

Final Exam Review

*** I based this document off of Lecture W16 L30. My answers and examples are from the book and previous lecture slides. I do think that the book provides some great chapter review questions. I would highly suggest reviewing the book's chapter review questions as well as the coding examples. ***

1. What is abstraction?

A general model of something. It is a definition that includes only the general characteristics of an object without the details that characterize specific instances of the object.

2. How is abstraction used in software development?

When using an object or a function, we are using abstraction. You do not have to know the algorithm it uses to compute the result it returns.

3. What is an abstract data type (ADT)?

This is any data type whose implementation details are kept separate from the logical properties needed to use it. Usually, this is used to refer to a data type made by the programmer. For example, classes are an abstract data type.

4. Describe procedural programming.

A method of writing software centered on the procedures, or functions, that carry out the actions of the program. Basically, function focused programming.

The program's data is normally stored in variables and is separate from these procedures.

CON: a program specification might change, which will require the format of the data to change as well as the data structure.

5. What is an object?

A software entity that combines both data and the procedures that work with it in a single unit. Its data items are referred to as its attributes and are stored in member variables. The procedures that an object performs are called member functions.

6. Describe object-oriented programming.

A method of programming that is centered around objects that encapsulate both data and the functions that operate on them.

7. Explain function overloading.

Two or more functions with the same name, but have different signatures

- i. The signature of a function is the function name and the data type parameters. This does not include the return type.
- ii. An easier way to describe an overloaded function, is to simply say that the parameter lists are different.

8. Pass by reference vs pass by value

When passing by reference, you are passing the original data type through the function. The function will be able to modify the original data type.

When passing by value, a copy of the original data type is made. Therefore, you cannot change the original. The function can only change the copy.

9. What are enumerated data types?

A programmer-defined data type whose legal values are a set of named integer constants

All the values in an enumerated data type have an associated integer.

- i. ex.) `enum People {Bill, Tom, Jerry};` // Jerry is associated with 2

10. When can an array be initialized?

When an array is defined it can be initialized.

- i. ex.) `int days[NUM_MONTHS] = {31, 28, 30};`

If the array has an initialization list, there is no need to specify the size.

- ii. ex.) `int days[] = {31, 28, 30}`

11. How can you copy one array to another?

To do this, you should use a for loop.

- i. ex.)

```
int arrayA[SIZE] = {1,2,3};
int arrayB[SIZE] = {5,6,7};
for ( int i = 0; i < SIZE; i++){
    arrayA[i] = arrayB[i];
}
```

12. How can you compare arrays?

To compare arrays you cannot just use `==`. Instead you must take the same approach as you would to copying an array to another. You must compare their individual elements.

- i. ex.)

```
const int SIZE = 5;
int arrayA[SIZE] = { 5, 10, 15, 20, 25 };
int arrayB[SIZE] = { 5, 10, 15, 20, 25 };
bool arraysEqual = true; // Flag variable
int count = 0;           // Loop counter variable
```

```
// Determine whether the elements contain the same data
while (arraysEqual && count < SIZE){
    if (arrayA[count] != arrayB[count]){
        arraysEqual = false;
        Count++;
    }
}
```

13. Demonstrate how to initialize a vector

First, make sure you have the `<vector>` library included

`vector<int> numbers(10);` // int is where you would put the data type and numbers is the name of the vector. 10 just means that numbers is a vector of 10 ints.

`vector <int> numbers(10, 2);` // numbers is a vector of 10 ints and each element is initialized to 2

`vector<int> numbers {1, 2, 3, 4, 5};` // when a vector is initialized this way each number is a value of an element. The first number does not tell us the size of the vector

You can also initialize a vector with the same elements as another vector.

- i. ex.) `vector<int> set2(set1);` // set2 has the same values as vector set1

14. How can you access the elements in a vector?

Like an array, you can use a for loop. It may be easier to use a ranged based for loop.

- i. ex.)

```
vector<int> numbers(5);
for (int &val : numbers){
    cout << val << " "; // val represents each element of the vector
}
```

If you need to add an element to a vector, but it is already full you can use `push_back`

- ii. ex.) `numbers.push_back(25);` \\ 25 represents the value not the element position

Using the `.at()` function, will tell you the value of the element located at the position you enter.

- iii. ex.)

```
vector<int> numbers {5, 6, 7};
int x = numbers.at(2); // x is now equal to 7
```

15. How can you remove elements from a vector?

To remove the last element of a vector, use `pop_back()`

- i. ex.) `numbers.pop_back();` // The size of the vector is now one element smaller.
- ii. The `pop_back` function is void, so you cannot return the value of the element that is being removed

If you need to clear all the contents of a vector use `.clear()`

If you want to see if a vector is empty use `.empty()`. This function will return true if the vector is empty.

16. Explain how a ranged-based for loop works.

This type of loop iterates once for each element in an array or vector. Each time the loop iterates, it copies an element from the array to a variable. See 14a for an example.

17. Explain data hiding and encapsulation.

Data hiding: restricting data access to certain members of an object. This is to allow only member functions to directly access and modify the object's data.

Encapsulation: the bundling of an object's data and procedures into a single entity

18. Explain the following in regard to accessing, protecting, and exchanging member data: this pointer, const, static, friends.

The compiler provides each member function of a class with an implicit parameter that points to the objects through which the member function is classed. The `this` pointer is that implicit parameter. In other words it is a pointer that points to the memory address of the object calling the function.

- i. `(*this).x` is the same as `this->x`
 - 1. `this->x` retrieves the value

Const is used with a parameter to prevent the called function from modifying it.

- ii. If const is placed after a function, the return value cannot be passed as a non-constant parameter.

Static members are accessible by all members of that same class. Static member functions can also be called independently of any class objects through the class name, and can be called before any objects of the class have been created.

However, the this pointer cannot be used on static members.

Friend functions are not members of a class but have access to the private members of the class.

19. What is a copy constructor?

A copy constructor is a special constructor that is called whenever a new object is created and initialized with the data of another object of the same class.

The default copy constructor copies field-to-field using memberwise assignment.

We can use a copy constructor when an object is initialized from an object of the same class, an object is passed by value to a function, and when an object is returned using a return statement from a function.

20. What does it mean to overload an operator?

Operators such as =, +, etc. can be redefined for use with objects of a class.

When overloading an operator you must have the same number of parameters as the standard version. To actually do this you have to use operator followed by the operator symbol you want to override.

- i. ex.) ClassName operator+ (ClassName right);

21. What are move operations?

A move assignment (overloaded = operator) and move constructor use an rvalue reference for the parameter. The dynamic memory locations can be “taken” from the parameter and assigned to the members in the object invoking the assignment.

22. Describe the difference between aggregation and composition.

Class aggregation and composition both have a “has-a” relationship between classes. Class aggregation is when an object of one class owns an object of another. Composition is a form of aggregation, where the owning class controls the lifetime of the objects of the owned class.

23. Explain inheritance.

Inheritance is a way of creating a new class by starting with an existing class and adding new members. The new class can replace or extend the functionality of the existing class. Inheritance has a “is-a” relationship between classes. For instance, a dog is an animal.

Base class access supports three inheritance models.

- i. Public, private, and protected

24. What is polymorphism?

Polymorphism is the provision of a single interface to access different objects. It can be implemented through virtual functions, which are used to override functions

25. What are abstract base classes?

An abstract class is a class that contains no objects that are not members of derived classes. They are an organizational tool for inheritance hierarchies. They can be used to specify an interface that must be implemented by all subclass. Member functions specified in an abstract class do not have to be implemented. This is left to the subclasses. If a function is not implemented is a pure virtual function or an abstract function.

26. How do you declare and use pointers?

Using the address operator & you can access a variable's address. The address of a memory location is called a pointer.

You can use pointers for objects and structure variables.

- i. ex.) (*structPtr).structName = 6;
- ii. A simpler way to do this is by using the structure pointer operator.
 1. structPtr -> structName(6);
 2. structPtr -> structName = 6;

A variable that stores an address is called a pointer variable. To do this properly you must specify the data type that the pointer variable will point to.

- iii. ex.) int *bleh; // bleh can only point to integer variable

Pointers can be used to access and modify the variable being pointed to.

- iv. ex.)

```
int x = 25;
int *ptr;
ptr = &x; // ptr now has the address of x stored
*ptr = 100; // now x is equal to 100
```

27. Explain dynamic memory allocation.

Allocating storage for a variable while a program is running. You can do this with the new operator. **new** returns the address of a memory location. The data type of the variable is indicated after new. You can also use it with an array.

- i. ex.)

```
double *dptr;
dptr = new double;
```

To release the dynamic memory use delete.

- ii. ex.) delete dptr;

If a pointer contains the address of memory that has been freed by a call to delete it is a dangling pointer.

Memory leaks occur if no-longer-needed dynamic memory is not freed. To prevent this free up dynamic memory after use.

28. Why are smart pointers so useful?

Unlike regular pointers, smart pointers can automatically delete dynamic memory that is no longer being used. A smart pointer manages the object that it points to.

- i. Unique pointers
 1. Points to a dynamically allocated object that has a single owner. Ownership can be transferred to another unique pointer.
 - a. ex.) uptr2 = move(uptr); // uptr2 now has ownership and uptr is now empty

2. When the owning unique pointer goes out of scope, memory for the object is deallocated. This also happens when the unique pointer takes ownership of a different object.
 3. ex.) `unique_ptr<int> uptr(new int);`
 4. You can manually deallocate memory by:
 - a. `Uptr = nullptr;`
 - b. `uptr.reset();`
- ii. Shared pointers
1. Points to a dynamically allocated object that may have multiple owners.
 2. Memory allocations can be combined by using the `make_shared` function.
 - a. ex.) `shared_ptr<int> uptr = make_shared<int>();`
- iii. Weak pointers

29. Explain the difference between classes and structs

Class	Structure
Members of a class are private by default.	Members of a structure are public by default.
Memory allocation happens on the heap.	Memory allocation happens on a stack.
It is a reference type data type.	It is a value type data type.
It is declared using the class keyword.	It is declared using the struct keyword.

30. When should you use a struct instead of a class?

31. Composition vs inheritance.

- a. Inheritance has a “is-a” relationship between classes and has inheritance types.
- b. Composition has a “has-a” relationship between classes