Lecture 5: Stack & Queue

Stack:

Always Remember:

1. When we Push, we:
   1. Store the new item in m\_stack[m\_top]
   2. Post-increment our m\_top variable
2. When we Pop, we:
   1. Pre-decrement our m\_top variable
   2. Return the item in m\_stack[m\_top]

Page 20: Infix to Postfix Conversion

Page 27: Solve a maze

Queue: Refer to Always Remember in Stack

Lecture 6: Inheritance

Reuse:

Only public member function can be reused/Protected 🡪 derived class can use it

Extension/Specialization

Inheritance & Construction:

* C++ always constructs the base part first, then the derived part second.
* It does this by secretly modifying your derived constructor – just as it did to construct your member variables (calling member variables constructors)

Inheritance & Destruction:

* C++ implicitly destructs all of an object’s member variables after the outer object’s destructor runs.
* C++ destructs the derived part first, then the base part second.

Inheritance & Initializer List

* If a superclass requires parameters for construction, then you must add an initializer list to the subclass constructor

Inheritance & Assignment Ops

* Assign one instance of a derived class to another
  + First copies the base data
  + The copies the derived data
  + If base/derived classes have dynamically allocated member variables – must define assignment ops and copy constructors for the base class & special versions for the derived class

Lecture 7: Polymorphism

Def: any time we use a **base pointer or a base reference** to access a derived object, this is called polymorphism (bad thing: chopping)

Where to use the virtual keyword?

1. Use the virtual keyword in both your base and derived classes any time you redefine a function in a derived class
2. Always use the virtual keyword for destructors in your base and derived classes
3. No virtual before constructor

Polymorphism & Pointers:

* May point a superclass pointer at a subclass variable
* Never point a derived class pointer/reference to a base class variable
* C++ always calls the most-derived version of a function associated with a variable, as long as it’s marked virtual!
* If one forget a virtual destructor, it only causes problems when one use polymorphism (to be safe, always use virtual destructors)

Pure Virtual Functions:

* Make a base class function pure virtual if you realize:
  + The base-class version of your function doesn’t (or cannot logically) do anything meaningful
* If you define at least one pure virtual function in a base class, then the class is called an “abstract base class”
* Purpose:
  + Avoid writing “dummy” logic
  + Force the programmer to implement function in a derived class to prevent bugs
  + Can still use ABCs like regular base classes to implement polymorphism.

Polymorphism Cheat Sheet: Page 50-51.

Lecture 8: Recursion

Two Rules of Recursion:

1. Must have a “stopping condition(aka base case)”
2. Must have a “simplifying step”: every time a recursive function calls itself, it must pass in a smaller sub-problem that ensures the algorithm will eventually reach its stopping condition.
3. Recursive functions should never use global, static or member variables???

Recursion Tips:

1. Your recursive function should generally only access the current node/array cell passed into it

Lecture 9: Generic Programming

Generic Functions & Operator

* Always place your template functions in a header file

Generic Classes:

* Add the prefix: template <typename xxx> before the CLASS DEFIINITION itself AND before EACH function definition, outside the class.
* Then update the types to use your template type
* Finally, place the postfix: <xxx> between the class name and the :: in all function def.

Template Cheat Sheet: page 20

Vector:

Push\_back/pop\_back/back/front/begin/end/size/empty

List:

Push\_back/pop\_back/back/front/begin/end/size/empty/push\_front/pop\_front

Iterator/Const Iterators

* Sometimes will pass a container as a const reference parameter; To iterate through such a container, you cannot use the regular iterator; list<string>::const\_iterator it;

Maps: begin/end/find 🡪 must define your own operator< method for the left-hand class/struct

Set:insert/erase/size 🡪 also need to define 🡪 both map and set are sorted

Most STL containers have an erase() method (for vector, this will disable the iterator variable)

Sort(first item, one pass last item, function address) 🡪 iterators