**Program No:** 01

**Program Topic:** *Constructor and Destructor Functions.*

**Program Title:**

Write an **program** in C++ to create database of the following items:

1. Name of the student

2. Roll number of the student

3. Height of the student

4. Weight of the student

**Objectives:**

To get basic knowledge about Constructor and Destructor Functions.

**Syntax:**

**Constructor:**

//within a class

public:

className {

//set fields and call methods

}

**Destructor:**

//within a class

public:

~className(void) {

//code to run on object deletion

}

**Source Code:**

#include <iostream>

#include <string>

using namespace std;

class student

{

private:

string name;

int roll;

float height, weight;

public:

student( );

~student( );

void display( );

};

student :: student ( ){

cout << “In Constructor..\n”

name = "NAim";

roll = 18;

height = 1.78;

weight = 64;

}

void student :: display ()

{

cout << "\nName: " << name;

cout << "\nRoll no: " << roll;

cout<< "\nHeight: " << height << "m";

cout<<"\nWeight: " << weight << "kg\n";

}

student :: ~ student ()

{

cout << "\ndestructing...\n";

}

int main ()

{

student obj;

student( );

obj.display( );

return 0;

}

**Output:**

In Constructor..

Name: NAim

Roll no: 18

Height: 1.78m

Weight: 64kg

destructing...

Process returned 0 (0x0) execution time : 0.033 s

Press any key to continue.

**Explanation:**

Constructor and Destructor function has the same name as class. And it has no return type. Constructor is called every time when an object is created. A memory is allocated when an object is created. Destructor frees that memory.

**Program No:** 02

**Program Topic:** *Constructor and Destructor Functions with stack.*

**Program Title:**

Write a C++ program to do stack operation using constructor and destructor functions.

**Objectives:**

To learn stack operation using Constructor and Destructor Functions.

**Syntax:**

**Constructor:**

//within a class

public:

className {

//set fields and call methods

}

**Destructor:**

//within a class

public:

~className(void) {

//code to run on object deletion

}

**Source Code:**

#include <iostream>

using namespace std ;

# define SIZE 100

class stack

{

char stck [SIZE];

int top;

public :

stack ( );

void push (char ch);

char pop ( );

};

stack :: stack ()

{

cout << " Constructing a stack \n";

top = 0;

}

void stack :: push (char ch)

{

if(top == SIZE)

{

cout << " Stack is full \n";

return;

}

stck[top] = ch;

top++;

}

char stack :: pop ()

{

if(top == 0)

{

cout << " Stack is empty \n";

return 0;

}

top --;

return stck [top];

}

int main ()

{

stack s;

int i, n1, n2;

char c;

cout << "How many times you want to push? ";

cin >> n1;

cout << endl;

cout << "Enter characters\n";

for(i = 0; i < n1; i++){

cin >> c;

s.push(c);

}

cout << "How many times you want to pop? ";

cin >> n2;

cout << endl;

for (i =0; i < n2; i ++)

cout << "Pop s: " << s.pop () << endl;

return 0;

}

**Output:**

Constructing a stack

How many times you want to push? 3

Enter characters

a

b

c

How many times you want to pop? 2

Pop s: c

Pop s: b

Process returned 0 (0x0) execution time : 8.694 s

Press any key to continue.

**Program No:** 03

**Program Topic:** *Constructor and Destructor Functions with time function.*

**Program Title:**

Write a C++ program to determine the time interval of a program.

**Objectives:**

To learn stack operation using Constructor and Destructor Functions.

**Syntax:**

**Constructor:**

//within a class

public:

className {

//set fields and call methods

}

**Destructor:**

//within a class

public:

~className(void) {

//code to run on object deletion

}

**Time:**

clock\_t start ;

start = clock( );

clock\_t end ;

end = clock ();

Elappsed time = (end – start) / CLOCKS\_PER\_SEC;

**Source Code:**

# include <iostream >

# include <ctime >

using namespace std ;

class timer

{

clock\_t start ;

public :

timer ();

~ timer ();

};

timer :: timer ()

{

start = clock ();

}

timer ::~ timer ()

{

clock\_t end ;

end = clock ();

cout << " Elapsed time : " << (end - start ) / CLOCKS\_PER\_SEC

<< "\n";

}

int main ()

{

timer ob;

char c;

cout << " Press a key followed by ENTER : ";

cin >> c;

return 0;

}

**Output:**

Press a key followed by ENTER : c

Elapsed time : 3

Process returned 0 (0x0) execution time : 3.463 s

Press any key to continue.

**Explanation:**

clock() function is used to determine the time interval from where the function started to where the function ended.

**Program No:** 04

**Program Topic:** *Object Pointers.*

**Program Title:**

Write a C++ program using pointers to objects.

**Objectives:**

To learn how to use pointers to objects.

**Syntax:**

typedef double\* DoubleArrayPtr;

DoubleArrayPtr a;

a = new double[array\_size];

delete [] a;

**Source Code:**

#include <iostream>

#include <string>

using namespace std;

class student

{

private:

int roll;

string name;

public:

student( ) : roll(0), name("")

{}

student(int r, string n) : roll(r), name (n)

{}

void get( )

{

cout<<"Enter Roll no: ";

cin>>roll;

cout<<"Enter Name: ";

cin>>name;

}

void print( )

{

cout<<"\nRoll no: " << roll << endl;

cout<<"Name: " << name << endl;

}

};

int main ( )

{

student \*ps = new student;

(\*ps).get( );

(\*ps).print( );

delete ps;

return 0;

}

**Output:**

Enter Roll no: 1703018

Enter Name: NAim

Roll no: 1703018

Name: NAim

Process returned 0 (0x0) execution time : 8.200 s

Press any key to continue.

**Explanation:**

A variable that holds an address value is called a pointer variable or simply pointer.

Pointer can point to objects as well as to simple data types and arrays.

**Program No:** 05

**Program Topic:** *Inline Functions.*

**Program Title:**

Write a C++ program to determine cubic area of a cube.

**Objective:**

To learn about in line Functions.

**Syntax:**

Inline datatype variable\_name(arguments);

**Source Code:**

#include <iostream>

using namespace std;

inline int cube(int s)

{

return s\*s\*s;

}

int main()

{

int n;

cout << "Enter a side of cube: ";

cin >> n;

cout << endl;

cout << "The cube of " << n <<" is: " << cube(n) << endl;

return 0;

}

**Output:**

Enter a side of cube: 3

The cube of 3 is: 27

Process returned 0 (0x0) execution time : 2.265 s

Press any key to continue.

**Explanation:**

The in-line function are not actually called but, rather, are expanded in line, at the point of each call.

**Program No:** 06

**Program Topic:** *Assigning Objects.*

**Program Title:**

Write a C++ program to assign and change objects.

**Objectives:**

To learn how to assign objects.

**Syntax:**

className ob1, ob2;

…. …. ….

ob1 = ob2;

**Source Code:**

#include <iostream>

using namespace std;

class MyClass

{

int a, b;

public:

void setAB(int i, int j) { a = i, b = j; }

void display()

{

cout << "\n a is " <<a << "\n";

cout << "\n b is " << b <<"\n\n";

}

};

int main()

{

MyClass ob1, ob2;

ob1.setAB(10, 20);

ob2.setAB(0, 0);

cout << "ob1 before assignment";

ob1.display();

cout << "ob2 before assignment";

ob2.display();

ob2 = ob1;

cout << "ob1 after assignment";

ob1.display();

cout << "ob2 after assignment";

ob2.display();

ob1.setAB(-1, -1);

cout << "ob1 after changing ob1 :";

ob1.display();

cout << "ob2 after changing ob1 :";

ob2.display();

cout << endl;

return 0;

}

**Output:**

ob2 before assignment

a is 0

b is 0

ob1 after assignment

a is 10

b is 20

ob2 after assignment

a is 10

b is 20

ob1 after changing ob1 :

a is -1

b is -1

ob2 after changing ob1 :

a is 10

b is 20

Process returned 0 (0x0) execution time : 0.105 s

Press any key to continue.

**Explanation:**

For assigning object, two objects must be from the same class.

**Program No:** 07

**Program Topic:** *Passing and returning objects from function.*

**Program Title:**

Write a C++ program to add two complex numbers.

**Objectives:**

To learn how to pass and return object from function.

**Syntax:**

class className{

……….

public:

className functionName(className arg1)

{

className obj;

… … …

return obj;

}

… … …

};

**Source Code:**

#include <iostream>

using namespace std;

class Complex

{

private:

int real;

int imag;

public:

Complex( );

void read( );

void display( );

Complex addComplexNumbers(Complex comp2)

{

Complex temp;

temp.real = real+comp2.real;

temp.imag = imag+comp2.imag;

return temp;

}

};

Complex :: Complex( ): real(0), imag(0)

{ }

void Complex :: read()

{

cout << "Enter real and imaginary number respectively:"<<endl;

cin >> real >> imag;

}

void Complex :: display()

{

cout << "Sum = " << real << "+" << imag << "i" << endl;

}

int main()

{

Complex c1, c2, c3;

c1.read();

c2.read();

c3 = c1.addComplexNumbers(c2);

c3.display();

return 0;

}

**Output:**

Enter real and imaginary number respectively:

2

3

Enter real and imaginary number respectively:

4

1

Sum = 6+4i

Process returned 0 (0x0) execution time : 3.855 s

Press any key to continue.

**Program No:** 08

**Program Topic:** *Friend Function.*

**Program Title:**

Write a C++ program using friend function.

**Objectives:**

To learn about Friend Function.

**Syntax:**

//within class

friend datatype functionName(className);

**Source Code:**

#include <iostream>

using namespace std;

class Distance

{

private:

int meter;

public:

Distance(): meter(0) { }

friend int addFive(Distance);

};

int addFive(Distance d)

{

d.meter += 5;

return d.meter;

}

int main()

{

Distance D;

cout<<"Distance: "<< addFive(D);

return 0;

}

**Output:**

Distance: 5

Process returned 0 (0x0) execution time : 0.134 s

Press any key to continue.

**Explanation:**

A friend is not a member of a class but still has access to its private elements. Friend functions are useful have to do with operator overloading and the creation of certain types of I/O functions. By friend function one function can have access to the private members of two or more different classes.

**Program No:** 09

**Program Topic:** *Object Arrays.*

**Program Title:**

Write a C++ program using arrays of object.

**Objectives:**

To learn how to use arrays of object.

**Syntax:**

class\_name array\_name [size] ;

**Source Code:**

#include <iostream>

#include <string>

using namespace std;

class books

{

string title;

float price ;

public:

void get( );

void put( );

};

void books :: get( )

{

cout << "Title: ";

cin >> title;

cout << "Price: ";

cin >> price;

}

void books :: put( )

{

cout << "Title: " << title << "\n";

cout << "Price: " << price << "\n";

}

int main ( )

{

books book[100];

int n;

cout << "How many books? ";

cin >> n;

cout << endl;

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

cout<<"Enter details of book " << (i+1) << "\n";

book[i].get( );

}

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

cout << "\nBook " << (i + 1) << "\n";

book[i].put( );

}

return 0;

}

**Output:**

How many books? 3

Enter details of book 1

Title: C++

Price: 500

Enter details of book 2

Title: C

Price: 300

Enter details of book 3

Title: Java

Price: 900

Book 1

Title: C++

Price: 500

Book 2

Title: C

Price: 300

Book 3

Title: Java

Price: 900

Process returned 0 (0x0) execution time : 22.563 s

Press any key to continue.

**Explanation:**

Like array of other user-defined data types, an array of type class can also be created. The array of type class contains the objects of the class as its individual elements. Thus, an array of a class type is also known as an array of objects.

**Program No:** 10

**Program Topic:** *this Pointer.*

**Program Title:**

Write a C++ program using this pointer.

**Objectives:**

To learn about this pointer.

**Syntax:**

//within class

return\_type functionname(arg){

this -> variable = variable;

return \*this;

}

**Source Code:**

#include<iostream>

using namespace std;

class Test

{

private:

int x;

public:

void setX (int x)

{

this -> x = x;

}

void print() { cout << "x = " << x << endl; }

};

int main()

{

Test obj;

int x = 20;

obj.setX(x);

obj.print();

return 0;

}

**Output:**

x = 20

Process returned 0 (0x0) execution time : 0.119 s

Press any key to continue.

**Explanation:**

The ‘this’ pointer is passed as a hidden argument to all nonstatic member function calls and is available as a local variable within the body of all nonstatic functions. ‘this’ pointer is a constant pointer that holds the memory address of the current object. ‘this’ pointer is not available in static member functions as static member functions can be called without any object (with class name).

**Program No:** 11

**Program Topic:** *Dynamic Memory Allocation and Deallocation.*

**Program Title:**

Write a C++ program to illustrate dynamic allocation and deallocation of memory using new and delete.

**Objectives:**

To learn how to do dynamic memory allocation and deallocation of memory using new and delete.

**Syntax:**

pointer-variable = new data-type;

pointer-variable = new data-type[size];

delete pointer-variable;

**Source Code:**

#include <iostream>

using namespace std;

int main ()

{

int\* p = NULL;

p = new int;

if (!p)

cout << "allocation of memory failed\n";

else

{

\*p = 30;

cout << "Value of p: " << \*p << endl;

}

float \*r = new float(75.25);

cout << "Value of r: " << \*r << endl;

int n = 5;

int \*q = new int[n];

if (!q)

cout << "allocation of memory failed\n";

else

{

for (int i = 0; i < n; i++)

q[i] = i+1;

cout << "Value store in block of memory: ";

for (int i = 0; i < n; i++)

cout << q[i] << " ";

}

delete p;

delete r;

delete[ ] q;

return 0;

}

**Output:**

Value of p: 30

Value of r: 75.25

Value store in block of memory: 1 2 3 4 5

Process returned 0 (0x0) execution time : 0.164 s

Press any key to continue.

**Explanation:**

Dynamic memory allocation is useful. One use of dynamically allocated memory is to allocate memory of variable size which is not possible with compiler allocated memory except variable length arrays. The most important use is flexibility provided to programmers. We are free to allocate and deallocate memory whenever we need and whenever we don’t need anymore. There are many cases where this flexibility helps