**Question set 2**

The following questions are from w3schools.com and are the C++ tutorials. They come from:

C++ Functions

C++ Function Parameters (except Pass By Reference)

C++ OOP

C++ Classes/Objects

C++ Class Methods

C++ Constructors

C++ Access Specifiers

They are in order. If you start with functions and work from there, you should be fine.

The subjects below are from the C++ tutorials at w3schools.com. They will be discussed and more questions will be created at a later date.

C++ Encapsulation

C++ Inheritance

C++ Polymorphism

C++ Exceptions

C++ References

C++ Pointers

C++ Function Overloading

C++ Function Parameters (Pass By Reference)

All questions are potential quiz and exam questions:

A \_\_\_\_\_\_\_\_\_\_ is a block of code which only runs when it is called. **function**

You can pass data, known as\_\_\_\_\_\_\_\_\_\_, into a function. **parameters**

To create (often referred to as \_**declare**\_) a function, specify the name of the function, followed by parentheses **()**

A C++ function consist of two parts. One is the \_**declaration**\_ which is the function's name, return type, and parameters (if any).

A C++ function consist of two parts. One is the \_**definition**\_ which is the body of the function (code to be executed).

Is it possible to separate the declaration and the definition of the function? **Yes**

Information can be passed to functions as a \_\_**parameters**\_\_\_\_\_\_\_\_. They act as variables inside the function.

You can also use a default parameter value, by using the \_\_**=**\_\_\_\_\_\_\_\_.

If you want the function to return a value, you can use a \_\_\_**data type**\_\_\_ (such as int, string, etc.) instead of void, and use the return keyword inside the function:

Procedural programming is about writing procedures or functions that perform operations on the data, while \_**OOP**\_ is about creating objects that contain both data and functions.

Given:

|  |  |
| --- | --- |
| **Class** | **Objects** |
| Fruit | Apple, Banana, Mango |

|  |  |
| --- | --- |
| **Class** | **Objects** |
| Car | Volvo, Audi, Toyota |

|  |  |
| --- | --- |
| **Class** | **Objects** |
| Starship | Galaxy, Runabout, Shuttle |

As observed in the tables above, a \_\_**class**\_ is a template for an object.

As observed in the tables above, an object is an \_**instance**\_ of a class.

When the individual objects are created, they inherit all the variables and functions from the \_**class**\_.

C++ is an \_**OOP**\_\_\_ programming language.

Everything in C++ is associated with \_\_**classes and objects**\_\_, along with its attributes and methods.

Attributes and methods are basically **variables** and **functions** that belongs to the class. These are often referred to as \_**class members**\_.

A class is a user-defined \_\_**data type**\_ that we can use in our program.

A class works as an object \_**object constructor**\_, or a "blueprint" for creating objects.

To create a class, use the \_**class**\_ keyword.

In C++, an \_\_**object**\_ is created from a class.

Given the following example:

// -----------------------------------------------  
class MyClass {       // The class  
  public:             // Access specifier  
    int myNum;        // Attribute (int variable)  
    string myString;  // Attribute (string variable)  
};  
  
int main() {  
  MyClass **myObj**;  // Create an object of MyClass  
  
  // Access attributes and set values  
  **myObj.myNum** = 15;   
  **myObj.myString** = "Some text";  
  
  // Print attribute values  
  cout << myObj.myNum << "\n";   
  cout << myObj.myString;   
  return 0;  
}

// -----------------------------------------------  
To access the class attributes (myNum and myString), use the \_\_.\_\_ on the object.

Methods are \_**functions**\_ that belongs to the class.

There are two ways to define functions that belongs to a class: \_\_**inside class def**\_ and \_**outside class def**\_.

Given the following example:

// -----------------------------------------------  
class MyClass {        // The class  
  public:              // Access specifier  
    void myMethod() {  // Method/function defined inside the class  
      cout << "Hello World!";  
    }  
};  
  
int main() {  
  MyClass myObj;     // Create an object of MyClass  
  myObj.myMethod();  // Call the method  
  return 0;  
}// -----------------------------------------------

You access methods just like you access attributes; by creating an object of the class and by using the \_\_\_.\_\_\_\_\_\_\_.

Given the following example:

// -----------------------------------------------  
class MyClass {        // The class  
  public:              // Access specifier  
    void myMethod();   // Method/function declaration  
};  
  
// Method/function definition outside the class  
void **MyClass::myMethod()** {  
  cout << "Hello World!";  
}  
  
int main() {  
  MyClass myObj;     // Create an object of MyClass  
  myObj.myMethod();  // Call the method  
  return 0;  
}

// -----------------------------------------------  
To define a function outside the class definition, you have to declare it inside the class and then define it outside of the class. This is done by specifying the name of the class, followed by the \_\_: : \_\_, followed by the name of the function.

A \_\_\_**constructor**\_\_\_\_\_\_\_ in C++ is a **special method** that is automatically called when an object of a class is created.

To create a \_\_**constructor**\_\_, use the same name as the class, followed by parentheses ().

The constructor has the same name as the class, it is always \_**public**\_\_\_, and it does not have any return value.

Just like functions, constructors can also be defined outside the class. First, declare the constructor inside the class, and then define it outside of the class by specifying the name of the class, followed by the scope resolution :: operator, followed by the name of the \_**constructor**\_ (which is the same as the class).

Given the following example:

// -----------------------------------------------  
class MyClass {  // The class  
  **public:**          
    // class members goes here  
};

// -----------------------------------------------  
Concerning classes and referring to the above example, the public keyword is an\_**access specifier**\_\_.

Concerning classes and referring to the above example, \_**access specifier**\_\_ define how the members (attributes and methods) of a class can be accessed.

Concerning classes and referring to the above example, the members are \_\_**public**\_\_ - which means that they can be accessed and modified from outside the code.

In C++, there are three access specifiers. One is \_**public**\_\_ where the members are accessible from outside the class.

In C++, there are three access specifiers. One is \_\_**private**\_\_ where the members cannot be accessed (or viewed) from outside the class.

In C++, there are three access specifiers. One is \_**protected**\_ where the members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

Given the following example:

// -----------------------------------------------  
class MyClass {  
  **public:**    // Public access specifier  
    int x;   // Public attribute  
  **private:**   // Private access specifier  
    int y;   // Private attribute  
};  
  
int main() {  
  MyClass myObj;  
  myObj.x = 25;  // Allowed (public)  
  myObj.y = 50;  // Not allowed (private)  
  return 0;  
}

// -----------------------------------------------

Referring to the above example, this program will run without error (T or F). **F**

By default, all members of a class are \_\_**private**\_ if you don't specify an access specifier.