Congratulations! You passed!

TO PASS 80% or higher

Keep Learning

GRADE
90%

Week 4 Quiz

LATEST SUBMISSION GRADE

90%

1. Which one of the following is NOT true?

1 / 1 point

- [✓ C++ allows a local variable to be declared in main() with an unknown type that can be defined when the program is executed.]
- C++ allows a member variable to be declared in a user-defined class with an unknown type that can by defined when an object of that class is created.
- C++ allows a variable to be declared in a user-defined member function of a user-defined class that can be defined when the function is called.
- C++ allows a variable to be declared in a user-defined function with an unknown type that can be defined when the function is called.

✓ **Correct**
Even though C++ allows functions and classes to use templated types that are defined when the function is called or an object of that class is created, every variable must have a type known at compile time.

2. Suppose you want to create a vector of integers. Which of the following creates an instance of the std::vector class that can contain integers?

1 / 1 point

- int<std::vector> v;
- int v[256];
- int *v;
- [✓ std::vector<int> v;]

✓ **Correct**
This correctly creates a std::vector using "int" as the template type of the elements of the vector.

3. Which of the following will generate an error at compile time?

1 / 1 point

- std::vector<char[256]> v;
- std::vector<double> v;
- std::vector<std::vector<int>> v;
- [✓ std::vector v;]

✓ **Correct**
This will generate a compile-time error because the compiler does not know what type should be used for the elements of the std::vector, and every variable (including v) has to have a type at compile time. You have to supply a type as the template parameter for the elements of the std::vector.

4.

1 / 1 point

```cpp
template <typename Type>
Type max(Type a, Type b) {
    return (a > b) ? a : b;
}
```

Which one of the following exampled is a proper way to call the max function declared above in template form?

- <Type = double>max(5.0,10.0)
- [✓ max(5.0,10.0)]
- max<double>(5.0,10.0)
- max<Type = double>(5.0,10.0)

✓ **Correct**
Whereas a class needs to explicitly identify the type, a templated function does not need to explicitly identify the type(s) used if the type of its arguments can be sufficiently matched to the templated types used in the function declaration.

5. 1 / 1 point

```
 1  template <typename Type>
 2  Type max(Type a, Type b) {
 3      return (a > b) ? a : b;
 4  }
 5
 6  class Just_a_double {
 7  public:
 8      double num;
 9  };
10
11  int main() {
12      Just_a_double a,b;
13      a.num = 5.0;
14      b.num = 10.0;
15      ...
16  }
```

Given the above code, which one of the expressions below, if used at line 15, will compile and not generate a compile error?

- ○ max("five",10.0)
- [✓ max(a.num,b.num)]
- ○ max(a,10.0)
- ○ max(a,b)

✓ **Correct**

Both arguments to max() are the same type and both can be compared using the greater-than operator.

6. **Which one of the following properly declares the class RubikCube derived from the base class Cube?** 1 / 1 point

- ○ class Cube : public RubikCube {...};
- ○ class Cube(RubikCube) {...};
- ○ class RubikCube(Cube) {...};
- [✓ class RubikCube : public Cube {...};]

✓ **Correct**

This correctly derives RubikCube as a specialization of base class Cube.

7. 1 / 1 point

```
class Pair {
public:
    double a,b;
    Pair(double x, double y) { a = x; b = y; }
};
```

If a class equalPair is derived from the above base class (but specializes it by adding a single boolean "isequal" member variable) then which one of the options below is a proper declaration of a constructor for equalPair?

(As a side note: Although the member variables are of type double, for the sake of this question, we are not concerned about making approximate comparisons of floating-point types, only exact comparisons. Usually, in practical usage, when you compare floating-point values, you should write a function for *approximate* comparison. That is, you should allow numbers to be considered equal if they have a very small absolute difference, even if they are not exactly the same.)

○
```
equalPair(double a, double b) {
    isequal = (a == b);
}
```

[✓
```
equalPair(double a, double b) : Pair(a,b) {
    isequal = (a == b);
}
```
]

○
```
equalPair(double a, double b) {
    this->Pair(a,b);
    isequal = (a == b);
}
```

○
```
equalPair(double a, double b) {
    Pair(a,b);
    isequal = (a == b);
}
```

✓ **Correct**

Correct. The class Pair does not have a default constructor so it needs to be explicitly constructed using Pair(a,b).

8. 0 / 1 point

```
class Pair {
private:
    double a,b;
};

class equalPair : public Pair {
private:
    bool isequal;
public:
    int status();
}
```

When the function status() is implemented, which variables will it have access to?

- No member variables of either equalPair or Pair.
- Just the member variables a,b of Pair.
- Just the member variable isequal of equalPair.
- [✓ Both the member variables a,b or Pair and isequal of equalPair.]

! **Incorrect**
The derived class equalPair still has access to its own private members, but even though Pair is indicated as a public base class, the derived class equalPair does not have access to the private members of Pair.

9.

1 / 1 point

```
1 class Just_a_double {
2 public:
3     double a;
4 
5     Just_a_double(double x) : a(x) { }
6     Just_a_double() : Just_a_double(0) { }
7 }
```

Which constructors, if any, compile properly?

- The constructor on line 5 results in a compiler error but the constructor on line 6 compiles properly,
- The constructor on line 5 compiles properly, but the constructor on line 6 results in a compiler error.
- Both constructors on lines 5 and 6 result in compiler errors.
- [✓ Both constructors on lines 5 and 6 compile properly]

✓ **Correct**
The initializer lists allow both member variable constructors as well as other declarations of the class constructor.

10. C++ is ...

1 / 1 point

- [✓ ... a great language for programming data structures.]
- ... the greatest language for programming data structures ever!
- ... meh.

✓ **Correct**