1. Write a statement to create a dynamic array of integers of size 5 using `new` operator in C++?
   int *p = new int[5];

2. Write a statement to create a dynamic array of pointers (to) integers of size 5 using `new` operator in C++?
   int **p = new int *[5];

3. State True or false. A destructor is called when a program that declares an object of a class ends.
   True

4. State True or false. While both `new` operator and `malloc` function can be used for dynamic memory allocation, the `new` operator calls the constructor but `malloc` does not.
   True

5. State True or False. A `friend function` can access all the private, protected and public members of a class.
   True

6. State True or False. Given a set `S` and a key `K`, the value of `S.count(K)` will either be 0 or 1.
   True

7. Given a `class A` as defined below:

```
class A{
   int x;
   int y;
   public:
   void disp();
}
```

Define a destructor (within the class) that displays the message `"Object destroyed"`

[~A(){cout<<"Object destroyed";}]

8. Given the following declaration of the `class Foo`, which of the following are valid ways to construct an object of `class Foo` (note the difference between parenthesis and curly braces)

```
class Foo
{
   public:
   int i = 0;

   Foo(int x):i(x){}
   };

(a)Foo f;
(b)Foo f{};
(c)Foo f(1);
(d)Foo f(Foo(1));
```

[c,d]

9. Define a class `complex` to maintain complex numbers of the form `a + ib.` The class should contain the following members:
   - Integer data members `a and b`.
   - A constructor with `zero` arguments and default values set to 1;
   - A constructor with `two` arguments.
   - A Friend function **`void addComplexNums(complex, complex)`** function that takes two complex numbers as arguments and displays their sum.

```
class complex{

      int a;
      int b;

  public:
      complex():a(1),b(1){}
      complex(int x, int y):a(x),b(y){}

      friend complex addComplexNums(complex,complex);
   };

  complex addComplexNums(complex c1, complex c2){
   complex sumc;
   sumc.a = c1.a +c2.a;
   sumc.b = c1.b +c2.b;
   return sumc;
  }
}
```

10. Define a Class template as follows:

```
template <class T>
class Comp{
    T dataitem_1;
    T dataitem_2;
public:
    Comp();
    Comp(T, T);
    T Max();
}
```

Include the following:

a) Definitions of the default and parameterized constructors inside class template.
b) Definition of function `Max()` which returns the maximum of the two data members, outside class template.
c) Include a `main()` function to test the functionality of the different member functions of the class template with `int` and `float` as template arguments..

```
#include<iostream>
using namespace std;
//Template Definition
template <class T>
class Comp{
    T dataitem_1, dataitem_2;
public:
    Comp(){};
    Comp(T a, T b):dataitem_1(a),dataitem_2(b){}
    T Max();
};
//Member function definition outside class
template<class T>
T Comp<T>::Max(){
   return dataitem_1>dataitem_2 ? dataitem_1 : dataitem_2;
}
int main(){
    // Integer Template parameter
     Comp<int> intComp(5,10);
     cout<<"Max for Integer template parameters:
"<<intComp.Max()<<endl;
     Comp<float> floatComp(5.20,10.50);
```