Compilers Lab 2

Melanie Palmer

September 17, 2015

Problem 1.

```
using namespace std;
```

Problem 2.

Since there was no name space specified, I will assume this is the default namespace. If no name space is specified it is considered to be in the global name space. To call it in our namespace we write the below:

```
::myFunct();
```

Problem 3.

```
int A::myMethod(){
        return 3;
}
```

Problem 4.

The difference between the two is that A is a class, while B is a struct. In other languages this may be pretty different, but in C++ this will only change what the default privacy setting is. In a class it will default the class to private, but a struct will default to public.

Problem 5.

Funtion 1 sets x to a string literal which means it stores the string on the stack. This means it will go away after when the function returns.

Function 2 sets x to a new stirng object which means it stores it on the heap. This means it will be fine after the function returns.

Problem 6.

In C++, NULL is still available, but it is just the integer 0. nullptr is the correct way to donate a pointer with an address of 0.

Problem 7.

The code below gets an integer in two different manners. One with the this keyword, the other just with a direct return. They both returned the same integer.

```
//
//  main.cpp
//  tester
//
//  Created by Melanie Palmer on 9/17/15.
//  Copyright (c) 2015 \mpalmer. All rights reserved.
//

#include <iostream>

class A {
    private :
        int x;
    public :
    A(int a) : x(a) {}

    int getX1(){
        return this->x;
    };
    int getX2(){
        return x;
    };
};

int main(int argc, const char * argv[]) {
    // insert code here...

    ::A x (9);

    std::cout << "X1 - " << x.getX1();
    std::cout << "\nX2 - " << x.getX2();
    return 0;
}
```

Problem 8.

The code below dictates a class A. It has 3 public variables: an int, a bool, and a string. The next part is a constructor for A which takes in 3 arguments. It's important to know that it fills in the values starting with val3(arg3) and working up to val1(arg1)

```
class A {
        public :
                int val1 ; bool val2 ; string val3 ;

        A(int arg1 , bool arg2 , string arg3): val1(arg1),
         val2(arg2), val3(arg3) {};
};
```

Problem 9.

It will print A Const, because it will start at any default constructors for its parent classes.

Problem 10.

```
delete a
```

Problem 11.

This is a destructor. It automates what to delete. You don't need to call this as it gets called automately when the instance is out of its scope.

Problem 12.

```
class A {
        private :
                classTy c ;
         public :
                A() : c(ROOT) {};
                A(classTy arg) : x(arg) {};
        };

class B : public A {
        public :
                B() : c(BCLASS) {};
 };
```

Problem 13.

If we excute this in another fucntion it will print A thing because A is not a virtual method.

Problem 14.

This will correctly display B thing.

Problem 15.

You need it to be virtual when an derrived class will be implmeneting it. The first that comes to my mind is toString().

Problem 16.

```
class A {
        public :
                void doThings() { thing(); }
                virtual void thing() =0;
};
```

Problem 17.

```
vector<int> i;
```

Problem 18.

It would fail as it was thinking it would have integers, but it's being called with floats.

Problem 19.

Very bad things would happen. Seriously though, this doesn't account for the fact that some types simple cannot be added or it could cause a lot of problems if they are added together.

Problem 20.

```
Template <typename T>
class Tree {
        private:
                tree<T> left;
                tree<T> right;
                T val;
        public:
                Tree() : left(nullprt), right(nullprt) {};
                Tree(Tree<T> left, Tree<T> right, T val):
                        left(left), right(right), val(val) {};
                virtural 'Tree() {
                        delete left;
                        delete right;
                }
}
```

Problem 21.

f1 does a pass by value A. f2 does a pass by value of a pointer A. f3 does a pass by reference of the address of A. (creates a pointer).

Problem 22.

A constant argument is a variable that the compiler recongnizes as constant/unchangeable. Its simislar to a define in C, but its for actual variables and the compiler recognizes it.

A const return lets the compiler know that this answer should not be changed. it could be very useful in circumstances where the function to attempt to alter or change the value without realizing it, such as arrays and strings.

Problem 23.

1. vector
2. stack
3. queue
4. deque
5. map

Problem 24.

1. You get a vector of A objects - you add an object
2. You get a vector of A object pointers - you add a pointer
3. You get a vecotor of by reference pointers of A. - you add the address of an object

Problem 25.

```
#include <vector>

int main(int argc, char *argv[])
{
  vector < vector <int >>  vec 1;

  for(vector < int > e : 1){};

vector<vector<int> >::iterator outer;
vector<int >::iterator inner;

for(outer = vec.begin();outer!=vec.end();++outer) {
    for(inner = (*outer).begin();inner!=(*outer).end();++inner) {

    }
}
```

```
  for ( auto it : vec ){};

  for( auto it = vec.begin(); it != vec.end(); i++){};
}
```

Problem 26.

When you go to remove the second element it will fail because it was already removed the first time. This is because they were shared pointers to a.

It is useful as you may want two things looking at the same object. It could be very useful for updating functions on completion of another (like a flag), but as we saw they can be dangerous as you could remove or alter everything looking at it without meaning to.

Problem 27.

This is legal because the * operator has been overloaded to act just like a dereference to a normal pointer.