# Chapter 17 Review

Ulysses Carlos

*<2019-12-31 Tue>*

## 1 Questions

### 1.1 Why do we need data structures with varying numbers of elements?

Because we may be in a situation where the amount of data is either increasing/decreasing.

### 1.2 What four kinds of storage do we have for a typical program?

- Code Storage
- Static Storage (For defined global variables)
- Stack/Automatic Storage (Memory used for calling functions)
- Free Store/Heap (Memory for the rest of the operating system)

### 1.3 What is the Free Store? What other name is commonly used for it? What operators support it?

The free store (Also known as the heap) is memory that is available for the operating system, and getting memory from the free store.

This free store is different from the heap used in C, as the C++ free store allocates/deallocates memory (and calls constructors/destructors if the pointer is of a object type)

## 1.4 What is a dereference operator and why do we need one?

The dereference operator (* in C/C++) allows access to the value of the object that a pointer is pointing to.

## 1.5 What is an address? How are memory addresses manipulated in C++?

An address is a location in memory of an object. These memory addresses are manipulated through the usage of pointers and references (which are dereferenced immutable pointers)

Addresses can be assigned to a pointer by using &T , where T refers to some type. An immutable address is assigned to a reference, which does not require dereferencing.

## 1.6 What information about a pointed-to object does a pointer have? What useful information does it lack?

A pointer only contains the address (in memory) where that object is located in. It does not know (in the case of an array) of how many items are in that array, which has to be tracked by the programmer.

## 1.7 What can a pointer do?

A pointer can point to any data object(or type)

## 1.8 What is a leak?

A memory leak is where allocated memory is not deallocated during a function/throughout the program, causing that memory to be unavail-

able.

## 1.9 What is a resource?

A resource can be something like a file or anything that is acquired by a class to preform an operation on.

## 1.10 How can we initialize a pointer?

We can initialize a pointer by

- Using **nullptr** to set an initial value for a pointer that is used later on
- Another pointer of the same type or compatible type.
- A data variable by using &

## 1.11 What is a null pointer? When do we need to use one?

A null pointer is a value (of nothing) that can be assigned to a pointer. It can be used as a initial value for a pointer, or as a way to prevent a pointer from being used. In C++, they are assigned as **nullptr**, but in older C++ and C code, **NULL** is used.

## 1.12 When do we need a pointer (instead of a reference or a named object) ?

Pointers should be used when you have to deal with null values or if a pointer has to have its address throughout a program.

## 1.13 What is a destructor? When do we want one?

A destructor specifies actions to be done at the end of an object's lifespan. These actions are then handled by the destructor than manually, so that we don't have situations where the programmer forgot to execute some function / whatever.

For example, in this chapter, our Vector class defined a destructor that deallocated all the memory allocated by its constructor.

## 1.14 When do we want a virtual destructor?

A Virtual destructor should be created when a virtual function is declared/defined in a class. Any class that is a derived class of this base will have a destructor that will take care of both members of the base class AS well of the derived class.

## 1.15 How are destructors for members called?

A class destructor is automatically generated by the compiler which calls all the destructors for each of the members.

## 1.16 What is a cast? When do we need to use one?

A cast is a why to allow one data type to be treated as a member of another data type. This would need to be used when dealing with data from two different data types, or if a cast is the only means to access some form of data.

C++ allows C style casting, but has its own versions of casting that will be displayed below:

- static_cast allows for casting between related pointer types (Such as void* and double*)
- reinterpret_cast allows for casting between unrelated types (Such as int and double*)
- const_cast can cast away a constant variable.

  These are a lot more specific than C-style casting which can be overkill.

## 1.17 How do we access a member of a class through a pointer?

Given a class A with members B and C, we create a class pointer p. A member of class can be accessed by this class pointer through **p->b** or **p->c**.

The **'->'** operation is shorthand for (**p).b** Which is something that you learned while working with C.

## 1.18 What is a doubly-linked list?

A doubly-linked list is a Linked List where each node has a next and previous pointer to the next and previous nodes in the list.

## 1.19 What is this and when do we need to use it?

**"this"** refers to a the current object that a member function is called for. It is immutable (You can't change the value of the this pointer (the address)) and can be used to "refer to the current object" instead of accessing the object indirectly.