**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**OBJECT ORIENTED PROGRAMMING (SE-203)**

**SE-A1-G2 LAB**



**DELHI TECHNOLOGICAL UNIVERSITY(DTU)**

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Bawana Road, Delhi-110042

Submitted by:-

Name:- ASHISH KUMAR

Roll number:- 2K18/SE/041

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| ***S.No.*** | ***PROGRAMES*** | ***DATE*** | ***SIGN.*** |
| **1.** |  |  |  |
| **2.** |  |  |  |
| **3.** |  |  |  |
| **4.** |  |  |  |
| **5.** |  |  |  |
| **6.** |  |  |  |
| **7.** |  |  |  |
| **8.** |  |  |  |
| **9.** |  |  |  |
| **10.** |  |  |  |
| **11.** |  |  |  |

**PROGRAM 1**

**Aim:-**To write a program to multiply two numbers using class.

**Description:-**

**Class:** The building block of C++ that leads to Object Oriented programming is a **Class**. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

An **Object** is an instance of a Class.

In this program, we take inputs(two numbers) from user by function named as get\_data() and display the output using function named as display\_result() .

**CODE:-**

#include <iostream>

#include <string>

using namespace std;

class multiply {

int val1,val2;

public:

int get\_data();

void display\_result(void);

};

int multiply :: get\_data()

{

cout << "enter two numbers :";

cin >> val1>> val2;

cout << "\n The Product is :" << (val1\*val2);

}

void multiply :: display\_result(void) {

int val=0;

val = get\_data();

}

int main() {

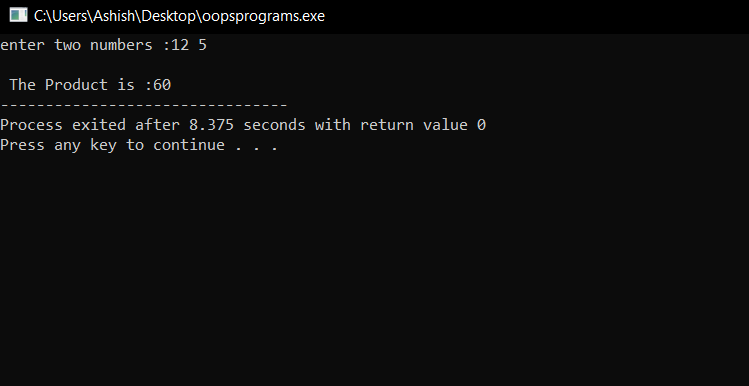
multiply numbers; //object creation

numbers.display\_result();

return 0;

}

**OUTPUT:-**

****

**PROGRAM 2**

**Aim:-** To write a program to demonstrate types of constructors in C++.

**Description:-** A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.

* Constructor has same name as the class itself
* Constructors don’t have return type, not even void.

In this program, we are going to learn about different types of constructors in C++, which r as follows:

1. [**Default Constructors:**](https://www.geeksforgeeks.org/c-internals-default-constructors-set-1/) Default constructor is the constructor which doesn’t take any argument. It has no parameters.
2. **Parameterized Constructors: that takes argument.** It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor’s body, use the parameters to initialize the object.
3. **Copy Constructor:** A copy constructor is a member function which initializes an object using another object of the same class.

**CODE:-**

#include <iostream>

using namespace std;

class Point {

private:

int x, y;

public:

// Parameterized Constructor

Point(int x1, int y1)

{

x = x1;

y = y1;

}

int getX()

{

return x;

}

int getY()

{

return y;

}

// Default Constructor

int a, b;

Point()

{

a = 10;

b = 20;

}

// Copy constructor

Point(const Point &p2) {

x = p2.x; y = p2.y;

}

};

int main()

{

Point p1(10, 15);

Point p;

Point p2 = p1; // Copy constructor is called here

cout<<"\*\*\*\*Default constructor example\*\*\*\*"<<endl;

cout << "a: " << p.a << endl

<< "b: " << p.b;

cout<<endl;

cout<<"\n\*\*\*\*Parametrized constructor example\*\*\*\*"<<endl;

cout << "x = " << p1.getX() << ", y = " << p1.getY();

cout<<"\n\n\*\*\*\*Copy constructor example\*\*\*\*"<<endl;

cout << "x = " << p2.getX() << ", y = " << p2.getY();

return 0;

}

**OUTPUT:-**



**PROGRAM 3**

**Aim:-**To write a program to overload unary operator and binary operator.

**Description:-** Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.

**CODE:-**

//binary operator overloading

#include<iostream>

#include<cstring>

using namespace std;

class String

{

char str[20];

public:

void input()

{

cout<<"\nEnter your string: ";

cin.getline(str,20);

}

void display()

{

cout<<"\nString after concatenation : "<<str;

}

String operator + (String s) //overloading

{

String obj;

strcat(str,s.str);

strcpy(obj.str,str);

return obj;

}

};

int main()

{

String str1,str2,str3;

str1.input();

str2.input();

str3=str1+str2;

str3.display();

return 0;

}

**OUTPUT:-**



// unary operator overloading

#include<iostream>

#include<cstring>

using namespace std;

class unary

{

int x;

int y;

int z;

public:

void getdata(int a,int b,int c) {

x=a;y=b;z=c;

}

void display(void) {

cout<<x<<" ";

cout<<y<<" ";

cout<<z<<" "<<endl;

}

void operator-();

};

void unary::operator-()

{

x=-x;

y=-y;

z=-z;

}

int main()

{

unary u;

u.getdata(10,-20,50);

cout<<"U: ";

u.display();

-u; //activates operator-() function

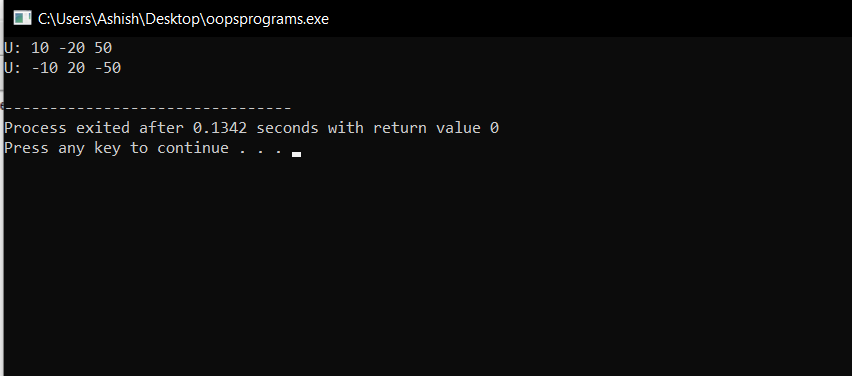
cout<<"U: ";

u.display();

return 0;

}

**OUTPUT:-**



**PROGRAM 4**

**Aim:-** To write a program to add two numbers using inline function.

**Description:-** C++ provides an inline functions to reduce the function call overhead. Inline function is a function that is expanded in line when it is called. When the inline function is called, whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. Inline function may increase efficiency if it is small.

To make any function as inline, start its definitions with the keyword “inline”.

**CODE:-**

#include<iostream.h>

#include<conio.h>

class summation {

int a,b,sum;

public:

void getdata();

void add();

};

inline void summation::getdata() {

cout<<"enter value of a & b:"<<endl;

cin>>a>>b;

}

inline void summation::add() {

sum=a+b;

cout<<"the sum of two no.is:-"<<sum<<endl;

}

void main()

{

summation s;

s.getdata();

s.add();

getch();

}

**OUTPUT:-**

****

**PROGRAM 5**

**Aim:-**To write a program using static keyword to demonstrate static member function.

**Description:-** Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime.

Static data members of class are those members which are shared by all the objects. Static data member has a single piece of storage, and is not available as separate copy with each object, like other non-static data members.

Static member variables (data members) are not initialized using constructor, because these are not dependent on object initialization.

**A static member function is a special member function, which is used to access only static data members**, any other normal data member cannot be accessed through static member function. Just like static data member, static member function is also a class function; it is not associated with any class object.

We can access a static member function with class name, by using following syntax:

class\_name:: function\_name(perameter);

**CODE:-**

#include<iostream>

using namespace std;

#define PI 3.14

class A

{

private:

int m,n;

static int a,b; //static variables

public:

static void showdata(int x,int y) //with static

{

a=x; b=y;

cout<<"Area of ellipse = "<<PI\*a\*b<<endl;

}

void show\_data(int n,int m) //without static

{

a=n;

b=m;

cout<<"Area of ellipse is : "<<PI\*a\*b<<endl;

}

};

int A::a;

int A::b;

int main()

{

A::showdata(3,4);

A t;

t.showdata(5,6);

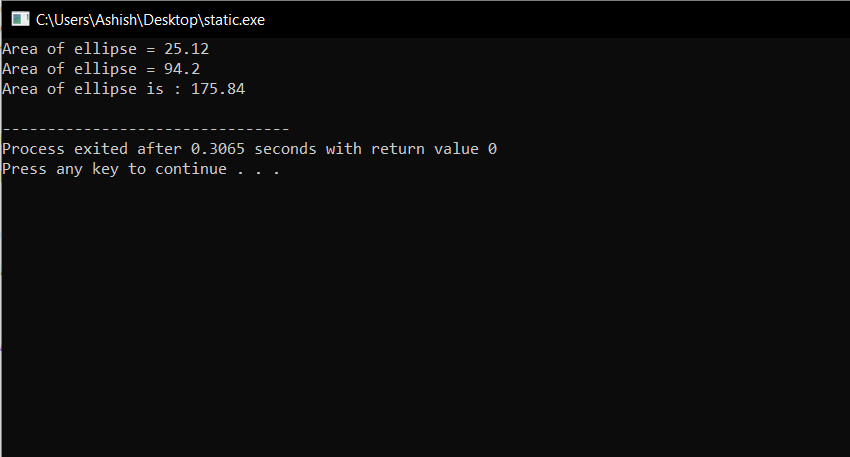
A n;

n.show\_data(5,8);

return 0;

}

**OUTPUT:-**



**PROGRAM 6**

**Aim:-**To write a program to access private members of a class using friend function.

**Description:-**

If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword “friend”.

**Characteristics of a Friend function:**

* The function is not in the scope of the class to which it has been declared as a friend.
* It cannot be called using the object as it is not in the scope of that class.
* It can be invoked like a normal function without using the object.
* It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
* It can be declared either in the private or the public part.

**CODE:-**

#include <iostream>

using namespace std;

class summation {

private:

int x, y,sum;

public:

friend void display(summation s);

void sum1(int x,int y);

};

void summation::sum1(int x,int y)

{

sum=x+y;

}

void display(summation s) {

cout<<"sum of two nos is: "<<s.sum<<endl;

}

int main()

{

summation s;

int a,b;

cout<<"enter two numbers :"<<endl;

cin>>a>>b;

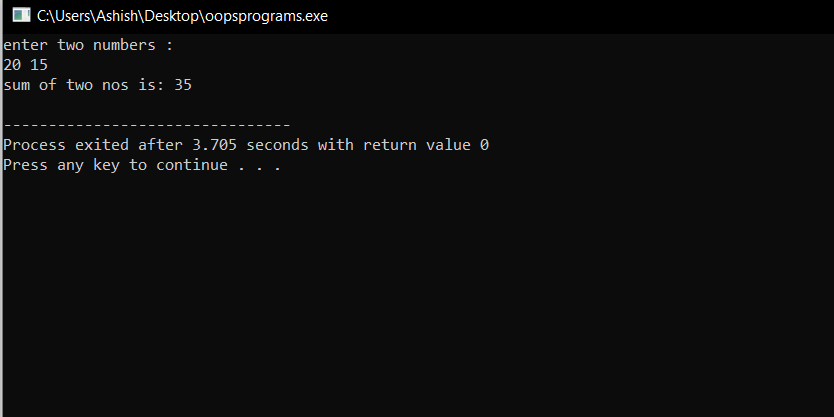
s.sum1(a,b);

display(s);

return 0;

}

**OUTPUT:-**



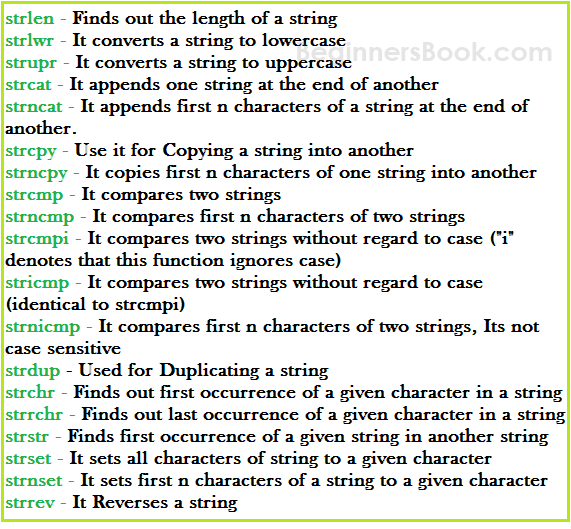
**PROGRAM 7**

**Aim:-**To write a program to do various String operations using built-in functions.

**Description:-**

String is a sequence of characters. char data type is used to represent one single character in C++.Any string ends with a terminating null character '\0'

**C++ various built-in String functions**



**CODE:-**

#include <iostream>

#include <cstring>

using namespace std;

int main () {

char str1[50],str2[50];

cout<<"Enter first string :";

cin>>str1;

cout<<"Enter second string :";

cin>>str2;

int n=strlen(str1);

cout<<"Length of 1st string is : "<<n<<endl;

int n2=strlen(str2);

cout<<"Length of 2nd string is : "<<n2<<endl;

if (strcmp(str1, str2) ==0)

{

printf("string 1 and string 2 are equal");

}else

{

printf("string 1 and 2 are different");

}

strcat(str1,str2);

cout<<"\n\nOutput string after concatenation: "<< str1<<endl;

cout<<"\nstring 1 after converted into UPPERCASE: ";

cout<<"\n"<< strupr (str1);

cout<<"\n\nThe given string is : \n"<<str1<<endl;

cout<<"\nAfter reversing ,string is ="<<strrev(str1)<<endl;

cout<<"\nModified String: \n"<<strnset(str1, '\*', 4)<<endl;

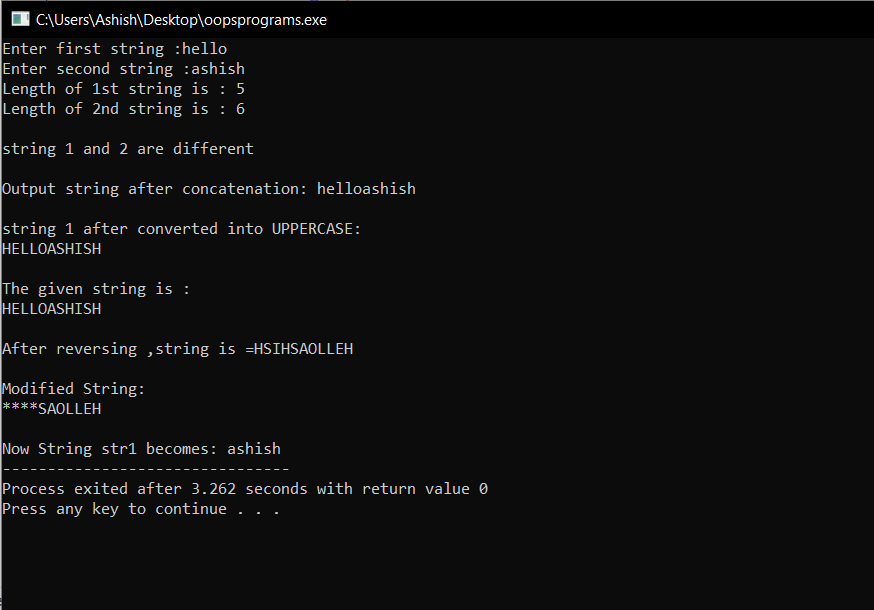
strcpy(str1,str2);

cout<<"\nNow String str1 becomes: "<< str1;

return 0;

}

**OUTPUT:-**



**PROGRAM 8**

**Aim:-**To write a program to demonstrate inheritance in OOPS.

**Description:-** The capability of a class to derive properties and characteristics from another class is called **Inheritance**.

**Implementing inheritance in C++**: For creating a sub-class which is inherited from the base class we have to follow the below syntax.  
**Syntax**:

class subclass\_name : access\_mode base\_class\_name

{

//body of subclass

};

Here, **subclass\_name**is the name of the sub class, **access\_mode**is the mode in which you want to inherit this sub class for example: public, private etc. and **base\_class\_name**is the name of the base class from which you want to inherit the sub class.

**Types of Inheritance in C++**

1. **Single Inheritance**: In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.
2. **Multiple Inheritance:** Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one **sub class** is inherited from more than one **base classes**.
3. **Multilevel Inheritance**: In this type of inheritance, a derived class is created from another derived class.
4. **Hierarchical Inheritance**: In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single base class.
5. **Hybrid (Virtual) Inheritance**: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

**CODE:-**

//this is an example of multiple inheritance

#include <iostream>

using namespace std;

class student {

protected:

int rno;

public:

void get\_no(int);

void put\_no(void);

};

void student::get\_no(int a) {

rno=a;

}

void student::put\_no() {

cout<<"Roll Number: "<<rno<<endl;

}

class test: public student {

protected:

float sub1,sub2;

public:

void get\_marks(float,float) ;

void put\_marks(void);

};

void test::get\_marks(float x,float y) {

sub1=x;

sub2=y;

}

void test::put\_marks()

{

cout<<"Marks in subject1 = "<<sub1<<endl;

cout<<"Marks in subject2 = "<<sub2<<endl;

}

class result:public test {

float total;

public:

void display(void) {

total=sub1 + sub2;

put\_no();

put\_marks();

cout<<"Total = "<<total<<endl;

}

};

int main () {

result stu;

int rollno;

cout<<"Enter your roll no.: ";

cin>>rollno;

cout<< "\nDetails are: "<<endl;

stu.get\_no(rollno);

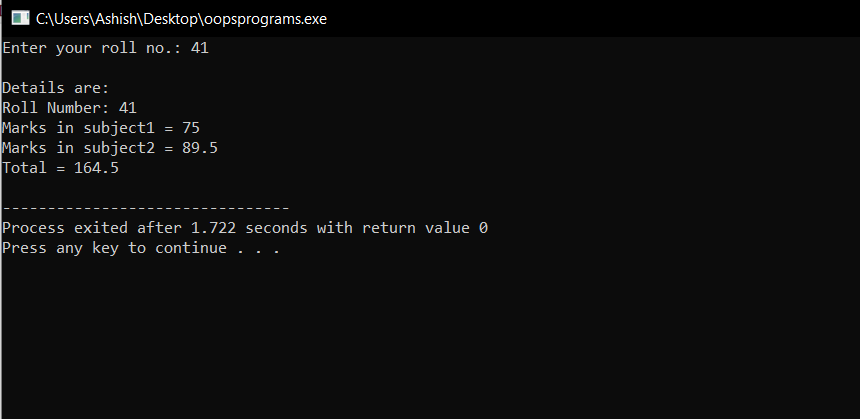
stu.get\_marks(75.0,89.5);

stu.display();

return 0;

}

**OUTPUT:-**

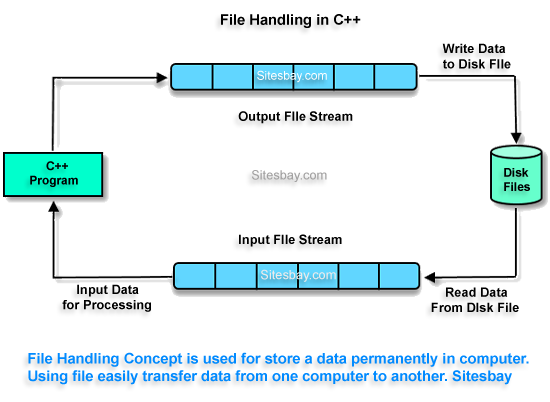
****

**PROGRAM 9**

**Aim:-**To write a program to open a file, edit the contents and close the file.

**Description:-**

**File Handling** concept in C++ language is used for store a data permanently in computer. Using file handling we can store our data in Secondary memory (Hard disk).



For read and write from a file you need another standard C++ library called **fstream**, which defines three new data types:

|  |  |
| --- | --- |
| **Datatype** | **Description** |
| ofstream | This is used to create a file and write data on files |
| ifstream | This is used to read data from files |
| fstream | This is used to both read and write data from/to files |
|  |  |

The function open() can be used to open multiple files that use the same stream object.

**Syntax**

file-stream-class stream-object;

stream-object.open("filename");

**Closing a File**

A file must be close after completion of all operation related to file. For closing file we need **close()** function.

**Syntax**

outfile.close();

**CODE:-**

//Working with multiple files(opening,writing,closing)

#include <iostream>

#include<fstream>

using namespace std;

int main () {

ofstream fout;

fout.open("country");

fout<<"United States of America(USA),\n";

fout<<"Austria,\n"<<"\n";

fout<<"India\n"<<endl;

fout.close();

fout.open("capital");

fout<<"\nWashington DC,\n"<<endl;

fout<<"Vienna,\n"<<endl;

fout<<"Delhi\n"<<endl;

fout.close();

//Reading the files

const int n=60; //size of line

char line[n];

ifstream fin;

fin.open("country");

cout<<"Contents of country file: \n";

while(fin) {

fin.getline(line,n); //read a line

cout<<line;

}

fin.close();

fin.open("capital");

cout<<"\n\nContents of capital file: \n";

while(fin) {

fin.getline(line,n);

cout<<line;

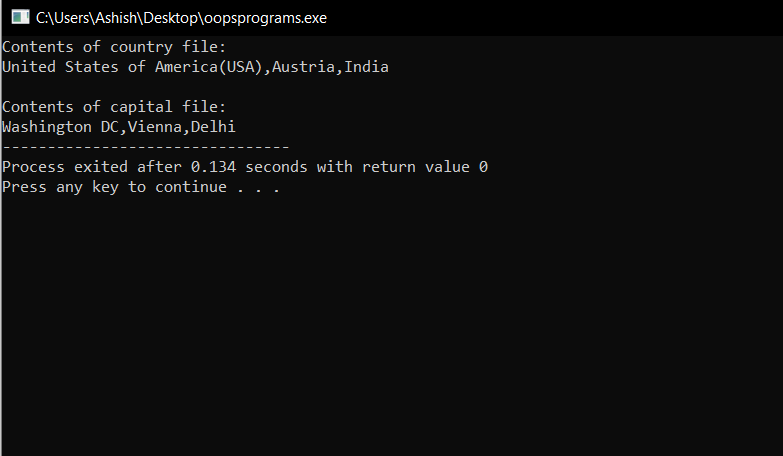
}

fin.close();

return 0;

}

**OUTPUT:-**



//Reading from two files simultaneously

#include <iostream>

#include<fstream>

#include<cstdlib>

using namespace std;

int main () {

const int size =80;

char line[size];

ifstream fin1,fin2;

fin1.open("country");

fin2.open("capital");

for(int i=0;i<10;i++) {

if(fin1.eof()!=0) {

cout<<"Exit from country\n";

exit(1);

}

fin1.getline(line,size);

cout<<"Capital of"<<line;

if(fin2.eof()!=0) {

cout<<"Exit from capital\n";

exit(1);

}

fin2.getline(line,size);

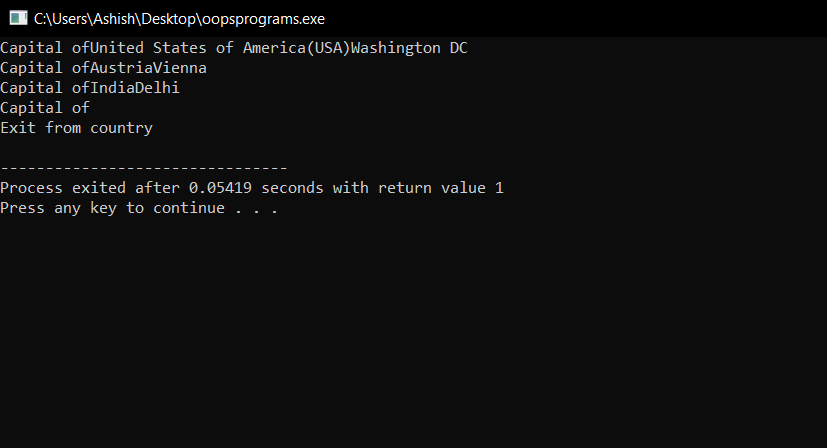
cout<<line<<endl;

}

return 0;

}

**OUTPUT:-**



**PROGRAM 10**

**Aim:-**To write a program to demonstrate “exception handling”.

**Description:-**

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch,** and **throw**.

* **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.
* **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.

**CODE:-**

#include <iostream>

using namespace std;

double division(int a, int b) {

if( b == 0 ) {

cout<<endl;

throw "Sorry u can't divide a number by zero!";

}

return (a/b);

}

int main () {

int x ;

cout<<"Enter numerator :";

cin>>x;

int y = 0;

double z = 0;

try {

z = division(x, y);

cout << z << endl;

} catch (const char\* msg) {

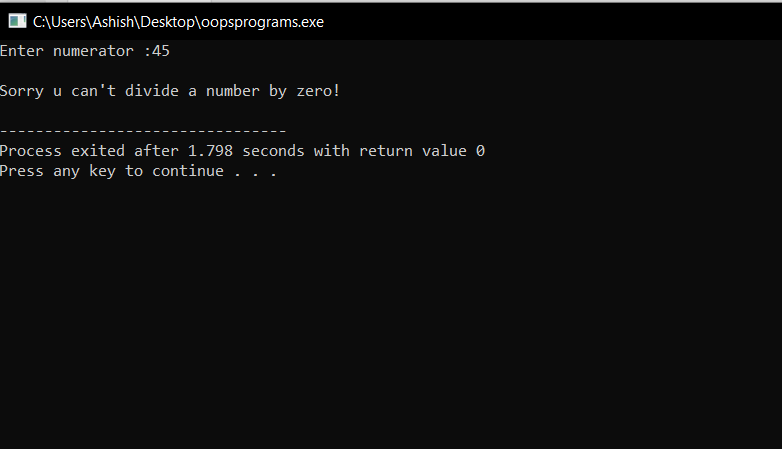
cout << msg << endl;

}

return 0;

}

**OUTPUT:-**



**PROGRAM 11**

**Aim:-** Write a program to demonstrate “Standard template library”.

**Description:-** The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays,maps etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized.

**STL has four components**

* Algorithms
* Containers
* Functions
* Iterators

**Algorithms**

The header algorithm defines a collection of functions especially designed to be used on ranges of elements.They act on containers and provide means for various operations for the contents of the containers.

* Algorithm
  + [Sorting](http://quiz.geeksforgeeks.org/sort-algorithms-the-c-standard-template-library-stl/)
  + [Searching](http://quiz.geeksforgeeks.org/binary-search-algorithms-the-c-standard-template-library-stl/)
  + [Important STL Algorithms](https://www.geeksforgeeks.org/c-magicians-stl-algorithms/)
  + [Useful Array algorithms](https://www.geeksforgeeks.org/useful-array-algorithms-in-c-stl/)
  + [Partition Operations](https://www.geeksforgeeks.org/stdpartition-in-c-stl/)
* Numeric
  + [valarray class](https://www.geeksforgeeks.org/std-valarray-class-c/)

**Containers**

Containers or container classes store objects and data. There are in total seven standard “first-class” container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors.

* Sequence Containers: implement data structures which can be accessed in a sequential manner.
  + [vector](http://quiz.geeksforgeeks.org/vector-sequence-containers-the-c-standard-template-library-stl-set-1/)
  + [list](http://quiz.geeksforgeeks.org/list-sequence-containers-the-c-standard-template-library-stl/)
  + [deque](http://quiz.geeksforgeeks.org/deque-sequence-containers-the-c-standard-template-library-stl/)
  + [arrays](https://www.geeksforgeeks.org/array-class-c/)
* Container Adaptors : provide a different interface for sequential containers.
  + [queue](http://quiz.geeksforgeeks.org/queue-container-adaptors-the-c-standard-template-library-stl/)
  + [priority\_queue](http://quiz.geeksforgeeks.org/priority-queue-container-adaptors-the-c-standard-template-library-stl/)
  + [stack](http://quiz.geeksforgeeks.org/stack-container-adaptors-the-c-standard-template-library-stl/)
* Associative Containers : implement sorted data structures that can be quickly searched (O(log n) complexity).
  + [set](http://quiz.geeksforgeeks.org/set-associative-containers-the-c-standard-template-library-stl/)
  + [multiset](http://quiz.geeksforgeeks.org/multiset-associative-containers-the-c-standard-template-library-stl/)
  + [map](http://quiz.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/)
  + [multimap](http://quiz.geeksforgeeks.org/multimap-associative-containers-the-c-standard-template-library-stl/)

**Functions**

The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters to be passed.

* [Functors](https://www.geeksforgeeks.org/functors-in-cpp/)

**Iterators**

As the name suggests, iterators are used for working upon a sequence of values. They are the major feature that allows generality in STL.

* [Iterators](https://www.geeksforgeeks.org/iterators-c-stl/)

**CODE:-**

\*/ Using of vector header file/\*

#include<iostream>

#include<vector>

using namespace std;

void display(vector<int>&v) {

for(int i=0;i<v.size();i++) {

cout<<v[i]<<" ";

}

cout<<endl;

}

int main()

{

vector<int> v; //create a vector of type int

cout<<"Initial size = "<<v.size()<<endl;

int x;

cout<<"Enter five integer values : ";

for(int i=0;i<5;i++) {

cin>>x;

v.push\_back(x);

}

cout<<" \nSize after adding 5 integers = "<<v.size()<<endl;

cout<<"\nCurrent contents : \n";

display(v);

//add one more value

v.push\_back(6.6); //float value truncated to int

cout<<"\nNew element is inserted successfully!!\n";

cout<<"\nNow SIZE = "<<v.size()<<endl;

cout<<"CONTENTS : \n";

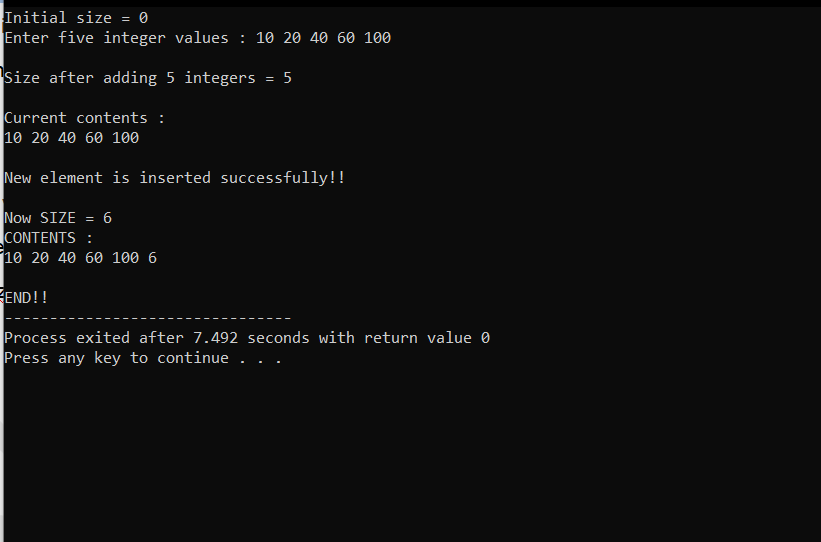
display(v);

cout<<"\nEND!!";

return 0;

}

**OUTPUT:-**



\*/ Using of map header file/\*

#include<iostream>

#include<map>

#include<string>

using namespace std;

typedef map<string,int> phoneMap; // key is type of string & value is type of int

int main()

{

string name;

int number;

phoneMap ph;

cout<<"Enter three sets of name and number: \n";

for(int i=0;i<3;i++) {

cin>>name; //get key

cin>>number; //get value

ph[name]=number; //put them in map

}

ph["ASHU"]=1001;

ph.insert(pair<string,int> ("ANKIT",2001));

int n=ph.size();

cout<<"\nSize of Map is : "<<n<<endl;

cout<<"\nList of numbers : \n";

phoneMap::iterator p;

for(p=ph.begin();p!=ph.end();p++) {

cout<<(\*p).first<<" "<<(\*p).second<<"\n";

}

cout<<endl;

cout<<"Enter Name : ";

cin>>name;

number=ph[name];

cout<<"Number : "<<number<<endl;

return 0;

}

**OUTPUT:-**

