor
```

```
Complex()
```

```
{
```

```
    real=0;
```

```
    img=0;
```

```
}
```

```
Complex(float a,float b)
```

```
{
```

```
    real=a;
```

```
    img=b;
```

```
}
```

```
//Addition of two complex numbers
```

```
Complex operator+(Complex c1)
```

```
{
```

```
    Complex temp;
```

```
    temp.real=real+c1.real;
```

```
    temp.img=img+c1.img;
```

```
    return temp;
```

```
}
```

```
//Subtraction of two complex numbers
```

```
Complex operator-(Complex c1)
```

```
{
```

```
    Complex temp;
```

```
    temp.real=real-c1.real;
```

```
    temp.img=img-c1.img;
```

```
    return temp;
```

```
}
```

```
//Multiplication of two complex numbers
```

```
Complex operator*(Complex c1)
```

```
{
```

```
    Complex temp;
```

```
    temp.real=(real*c1.real)-(img*c1.img);
```

```
    temp.img=(img*c1.real)+(real*c1.img);
```

```
    return temp;
```

```
}
```

```
//Division of two complex numbers
```

```
Complex operator/(Complex c1)
```

```
{
```

```
    Complex temp,c2;
```

```
    c2.img=-c1.img;
```

```
    float x;
```

```

    temp.real=(real*c1.real)-(img*(c2.img));
    temp.img=(real*c1.real)+(real*(c2.img));
    x=(c1.real)*(c1.real)+(c1.img)*(c1.img);
    temp.real=temp.real/x;
    temp.img=temp.img/x;
    return temp;
}
//overloaded insertion (<<) opertor for class Complex
friend ostream &operator<<(ostream &out, Complex &c)
{
    out << c.real<<" + "<< c.img<<" i";
    return out;
}
//overloaded extraction (>>) opertor for class Complex
friend istream &operator>>(istream &in, Complex &c)
{
    in>> c.real>>c.img;
    return in;
}
};
int main()
{
    Complex c1,c2,c3;
    int choice;
    char ans;
    do
    {
        cout<<"\n***** MENU *****\n";
        cout<<"\n\t1.Addition\n\t2.Subtraction\n\t3.Multiplication\n\t4.Division";
        cout<<"\n\nEnter your choice: ";
        cin>>choice;
        cout<< "Enter real and img part of first complex number\n";
        cin>>c1;
        cout<< "Enter real and img part of second complex number\n";
        cin>>c2;
        switch(choice)
        {
            case 1:
                c3=c1+c2;
                cout<<"\n\nAddition is: ";
                cout<<c3;
                break;
            case 2:
                c3=c1-c2;
                cout<<"\n\nSubtraction is: ";
                cout<<c3;
                break;
            case 3:
                c3=c1*c2;
                cout<<"\n\nMultiplication is: ";
                cout<<c3;
                break;

```

```
    case 4:
        c3=c1/c2;
        cout<<"\n\nDivision is: ";
        cout<<c3;
        break;
    default:
        cout<<"\nWrong choice";
    }
    cout<<"\nDo you want to continue?(y/n): ";
    cin>>ans;
    } while(ans=='y' | | ans=='Y');
    return 0;
}
```


OOPL – ASSIGNMENT 3 (Batch S5)

Title:

Demonstrate use of operator overloading in object oriented programming for Quadratic equation that represents degree two polynomials.

Objectives:

- 1) To understand concept of operator overloading.
- 2) To demonstrate overloading of binary operator, insertion and extraction operator.

Problem Statement:

Implement a class Quadratic that represents degree two polynomials i.e., polynomials of type ax^2+bx+c . The class will require three data members corresponding to a, b and c. Implement the following operations:

1. A constructor (including a default constructor which creates the 0 polynomial).
2. Overloaded **operator+** to add two polynomials of degree 2.
3. Overloaded **<<** and **>>** to print and read polynomials. To do this, you will need to decide what you want your input and output format to look like.
4. A function eval that computes the value of a polynomial for a given value of x.
5. A function that computes the two solutions of the equation $ax^2+bx+c=0$.

Outcomes:

- 1) Students will be able to demonstrate use of constructor
- 2) Students will be able to demonstrate binary operator overloading
- 3) Students will be able to demonstrate overloading of insertion and extraction operator using friend function.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64 bit Linux/Windows Operating System, G++ compiler

Theory:**a. Quadratic Equation:**

- In algebra, a quadratic equation (from the Latin quadratus for "square") is any equation having the form

$$ax^2 + bx + c = 0$$

- where x represents an unknown, and a, b, and c represent known numbers such that a is not equal to 0. If $a = 0$, then the equation is linear, not quadratic.
- The numbers a, b, and c are the coefficients of the equation, and may be distinguished by calling them, respectively, the quadratic coefficient, the linear coefficient and the constant or free term.
- The standard form of a quadratic equation is:

$$ax^2 + bx + c = 0,$$

where a, b and c are real numbers and $a \neq 0$

- The term b^2-4ac is known as the determinant of a quadratic equation. The determinant tells the nature of the roots.
 - If determinant is greater than 0, the roots are real and different.
 - If determinant is equal to 0, the roots are real and equal.
 - If determinant is less than 0, the roots are complex and different.

If determinant > 0 ,

$$\text{root1} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$\text{root2} = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

If determinant $= 0$,

$$\text{root1} = \text{root2} = \frac{-b}{2a}$$

If determinant < 0 ,

$$\text{root1} = \frac{-b}{2a} + i \frac{\sqrt{-(b^2 - 4ac)}}{2a}$$

$$\text{root2} = \frac{-b}{2a} - i \frac{\sqrt{-(b^2 - 4ac)}}{2a}$$

- **Worked example**

- First, we need to identify the values for a, b, and c (the coefficients).
- First step, make sure the equation is in the format from above, $ax^2+bx+c=0$
- $x^2+4x-21=0$
 - a is the coefficient in front of x^2 , so here $a=1$ (note that a can't equal 0 -- the x^2 is what makes it a quadratic).
 - b is the coefficient in front of the x, so here $b=4$.
 - c is the constant, or the term without any x, so here $c=-21$
- Then we plug a, b, and c into the formula:

$$x = \frac{-4 \pm \sqrt{16 - 4 \cdot 1 \cdot (-21)}}{2}$$

solving this looks like:

$$x = \frac{-4 \pm \sqrt{100}}{2}$$

$$= \frac{-4 \pm 10}{2}$$

$$= -2 \pm 5$$

Therefore $x = 3$ or $x = -7$.

b. The sqrt () function:

- The sqrt () function computes the square root of a number.
- Function prototype of sqrt ()

$$\text{double sqrt (double arg);}$$
- The function sqrt () takes a single argument (in double) and returns the square root (also in double).

$$[\text{Mathematics}] \sqrt{x} = \text{sqrt}(x) \quad [\text{In C Programming}]$$

- The sqrt () function is defined in math.h header file.

To find square root of type int, float or long double, you can explicitly convert the type to double using cast operator.

c. Operator Overloading:

- Operator overloading is one of the special feature of C++.
- It also shows the extensibility of C++. C++ permits us to add two variables of user defined types with the same way that is applied with built in type data type. This refers to ability to provide special meaning for existing data type. This mechanism of giving such special meaning to an operator is known as overloading.
- Operator overloading provides a flexible option for creation of new definition for most of the C++ operators. We can assign additional meaning to all existing C++ operators.
- To define an additional task to an operator, we must specify what it means in relation to class to which the operator is applied. This is done with the help of a special function, called operator function which describes the task.

```
Return type class_name :: operator op( argument list)
{
    // Body of the function
    // The task defined by overloaded operator
}
```

- where:
 - return type is type of value returned by specified operation.
 - op is operator being overloaded.
 - operator is a keyword in C++
- The Syntax of declaration of an Operator function is as follows:
operator operator_name
- For example, suppose that we want to declare an Operator function for ‘=’. We can do it as follows:

```
operator =
```

Rules for overloading operators:

- 1) Only existing operators can be overloaded. New operators cannot be overloaded.
- 2) The overloaded operator must have at least one operand that is of user defined type.
- 3) We cannot change the basic meaning of an operator. That is to say, We cannot redefine the plus(+) operator to subtract one value from the other.
- 4) Overloaded operators follow the syntax rules of the original operators. They cannot be overridden.
- 5) There are some operators that cannot be overloaded like size of operator(sizeof), membership operator(.), pointer to member operator(.*), scope resolution operator(::), conditional operators(?:) etc
- 6) We cannot use “friend” functions to overload certain operators. However, member function can be used to overload them. Friend Functions can not be used with assignment operator(=), function call operator(()), subscripting operator([]), class member access operator(->) etc.
- 7) Unary operators, overloaded by means of a member function, take no explicit arguments and return no explicit values, but, those overloaded by means of a friend function, take one reference argument (the object of the relevant class).
- 8) Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.
- 9) When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.
- 10) Binary arithmetic operators such as +, -, * and / must explicitly return a value. They must not attempt to change their own arguments.

Algorithm:

1. Create a class Quadratic with variable a, b and c to represent degree two polynomial.
2. Define default and parameterized constructors to initialize a, b and c variables
3. Define function getData() to accept values of a,b and c from user
4. Define a function eval to calculate result of quadratic equation.
5. eval function must accept value of x from user for calculating result of quadratic equation.
6. Define function findRoots() to calculate roots of quadratic equation.
7. In findRoots(), calculate determinant using formula $(b*b)-4*a*c$
8. If determinant is greater than zero, calculate real and different roots and display roots.
9. If determinant is equal to zero, calculate real and equal roots and display roots.
10. If determinant is less than zero, calculate complex and different roots and display.
11. Create a function to overload binary operator + and display addition of two polynomials.
12. Define a friend function to overload insertion operator << to print quadratic equation
13. Define a friend function to overload extraction operator >> to enter value of quadratic equation variables a, b and c.
14. In main function, create three objects of Quadratic class to represent three equations.
15. Find roots of all quadratic equations and evaluate their result.
16. Display addition of two quadratic equations in third equation.

Test Cases:

Refer test cases given in previous assignments and write down test cases for this assignment.

Conclusion:

Hence, we have studied, used and demonstrated use of binary operator overloading and insertion-extraction operator overloading using friend function.

PROGRAM

```
/*
```

Problem Statement: Implement a class Quadratic that represents degree two polynomials i.e., polynomials of type ax^2+bx+c . The class will require three data members corresponding to a, b and c. Implement the following operations:

1. A constructor (including a default constructor which creates the 0 polynomial).
2. Overloaded operator+ to add two polynomials of degree 2.
3. Overloaded << and >> to print and read polynomials. To do this, you will need to decide what you want your input and output format to look like.
4. A function eval that computes the value of a polynomial for a given value of x.
5. A function that computes the two solutions of the equation $ax^2+bx+c=0$.

```
*/
```

```
#include<iostream>
```

```
#include<math.h> //used for sqrt function
```

```
using namespace std;
```

```
class Quadratic {
```

```
    private :
```

```
    int a,b,c,x;
```

```
    public:
```

```

    Quadratic() {          a=b=c=0;          }
    Quadratic(int a,int b,int c) {
        this->a=a;
        this->b=b;
        this->c=c;
    }
    //overloaded insertion (<<) opertor
    friend ostream &operator<<(ostream &out, Quadratic &q) {
        out<<"\nQuadratic equation is\t";
        out <<q.a<<" X * X + "<<q.b<<" X + "<<q.c ;
        return out;
    }
    //overloaded extraction (>>) opertor
    friend istream &operator>>(istream &in, Quadratic &q) {
        cout<<"\nEnter values of a,b,c of quadratic equation:\n";
        in>>q.a>>q.b>>q.c;
        return in;
    }
    Quadratic operator +(Quadratic);
    void eval();
    void findRoots();
};

void Quadratic::eval() {
    cout<<"\nEnter value of x\n";
    cin>>x;
    int result=(a*x*x)+(b*x)+c;
    cout<<"\nFor x= "<<x<<" Answer is : "<<result<<endl;
}

void Quadratic::findRoots() {
    int temp=(b*b)-(4*a*c);
    if(temp>0) {
        float r1=(-b+sqrt(temp))/(2.0*a);
        float r2=(-b-sqrt(temp))/(2.0*a);
        cout<<"\nRoots are real and different";
        cout<<"\nRoot1= "<<r1;
        cout<<"\nRoot2= "<<r2;
    }
    else if(temp==0) {
        cout<<"\nRoots are real and equal";
        float r=(-b+sqrt(temp))/(2.0*a);
        cout<<"\nEqual Roots are"<<r;
    }
    else {
        cout << "\nRoots are complex and different." << endl;
        float real = -b/(2*a);
        float imaginary =sqrt(-temp)/(2*a);
    }
}

```

```

        cout << "Root1 = " << real << "+" << imaginary << "i" << endl;
        cout << "Root2 = " << real << "-" << imaginary << "i" << endl;
    }
}

Quadratic Quadratic::operator +(Quadratic q) {
    Quadratic temp;
    temp.a=this->a+q.a;
    temp.b=this->b+q.b;
    temp.c=this->c+q.c;
    return temp;
}

int main()
{
    Quadratic q1,q2,q3;
    cin>>q1;
    cin>>q2;
    q3=q1+q2;
    cout<<"\nQuadratic equation 1, its roots and result\n";
    cout<<q1;
    q1.findRoots();
    q1.eval();
    cout<<"\nQuadratic equation 2, its roots and result\n";
    cout<<q2;
    q2.findRoots();
    q2.eval();
    cout<<"\nQuadratic equation 3, its roots and result\n";
    q3.findRoots();
    q3.eval();
    return 0;
}

```

*******OUTPUT*******

Enter values of a,b,c of quadratic equation:

1 5 3

Enter values of a,b,c of quadratic equation:

2 2 6

Quadratic equation 1, its roots and result

Quadratic equation is $1 X^2 + 5 X + 3$

Roots are real and different

Root1= -0.697224

Root2= -4.30278

Enter value of x

3

For x= 3 Answer is : 27

Quadratic equation 2, its roots and result

Quadratic equation is $2 X^2 + 2 X + 6$

Roots are complex and different.

Root1 = $0+1.65831i$

Root2 = $0-1.65831i$

Enter value of x

3

For x= 3 Answer is : 30

Quadratic equation 3, its roots and result

Roots are complex and different.

Root1 = $-1+1.28019i$

Root2 = $-1-1.28019i$

Enter value of x

2

For x= 2 Answer is : 35

OOPL – ASSIGNMENT 3(Batch S6)

Title: Demonstrate use of operator overloading for Rational class.

Objectives:

- 1) To understand concept of operator overloading.
- 2) To demonstrate overloading of binary arithmetic operators, relational and equality operators for rational class.

Problem Statement:

Create a class Rational Number (fractions) with the following capabilities:

- a) Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifies fractions that are not in reduced form and avoids negative denominators.
- b) Overload the addition, subtraction, multiplication and division operators for this class.
- c) Overload the relational and equality operators for this class.

Outcomes:

1. Students will be able to demonstrate use of constructor
2. Students will be able to demonstrate arithmetic binary operators overloading
3. Students will be able to demonstrate overloading of relational and equality operator for rational class.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64 bit Linux/Windows Operating System, G++ compiler

Theory:

a) Rational Number:

- In mathematics, a rational number is any number that can be expressed as the quotient or fraction p/q of two integers, a numerator p and a non-zero denominator q . Since q may be equal to 1, every integer is a rational number.
- The set of all rational numbers, often referred to as “the rationals”, is usually denoted by a boldface Q (or blackboard bold \mathbb{Q}), Unicode \mathbb{Q} ; it was thus denoted in 1895 by Giuseppe Peano after *quoziente*, Italian for “quotient”.
- The decimal expansion of a rational number always either terminates after a finite number of digits or begins to repeat the same finite sequence of digits over and over.
- Moreover, any repeating or terminating decimal represents a rational number. These statements hold true not just for base 10, but also for any other integer base (e.g. binary, hexadecimal).

b) Operator Overloading:

- Operator overloading is one of the special feature of C++.
- It also shows the extensibility of C++. C++ permits us to add two variables of user defined types with the same way that is applied with built in type data type. This refers to ability to provide special meaning for existing data type. This mechanism of giving such special meaning to an operator is known as overloading.
- Operator overloading provides a flexible option for creation of new definition for most of the C++ operators. We can assign additional meaning to all existing C++ operators.

- To define an additional task to an operator, we must specify what it means in relation to class to which the operator is applied. This is done with the help of a special function, called operator function which describes the task.

```
Return type class_name :: operator op( argument list)
{
    // Body of the function
    // The task defined by overloaded operator
}
```

- where:
 - return type is type of value returned by specified operation.
 - op is operator being overloaded.
 - operator is a keyword in C++
- The Syntax of declaration of an Operator function is as follows:


```
operator operator_name
```
- For example, suppose that we want to declare an Operator function for '='. We can do it as follows:

```
operator =
```

Rules for overloading operators:

- 1) Only existing operators can be overloaded. New operators cannot be overloaded.
- 2) The overloaded operator must have at least one operand that is of user defined type.
- 3) We cannot change the basic meaning of an operator. That is to say, We cannot redefine the plus(+) operator to subtract one value from the other.
- 4) Overloaded operators follow the syntax rules of the original operators. They cannot be overridden.
- 5) There are some operators that cannot be overloaded like size of operator(sizeof), membership operator(.), pointer to member operator(.*), scope resolution operator(::), conditional operators(?:) etc
- 6) We cannot use "friend" functions to overload certain operators. However, member function can be used to overload them. Friend Functions can not be used with assignment operator(=), function call operator(()), subscripting operator([]), class member access operator(->) etc.
- 7) Unary operators, overloaded by means of a member function, take no explicit arguments and return no explicit values, but, those overloaded by means of a friend function, take one reference argument (the object of the relevant class).
- 8) Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.
- 9) When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.
- 10) Binary arithmetic operators such as +, -, * and / must explicitly return a value. They must not attempt to change their own arguments.

Algorithm/ Flowchart/Pseudocode

Test Cases:

Refer test cases given in previous assignments and write down test cases for this assignment.

Conclusion:

Hence, we have studied, used and demonstrated use of binary arithmetic operator overloading and overloading equality and relational operators for rational class.

PROGRAM

/*

Create a class Rational Number (fractions) with the following capabilities:

a) Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifies fractions that are not in reduced form and avoids negative denominators.

b) Overload the addition, subtraction, multiplication and division operators for this class.

c) Overload the relational and equality operators for this class.

*/

```
#include<iostream>
```

```
#include<cstdlib>
```

```
using namespace std;
```

```
class Rationall{
```

```
    int numerator,denominator;
```

```
    public:
```

```
        Rationall();
```

```
        Rationall(int,int);
```

```
        Rationall operator+(Rationall);
```

```
        Rationall operator-(Rationall);
```

```
        Rationall operator*(Rationall);
```

```
        Rationall operator/(Rationall);
```

```
        bool operator>(Rationall);
```

```
        bool operator<(Rationall);
```

```
        bool operator>=(Rationall);
```

```
        bool operator<=(Rationall);
```

```
        bool operator==(Rationall);
```

```
        bool operator!=(Rationall);
```

```
        void printRationall();
```

```
        void reduce();
```

```
};
```

```
// default constructor: parameters are numerator and denominator respectively
```

```
// if the number is negative, the negative is always stored in the numerator
```

```
Rationall::Rationall(int n, int d) {
```

```
    numerator = d < 0 ? -n : n;
```

```
    denominator = d < 0 ? -d : d;
```

```
    reduce();
```

```
}
```

```
Rationall::Rationall() {
```

```
    numerator = 0;
```

```
    denominator =1;
```

```
}
```

```
//----- add -----
```

```
// overloaded +: addition of 2 Rationalls, current object and parameter
```

```
Rationall Rationall::operator+(Rationall a) {
```

```
    Rationall t;
```

```
    t.numerator = a.numerator * denominator + a.denominator * numerator;
```

```
    t.denominator = a.denominator * denominator;
```

```

        t.reduce();
        return t;
    }
    //----- subtract -----
    // subtraction of 2 Rationall s, current object and parameter
    Rationall Rationall::operator-(Rationall s) {
        Rationall t;
        t.numerator = s.denominator * numerator - denominator * s.numerator;
        t.denominator = s.denominator * denominator;
        t.reduce();
        return t;
    }
    //----- multiply -----
    // multiplication of 2 Rationall s, current object and parameter
    Rationall Rationall::operator*(Rationall m) {
        Rationall t;
        t.numerator = m.numerator * numerator;
        t.denominator = m.denominator * denominator;
        t.reduce();
        return t;
    }

    //----- divide -----
    // division of 2 Rationall s, current object and parameter, division by zero crashes
    Rationall Rationall::operator/(Rationall v) {
        Rationall t;
        t.numerator = v.denominator * numerator;
        t.denominator = denominator * v.numerator;
        t.reduce();
        return t;
    }
    bool Rationall::operator>(Rationall v) {
        float f1=numerator/float(denominator);
        float f2=v.numerator/float(v.denominator);
        if(f1>f2)
            return true;
        else return false;
    }
    bool Rationall::operator<(Rationall v) {
        float f1=numerator/float(denominator);
        float f2=v.numerator/float(v.denominator);
        if(f1<f2)
            return true;
        else return false;
    }
    bool Rationall::operator>=(Rationall v) {

```

```

        float f1=numerator/float(denominator);
        float f2=v.numerator/float(v.denominator);
        if(f1>=f2)
            return true;
        else return false;
    }
    bool Rationall::operator<=(Rationall v) {
        float f1=numerator/float(denominator);
        float f2=v.numerator/float(v.denominator);
        if(f1<=f2)
            return true;
        else
            return false;
    }
    bool Rationall::operator==(Rationall v) {
        float f1=numerator/float(denominator);
        float f2=v.numerator/float(v.denominator);
        if(f1==f2)
            return true;
        else
            return false;
    }
    bool Rationall::operator!=(Rationall v) {
        float f1=numerator/float(denominator);
        float f2=v.numerator/float(v.denominator);
        if(f1!=f2)
            return true;
        else return false;
    }
}
//----- printRationall -----
void Rationall::printRationall() {
    if (denominator == 0)
        cout << endl << "DIVIDE BY ZERO ERROR!!!" << endl;
    else if (numerator == 0)
        cout << 0;
    else
        cout << numerator << "/" << denominator;
}

//----- reduce -----
// reduce fraction to lowest terms
void Rationall::reduce() {
    int n = numerator < 0 ? -numerator : numerator;
    int d = denominator;
    int largest = n > d ? n : d;
    int gcd = 0; // greatest common divisor

```

```

        for (int loop = largest; loop >= 2; loop--)
            if (numerator % loop == 0 && denominator % loop == 0) {
                gcd = loop;
                break;
            }
        if (gcd != 0) {
            numerator /= gcd;
            denominator /= gcd;
        }
    }
}

int main()
{
    Rationall r1,r2,r3;
    int n,d;
    int choice;
    char ch;
    cout<<"Enter first Rationall number - numerator and denominator";
    cin>>n>>d;
    r1=Rationall(n,d);
    cout<<"Enter second Rationall number - numerator and denominator";
    cin>>n>>d;
    r2=Rationall(n,d);
    do{
        cout<<"*****MENU*****\n";
        cout<<"1. +\t2. -\t3. *\t4. /\t5. >\t6. <\n7. >=\t8. <=\t9. !=\t10. ==";
        cout<<"\n11. Print\t12.Exit\n";
        cout<<"Enter your choice";
        cin>>choice;
        switch(choice)
        {
            case 1:
                r3=r1+r2;
                r3.printRationall();
                break;
            case 2:
                r3=r1-r2;
                r3.printRationall();
                break;
            case 3:
                r3=r1*r2;
                r3.printRationall();
                break;
            case 4:
                r3=r1/r2;
                r3.printRationall();
                break;

```

```

        case 5:
            if(r1>r2)
                r1.printRational1();cout<<" is greater than ";
                r2.printRational1();
            break;
        case 6:
            if(r1<r2)
                r2.printRational1();cout<<" is greater than ";
                r1.printRational1();
            break;
        case 7:
            if(r1>=r2)
                r1.printRational1();cout<<" is greater than or equal to ";
                r2.printRational1();
            break;
        case 8:
            if(r1<=r2)
                r2.printRational1();
                cout<<" is greater than or equal to ";
                r1.printRational1();
            break;
        case 9:
            if(r1!=r2)
                r1.printRational1();cout<<" is not equal to";
                r2.printRational1();
            break;
        case 10:
            if(r1==r2)
                r1.printRational1();cout<<" is equal to ";
                r2.printRational1();
            break;
        case 11:
            cout<<"\nFirst Rational Number\n";
            r1.printRational1();
            cout<<"\nSecond Rational Number\n";
            r2.printRational1();
            cout<<"\nThird Rational Number\n";
            r3.printRational1();
            break;
        case 12:exit(0);;
    }
    cout<<"\nDo you want to continue(y/n)\n";
    cin>>ch;
}while(ch=='y' || ch=='Y');
return 0;
}

```


OOPL – ASSIGNMENT 4(Batch S4)

Title: Demonstrate use of Inheritance & Virtual Function.

Objectives:

1. To understand concept of Inheritance & Pure Virtual Function.
2. To demonstrate concept of Pure Virtual function through unit conversion.

Problem Statement:

Write C++ Program with base class convert declares two variables, val1 and val2, which hold the initial and converted values, respectively. It also defines the functions getinit() & getconv(), which return the initial value and the converted value. These elements of convert are fixed and applicable to all derived classes that will inherit convert. However, the function that will actually perform the conversion, compute(), is a pure virtual function that must be defined by the classes derived from convert. The specific nature of compute() will be determined by what type of conversion is taking place.

Outcomes:

- a) Students will be able to demonstrate use of Hierarchical Inheritance.
- b) Students will be able to demonstrate pure virtual function.
- c) Students will be able to understand how to create abstract class.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) Inheritance:

- C++ supports the concept of reusability once a class has been written and tested, it can be adapted by other programmers to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones.
- The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as base class or super class. And the new one is called as derived class or subclass.
- A derived class represents a more specialized group of objects. Typically, a derived class contains behaviors inherited from its base class plus additional behaviors.

b) Base Class & Derived Class:

- To define a derived class, we use a class derivation list to specify the base class(es).
- A class derivation list names one or more base classes and has the form:

class derived-class : access-specifier base-class

- Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.
- C++ offers three forms of inheritance—public, protected and private.

c) Access Control and Inheritance:

- A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.
- We can summarize the different access types according to who can access them in the following way:

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

- When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance. The type of inheritance is specified by the access-specifier as explained above. We hardly use protected or private inheritance, but public inheritance is commonly used.
- While using different type of inheritance, following rules are applied:
 - Public Inheritance:** When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.
 - Protected Inheritance:** When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.
 - Private Inheritance:** When deriving from a private base class, public and protected members of the base class become private members of the derived class.

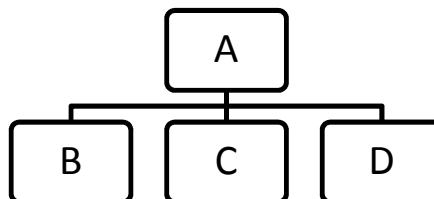
d) Types of Inheritances:

- Following are the types of Inheritance:

- Single Level Inheritance
- Multiple Inheritances
- Hierarchical inheritance
- Multilevel Inheritance
- Hybrid Inheritance.

e) Hierarchical Inheritance:

- Properties of one class is inherited in more derived class is known as “hierarchical inheritance”



- Syntax:

```

class base_class_name{ -----};
class derived_class_name1: visibility mode base_class_name{ -----};
class derived_class_name2: visibility mode base_class_name{-----};
class derived_class_name3: visibility mode base_class_name{-----};
  
```

- Here base_class_name is Super class and derived_class1, derived_class2, derived_class3 are the derived classes which can inherits the base_class_name class.
- Visibility mode has been either from public, protected or private.
- Here we can create objects of the derived classes from these objects we can call the base class member functions.
- Example:

```

class Student {-----};
class Marksheet : public Student{-----};
class Sport : public Student{-----};
  
```

f) Pure virtual functions:-

Virtual function which does not have any body or statements can be declared as 'pure virtual function' we can implement it by assigning the value zero(0) to a virtual function

e.g.

```
Virtual void display()=0;
```

Here the display functions is known pure virtual because it is having value.

g) Abstract classes:-

An *abstract class* is a class that is designed to be specifically used as a base class. An abstract class contains at least one *pure virtual function*. You declare a pure virtual function by using a *pure specifier* (= 0) in the declaration of a virtual member function in the class declaration. The following is an example of an abstract class:

```
class ab
{
    public:
    virtual void fun()=0;
};
```

Algorithm:

Flowchart: Draw flowchart and get it corrected from faculty.

Test Cases: Refer test cases given in previous assignments and write down test cases for this assignment.

Conclusion:

In this assignment we have studied the concept of Hierarchical inheritance, Pure virtual function and abstract class concepts also applied it for given problem statement.

Program

```
#include<iostream>
using namespace std;
class Convert{
    protected:
        double val1,val2;
        double getinit(){
            return val1;
        }
        double getconv(){
            return val2;
        }
    public:virtual void compute()=0;
};
class KMtoM:public Convert{
    double temp;
    public:
        void compute(){
            cout<<"\n\t\t Kilometer to Meter Convertor";
            cout<<"\n\t Enter the Kilometer value:";
            cin>>val1;
            cout<<"\n\t Kilometer value:"<<getinit();
            val2=val1*1000;
            cout<<"\n\t Meter value:"<<getconv();
        }
};
```

```

class CMtoIN:public Convert{
    double temp;
public:
    void compute(){
        cout<<"\n\t\t Centimeter to Inch Convertor";
        cout<<"\n\t Enter the Centimeter value:";
        cin>>vall;
        cout<<"\n\t Centimeter value:"<<getinit();
        val2=vall/2.54;
        cout<<"\n\t Inch value:"<<getconv();
    }
};

class CLtoF:public Convert{
    double temp;
public:
    void compute(){
        cout<<"\n\t\t Celsius to Fahrenheit Convertor";
        cout<<"\n\t Enter the Celsius value:";
        cin>>vall;
        cout<<"\n\t Celsius value:"<<getinit();
        val2=vall*1.8+32;
        cout<<"\n\t Fahrenheit value:"<<getconv();
    }
};

int main(){
    Convert *c;
    KMtoM km;
    c=&km;
    c->compute();
    CMtoIN ci;
    c=&ci;
    c->compute();
    CLtoF cf;
    c=&cf;
    c->compute();
    return 0;
}

```

Output:

```

    Kilometer to Meter Convertor
Enter the Kilometer value:2.6
Kilometer value:2.6
Meter value:2600
    Centimeter to Inch Convertor
Enter the Centimeter value:5
Centimeter value:5
Inch value:1.9685
    Celsius to Fahrenheit Convertor
Enter the Celsius value:3
Celsius value:3
Fahrenheit value:37.4

```

OOPL – ASSIGNMENT 4(Batch S5)

Title: Demonstrate reusability of code thru Inheritance and use of exception handling.

Objectives:

- 1) To learn and understand code reusability and demonstrate it using Inheritance concepts.
- 2) To learn, understand and demonstrate exception handling in object oriented environment.

Problem Statement:

Imagine a publishing company which does marketing for book and audiocassette versions. Create a class publication that stores the title (a string) and price (type float) of a publication. From this class derive two classes: book, which adds a page count (type int), and tape, which adds a playing time in minutes (type float).

Write a program that instantiates the book and tape classes, allows user to enter data and displays the data members. If an exception is caught, replace all the data member values with zero values.

Outcomes:

- 1) Students will be able to learn and understand inheritance and exception handling.
- 2) Students will be able to demonstrate inheritance and exception handling.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) Inheritance:

- C++ supports the concept of reusability once a class has been written and tested, it can be adapted by other programmers to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones.
- The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as base class or super class. And the new one is called as derived class or subclass.
- A derived class represents a more specialized group of objects. Typically, a derived class contains behaviors inherited from its base class plus additional behaviors.

b) Base Class & Derived Class:

- To define a derived class, we use a class derivation list to specify the base class(es).
- A class derivation list names one or more base classes and has the form:

class derived-class : access-specifier base-class

- Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.
- C++ offers three forms of inheritance—public, protected and private.

c) Access Control and Inheritance:

- A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.
- We can summarize the different access types according to who can access them in the following way:

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

- When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance. The type of inheritance is specified by the access-specifier as explained above. We hardly use protected or private inheritance, but public inheritance is commonly used.
- While using different type of inheritance, following rules are applied:
 - i. **Public Inheritance:** When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.
 - ii. **Protected Inheritance:** When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.
 - iii. **Private Inheritance:** When deriving from a private base class, public and protected members of the base class become private members of the derived class.

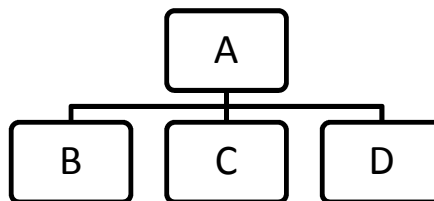
d) Types of Inheritances:

- Following are the types of Inheritance:

- A. Single Level Inheritance
- B. Multiple Inheritances
- C. Hierarchical inheritance
- D. Multilevel Inheritance
- E. Hybrid Inheritance.

e) Hierarchical Inheritance:

- Properties of one class is inherited in more derived class is known as “hierarchical inheritance”



- Syntax:

```

class base_class_name{ -----};
class derived_class_name1: visibility mode base_class_name{ -----};
class derived_class_name2: visibility mode base_class_name{-----};
class derived_class_name3: visibility mode base_class_name{-----};
  
```

- Here base_class_name is Super class and derived_class1, derived_class2, derived_class3 are the derived classes which can inherits the base_class_name class.
- Visibility mode has been either from public, protected or private.
- Here we can create objects of the derived classes from these objects we can call the base class member functions.

- Example:

```

class Student {-----};
class Marksheet : public Student{-----};
class Sport : public Student{-----};
  
```

f) Exception Handling:

- An exception occurs when an unexpected error or unpredictable behaviors happened on your program not caused by the operating system itself. These exceptions are handled by code which is outside the normal flow of control and it needs an emergency exit.

- Compared to the structured exception handling, returning an integer as an error flag is problematic when dealing with objects.
- The C++ exception-handling can be a full-fledged object, with data members and member functions. Such an object can provide the exception handler with more options for recovery. A clever exception object, for example, can have a member function that returns a detailed verbal description of the error, instead of letting the handler look it up in a table or a file.
- C++ has incorporated three operators to help us handle these situations: try, throw and catch. The following is the try, throw...catch program segment example:
- Syntax:

```

try
{
    Compound-statement handler-list
    handler-list here
    The throw-expression:
    throw expression
}
catch (exception-declaration) compound-statement
{
    Exception-declaration:
    type-specifier-list here
}

```

- **try:** A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.
- **throw:** A program throws an exception when a problem shows up. This is done using a throw keyword.
- **catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

Algorithm:

- 1) Create a class Marketing with variable title and price.
- 2) Define default and parameterized constructors to initialize title and price variables
- 3) Define function getData() to accept values from user
- 4) Define a function putData() to print these values.
- 5) In putData(), check if length of title is more than 3 chars else generate exception and handle exception.
- 6) In putData(), check if price is non negative else generate exception and handle exception.
- 7) Create a derived class Book from base Marketing with variable number of pages.
- 8) Define default and parameterized constructors to initialize variables
- 9) Define function getData() to accept values from user
- 10) Define a function putData() to print these values.
- 11) In putData(), check if number of pages is non negative else generate exception and handle exception.
- 12) Create a derived class Cassette from base Marketing with variable number of pages.
- 13) Define default and parameterized constructors to initialize variables
- 14) Define function getData() to accept values from user
- 15) Define a function putData() to print these values.
- 16) In putData(), check if number of pages is non negative else generate exception and handle exception.
- 17) In main function, create two objects, one of Book class and other of Cassette class. Use appropriate constructors and methods to print details of Book and Cassette.

Flowchart:

Draw proper flowchart and get it corrected from faculty.

Test Cases:

Test Case 1: (All valid inputs are given and desired output is shown)

Enter Title : C++

Enter Price : 250.34

Enter No of pages of book : 356

Enter Title : Vande Mataram

Enter Price : 156

Enter Play time in minutes : 34.54

Test Case 2: (Base class require title length more than 3 chars and price more than 0 otherwise exception will be generated)

Enter Title : C

Enter Price : -250.34

Enter No of pages of book : 356

Enter Title : Vande Mataram

Enter Price : -156

Enter Play time in minutes : 34.54

Error Msg : Title can not be less than three characters Price can't be negative

Test Case 3: (Book require non negative number of pages and Cassette needs non negative play time)

Enter Title : C++

Enter Price : 250.34

Enter No of pages of book : -356

Enter Title : Vande Mataram

Enter Price : 156

Enter Play time in minutes : -34.54

Error Msg : Number of Pages can't be negative

Play time must be more than 0

Conclusion: Hence, we learnt to use and demonstrate concepts of inheritance and exception handling.

Program

```
#include <iostream>
using namespace std;
class Marketing{
public:
    Marketing(){
        title="";
        price=0.0;
    }
    Marketing(string title, float price){
        this->title=title;
        this->price=price;
    }
    void getData(){
        cout<<"\nEnter title and price\n";
        cin>>title>>price;
    }
    void putData(){
        try
        {
            if(title.length()<3)
                throw title;
```



```

        if(price<=0.0)
            throw price;
    }
    catch(string){
        cout<<"\nError: Title below 3 characters is not allowed";
        title="";
    }
    catch(float f)
    {
        cout<<"\nError: Price not valid: \t"<<f;
        price=0.0;
    }
    cout<<"\nTitle is : "<<title;
    cout<<"\nPrice is : "<<price;
}
private:
    string title;
    float price;
};

class Book: public Marketing{
public:
    Book():Marketing(){
        pages=0;
    }
    Book(string title, float price, int pages):Marketing(title,price){
        this->pages=pages;
    }
    void getData(){
        Marketing::getData();
        cout<<"\nEnter no. of pages in book\n";
        cin>>pages;
    }
    void putData(){
        Marketing::putData();
        try
        {
            if(pages<0)
                throw pages;
        }
        catch(int f){
            cout<<"\nError: Pages not valid: \t"<<f;
            pages=0;
        }
        cout<<"\nPages are : "<<pages;
    }
private:
    int pages;
};

class Cassette: public Marketing{
public:
    Cassette():Marketing(){
        playtime=0.0;
    }
};

```

```

    }
    Cassette(string title, float price, float
    playtime):Marketing(title,price){
        this->playtime=playtime;
    }
    void getData(){
        Marketing::getData();
        cout<<"\nEnter play time of cassette\n";
        cin>>playtime;
    }
    void putData(){
        Marketing::putData();
        try
        {
            if(playtime<0.0)
                throw playtime;
        }
        catch(float f){
            cout<<"\nError: Playtime not valid: \t"<<f;
            playtime=0.0;
        }
        cout<<"\nPlaytime is :"<<playtime;
    }
private:
    float playtime;
};

int main(){
    Book book;
    cout<<"\n*****BOOK*****\n";
    book.getData();
    cout<<"\n*****CASSETTE*****\n";
    Cassette cassette;
    cassette.getData();
    cout<<"\n*****BOOK*****\n";
    book.putData();
    cout<<"\n*****CASSETTE*****\n";
    cassette.putData();
    return 0;
}

```

Output

```

*****BOOK*****
Enter title and price c++ 111.44
Enter no. of pages in book 234
*****CASSETTE*****
Enter title and price ddlj 100
Enter play time of cassette 23.4
*****BOOK*****
Title is :c++
Price is :111.44
Pages are :234

```

*****CASSETTE*****

Title is :ddlj

Price is :100

Playtime is :23.4

OUTPUT 2:

*****BOOK*****

Enter title and price C 100

Enter no. of pages in book -34

*****CASSETTE*****

Enter title and price DDLJ 100

Enter play time of cassette 23.4

*****BOOK*****

Error: Title below 3 characters is not allowed

Title is :

Price is :100

Error: Pages not valid: -34

Pages are :0

*****CASSETTE*****

Error: Price not valid: -100

Title is :DDLJ

Price is :0

Playtime is :23.4

OOPL – ASSIGNMENT 4 (Batch S6)

Title: Demonstration of multiple Inheritance to create Bio-data of student

Objectives:

1. To learn and understand code reusability and demonstrate it using Inheritance

Problem Statement:

Create Student\Employee bio-data using following classes i) Personal record ii) Professional record iii) Academic record Assume appropriate data members and member function to accept required data & print bio-data. Create bio-data using multiple inheritance using C++.

Outcomes:

- 1 Students will be able to learn and understand multiple inheritance.
- 3) Students will be able to demonstrate multiple inheritance.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) Inheritance:

- C++ supports the concept of reusability once a class has been written and tested, it can be adapted by other programmers to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones.
- The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as base class or super class. And the new one is called as derived class or subclass.
- A derived class represents a more specialized group of objects. Typically, a derived class contains behaviors inherited from its base class plus additional behaviors.

b) Base Class & Derived Class:

- To define a derived class, we use a class derivation list to specify the base class(es).
- A class derivation list names one or more base classes and has the form:

class derived-class : access-specifier base-class

- Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.
- C++ offers three forms of inheritance—public, protected and private.

c) Access Control and Inheritance:

- A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.
- We can summarize the different access types according to who can access them in the following way:

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

- When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance. The type of inheritance is specified by the access-specifier as explained above. We hardly use protected or private inheritance, but public inheritance is commonly used.

- While using different type of inheritance, following rules are applied:
 - Public Inheritance:** When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.
 - Protected Inheritance:** When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.
 - Private Inheritance:** When deriving from a private base class, public and protected members of the base class become private members of the derived class.

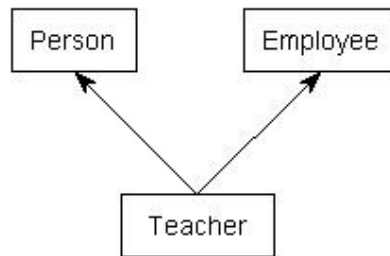
d) Types of Inheritances:

- Following are the types of Inheritance:

- F. Single Level Inheritance
- G. Multiple Inheritances
- H. Hierarchical inheritance
- I. Multilevel Inheritance
- J. Hybrid Inheritance.

g) Multiple Inheritance:

- When a derived class is created from more than one base class then that inheritance is called as multiple inheritance. But multiple inheritance is not supported by .net using classes and can be done using interfaces.



- Let's say we wanted to write a program to keep track of a bunch of teachers. A teacher is a person. However, a teacher is also an employee (they are their own employer if working for themselves). Multiple inheritance can be used to create a Teacher class that inherits properties from both Person and Employee.
- To use multiple inheritance, simply specify each base class (just like in single inheritance), separated by a comma.


```

class Person { };
class Employee { };
class Teacher : public Person, public Employee { };
      
```

Algorithm:

1. Create a class Personal to enter personal details
2. Declare name, address and gender as string variables while contact as integer variables.
3. Write default constructor to initialize all variables.
4. Define getData() to enter all values from user.
5. Define putData() to display values entered.
6. Create a class Professional to enter professional details
7. Declare variables to enter operating system, programming lang details.
8. Write default constructor to initialize all variables.
9. Define getData() to enter all values from user.
10. Define putData() to display values entered.
11. Create a class Academic to enter academic details

12. Declare variables to enter academic details of user.
13. Write default constructor to initialize all variables.
14. Define getData() to enter all values from user.
15. Define putData() to display values entered.
16. Create a class CV from base class Personal, Professional and Academic and inherit publicly.
17. Declare string variable to enter title of your CV.
18. Write default constructor to initialize all variables of all base classes and derived class too.
19. Define getData() to enter values from user and to call base class method
20. Define putData() to display values entered and to call base class method.
21. In main(), create object of CV class.
22. Call getData() method of CV class to enter personal, professional and academic details and to enter title for CV.
23. Call putData() method of CV class to display personal, professional and academic details and title for CV.

Flowchart: Draw flowchart and get it corrected from faculty.

Test Cases:

Refer test cases given in previous assignments and write down test cases for this assignment.

Conclusion:

In this assignment we have studied the concept of multiple inheritance and applied it for given problem statement.

Program

//Demonstration of Multiple Inheritance

```
#include<iostream>
```

```
#include<stdlib.h>
```

```
using namespace std;
```

```
class Personal{
```

```
private:
```

```
string name,address,gender;
```

```
int contact;
```

```
public:
```

```
Personal(){
```

```
name="";
```

```
address="";
```

```
gender="";
```

```
contact=-1;
```

```
}
```

```
void getData(){
```

```
cout<<"\n***** Enter your personal details*****";
```

```
cout<<"\nEnter your name:";
```

```
cin>>name;
```

```
cout<<"\nEnter gender:";
```

```
cin>>gender;
```

```
cout<<"\nEnter address:";
```

```
cin>>address;
```

```
cout<<"\nEnter contact number:";
```

```
cin>>contact;
```

```
}
```

```
void putData(){
```

```
cout<<"\n your Personal Details are:- ";
```

```
cout<<"\n\nName of the Student is:\t"<<name;
```

```
cout<<"\nGender of the Student is:\t"<<gender;
```

```

        cout<<"\nAddress of the Student is:\t"<<address;
        cout<<"\nContact number of the Student is:\t"<<contact;
    }
};
class Professional{
private:
    string os,pl;
    string classname;
public:
    Professional(){
        os="";
        pl="";
        classname="";
    }
    void getData(){
        cout<<"\n\nEnter your Professional Deatails: ";
        cout<<"\n\nEnter name of the class: ";
        cin>>classname;
        cout<<"\nEnter which operating system do you know?:";
        cin>>os;
        cout<<"\nEnter which progrmming language do u know?:";
        cin>>pl;
    }
    void putData(){
        cout<<"\n your Professional Details are:- \n ";
        cout<<"\n\n Name of the Class is:\t"<<classname;
        cout<<"\n known OS is:\t"<<os;
        cout<<"\n known Programming language is:\t"<<pl;
    }
};
class Academic{
private:
    float ten,twe,fe,se;
public:
    Academic(){
        ten=-1;
        twe=-1;
        fe=-1;
        se=-1;
    }
    void getData(){
        cout<<"\n\nEnter 10th,12th,F.E,S.E Percent: \n";
        cin>>ten>>twe>>fe>>se;
    }
    void putData(){
        cout<<"\n\n Your Academic Details are:- ";
        cout<<"\n 10th marks are"<<ten;
        cout<<"\n 12th marks are"<<twe;
        cout<<"\n FE marks are"<<fe;
        cout<<"\n SE marks are"<<se;
    }
};

```

```

class CV:public Personal,public Professional,public Academic{
private:
string title;
public:
CV():Personal(),Professional(),Academic(){
    title="";
}
void getData(){
    Personal::getData();
    Professional::getData();
    Academic::getData();
    cout<<"\nEnter title for Resume";
    cin>>title;
}
void putData(){
    cout<<"\n*****<<title<<"*****\n";
    Personal::putData();
    Professional::putData();
    Academic::putData();
}
};
int main(){
    CV c1;
    c1.getData();
    c1.putData();
    return 0;
}

```


OOPL – ASSIGNMENT 5

Title: Demonstrate class template for vector class.

Objectives:

- 1) To learn and understand templates.
- 2) To demonstrate class template for vector class and its various operations.

Problem Statement:

Create a class template to represent a generic vector. Include following member functions:

1. To create the vector.
2. To modify the value of a given element
3. To multiply by a scalar value
4. To display the vector in the form (10,20,30,...)

Outcomes:

- 1) Students will be able to learn and understand working and use of class template.
- 2) Students will be able to demonstrate class template for vector class with various operations.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) Template

- Templates are a feature of the C++ programming language that allows functions and classes to operate with generic types.
- This allows a function or class to work on many different data types without being rewritten for each one. Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.
- A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept.
- There is a single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector <int> or vector <string>.
- Templates are a way of making your classes more abstract by letting you define the behavior of the class without actually knowing what datatype will be handled by the operations of the class.
- In essence, this is what is known as generic programming; this term is a useful way to think about templates because it helps remind the programmer that a templated class does not depend on the datatype (or types) it deals with. To a large degree, a templated class is more focused on the algorithmic thought rather than the specific nuances of a single datatype.
- Templates can be used in conjunction with abstract datatypes in order to allow them to handle any type of data.
- For example, you could make a templated stack class that can handle a stack of any datatype, rather than having to create a stack class for every different datatype for which you want the stack to function.
- The ability to have a single class that can handle several different datatypes means the code is easier to maintain, and it makes classes more reusable.
- The basic syntax for declaring a templated class is as follows:
template <class a_type> class a_class {...};
- The keyword 'class' above simply means that the identifier a_type will stand for a datatype.

- NB: `a_type` is not a keyword; it is an identifier that during the execution of the program will represent a single datatype. For example, you could, when defining variables in the class, use the following line:

```
a_type a_var;
```

- and when the programmer defines which datatype '`a_type`' is to be when the program instantiates a particular instance of `a_class`, `a_var` will be of that type.
- When defining a function as a member of a templated class, it is necessary to define it as a templated function

```
template<class a_type> void a_class<a_type>::a_function(){...}
```

- When declaring an instance of a templated class, the syntax is as follows:

```
a_class<int> an_example_class;
```
- An instantiated object of a templated class is called a specialization; the term specialization is useful to remember because it reminds us that the original class is a generic class, whereas a specific instantiation of a class is specialized for a single datatype (although it is possible to template multiple types).
- Usually when writing code it is easiest to precede from concrete to abstract; therefore, it is easier to write a class for a specific datatype and then proceed to a templated – generic – class.
- For that brevity is the soul of wit, this example will be brief and therefore of little practical application.
- We will define the first class to act only on integers.

```
class calc{
    public:
    int multiply(int x, int y);
    int add(int x, int y);
};
int calc::multiply(int x, int y){
    return x*y;
}
int calc::add(int x, int y){
    return x+y;
}
```

- We now have a perfectly harmless little class that functions perfectly well for integers; but what if we decided we wanted a generic class that would work equally well for floating point numbers? We would use a template.

```
template <class A_Type>
class calc{
    public:
    A_Type multiply(A_Type x, A_Type y);
    A_Type add(A_Type x, A_Type y);
};
template <class A_Type>
A_Type calc<A_Type>::multiply(A_Type x,A_Type y){
    return x*y;
}
template <class A_Type>
A_Type calc<A_Type>::add(A_Type x, A_Type y){
    return x+y;
}
```

- To understand the templated class, just think about replacing the identifier `A_Type` everywhere it appears, except as part of the template or class definition, with the keyword `int`.

It would be the same as the above class; now when you instantiate an object of class calc you can choose which datatype the class will handle.

calc <double> a_calc_class;

- Templates are handy for making your programs more generic and allowing your code to be reused later.

b) Vector:

- A vector, in programming, is a type of array that is one dimensional. Vectors are a logical element in programming languages that are used for storing data. Vectors are similar to arrays but their actual implementation and operation differs.
- Vectors are primarily used within the programming context of most programming languages and serve as data structure containers. Being a data structure, vectors are used for storing objects and collections of objects in an organized structure.
- The major difference between an array and a vector is that, unlike typical arrays, the container size of a vector can be easily increased and decreased to complement different data storage types. Vectors have a dynamic structure and provide the ability to assign container size up front and enable allocation of memory space quickly. Vectors can be thought of as dynamic arrays.

Algorithm:

1. Define a class Vector of type template
2. Declare one parametrized constructor and four methods to create vector, display vector, modify element in vector and multiply vector by a scalar value.
3. Define constructor outside the class with size passed as parameter
4. Define method create to input size number of elements in vector.
5. Define display method to display size number of elements in vector.
6. Define modify method where index position of element to be modified is passed as argument.
7. Check if index position is not greater than size of vector.
8. Accept new element from user and update element at given index position with new value.
9. Define multiply method with parameter as scalar value and multiply all elements in vector with given scalar value.
10. Define main method
11. Accept size of vector from user
12. Accept data type of vector as either integer or float
13. Display menu of all available functions.
14. Accept choice of user
15. If choice is 1 which must be first choice to input elements in vector, call create method of vector class.
16. For choice 2, enter index position to modify and call modify method
17. For choice 3, enter scalar value for multiplication and call multiply the method
18. If choice is 4, display current elements in the vector
19. For choice 5, exit the program.
20. If user wants to continue, Repeat steps 13 to 20 till user enters 'n' or 5 as choice to exit program
21. End the program.

Flowchart:

Draw proper flowchart and get it corrected from faculty.

Test Cases:

Test Case 1: (All valid inputs are given and desired output is shown)

Enter size of vector: 5

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR

3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation. Enter your choice:

1

Enter 5 elements in vector

2 5 3 12 55

Do you want to continue(y/n)? y

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation. Enter your choice:

4 E

lements in vector:

2 5 3 12 55

Do you want to continue(y/n)? y

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation. Enter your choice:

2 E

Enter index of element you want to modify: 3

Enter new value: 33

Do you want to continue(y/n)? y

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation. Enter your choice:

4 E

lements in vector:

2 5 33 12 55

Do you want to continue(y/n)? y

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation. Enter your choice:

3 E

Enter scalar value to multiply vector

2 D

Do you want to continue(y/n)? y

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation. Enter your choice:

4 E

lements in vector:

4 10 66 24 110

Do you want to continue(y/n)? n

Test Case 2: (If size is negative)

Enter size of vector: -5

Size can't be negative. Initializing to 1.

Test Case 3: (If index of element to modify is greater than size.)

Enter index of element you want to modify: 30

Index position can't be greater than size of vector

Test Case 4: (If index of element to modify is less than 0)

Enter index of element you want to modify: -3

Index can not be negative. So initializing as 0

Conclusion:

Hence, we have demonstrated use of class template for vector class.

Program:

```
#include<iostream>
#include<cstdlib>
using namespace std;
template<class T>
class Vector{
    int size;
    T *v;
public:
    Vector(int size);
    void create();
    void modify(int elem);
    void multiply(int val);
    void display();
};
template<class T>
Vector<T>::Vector(int size){
    this->size=size;
    v=new T[this->size];
}
template<class T>
void Vector<T>::create(){
    cout<<"Enter "<<size<<" elements in vector:\n";
    for(int i=0;i<size;i++)
        cin>>v[i];
}
template<class T>
void Vector<T>::display(){
```

```

        cout<<"Elements in vector:\n( ";
        for(int i=0;i<size;i++)
            cout<<v[i]<<" ";
        cout<<")";
    }
    template<class T>
    void Vector<T>::modify(int elem){
        if(elem>size){
            cout<<"Index position cann't be greater than size of vecotr";
        }
        else{
            if(elem<0){
                cout<<"Index can not be negative. So initializing as 0";
                elem=1;
            }
            cout<<"\nEnter new value:\n";
            cin>>v[elem-1];
        }
    }
}
template<class T>
void Vector<T>::multiply(int value){
    for(int i=0;i<size;i++)
        v[i]=v[i]*value;
}
int main() {
    char ch;
    int size;
    cout<<"\nEnter size of vector:\n ";
    cin>>size;
    if(size<0){
        cout<<"\nSize cann't be negative. Initializing to 1";
        size=1;
    }
    Vector<int> v(size);
    do{
        cout<<"\n*****MENU*****\n";
        cout<<"1. CREATE VECTOR\n2. MODIFY VECTOR\n";
        cout<<"3. MULTIPLY VECTOR WITH SCALAR\n4. DISPLAY VECTOR\n5. EXIT";
        cout<<"\nNote: Create vector before doing any other operation";
        cout<<"\nEnter your choice:\n";
        int choice;
        cin>>choice;
        switch(choice){
            case 1:
                v.create();
                break;
            case 2: int elem;
                cout<<"\nEnter index of element you want to modify ";
                cin>>elem;
                v.modify(elem);
                break;

```

```

        case 3:
            int val;
            cout<<"\nEnter scalar value to multiply vector\n ";
            cin>>val;
            v.multiply(val);
            break;
        case 4: v.display();
            break;
        case 5:
            exit(0);
            break;
    }
    cout<<"\nDo you want to continue(y/n)?\n";
    cin>>ch;
}while(ch == 'y' || ch == 'Y');
return 0;
}

```

OUTPUT:

```

Enter size of vector: 4
*****MENU*****
1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT
Note: Create vector before doing any other operation
Enter your choice: 1
Enter 4 elements in vector:
33 44 22 55
Do you want to continue(y/n)? Y
*****MENU*****
1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT
Note: Create vector before doing any other operation
Enter your choice: 4
Elements in vector:
( 33 44 22 55 )
Do you want to continue(y/n)? Y
*****MENU*****
1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT
Note: Create vector before doing any other operation
Enter your choice: 2
Enter index of element you want to modify 3
Enter new value: 44
Do you want to continue(y/n)? Y

```

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation

Enter your choice: 4

Elements in vector:

(33 44 44 55)

Do you want to continue(y/n)? Y

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation

Enter your choice: 3

Enter scalar value to multiply vector 2

Do you want to continue(y/n)? Y

*****MENU*****

1. CREATE VECTOR
2. MODIFY VECTOR
3. MULTIPLY VECTOR WITH SCALAR
4. DISPLAY VECTOR
5. EXIT

Note: Create vector before doing any other operation

Enter your choice: 4

Elements in vector:

(66 88 88 110)

Do you want to continue(y/n)? N

OOPL – ASSIGNMENT 6

Title: Demonstrate file handling operations to count number of lines in a file.

Objectives:

- 1) To learn and understand file handling functions.
- 2) To demonstrate file reading and writing functions on a file.

Problem Statement:

Write a function in C++ to count and display the number of lines not starting with alphabet 'A' present in a text file "STORY.TXT". Example: If the file "STORY.TXT" contains the following lines,

The roses are red.

A girl is playing there.

There is a playground.

An aeroplane is in the sky.

Numbers are not allowed in the password.

The function should display the output as 3.

Outcomes:

- 1) Students will be able to learn and understand file handling operations.
- 2) Students will be able to demonstrate counting number of lines in a file matching with criteria.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

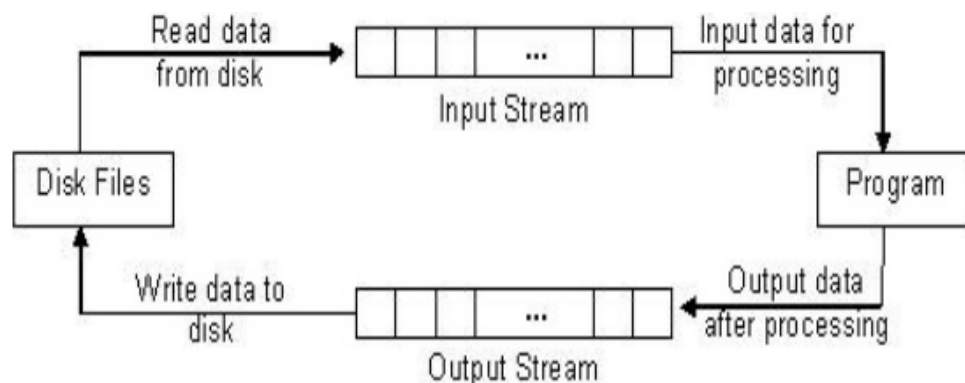
Software requirements:

64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) File

- A file is a stream of bytes/ set of characters stored on some secondary storage devices.
- Types of Files:
 - **Text file:** A text file stores information in readable and printable form. Each line of text is terminated with an EOL (End of Line) character.
 - **Binary file:** A binary file contains information in the non-readable form i.e. in the same format in which it is held in memory.
- We can read as well as write data in the file whenever required.
- File handling possible with stream classes. I/O system handles file more like input output with console.
- Stream which feeds data to program through file is known as file input stream.
- Stream which stores program data to file is known as output file stream.



b) Opening a File:

- A file must be opened before you can read from it or write to it.
- Either the ofstream or fstream object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only.
- Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.

void open(const char *filename, ios::openmode mode);

- Here, the first argument specifies the name and location of the file to be opened and the second argument of the open() member function defines the mode in which the file should be opened.

c) Closing a File:

- When a C++ program terminates it automatically closes flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.
- Following is the standard syntax for close() function, which is a member of fstream, ifstream, and ofstream objects.

void close();

d) Writing to a File:

- While doing C++ programming, you write information to a file from your program using the stream insertion operator (<<) just as you use that operator to output information to the screen. The only difference is that you use an ofstream or fstream object instead of the cout object.

e) Reading from a File:

- You read information from a file into your program using the stream extraction operator (>>) just as you use that operator to input information from the keyboard. The only difference is that you use an ifstream or fstream object instead of the cin object.

f) getline() Function:

- getline reads characters from an input stream and places them into a string:
 1. Behaves as UnformattedInputFunction. After constructing and checking the sentry object, performs the following:
 - a. Calls str.erase()
 - b. Extracts characters from input and appends them to str until one of the following occurs
 - i. end-of-file condition on input, in which case, getline sets eofbit.
 - ii. the next available input character is delim, as tested by Traits::eq(c, delim), in which case the delimiter character is extracted from input, but is not appended to str.
 - iii. str.max_size() characters have been stored, in which case getline sets failbit and returns.
 - c. If no characters were extracted for whatever reason (not even the discarded delimiter), getline sets failbit and returns.
 2. Same as getline(input, str, input.widen('\n')), that is, the default delimiter is the endline character.

g) get() function:

- The get() function reads a single character from the file and puts that value in ch. Following is the general syntax for get() function:

ifstream.get(char);

Algorithm:

1. Include header files iostream, fstream and cstdlib.
2. Define a function count with two parameters : first parameter as name of file to read and second as character to search as beginning of line
3. Open given file for reading
4. Check if file is opened correctly or generate error
5. Initialize counter to -1
6. Read current line of file and check it does not start with given character.
7. If not starting with given character, increment counter by 1 else continue.
8. Display current line
9. Repeat step 6 to 8 till end of file is encountered
10. Return value of counter
11. In main(), enter name of file to open for reading – parameter 1
12. Enter character to search as first character of every line in file – parameter 2
13. Call count function with these two parameters
14. Display value returned by count function.
15. End program

Flowchart: Draw proper flowchart and get it corrected from faculty.

Test Cases:

Test Case 1: (All valid inputs are given and desired output is shown)

Enter name of your file to read : story.txt

Enter character to check as first character of line : A

Contents of file are:

The roses are red.

A girl is playing there.

There is a playground.

An aeroplane is in the sky.

Numbers are not allowed in the password.

No of lines not starting with A are : 4

Test Case 2: (File opening error)

Enter name of your file to read : story.dat

Enter character to check as first character of line : A

Error reading file

Test Case 3: (No line starts with given character)

Enter name of your file to read : story.txt

Enter character to check as first character of line : Z

Contents of file are:

The roses are red.

A girl is playing there.

There is a playground.

An aeroplane is in the sky.

Numbers are not allowed in the password.

No of lines not starting with A are : 5

Conclusion:

Hence, we have demonstrated file operations and searching particular line in a file.

PROGRAM

```
#include<iostream>
#include<fstream>
#include<cstdlib>
using namespace std;
int count(string n,char ch)
{
    fstream file;
    file.open(n.c_str(),ios::in);
    if(!file)
    {
        cout<<"Error reading file\n";
        return(1);
    }
    int counter=-1;
    char data[80];
    cout<<"Contents of file are:\n";
    while(file)
    {
        file.getline(data,80);
        cout<<data<<endl;
        if(data[0] != ch)
            counter++;
    }
    return counter;
}
int main()
{
    string name;
    cout<<"Enter name of your file to read";
    cin>>name;
    char ch;
    cout<<"Enter character to check as first character of line\n";
    cin >> ch;
    int c=count(name,ch);
    cout<<"\nNo of lines not starting with "<<ch<<" are :"<< c<<endl;
    return 0;
}
```

OOPL – ASSIGNMENT 7

Title: Demonstrate exception handling through user defined exception class

Objectives:

- 1) To learn and understand exception handling
- 2) To demonstrate exception handling through user defined exception class.

Problem Statement:

Create User defined exception to check the following conditions and throw the exception if the criterion does not meet.

1. User has age between 18 and 55
2. User stays has income between Rs. 50,000 – Rs. 1,00,000 per month
3. User stays in Pune/ Mumbai/ Bangalore / Chennai
4. User has 4-wheeler

Accept age, Income, City, Vehicle from the user and check for the conditions mentioned above. If any of the condition not met then throw the exception

Outcomes:

- 1) Students will be able to learn and understand exception handling.
- 2) Students will be able to demonstrate exception handling by creating user defined class.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

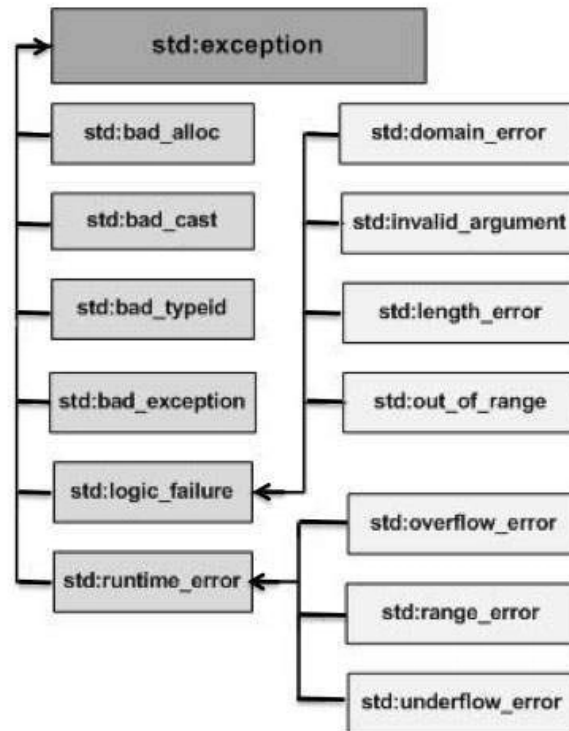
64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) Exception Handling

- An exception occurs when an unexpected error or unpredictable behaviors happened on your program not caused by the operating system itself.
- Those exceptions are handled by code which is outside the normal flow of control and it needs an emergency exit. C++ has incorporated three operators to help us handle these situations: try, throw and catch.
- C++ provides a list of standard exceptions defined in <exception> which we can use in our programs. These are arranged in a parent-child class hierarchy shown below:

Exception	Description
<code>std::exception</code>	An exception and parent class of all the standard C++ exceptions.
<code>std::bad_alloc</code>	This can be thrown by <code>new</code> .
<code>std::bad_cast</code>	This can be thrown by <code>dynamic_cast</code> .
<code>std::bad_exception</code>	This is useful device to handle unexpected exceptions in a C++ program
<code>std::bad_typeid</code>	This can be thrown by <code>typeid</code> .
<code>std::logic_error</code>	An exception that theoretically can be detected by reading the code.
<code>std::domain_error</code>	This is an exception thrown when a mathematically invalid domain is used
<code>std::invalid_argument</code>	This is thrown due to invalid arguments.
<code>std::length_error</code>	This is thrown when a too big <code>std::string</code> is created
<code>std::out_of_range</code>	This can be thrown by the <code>at</code> method from for example a <code>std::vector</code> and <code>std::bitset<>::operator[]()</code> .
<code>std::runtime_error</code>	An exception that theoretically can not be detected by reading the code.
<code>std::overflow_error</code>	This is thrown if a mathematical overflow occurs.
<code>std::range_error</code>	This is occurred when you try to store a value which is out of range.
<code>std::underflow_error</code>	This is thrown if a mathematical underflow occurs.



- The C++ Standard library provides a base class specifically designed to declare objects to be thrown as exceptions. It is called `std::exception` and is defined in the `<exception>` header. This class has a virtual member function called `what` that returns a null-terminated character sequence (of type `char *`) and that can be overwritten in derived classes to contain some sort of description of the exception.
- Other way is to create your own class without inheriting exception class and write your own methods to handle exception as below :

```

#include <iostream>
using namespace std;
class Error
{
public:
    Error(string s)
    {
        str=s;
    }
    void err_disp()
    {
        cout<<"Error message: "<<str<<endl;
    }
private:
    string str;
};

int main() {
    cout<<"Enter a number between 1 to 10";
    int num;
    cin>>num;
}
  
```

```

        try
        {
            if(num<1 || num>10)
                throw Error("Value is not between 1 to 10");
            else
                cout<<"You entered "<<num<<endl;
        }
        catch(Error e)
        {
            e.err_disp();
        }
        return 0;
    }
}

```

Algorithm:

1. Create a class Error with variables of type int, float and string to handle three different errors.
2. Define default constructor to initialize all variables.
3. Define parameterized constructor with an int value and assigned local variable to member variable
4. Define second constructor with float value and assigned local variable to float type member variable
5. Similarly, define next constructor of string variable type to assign its value to member variable of type string.
6. Define three different methods to display three different messages for all three different member variables.
7. In main method, accept age, income, city and vehicle type from user.
8. If age is not between 18 to 55, generate exception and use user defined Error class to catch int type of exception and display relevant message.
9. If income is not between 50000 to 100000, generate exception and use user defined Error class to catch float type of exception and display relevant message.
10. If city is other than Pune, Mumbai, Chennai or Bangalore; generate exception and catch string type of exception to display relevant message.
11. If vehicle type is entered other than 4-wheeler; generate exception and catch string type of exception to display relevant message.

Flowchart: Draw proper flowchart and get it corrected from faculty.

Test Cases:

Enter Age (between 18 to 55): 18

Age is : 18

Enter monthly income in INR (between 50000 to 100000) : 66666

Monthly income in INR is :66666

Enter City (pune or mumbai or bangalore or chennai): pune

City is : pune

Enter type of vehicle (2-wheeler/4-wheeler):

Vehicle is : 4-wheeler

Test Case 2:

Enter Age (between 18 to 55):68
Invalid age : 68
Enter monthly income in INR (between 50000 to 100000) :25000
Invalid income : 25000
Enter City (pune or mumbai or bangalore or chennai):nasik
City not allowed; Invalid Vallue: nasik
Enter type of vehicle (2-wheeler/4-wheeler):2-wheeler
Invalid Vehicle;Invalid Vallue: 2-wheeler

Conclusion:

Hence, we learnt used defined exception handling and demonstrated user defined class to handle exception of different types.

PROGRAM

```
#include<iostream>
using namespace std;
class Error
{
    int eage;;
    float eincome;
    string estr;
public:
    Error(int a){
        eage=a;
    }
    Error(float i){
        eincome=i;
    }
    Error(string s){
        estr=s;
    }
    void err_age(){
        cout<<"Invalid age : "<<eage<<endl;
    }
    void err_income(){
        cout<<"Invalid income : "<<eincome<<endl;
    }
    void err_str(){
        cout<<"Invalid Vallue: "<<estr<<endl;
    }
};
int main()
{
    int age;
    float income;
    string city,vehicle;
    try{
        cout<<"Enter Age (between 18 to 55):\n";
        cin>>age;
        if(age<18 | age>55)
            throw Error(age);
        else
```



```

        cout<<"Age is : "<<age<<endl;
    }
    catch(Error e) {
        e.err_age();
    }
    try {
        cout<<"Enter monthly income in INR (between 50000 to 100000) :\n";
        cin>>income;
        if(income<50000 || income>100000)
            throw Error(income);
        else
            cout<<"Monthly income in INR is : "<<income<<endl;
    }
    catch(Error e) {
        e.err_income();
    }
    try{
        cout<<"Enter City (pune or mumbai or bangalore or chennai):\n";
        cin>>city;
        if((city != "pune") || (city != "mumbai") || (city != "bangalore") || (city !=
"chennai"))
            throw Error(city);
        else
            cout<<"City is : "<<city<<endl;
    }
    catch(Error e) {
        cout<<"\nCity not allowed; ";
        e.err_str();
    }
    try {
        cout<<"Enter type of vehicle (2-wheeler/4-wheeler):\n";
        cin>>vehicle;
        if(vehicle != "4-wheeler")
            throw Error(vehicle);
        else
            cout<<"Vehicle is : "<<vehicle<<endl;
    }
    catch(Error e) {
        cout<<"\nInvalid Vehicle;";
        e.err_str();
    }
}

```

*****Output*****

Enter Age (between 18 to 55):68

Invalid age : 68

Enter monthly income in INR (between 50000 to 100000) :25000

Invalid income : 25000

Enter City (pune or mumbai or bangalore or chennai):nasik

City not allowed; Invalid Vallue: nasik

Enter type of vehicle (2-wheeler/4-wheeler):2-wheeler

Invalid Vehicle;Invalid Vallue: 2-wheeler

OOPL – ASSIGNMENT 8

Title: Demonstrate file handling operations to create phonebook.

Objectives:

- 1) To learn and understand file handling functions.
- 2) To demonstrate random accessing file.

Problem Statement:

Write a menu driven program that will create a data file containing the list of telephone numbers in the following form

John 23456

Ahmed 9876

.....

Use a class object to store each set of data, access the file created and implement the following tasks

- a. Determine the telephone number of specified person
- b. Determine the name if telephone number is known
- c. Update the telephone number, whenever there is a change.

Outcomes:

- 1) Students will be able to learn and understand file handling operations.
- 2) Students will be able to demonstrate random accessing file.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

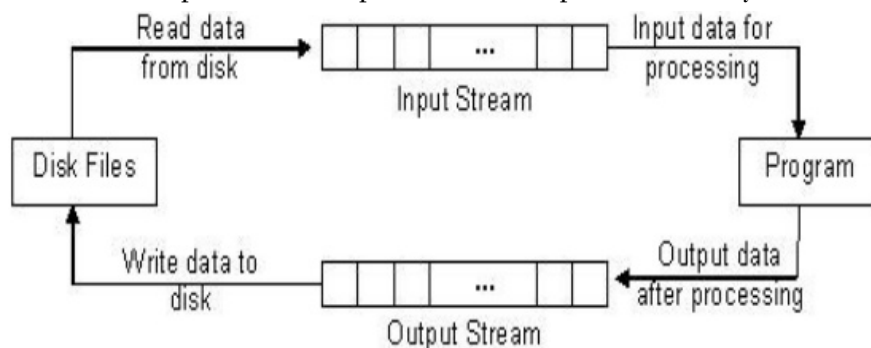
Software requirements:

64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) File

- A file is a stream of bytes/ set of characters stored on some secondary storage devices.
- Types of Files:
 - **Text file:** A text file stores information in readable and printable form. Each line of text is terminated with an EOL (End of Line) character.
 - **Binary file:** A binary file contains information in the non-readable form i.e. in the same format in which it is held in memory.
- We can read as well as write data in the file whenever required.
- File handling possible with stream classes. I/O system handles file more like input output with console.
- The fstream class can provide both input as well as output functionality.



b) Opening a File:

- A file must be opened before you can read from it or write to it.
- Either the ofstream or fstream object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only.

- Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.

```
void open(const char *filename, ios::openmode mode);
```

- Here, the first argument specifies the name and location of the file to be opened and the second argument of the open() member function defines the mode in which the file should be opened.

c) Closing a File:

- When a C++ program terminates it automatically closes flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.
- Following is the standard syntax for close() function, which is a member of fstream, ifstream, and ofstream objects.

```
void close();
```

d) Working on Binary Files

- A binary file contains information in the non-readable form i.e. in the same format in which it is held in memory.
- To achieve operations on binary files the file must be opened in ios::binary mode.

• Binary file functions:

1. read()

- read a block of binary data or reads a fixed number of bytes from the specified stream and store in a buffer.
- Syntax: Stream_object.read((char *)& Object, sizeof(Object));
- Example: filestream.read((char *)&s, sizeof(s));

2. write()

- write a block of binary data or writes fixed number of bytes from a specific memory location to the specified stream.
- Syntax : Stream_object.write((char *)& Object, sizeof(Object));
- Example: filestream.write((char *)&s, sizeof(s));

• Note: Both functions take two arguments.

- a. Address of the variable-(must be cast to type char *(i.e. pointer to character type)).
- b. Length of that variable in bytes.

e) File Pointers and Manipulators:

- Each file has two pointers known as file pointers, one is called the input pointer and the other is called output pointer.
- The input pointer is used for reading the contents of of a given file location and the output pointer is used for writing to a given file location.
- Each time an input or output operation takes place, the appropriate pointer is automatically advanced.

f) Functions for Manipulations of File pointer:

- All the actions on the file pointers takes place by default.
- For controlling the movement of file pointers file stream classes support the following functions
 - a) seekg() Moves get pointer (input)to a specified location.
 - b) seekp() Moves put pointer (output) to a specified location.
 - c) tellg() Give the current position of the get pointer.
 - d) tellp() Give the current position of the put pointer.

For example, the statement seekg(10); moves the pointer to the byte number 10.The bytes in a file are numbered beginning from zero. Therefore ,the pointer to the 11 byte in the file.

Following table shows the file pointer manipulation at various stages:

Function	Description
seekg(long)	moves get pointer (input) to a specified location.
seekp(long)	moves put pointer (output) to a specified location.
tellg()	gives the current position of the get pointer.
tellp()	gives the current position of the put pointer.
fout . seekg(0, ios :: beg)	go to start
fout . seekg(0, ios :: cur)	stay at current position
fout . seekg(0, ios :: end)	go to the end of file
fout . seekg(m, ios :: beg)	move to m+1 byte in the file
fout . seekg(m, ios :: cur)	go forward by m bytes from the current position
fout . seekg(-m, ios :: cur)	go backward by m bytes from the current position
fout . seekg(-m, ios :: end)	go backward by m bytes from the end

Algorithm: Write appropriate algorithm/ pseudocode & get it corrected from faculty.

Flowchart: Draw proper flowchart and get it corrected from faculty.

Test Cases:

Conclusion:

Hence, we have studied and demonstrated file operations and randomly accessing file for performing various file operations.

PROGRAM:

```
#include<iostream>
#include<fstream>
#include<iomanip>
#include<cstring>
using namespace std;
class Telephone {
    char na[20],tel[15];
    public:
        void getdata() {
            cout<<"\n Enter the Telephone details:";
            cout<<"\n Name:";
            cin>>na;
            cout<<"\n Telephone Number:";
            cin>>tel;
        }
        void putdata() {
            cout<<"\n"<<left<<setw(20)<<na;
            cout<<left<<setw(15)<<tel;
        }
        char *getName(){    return na;    }
        char *gettel(){    return tel;    }
        void update(char *nam,char *telno){
```

```

        strcpy(nam,na);
        strcpy(tel,telno);
    }
};

int main() {
    char telno[15],name[20],ch;
    int cho,flag=0,cnt=0;
    fstream f;
    Telephone t;
    f.open("TData.dat",ios::out | ios::in | ios::binary);
    do{
        cout<<"\n Telephone Directory";
        cout<<"\n 1. Create Record";
        cout<<"\n 2. Display All Records";
        cout<<"\n 3. Search by Name";
        cout<<"\n 4. Search by Telephone Number";
        cout<<"\n 5. Update Telephone Number";
        cout<<"\n 6. Exit";
        cout<<"\n Enter your choice:";
        cin>>cho;
        switch(cho){
            case 1: t.getdata();
                    f.write((char*)&t,sizeof(t));
                    break;
            case 2: cout<<"\n All Telephone Records are:";
                    cout<<"\n  Name \t\tTelephoneNumber";
                    f.seekg(0,ios::beg);
                    for(f.read((char*)&t,sizeof(t));!f.eof();f.read((char*)&t,sizeof(t)))
                        t.putdata();
                    f.clear();
                    break;
            case 3: cout<<"\n Enter Name for Searching:";
                    cin>>name;
                    f.seekg(0,ios::beg);
                    for(f.read((char*)&t,sizeof(t));!f.eof();f.read((char*)&t,sizeof(t))) {
                        if(strcmp(name,t.getName())==0) {
                            cout<<"\n All Telephone Records are:";
                            cout<<"\n  Name \t\tTelephoneNumber";
                            t.putdata();
                            flag=1;
                        }
                    }
                    f.clear();
                    if(flag==0)
                        cout<<"\n Record not Found.";
                    break;
            case 4: cout<<"\n Enter Telephone Number for Searching:";
                    cin>>telno;
                    f.seekg(0,ios::beg);
                    flag=0;
                    for(f.read((char*)&t,sizeof(t));!f.eof();f.read((char*)&t,sizeof(t)))
                    {

```

```

        if(strcmp(telno,t.gettel())==0) {
            cout<<"\n All Telephone Records are:";
            cout<<"\n  Name \t\tTelephoneNumber";
            t.putdata();
            flag=1;
        }
    }
    f.clear();
    if(flag==0)
        cout<<"\n Record not Found.";
    break;
case 5: cout<<"\n Enter Name for Updation:";
    cin>>name;
    flag=0;
    f.seekg(0,ios::beg);
    for(f.read((char*)&t,sizeof(t));!f.eof();f.read((char*)&t,sizeof(t))) {
        cnt++;
        if(strcmp(name,t.getName())==0) {
            flag=1;
            break;
        }
    }
    f.clear();
    if(flag==0)
        cout<<"\n Record not Found.";
    else
    {
        int loc=(cnt-1)*sizeof(t);
        if(f.eof())
            f.clear();
        cout<<"Enter New Telephone No : ";
        cin>>telno;
        f.seekp(loc);
        t.update(name,telno);
        f.flush();
        f.write((char *) &t, sizeof(t));
    }
    break;
case 6: return 0;
default:cout<<"\n Enter valid Choice.";
    break;
}
cout<<"\n Do you want to continue(y/n):";
cin>>ch;
}while(ch=='Y' || ch=='y');
return 0;
}

```

Output

Telephone Directory

1. Create Record
2. Display All Records
3. Search by Name
4. Search by Telephone Number
5. Update Telephone Number
6. Exit

Enter your choice:2

All Telephone Records are:

Name	Telephone Number
Kailas	9621362321
Rocky	8422332211
Anil	9421913477
Raj	9422650021
Kuldeep	9421967989

Do you want to continue(y/n):y

Telephone Directory

1. Create Record
2. Display All Records
3. Search by Name
4. Search by Telephone Number
5. Update Telephone Number
6. Exit

Enter your choice:3

Enter Name for Searching: Kuldeep

All Telephone Records are:

Name	Telephone Number
Kuldeep	9421967989

Do you want to continue(y/n):y

Telephone Directory

1. Create Record
2. Display All Records
3. Search by Name
4. Search by Telephone Number
5. Update Telephone Number
6. Exit

Enter your choice:4

Enter Telephone Number for Searching:9421913477

All Telephone Records are:

Name	Telephone Number
Anil	9421913477

Do you want to continue(y/n):n

OOPL – ASSIGNMENT 9

Title: Demonstration of command line arguments for finding and replacing string in a file.

Objectives:

- 1) To learn and understand command line arguments and their use.
- 2) To demonstrate string functions and command line arguments.

Problem Statement:

Write a C++ program using command line arguments to search for a word in a file and replace it with the specified word. The usage of the program is shown below.

\$ change <file name> <old word> <new word>

Outcomes:

- 1) Students will be able to learn and understand command line arguments
- 2) Students will be able to demonstrate command line arguments and some string functions for finding and replacing contents in a file.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) Command Line Argument:

- Most important function of C/C++ is main() function. It is mostly defined with a return type of int and without parameters :

```
int main() { /* ... */ }
```

- Similar to other functions, main() can also take arguments. These arguments are known as command line arguments and they are passed to the program at runtime.

- To pass command line arguments, we typically define main() with two arguments as :

```
int main(int argc, char *argv[]) { /* ... */ }
```

or

```
int main(int argc, char **argv) { /* ... */ }
```

- a) argc (ARGument Count) is int and stores number of command-line arguments passed by the user including the name of the program. It should be positive value.
- b) argv(ARGument Vector) is array of character pointers listing all the arguments. If argc is greater than zero, Then array elements from argv[0] to argv[argc-1] will contain pointers to strings.

- Consider the following example:

```
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
    cout << "You have entered " << argc << " arguments:" << "\n";
    for (int i = 0; i < argc; ++i)
        cout << argv[i] << "\t";
    return 0;
}
```

- Assuming we are on linux system and the program is saved as prog.cpp

Compile : g++ -o prog prog.cpp

Executable is generated with name **prog**

Run : ./prog Hello World

where the command line arguments are “./prog” , “Hello” and “World”

Output :

You have entered 3 arguments:

./prog Hello World

b) String Functions:

I. find() :

- `std::string::find()` is a public member function of `string` class.
- You can use the `find()` function to locate the first occurrence of a string, a character array, or a character. If the target is not found, the functions return `string::npos`.
- These functions are useful in a situation where you need to parse a string which contains some sort of delimiter.
- Syntax:

```
int find(const string & s, int pos = 0) const;
int find(const char *cs, int pos = 0) const;
int find(char c, int pos = 0) const;
```
- The first function searches the invoking object for `s` starting at index `pos` and returns the index where `s` is first located.
- The second one searches invoking object for `cs` starting at index `pos` and returns the index where `cs` is first located.
- The third one searches invoking object for `c` starting at index `pos` and returns the index where `c` is first located.
- As noted before, if the target is not found, all functions return `string::npos`.
- Here are some examples:

```
string s = "Ja yo Afu! Yo yo yo!";
string s2 = "yo"; char *s3 = "you";
char c = '\\';
unsigned int temp; /* string::npos is a big number */
temp = s.find(s2); /* temp is 3 */
temp = s.find(s2, 4); /* temp is 14 */
temp = s.find(s3); /* temp is string::npos */
temp = s.find(s3, 6); /* temp is string::npos */
temp = s.find(c); /* temp is 9 */
temp = s.find(c, 10); /* temp is 19 */
```

II. replace():

- `std::string::replace()` is a public member function of `string` class.
- It replaces the portion of the string that begins at character `pos` and spans `len` characters (or the part of the string in the range between `[il,i2)`) by new contents:
- You can replace a substring of a string with a given string or char array by using the `replace()` function! There is a variety of the `replace()` function constructors. Here are their prototypes:

```
string & replace(int pos, int n, const string & s);
string & replace(int pos, int n, const char* cs);
```
- The first function replaces with `s` from index `pos` spanning `n` characters of the invoking string.
- The second one replaces with `cs` contents from index `pos` spanning `n` characters.
- Here are several examples:

```
string a = "Best Worst";
string b = "wishes to Demon";
a.replace(5, 5, b); /* a is "Best wishes to Demon" */
a.replace(15, 5, c); /* a is "Best wishes to Lali" */
```

Algorithm:

- 1) Include required header files like `iostream`, `fstream`(for file operations), `cstdlib` (for `exit` function), `string.h` (for string functions)
- 2) Define main method with parameters `argc` and `argv` as `int main(int argc, char *argv[])`

- 3) Check if number of parameters passed on command line are not less than 4 (i.e. value of argc variable)
- 4) If yes generate error and exit program
- 5) Otherwise, open file having name given as first argument on command line.
- 6) Display error and exit if not able to open file for reading.
- 7) Display contents of file on command prompt.
- 8) Use getline() to read first line from file.
- 9) Use find() method of string class to check first occurrence of string_to_find argv[2].
- 10) If not available, read next line from file.
- 11) If string_to_find exists, use replace() method to replace it with string_to_replace argv[3].
- 12) To check, next occurrence of string_to_find in same line, use find() method to start searching from next index position.
- 13) Concatenate line (updated/ not updated) in a temporary string.
- 14) Repeat step 8 to step 13 till end of file.
- 15) Close the file
- 16) Open same file in write mode and check if no error.
- 17) If error, display error message and exit the program.
- 18) Otherwise write contents of temporary string in file.
- 19) Close the file
- 20) Open same file in read mode and check if no error.
- 21) If error, display error message and exit the program.
- 22) Otherwise read contents of file and display on command line.
- 23) Close the file
- 24) Exit program

Flowchart: Draw proper flowchart and get it corrected from faculty.

Test Cases:

Test Case 1: (Enter filename string_to_find string_to_replace on command line)

```
$ g++ -o cmdline commandline.cpp
$ ./cmdline trial.txt
```

Output will be error message : Insufficient number of arguments on command line

Test Case 2: (Enter filename string_to_find string_to_replace on command line)

```
$ g++ -o cmdline commandline.cpp
$ ./cmdline trial.txt hi bye
```

Output will be error message if file does not exists : Error opening file

Test Case 3: (Enter filename string_to_find string_to_replace on command line)

```
$ g++ -o cmdline commandline.cpp
$ ./cmdline trial.txt hi bye
```

Output will be : Old contents of file will be displayed and then new contents after replacement will be shown. Program replaces all occurrences of existing old string with new string.

Conclusion:

Hence, we learnt passing command line arguments and use them to perform find and replace operation on given file.

PROGRAM:

```
#include<iostream>
#include<fstream>
#include<string.h>
using namespace std;
int main(int argc,char *argv[])
{
```

```

        if(argc<4){
            cout<<"\n\t Insufficient number of arguments from command line.";
            return 1;
        }
        string data,newdata;
        fstream fin;
        cout<<"\n\t Find String "<<argv[2]<<" and replace with "<<argv[3];
        cout<<"\n\t *** Contents of File Before Replacing ***";
        fin.open(argv[1],ios::in);
        if(!fin){
            cout<<"\n\t Unable to open a file.";
            return 1;
        }
        while(fin){
            getline(fin,data);
            cout<<data<<endl;
        }
        fin.close();
        fin.open(argv[1],ios::in);
        newdata="";
        if(!fin) {
            cout<<"\n\t Unable to open a file.";
            return 1;
        }
        while(fin){
            getline(fin,data);
            int b;
            int a=(int) data.find(argv[2],0);
            while(a!=string::npos){
                data.replace(a,strlen(argv[2]),argv[3]);
                a=(int)data.find(argv[2],(a+1));
            }
            newdata=newdata+data+"\n";
        }
        fin.close();
        fin.open(argv[1],ios::out);
        fin<<newdata;
        cout<<"\n\t Replacement Done Successfully.";
        fin.close();
        cout<<"\n\t *** Contents of File After Replacing ***";
        fin.open(argv[1],ios::in);
        if(!fin) {
            cout<<"\n\t Unable to open a file.";
            return 1;
        }
        while(fin){
            getline(fin,data);
            cout<<data<<endl;
        }
        fin.close();
        return 0;
    }
}

```

OUTPUT

Find string “you” and replace with “me” in file “trial.txt”

*****Contents of file before replacing*****

hi

how are you?

not you!

yes you

whats going on?

i m not getting anything

everything is you you only

I am asking you

Replacement successful

*****Contents of file after replacing*****

hi

how are me?

not me!

yes me

whats going on?

i m not getting anything

everything is me me only

I am asking me

OOPL – ASSIGNMENT 10

Title: Demonstrate function template for selection sorting algorithm.

Objectives:

- 1) To learn and understand templates.
- 2) To demonstrate function template for selection sort.

Problem Statement:

Write a function template selection Sort. Write a program that inputs, sorts and outputs an integer array and a float array.

Outcomes:

- 1) Students will be able to learn and understand working and use of function template.
- 2) Students will be able to demonstrate function template for selection sort.

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

64-bit Linux/Windows Operating System, G++ compiler

Theory:

a) Template

- Template allows a function or class to work on many different data types without being rewritten for each one. Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.
- A template is a blueprint or formula for creating a generic class or a function.
- The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept.
- There is a single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector <int> or vector <string>.
- You can use templates to define functions as well as classes, let us see how do they work:
- Function Template: The general form of a template function definition is shown here:

```
template <typename type> ret-type func-name(parameter list)
{
    // body of function
}
```

- Here, type is a placeholder name for a data type used by the function. This name can be used within the function definition.
- A function template behaves like a function except that the template can have arguments of many different types (see example). In other words, a function template represents a family of functions. The format for declaring function templates with type parameters is:

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

- Both expressions have the same meaning and behave in exactly the same way. The latter form was introduced to avoid confusion, since a type parameter need not be a class. (it can also be a basic type such as int or double.)
- For example, the C++ Standard Library contains the function template max(x, y) which returns the larger of x and y. That function template could be defined like this:

```
template <typename T>
inline T max(T a, T b) {
    return a > b ? a : b;
}
```

- This single function definition works with many data types. The usage of a function template saves space in the source code file in addition to limiting changes to one function description and making the code easier to read.

b) Selection Sort

- Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison based algorithm in which the list is divided into two parts, sorted part at left end and unsorted part at right end. Initially sorted part is empty and unsorted part is entire list.
- Smallest element is selected from the unsorted array and swapped with the leftmost element and that element becomes part of sorted array. This process continues moving unsorted array boundary by one element to the right.
- This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where n are no. of items.

Step 1 – Set MIN to location 0

Step 2 – Search the minimum element in the list

Step 3 – Swap with value at location MIN

Step 4 – Increment MIN to point to next element

Step 5 – Repeat until list is sorted

Algorithm:

- 1) Define a function template with name `input()` and two arguments, one argument for array to sort and other for size of array
- 2) Accept size number of elements from user for sorting in `input()`
- 3) Define other function template `display()` with two arguments, one pointer of array to sort and other number of elements to sort
- 4) Print sorted elements stored in array
- 5) Define another function template `sort()` to implement selection sort for generic datatype.
- 6) Declare a variable `min` to hold index position of smallest element in array to sort.
- 7) Define for loop with loop variable “`i`” from 0 to size, incremented by one.
- 8) Assign first index position to `min` variable i.e. 0.
- 9) Define second for loop with variable “`j`” starting from “`i + 1`” to size, incremented by one.
- 10) In inner for loop, check if element at index position “`j`” is smaller than element at index position “`min`”.
- 11) If condition is true, assign `min` as `j` else continue
- 12) In outer for loop, swap element at index position “`i`” with element at index position at “`min`”.
- 13) In `main()` method, declare variable `size` to accept size of array.
- 14) Input number of elements to sort and declare an array of given size.
- 15) Use `int` as datatype of array
- 16) Call `input()` function and pass array name and size as argument
- 17) Call `sort()` function and pass array name and size as argument to sort
- 18) Call `display()` function and pass array name and size as argument to display sorted array.
- 19) Repeat steps from 15 to 19 with datatype `float`
- 20) End program.

Flowchart: Draw proper flowchart and get it corrected from faculty.

Test Cases:

Test Case 1: (All valid inputs are given and desired output is shown)

How many elements you want to sort : 5

Enter 5 elements in array : 10 32 2 -32 11

Elements before sorting : 10 32 2 -32 11

Elements after sorting : -32 2 10 11 32

Enter 5 elements in array : 1.30 3.2 4.2 -4.32 1.1

Elements before sorting : 1.30 3.2 4.2 -4.32 1.1

Elements after sorting : -4.32 1.1 1.30 3.2 4.2
Test Case 2: (When size is given as negative value)
How many elements you want to sort : -4
Array size can not be negative. Initializing size to 1
Enter 1 elements in array : 10
Elements before sorting : 10
Elements after sorting : 10
Enter 1 elements in array : 1.1
Elements before sorting : 1.1
Elements after sorting : 1.1

Conclusion:

Hence, we demonstrated use of function template for selection sort

PROGRAM

```
/*Write a function template selection Sort. Write a program that inputs, sorts and outputs an
integer array and a float array. */
#include<iostream>
using namespace std;
template<typename T>
void input(T *a, int size) {
    cout<<"\nEnter "<<size<<" elements in array";
    for(int i=0;i<size;i++)
        cin>>a[i];
}
template<typename T>
void sorting(T *a, int size) {
    int min;
    for(int i=0;i<size;i++){
        min=i;
        for(int j=i+1;j<size;j++){
            if(a[j]<a[min])
                min=j;
        }
        swap(a[i],a[min]);
    }
}
template<typename T>
void display(T *a, int size){
    for(int i=0;i<size;i++)
        cout<<a[i]<<"\t";
}
int main()
{
    cout<<"\nHow many elements you want to sort\n";
    int size;
    cin>>size;
    if(size < ) {
        cout<<"Array size can not be negative. Initializing size to 1";
        size=1;
    }
    int a[size];
```

```
    input(a,size);
    cout<<"\nElements before sorting\n";
    display(a,size);
    cout<<"\nElements after sorting\n";
    sorting(a,size);
    display(a,size);
    float b[size];
    input(b,size);
    cout<<"\nElements before sorting\n";
    display(b,size);
    cout<<"\nElements after sorting\n";
    sorting(b,size);
    display(b,size);
    return 0;
}
```