CS162 Lecture 21

Exam 2 Practice
True/False

1] Pointers to a base class may be assigned the address of a derived class object

2] In C++ polymorphism is very difficult to achieve unless you also use inheritance

3] You can overload operators as a friend function

4] The standard template library (STL) vector and list classes are examples of sequential containers.

5] Nodes in a linked list are stored in contiguous memory.
Slide 2 answers

1] True -- This is how different child classes can be stored in one array!

2] True -- Polymorphism is: determining a virtual function’s implementation at runtime with late binding. You have to use inheritance because it is declared in the abstract base class and implemented in the child classes.

3] True, overloading operators is like any other function, you just can’t make new operators, and there are some operators that cannot be overloaded (Ex: the ‘.’ operator)

4] True -- sequential (vector, LinkList, Array), interface (Stack, Queue), or relational (Map, database)!

5] False
True/False

1] C++ enables you to use the friend keyword to declare friend functions and friend classes for a class so these functions and classes can access the class’s private members.

2] If a function is declared as: returnType funcName(paramList) throw (type), this function can only throw the exception of the specified type.

3] A template class allows the class to be used with different data types.

4] If an exception is not caught, it is stored for later use.

5] The following code declares a vector of characters:

   vector characters<char>;
Slide 4 answers

1] True

2] True

3] True

4] False -- To store an exception for later use you have to catch it!

5] False -- correct syntax is: vector<char> characters; the vector container is what needs a type parameter!
True/False

1] Destructors are not inherited into the derived class.

2] The assignment operator is inherited from the base class.

3] If a function throws an exception, it must be caught inside that function.

4] In a try block, the throw statement is always executed.

5] The catch block is the group of statements that handle an exception.

6] An object of a base class type can be passed to a function that takes an object of the derived class type.
Slide 6 answers

1] True -- The big 3 are not inherited, if you don’t implement they are automatically generated just link for non-inheriting classes

2] False -- The big 3 are not inherited, if you don’t implement in the child they are automatically generated just like for non-inheriting classes

3] False -- The exception is thrown by the function in the try() block! It is caught by the catch() block outside the function

4] False -- Hopefully the function doesn’t throw! try/catch is to handle if it does

5] True -- see #3

6] False -- You can pass children (derived) classes to functions that are expecting parent (base) classes as input!
Multiple Choice

To add an int value 5 to a vector v of integers, use ______.

A] v.append(5);
B] v.insert(5);
C] v.add(5);
D] v.push_back(5);
Answer: D

Append is an array operation!
Multiple Choice

Who can access private data in a class?

A] classes derived from the class
B] friends of the class
C] everyone
D] A and B
E] no one
Answer: B

friends of the class -- This is what declaring a friend is for!
Multiple Choice

Which is the proper way to declare an iterator for an STL container?

A] list<int>::iterator listIterator;
B] <int>list::iterator listIterator;
C] list::iterator<int> listIterator;
D] list::iterator listIterator<int>;
Answer: A

`list<int>::iterator listIterator;`

Our **list container** is of type `int`, we are using the **list class iterator** implementation, and we are calling the resulting **iterator object** ‘listIterator’.
Multiple Choice

Which of the following statements are true?

A] A custom exception class must always be derived from class exception.

B] A custom exception class must always be derived from a derived class of class exception.

C] A custom exception class is just like a regular class in C++.

D] A custom exception class must always be derived from class runtime_error.

E] Runtime Error
Answer: C

Custom exception classes are like any other custom class--it could be a parent, child, or standalone, depending on what/how we implement as other classes!

also--C is a better guess than runtime error!
Multiple Choice

Which of the following is a pure virtual function?

A] virtual double getArea()

B] double getArea() = 0;

C] virtual double getArea() = 0;

D] virtual double getArea() { };

Answer: C

You have to have the ‘virtual’ keyword in your base class to be virtual, even though you don’t need the keyword in your child class.

You have to have the ‘= 0;’ in your parent to be pure virtual!

Because pure virtual function declarations only appear in the abstract base class, we know we need both the ‘virtual’ keyword and ‘= 0;’ ending!
Multiple Choice

Suppose Circle and Rectangle classes are derived from Shape and you declared

```cpp
void displayShape(Shape S) {
    cout << S.toString() << endl;
}
```

Which of the following function calls is incorrect?

A] displayShape(Rectangle(2, 3));

B] displayShape(Shape(“black”, true));

C] displayShape(string());

D] displayShape(Circle(5));
Answer: C

string is neither a Shape nor a child class of shape!
Multiple Choice

Which is the correct way to tell the compiler that the class being declared (ChildClass) is derived from the base class (BaseClass)?

A] class ChildClass::public BaseClass

B] class ChildClass::public BaseClass

C] class ChildClass childOf public BaseClass

D] class ChildClass derived BaseClass
Answer: B

- :: is the scope resolution operator! Use to call non-inherited functions!
- derived is not a keyword!
Multiple Choice

Given a base class with at least one public member function, how many child classes can redefine that member function?

A] 1
B] 0
C] all of them
D] none of the above
Answer: C

All child classes can redefine member functions of the base class!
Multiple Choice

If the member variables in the base class are listed as protected, then who can access or modify those variables?

A] members of the base class

B] members of the derived class

C] outside the base or derived classes

D] A and B

E] All of the above
Answer: D

A base class can access its own members, and making them protected allows the child class access as well, despite keeping them private to other non-child classes!
Multiple Choice

If a base class has public member functions that are not listed by a derived class, then these functions

A] are not available to the derived class
B] are inherited unchanged in the derived class
C] are private to the derived class
D] do not exist in the derived class
Answer: B

Public member functions that are not changed in the derived class are inherited as-is!
Multiple Choice

If you have a copy constructor in the base class, but to not have a copy constructor for the derived class, then

A] you will have a syntax error

B] a copy constructor for the derived class is automatically created for you

C] you cannot use pointer variables

D] the default constructor is used
Answer: B

The big 3 are always automatically generated when not implemented, for inheriting and standalone classes!
Multiple Choice

Given a class A that derives from a class B that derives from a class C, when an object of class A goes out of scope, in which order are the destructors called?

A] C, B, then A

B] A, B, then C

C] unable to determine

D] depends on how the code is written for the destructors
Answer: B

A is child of (derives from) B; B is child of C.

Grandparent C → Parent B → Child A

A goes out of scope, calling A’s destructor.

A’s destructor calls parent destructor B

B’s destructor calls parent destructor C.

So: A, B, then C.
Multiple Choice

If the Pet class had a non-virtual member function named print, and a pointer variable of that class is pointing to a Dog object, then the code `Ptr->print();` calls

A] the base class print function

B] the derived print function

C] both the derived and base print functions

D] it causes a run-time error
Answer: A

The function is *non-virtual*, meaning it is called on the pointer type (base class), not the object type (child class)
Multiple Choice

Polymorphism refers to

A] the ability to change the behavior of a function at runtime.
B] overriding base class functions.
C] overloading functions
D] none of the above
Answer: A

“Polymorphism is using Late Binding to define the behavior of a virtual function at runtime”

“Polymorphism is using Late Binding to define the behavior of a virtual function at runtime”

“Polymorphism is using Late Binding to define the behavior of a virtual function at runtime”
Multiple Choice

In order to tell the compiler to wait to decide which version of a function to use, you must precede the function declaration in the base class with the keyword

A] operator
B] friend
C] virtual
D] derived
Answer: C

“Polymorphism is using Late Binding to define the behavior of a virtual function at runtime”

“Polymorphism is using Late Binding to define the behavior of a virtual function at runtime”

“Polymorphism is using Late Binding to define the behavior of a virtual function at runtime”
Multiple Choice

Which of the following operations do forward iterators have?

A] Overloaded operator+ to add an int value to the iterator to move the place the iterator points forward by the argument number of elements.

B] Overloaded operator* to multiply the iterator by an int value to move the place the iterator points by a number of elements equal to the argument.

C] Overloaded operator++ to move the place the iterator points forward by one element.

D] Overloaded operator-- to move the place the iterator points backward by one element.
Answer: C

A] No - this operator is not commonly overloaded for iterators

B] No - this operator is not commonly overloaded for iterators

C] Yes - This is a forward indexing behavior

D] No - This is a backward indexing behavior, would be present in backward and bidirectional iterators.