CS 162-010 Exam II Spring 2022

Part I: True (T) / False (F), put T/F on the answer sheet (28 pts, 2 pts each)

1. In C++, you cannot create new operators (such as **).

2. Neither destructor nor constructor(s) of the base class is inherited into its derived class(es).

3. Recursive functions usually use more memory space than non-recursive functions.

4. If you have a class template declared, and you instantiate it in your program twice (one with an integer, one with a string), the compiler creates 2 versions of the class.

5. The **`catch`** block is the group of statements that handle an exception, and in a **`try`** block, the **`throw`** statement is always executed.

6. Vector is an array that can grow and shrink in length while the program is running, where **`.size()`** refers to the number of elements in the vector.

7. When using objects polymorphically, C++ will automatically use the correct destructor, regardless of whether it is declared as virtual.

8. A class containing a single virtual function is an abstract class.

9. For the **`std::vector`** class, both the **`at()`** function and the **`operator[]()`** function generate exceptions when trying to access an element beyond the bounds of the vector.

10. If the member variables in a base class are marked as **`private`**, the derived class may directly access those variables.

11. Recursive merge sort has a constant space complexity O(1).

12. When using a stack, the order of inserting and removing elements are the same.

13. You can directly access the nth node of a linked list by [].

14. In a program that often needs to insert an element into the front of the container, we'd better choose a linked list over an array.

Part II: Multiple Choices. Put your answers on the answer sheet (72 pts, 3 pts each)

1. Given a class C that derives from a class B that derives from a class A, when an object of class C is created, in which order are the constructors called?
   A. A, B then C
   B. C, B then A
   C. depends on how the code is written for the constructors
   D. unable to determine

2. You should make a function a virtual function if _____.
   A. every class that is derived from this class use all the member functions from this class
   B. every class that is derived from this class needs to re-define this function
   C. that function is an operator
   D. only in the derived classes

3. What would be printed by the following snippet:

```
vector <int> array(10);
int i;
try {
    cout << "Grabbed i." << endl;
    i = array.at(100000);
}
catch (...) {
    cout << "Caught an exception." << endl;
}
```

   A. `Caught an exception.`
   B. `Grabbed i.`
   C. `Grabbed i.`
      `Caught an exception.`
   D. Nothing would be printed by this snippet

4. You are experimenting with a sorting algorithm which has average-case runtime complexity $O(n^3)$, and you've found that, on average, this algorithm runs in 10ms on arrays of size 10,000. How long, then, would you expect insertion sort to run on arrays of size 20,000?
   A. 40 ms
   B. 80 ms
   C. 90 ms
   D. 100 ms

5. Given the following function,

```
int F(int a, int b){
     if (b == 0)
          return a;
     return F(b, a % b);
}
```

   What is the output of the following statement?

```
cout << F(60, 36) << endl;
```

   A. 6
   B. 12
   C. 3
   D. No output, the program crashes

6. A catch block that expects an integer argument will catch _____.
   A. all exceptions
   B. all integer exceptions
   C. any exception value that can be converted into an integer
   D. None of the above

7. Using inheritance allows us to _____.
   A. eliminate duplicate code
   B. make our class more modular
   C. implement an "is-a" relationship
   D. all of the above

8. What is the runtime complexity of the following snippet of code?

```
double v = 0;
for (int i = 0; i < n; i++) {
    for (int j = n; j > 0; j/=2) {
        v += i * j;
    }
}
for (int i = 0; i < n; i++) {
    v -= i * i;
}
```

   A. O(n)
   B. O(n logn)
   C. $O(n^2+n)$
   D. $O(n^2)$

For Questions 9-12, consider the template class below:

```
template <class T>
class Array{
private:
   T* array;
   int size;
public:
   Array(int size);
   ~Array();
   T& at(int i);
};
```

9. What is the purpose of the template parameter **T**.
   A. It allows the class to be agnostic about the type of the array, which specified by **T**.
   B. It represents a specific class from the C++ Standard Template Library
   C. It's a C++ keyword that must be used for all template definitions
   D. It has no purpose

10. What would be the correct implementation of a constructor for the Array class that implemented enough space for an array of **`size`** elements of the template parameter type?

A.
```
template <class T>
Array<T>::Array(int size) : size(size){
     this->array = new T[size];
}
```

B.
```
template <class T>
Array::Array(int size) : size(size){
     this->array = new T[size];
}
```

C.
```
template <class T>
Array<T>::Array(int size) : size(size){
     this->array = new <T>[size];
}
```

D.
```
template <class T>
Array::Array<T>(int size) : size(size){
   this->array = new T[size];
}
```

11. What would be the proper way to use the **`Array`** class above to create an array of 10 integers?

A. `Array<int> arr(10);`

B. `Array<int, 10> arr;`

C. `Array arr(10);`

D. `Array arr(int, 10);`

12. How would you implement the function body of the destructor to properly free the memory allocated by the **`Array`** class?

A. `delete [] array;`

B. `delete array;`

C. `delete <int> [] array;`

D. `delete <T> [] array;`

For Questions 13-18, refer to the following three classes:

```
class Animal {
private:
    string name;
public:
    Animal(const char* name) : name(string(name)) {}
    ~Animal() {}
    string get_name() { return name; };
    virtual string says() = 0;
};

class Cat : public Animal {
public:
```

```
    Cat(const char* name) : Animal(name) {}
    string says() { return "Meow"; }
};

class GrumpyCat : public Cat{
public:
    GrumpyCat(const char* name) : Cat(name) {}
    string says() { return "Awful"; }
};
```

13. What would be printed by the following snippet?

```
GrumpyCat g("grumpycat");
cout << g.get_name() << " says " << g.says() << endl;
```

A. **grumpycat says Awful**
B. **grumpycat says**
C. **grumpycat says Meow**
D. This snippet would not compile, so nothing would be printed.

14. What would be printed by the following snippet?

```
Animal* a = new Cat("cat");
cout << a->get_name() << " says " << a->says() << endl;
```

A. **cat says Meow**
B. **cat says**
C. **cat says Awful**
D. This snippet would not compile, so nothing would be printed.

15. What would be printed by the following snippet?

```
Animal* a = new Animal("animal");
cout << a->get_name() << " says " << a->says() << endl;
```

A. **animal says**
B. **says**
C. This snippet would compile, but it would not print anything.
D. This snippet would not compile, so nothing would be printed.

16. What can we say about the memory usage of the following snippet?

```
Animal* a = new Cat("cat");
delete a;
```

A. This snippet correctly frees all of the memory it allocates.
B. This snippet does not free all of the memory it allocates.
C. This snippet would not compile, so we can't say anything about its memory usage.
D. None of the above

17. What term best describes the **`Animal`** class?
    A. Abstract class
    B. Pure virtual class
    C. Virtual class
    D. None of the above

18. What term best describes the **`Animal`** class's **`says`** function?
    A. Pure virtual function
    B. Virtual function
    C. Abstract function
    D. None of the above

For Question 19-24, refer to the following three classes

```
class Base {
public:
    virtual string print() const{
        return "This is Base class";
    }
    virtual ~Base() { cout << "Base destructor" << endl;}
};

class Derived : public Base {
public:
    virtual string print() const {
       return "This is Derived class";
    }
    ~Derived() { cout << "Derived destructor" << endl;}
};

void describe (Base& p){
     cout << p.print() << endl;
}
```

19. What would the output of the following?

```
int main(){
     Base b;
}
```

    A. Nothing
    B. **`This is Base class`**
       `Base destructor`
    C. **`This is Base class`**
    D. **`Base destructor`**

20. What would the output of the following?

```
int main(){
  Derived d;
}
```

A. Nothing
B. **This is Derived class**
   Derived destructor
C. **Derived destructor**
   Base destructor
D. **Derived destructor**

21. What would the output of the following statement? (Assume **b** is a **Base** class object):

```
describe (b);
```

A. **This is Base class**
B. **This is Derived class**
C. This snippet would compile, but nothing would be printed.
D. This snippet would not compile, so nothing would be printed.

22. What would the output of the following statement? (Assume **d** is a **Derived** class object)

```
describe (d);
```

A. **This is Base class**
B. **This is Derived class**
C. This snippet would compile, but nothing would be printed.
D. This snippet would not compile, so nothing would be printed.

23. What would the output of the following statement? (Assume **d** is a **Derived** class object)?

```
Base* b_ptr = &d;
describe (*b_ptr);
```

A. **This is Base class**
B. **This is Derived class**
C. This snippet would compile, but nothing would be printed.
D. This snippet would not compile, so nothing would be printed.

24. What would the output of the following statement? (Assume **d** is a **Derived** class object)?

```
Derived* d_ptr = &b;
describe (*d_ptr);
```

A. **This is Base class**
B. **This is Derived class**
C. This snippet would compile, but nothing would be printed.
D. This snippet would not compile, so nothing would be printed.