1. Define and explain the following items related to the C++ programming language. Illustrate them with examples. Explain your examples.

1. lvalue vs. rvalue

Lvalue represents an object that has an address and occupies some identifiable location in memory whereas rvalue is an expression that does not represent an object occupying some identifiable location in memory.

Answer:

Example of lvalue.

Int i= 10;

The variable 'I' has an address in memory and is a lvalue,

But we cant do the following:

int i ;

10 = i;

Since 10 doesn't have an identifiable memory location it is an rvalue, and you cant assign the value of I to 10.

2. pointer vs. reference

Answer:

A pointer is a variable that holds the memory address of another variable, whereas a reference is an alias or another name for an already existing variable, normally regarded as a constant pointer. A pointer can be reassigned to hold another variable but a reference cannot be reassigned and must be assigned at initialization.

Pointer:

Int a = 10;

Int *p = &a;

Reference:

Int &ref = a;

3. typedef vs. typeid vs. typename

Answer:

Typedef allows the programmers to define new names for types such as int in C++, it is used to provide more clarity to the code and to make it easier to make changes to the underlying data types.

E.g.

typedef unsigned long ulong;

unsigned long a; and ulong b; would both create variables that hold unsigned long data.

Typeid operator would allow the type of an object to be determined at runtime.

E.g.

```
int i,j;

const type_info& ti1 = typeid(i);

const type_info& ti2 = typeid(j);

if (ti1==ti2) cout << “they are of the same type”;
```

typename is a keyword used when writing templates to specify that a dependent name in a template definition is a type.

E.g.

```
template <typename T>

const T& max(const T& x, const T& y)

{

 if (y < x)

   return x;

 return y;

}
```

4. explicit

Answer:

Using explicit keyword would prevent the compiler from using implicit conversion for a constructor, conversion function (since c++11).

Say we have a class String:

```
Class String {

Public:

        String( int n);

        String (const char *p);

};
```

If we do String a = ‘x’; the character ‘x’ would be implicitly converted to int and the String (int) constructor will be called.

So we need to have explicit keyword to the constructor to prevent implicit conversion

```
Class String{

        explicit String (int n );

        String(const char *p);

};
```

5. noexcept

Answer:

The noexcept operator performs a compile-time check that returns true if an expression is declared to not throw any exceptions.

noexcept(foo()) // would get 0 if it can throw an exception or 1 if it can never throw an exception, but it might yield false negative even if it cant throw an exception.

6. auto

Answer:

The auto keyword specifies that the type of the variable that is being declared will be automatically deducted from its initializer.

Int x = 4;

auto y = 5;

typeid(x).name()

and typeid(y).name() are both integer as the y is automatically deduced to be integer.

7. inline

Answer:

The compiler would place a copy of the code of that inline function at each point where the function is called at compile time.

Inline int foo()

You can make the function inline by using it as prefix to the function.

8. using declaration vs. using directive

Answer:

Using declaration can be used to introduce namespace members into other namespaces and block scopes or to introduce base class members into derived class definition whereas the **using** directive allows all the names in a namespace to be used without the namespace-name as an explicit qualifier.

e.g.

Using declaration

```
#include <stdio.h>

class B {

public:

  void f(char) {

    printf_s("In B::f()\n");

  }
```

```
};
class D : B {
public:
  using B::f;    // B::f(char) is now visible as D::f(char)
  void f(int) {
    printf_s("In D::f()\n");
    f('c');     // Invokes B::f(char) instead of recursing
  }
};
```

Using directive example:

```
namespace Foo
{
   void bar() {}
}

//Have to use Foo::bar()
Foo::bar();
using namespace Foo;
bar();
```

We can use the using namespace to import everything out of Foo namespace and don't have to write Foo::bar() to use the function.

2. (a) Write a C++ program that asks input from the keyboard and prints the result on the screen and writes it to a file. Use the C++ standard overloaded extraction >> (input) , and insertion << (output) operations. The question is: "What is your age?". Make your C++ program to compile and run.

Answer:

```
#include <iostream>

#include <fstream>


using namespace std;


int main()

{

   string s = "What is your age?";

   cout << s<<endl;

   string input;

   cin >> input;

   cout << input;

   ofstream MyFile("filename.txt");

   MyFile << input;

   MyFile.close();

}
```

I use string variable to store the answer as I didn't specify the users to type the age in number or words, it would then output it on the screen and write it to the file called filename.txt.

(b) After you get the program to work, make some purposeful mistakes to see how the development tools respond. For example, forget to include a header, forget to terminate a string, misspell some keywords, omit a semicolon ';' or a brace ('{' or '}'), etc. Inspect the error diagnostics from the point of view of a programmer. Give examples and explain some of good (helpful or informative) diagnostic messages. Give and explain examples of perhaps not so good (obscure, misleading,or superfluous) diagnostic messages.

Answer:

Forgetting a semicolon or brace would make the error message to show expected a ';' at the particular line which is helpful and informative.

Example of not so good one is such as misspelling an identified such as misspelling string as sring or String, the error message would only show identifier sring is undefined but never suggested (did you mean string) for the error message, if the user were to not know the actual name of the identifier he

wouldn’t be able to figure it out. Another example is that forgetting to include a header would make the identifiers to be undefined such as string cin etc, and it doesn’t suggest to include the headers.

3. Explain the following items concerning the C++ programming language:

1. What is a constructor? What is a destructor? Explain their purpose in C++ programs. What can we (generally) assume about the state of an object and its invariants after completing the execution of its constructor? And after executing its destructor?

Answer:

A constructor is a method that is automatically called when an object of a class is created, destructor on the other hand is a method that is automatically invoked when the object is destroyed. Their purpose in c++ is to initialize the class object and destroy the class object.

After an object is created, the memory is allocated and then the constructor runs. When the constructor runs to completion, the lifetime or the scope of the object begins, the object is also initialized with some default values.If the object goes out of scope or the lifetime of the object ends, the destructor function is called, in which the memory will be freed.

After the constructor runs, the object is said to have the properties, behaviour and structure of the class, which we can use the function of the class of the object.

After executioning its destructor, the object is destroyed and the memory of the object is freed.

2. Consider the overloading of assignment operator in C++. Explain why it is useful. When is it necessary to overload assignment? If we overload assignment, in what circumstances do we need to overload other operations, too?

Answer:

Overloading allows us to specify more than one definition for a function name or an operator in the same scope. It is useful to ensure that the code is clean and easy to understand by not having to use different function names for same kind of operations. The overloaded functions must have different types of arguments or a different number of arguments so that the overload resolution can differentiate between he different functions of the same name. If we have defined our own data types and we want to allow operation such as addition to our data type, we can have operator overloading to achieve that.

3. What different kinds of constructors does a C++ class typically need? Especially, what is a default constructor? What does the term "default" really mean here? What are the conditions under which it will be called? For what kinds of situations is the default constructor essential?

Answer:

C++ have three types of constructors namely default constructors, parametrized constructor, copy constructor, default constructor is the constructor which does not take any argument and has no parameter. The default means that the compiler will provide a default constructor implicitly even if we do not define a constructor explicitly. The default constructor will be called when an object of a class is created and the constructor is not defined explicitly in the class. The default constructor is significant when we declare an object with no argument list e.g. MyClass x; or new MyClass; or new MyClass();. When the derived class constructor does not explicitly call the base class constructor in its initializer list, the default constructor for the base class is called.

4. Compare C++ destructors to facilities in other programming languages that you know? Why don't they need to provide C++-style destructors? Or do they both need and (should) actually provide? Hint. Consider the finally block in Java and the C# using statement .

Answer: Java has no destructor because it is a garbage collected language and we cant really predict if and when the object will be destroyed (it will automatically reclaim any no longer being accessed memory), however we can define a close method and use it in the try_catch_finally clause in which we close it at the finally close to properly close any streams/sockets. C# has finalizer/destructor which is also used to perform final clean-up when a class instance is being collected by the garbage collector, it is automatically invoked by the garbage collector when it is no longer being used by the application.

4. Explore how C++ objects, often as instances of user-defined classes, can be created and destroyed. Consider the different possibilities: as either local or global variables/fields/elements, as temporary values in expressions, as passed arguments in function calls, or as heap-allocated dynamic "objects". In each case, explain how the lifetime ("life cycle") of the object is determined (either manually or automatically). Provide short illustrative examples, and explain them.

Answer:

As global variable c++ can be defined by the following, then it will be accessible to all the files that include the header.

In the Global_obj.cpp

Class obj;

In the header file Global_obj.h

Extern Class obj;

In Main.cpp

#include "global_obj.h"

You can create a local obj in the function call, and the life cycle of the object would end after the function call is finished.

```
Foo (){
  Class obj;
}
```

You can dynamically allocate and deallocate the objects this way

```
using namespace std;
int main()
{
        Class* a = new Class[4];
        delete [] a;
        return 0;
}
```

By having a pointer, we just need to delete the array of objects a then we can allocate and deallocate the objects dynamically.