Things you should be able to do in C++

1. Write simple classes and functions in C++.
2. Understand the difference between call-by-value and call-by-reference.
3. Understand function overloading - when each function will be called, when there is an ambiguity in definition (compile time errors).
4. Understand the use of friend functions - when they must be used, how can you create a workaround and not use them.
5. Appreciate the various meanings of const in C++, and know when to use them. (const object, const pointer, pointer to a const object, const member function, const reference).
6. Overload operators (what can be overloaded and what not).
7. Read programs using overloaded operators, by identifying which methods or independent functions are called.
8. Know how to use static, local, dynamic and temporary allocation, appreciating their properties and distinctive features.
9. Understand the properties of sub-objects (objects that are fields of other objects - order of initialization)
10. Use inheritance, method redefinition(overriding/hiding/virtual) and abstract classes in C++.
11. Write generic classes and functions in C++ (templates).
12. Explain what the automatically generated constructors, destructors and assignment operators do, when they are inadequate, and if so how they should be replaced.
13. Understand when constructors and destructors are called (order of calls to constructors/destructors).

1. What is a class?
   - 1.1. What are the differences between a struct and a class in C++?
   - 1.2. What is the difference between public, private, and protected access?
   - 1.3. For class Foo { }; what default methods will the compiler generate for you?
   - 1.4. How can you force the compiler to not generate them?
   - 1.5. What is a constructor initializer list?
   - 1.6. When must you use a constructor initializer list?
   - 1.7. What is a:
     - 1.7.1. Constructor?
     - 1.7.2. Destructor?
     - 1.7.3. Default constructor?
     - 1.7.4. Copy constructor?
     - 1.7.5. Conversion constructor?
   - 1.8. What does it mean to declare a...
     - 1.8.1. member function as virtual?
     - 1.8.2. member function as pure virtual?
     - 1.8.3. member function as static?
     - 1.8.4. member variable as static?
     - 1.8.5. constructor as static?
     - 1.8.6. destructor as static?

2. What is exception handling?
   - 2.1. What happens when an exception is thrown in C++.
   - 2.2. What happens if an exception is not caught?
   - 2.3. What happens if an exception is thrown from an object's constructor?
   - 2.4. What happens if an exception is thrown from an object's destructor?

3. What output does the following code generate? Why?
What output does it generate if you make A::Foo() a pure virtual function?

```
#include <iostream>
using namespace std;

class A {
     public:
          A() { this->Foo(); }
          virtual void Foo() { cout << "A::Foo()" << endl; }
};

class B : public A {
     public:
          B() { this->Foo(); }
          virtual void Foo() { cout << "B::Foo()" << endl; }
};

int main(int, char**)
{
     B objectB;
     return 0;
}
```

4.What output does this program generate? Why?

```
#include <iostream>
using namespace std;
class A {
     public:
          A()  { cout << "A::A()" << endl; }
          ~A() { cout << "A::~A()" << endl; throw "A::exception"; }
     };

class B {
     public:
          B()  { cout << "B::B()" << endl; throw "B::exception"; }
          ~B() { cout << "B::~B()"; }
     };

int main(int, char**)
{
     try {
          cout << "Entering try...catch block" << endl;
          A objectA;
          B objectB;

          cout << "Exiting try...catch block" << endl;
     }
     catch (char* ex) {
          cout << ex << endl;
     }
     return 0;
}
```

5. **Type conversions**

5.1. **Provide class methods which allow type conversion between the celcius and farenheit classes below. (Note: to convert from Celcius to Farenheit multiply by 1.8 then add 32, and to convert from Farenheit to Celcius subtract 32 then divide by 1.8.)**

5.2. **Provide a main routine demonstrating the following type conversions:**
farenheit to celcius
celcius to farenheit

```
class celcius {
     public:
          celcius(double c = 0): deg(c) {}
          ~celcius() {}
          void print() { cout << f << " degrees Celcius" << endl; }
     private:
          double deg;
}

class farenheit {
     public:
          farenheit(double f = 0): deg(f) {}
          ~farenheit() {}
          void print() { cout << f << " degrees Farenheit" << endl; }
     private:
          double deg;
}
```

6. **Type conversions**

6.1. **For the MyString class described below, create conversion methods that permit type conversion between type char * and instances of class MyString.**

6.2. **Using the methods you created in part 1, write a main routine which includes examples of:**
1. **A conversion from a variable of type char * to an instance of class MyString**
2. **A conversion from an instance of class MyString to a variable of type char ***

```
class MyString {
public:
   MyString(int size = 0) {
      if (size < 0) AllocatedSpace = 0;
      else AllocatedSpace = size;
      Text = new char[size];
   }
   ~MyString() { delete[] Text; }
   void SetString(char *str) { strncpy(Text, str, AllocatedSpace); }
   char *GetString() const { return Text; }
private:
   char *Text;
    int  AllocatedSpace;
};
```

7. **Testing, exceptions**
Show the output from the following program.

```
#include <iostream>
using namespace std;
void f1(double t);

int main()
{
   try {
     cout << "stage 1" << endl;
     f1(80);
     cout << "stage 2" << endl;
     f1(40);
     cout << "stage 3" << endl;
   }
   catch (int x) {
      cout << x << endl;
   }
   cout << "stage 4" << endl;
}

void f1(double t)
{
   cout << "Beginning f1" << endl;
   if (t < 50) throw 25;
}
```

8. **Function and method overloading.**
Clearly identify the ambiguities which prevent the compiler from successfully compiling the following code segment.

```
#include <iostream>
using namespace std;
int foo(int x = 0, char y = 'c');
char foo(int x, char y);
char foo(char *x, char y);
void foo(float x);
void foo(float x, int y = 0);
void foo(float x, char y);

int main()
{
     cout << foo(3, 'x');
     cout << foo("foo", 'x');
     cout << foo(3, 4);
     cout << foo(3);
     cout << foo('x');
     foo(3);
     foo(3, '4');
}
// Assume the function implementations follow below
```

Provide an alternative set of methods which solve the problems, and explain your solution.

9. An experimental psychologist has designed the following class in C++ to perform a (very) crude simulation of short-term memory:

```
class Brain {
public:
     Brain();
     void add_an_item(string s);
     void bump( );
     void display( ) const;
  private:
     static const int MAX = 7;
     vector<string> memory;
};
```

A Brain is initially empty. add_an_item adds a string to the memory. The memory can only contain up to seven strings. The addition of another string causes the first (oldest) string to be forgotten. A bump on the head causes the most recently added string to be forgotten. display displays the current contents of memory. Write the definitions of the interface functions.

10. What is the output of the following program?

```
#include <iostream>
using namespace std;

struct A {
     A() { cerr << "I'm the constructor of A - a member object\n"; }
     ~A() { cerr << "I'm the destructor of A - a member object\n"; }
};

struct B {
     B() { cerr << "I'm the constructor of B - a super-class\n"; }
     ~B() { cerr << "I'm the destructor of B - a super-class\n"; }
};

struct C : public B {
     A a;
     C() { cerr << "I'm the constructor of C - a sub-class\n"; }
     ~C() { cerr << "I'm the destructor of C - a sub-class\n"; }
};

int main()
{
     C c;
     return 0;
}
```

11. What is the primary value in using virtual functions within C++?

a) They prevent the base class function from ever being called.

b) They allow you to provide unique behavior for derived classes.

c) Since they are "virtual" they sometimes don't really exist.

d) Internet technologies.

12. In C++ what is the difference between a struct and a class?

a) There is no comparison. Structs are used in C while classes are used in C++.

b) Classes use less memory since it is newer technology.

c) You cannot implement functions when you use a struct.

d) The default access specifier.

13. What is the difference between a C++ pointer and a reference?

a) A reference is the entire object while a pointer is only the address of it.

b) References are used only in Visual C++.

c) The syntax used to access the object.

d) Pointers are simple address to the object while a reference uses the virtual table.

14. What does the term “default parameter” mean in C++?

a) It means that the compiler supplies default functionality that you get for "free".

b) A parameter that C++ automatically supplies a default value for.

c) Some C++ functions have all the parameters written for you by the compiler

d) A parameter based on primitive C++ variable types.

15. What is the point in making a destructor "virtual"?

a) There is no value.

b) C++ uses it for memory leak detection.

c) It causes the derived class destructor to be called upon delete.

d) A derived class destructor is always called, it just changes the order of calling.

16. If two functions within the same class have the same name what happens?

a) Either one will be called depending on the parameters.

b) Nether function will be called. This is dead code and a very difficult problem to debug.

c) Both functions will be called in order.

d) The base class function of the same name will be called.

17. What is the primary purpose of a default constructor?

a) To allow multiple classes to be used in a single program.

b) To copy an actual argument to a function's parameter.

c) To initialize each object as it is declared.

d) To maintain a count of how many objects of a class have been created.

18. Suppose that the foo class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two foo objects?

a) The automatic assignment operator is used

b) The copy constructor is used

c) Compiler error

d) Run-time error

19. Using the following declarations, which of the following assignment statements are legal?

class A {...} ; class B: public A {...};

A a; B b, b1, b2;

a) a = b;

b) b = a;

c) b1 = b2;

d) Both (a) and (b) are legal, but not (c).

e) Both (a) and (c) are legal, but not (b).

f) Both (b) and (c) are legal, but not (a).

20. Consider the assignment statement a=b; (with the variable declarations in the previous question). Which answer is true?

a) The assignment statement is illegal.

b) The assignment statement activates the A assignment operator.

c) The assignment statement activates the B assignment operator.

d) The assignment statement activates both A and B assignment operators.

21. Consider the assignment statement b=a; (with the variable declarations in question 19). Which answer is true?

a) The assignment statement is illegal.

b) The assignment statement activates the A assignment operator.

c) The assignment statement activates the B assignment operator.

d) The assignment statement activates both A and B assignment operators.

22. Consider the declarations in question 19. Suppose there are two functions: f has an argument of type A and g has an argument of type B.

Which statement is correct?

a) Both f(b) and g(b) are legal function calls.

b) f(b) is legal, but g(b) is not legal.

c) f(b) is not legal, but g(b) is legal.

d) Neither f(b) nor g(b) is a legal

23. In C++ what does the operator over loading means.

a) Giving new meaning to existing C++ operators

b) Defining functionality of existing C++ operator for user define objects.

c) Defining new operators.

24. Which statement is true of "pure virtual" functions?

a) They force you to derive another class from the base class which contains the function.

b) They contain no code.

c) Neither of these

d) Both of these

25. When should you use a const reference parameter?

a) Whenever the data type might be many bytes.

b) Whenever the data type might be many bytes, the function changes the parameter within its body, and you do NOT want these changes to alter the actual argument.

c) Whenever the data type might be many bytes, the function changes the parameter within its body, and you DO want these changes to alter the actual argument.

d) Whenever the data type might be many bytes, and the function does not change the parameter within its body.

26. If there is one class template which has one static member variable, that static variable will belong to:

a) Every instance of class template will share the same copy of static variable.

b) Every instance of class template will have its own copy of static variable.

c) Compilation error