Assignment -2

Choose the correct answers:

1) D

2) C

3) A

4) A

5) A

Short answers type questions:

1).

The new operator

The new operator requests for the memory allocation in heap. If the sufficient memory is available, it initializes the memory to the pointer variable and returns its address.

Here is the syntax of new operator in C++ language,

```
pointer_variable = new datatype;
```

Here is the syntax to initialize the memory,

```
pointer_variable = new datatype(value);
```

Here is the syntax to allocate a block of memory,

```
pointer_variable = new datatype[size];
```

Here is an example of new operator in C++ language,

Example

```
#include
using namespace std;
int main () {
int *ptr1 = NULL;
ptr1 = new int;
float *ptr2 = new float(223.324);
int *ptr3 = new int[28];
*ptr1 = 28;
cout << "Value of pointer variable 1  ": << *ptr1 << endl;
cout << "Value of pointer variable 2  ": << *ptr2 << endl;
if (!ptr3)
cout << "Allocation of memory failed\n";
else {
for (int i = 10; i < 15; i++)
ptr3[i] = i+1;
cout << "Value of store in block of memory: ";
for (int i = 10; i < 15; i++)
cout << ptr3[i] << " ";
}
return 0;
}
```

Output

```
Value of pointer variable 1   28
Value of pointer variable 2   223.324
Value of store in block of memory: 11 12 13 14 15
```

The delete operator

The delete operator is used to deallocate the memory. User has privilege to deallocate the created pointer variable by this delete operator.

Here is the syntax of delete operator in C++ language,

```
delete pointer_variable;
```

Here is the syntax to delete the block of allocated memory,

```
delete[ ]pointer_variable;
```

Here is an example of delete operator in C++ language,

Example

```
#include
using namespace std;
int main () {
int *ptr1 = NULL;
ptr1 = new int;
float *ptr2 = new float(299.121);
int *ptr3 = new int[28];
*ptr1 = 28;
cout << "Value of pointer variable 1  ": << *ptr1 << endl;
cout << "Value of pointer variable 2  ": << *ptr2 << endl;
if (!ptr3)
cout << "Allocation of memory failed\n";
else {
for (int i = 10; i < 15; i++)
ptr3[i] = i+1;
cout << "Value of store in block of memory: ";
for (int i = 10; i < 15; i++)
cout << ptr3[i] << " ";
}
delete ptr1;
delete ptr2;
delete[] ptr3;
return 0;
}
```

Output

```
Value of pointer variable 1   28
Value of pointer variable 2   299.121
Value of store in block of memory: 11 12 13 14 15
```

2).

Constructor:

A constructor is a special type of function with no return type. Name of constructor should be same as the name of the class. we define a method inside the class and constructor is also defined inside a class. A constructor is called automatically when we create an object of a class. we can't call a constructor explicitly. Let us see the types of constructor.

Constructor Types

1.Default Constructor

2.Parameterized Constructor

3.Copy Constructor

4.Static constructor
5.Private constructor

constructor is required for:
1.constructor is called automatically when we create an object of the class.
2.Name of constructor should be same as the name of the class.
3.constructor does not return any value.
4.constructor should have a public access modifier.

3).
Differences between Procedural and Object Oriented Programming:

Procedural Programming:
Procedural Programming can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure. Procedures, also known as routines, subroutines or functions, simply consist of a series of computational steps to be carried out. During a program's execution, any given procedure might be called at any point, including by other procedures or itself.

Languages used in Procedural Programming:
FORTRAN, ALGOL, COBOL,

BASIC, Pascal and C.

Object Oriented Programming:

Object oriented programming can be defined as a programming model which is based upon the concept of objects. Objects contain data in the form of attributes and code in the form of methods. In object oriented programming, computer programs are designed using the concept of objects that interact with real world. Object oriented programming languages are various but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.

Languages used in Object Oriented Programming:

Java, C++, C#, Python,
PHP, JavaScript, Ruby, Perl,
Objective-C, Dart, Swift, Scala.

Long answer type questions:

A).

Polymorphism in C++:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A real-life example of polymorphism, a person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism. Polymorphism is considered as one of the important features of Object Oriented Programming.

In C++ polymorphism is mainly divided into two types:
1.Compile time Polymorphism
2.Runtime Polymorphism

Compile time polymorphism:
This type of polymorphism is achieved by function overloading or operator overloading.
Function Overloading: When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments.
Rules of Function Overloading
Example:

```
// C++ program for function overloading
#include

using namespace std;
class Geeks
```

```
{
public:

// function with 1 int parameter
void func(int p)
{
cout << "value of p is " << p << endl;
}

// function with same name but 1 double parameter
void func(double p)
{
cout << "value of p is " << p << endl;
}

// function with same name and 2 int parameters
void func(int p, int q)
{
cout << "value of p and q is " << p << ", " << q << endl;
}
};

int main() {
```

```
Geeks obj1;

// which function is called will depend on the parameters passed
// The first 'func' is called
obj1.func(7);

// The second 'func' is called
obj1.func(9.132);

// The third 'func' is called
obj1.func(85,64);
return 0;
}
```

Output:

```
value of x is 7
value of x is 9.132
value of x and y is 85, 64
```

In the above example, a single function named func acts differently in three different situations which is the property of polymorphism.

Operator Overloading:

C++ also provide option to overload operators. For example, we can make the operator ('+') for string class to concatenate two strings.

we know that this is the addition operator whose task is to add two operands. So a single operator '+' when placed between integer operands adds them and when placed between string operands, concatenates them.

Example:

```
// CPP program to illustrate
// Operator Overloading
#include
using namespace std;

class complex {
private:
int real, imag;
public:
complex(int r = 0, int i =0) {real = r; imag = i;}

// This is automatically called when '+' is used with
// between two complex objects
complex operator + (complex const &obj) {
complex res;
res.real = real + obj.real;
res.imag = imag + obj.imag;
return res;
}
void print() { cout << real << " + i" << imag << endl; }
};

int main()
{
complex c1(10, 5), c2(2, 4);
complex c3 = c1 + c2;   //An example call to "operator+"
c3.print();
}
```

Output:

```
12 + i9
```

In the above example the operator '+' is overloaded. The operator '+' is an addition operator and can add two numbers(integers or floating point) but here the operator is made to perform addition of two imaginary or complex numbers. To learn operator overloading in details visit this link.

Runtime polymorphism:

This type of polymorphism is achieved by Function Overriding. Function overriding on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

Example:

```
// C++ program for function overriding
```

```
#include
using namespace std;

class base
{
public:
virtual void print ()
{ cout<< "print base class" <
void show ()
{ cout<< "show base class" <};

class derived:public base
{
public:
void print ()   print () is already virtual function in derived class,
we could also declared as virtual void print () explicitly
{ cout<< "print derived class" <
void show ()
{ cout<< "show derived class" <};

//main function
int main()
{
base *bptr;
derived d;
bptr = &d;

//virtual function, binded at runtime (Runtime polymorphism)
bptr->print();

// Non-virtual function, binded at compile time
bptr->show();

return 0;
}
```

Output:

```
print derived class
show base class
```