COMSC-110 Chapter 15

Some Advanced Topics

Valerie Colber, MBA, PMP, SCPM

Storing values in a program

- Literal values
- Variables: store one value
- Arrays: store multiple values of the *same data type*
- Structs: database records
- Array-Based Lists: arrays in which not all of the elements are used, but are available for use instead.
- Linked Lists: each value is separate, containing a "link" to the next value in the list.
- Collections: like arrays and lists, but with *system-managed storage* and a library of subprograms

Collections

Collections

- very similar to arrays -- uses indexable elements
- like arrays and lists, but with *system-managed storage*
- ...and library of subprograms

Double Ended Queue

- The library-defined data type "deque", and others like it, have library functions to create lists and manage values -- without the programmer having to worry about capacity or start and next pointers.

The standard template library

- #include <deque>

Declaring a List using a Collection

- The required include is **#include <deque>**.

deque<Student> student;

Declaring a List using a Collection

- The use of the data type "Student" enclosed in "angle brackets" specifies that the list is for values of that data type.
- This can be used for integer and floating point variables as well as struct variables.
- For struct variables, the definition need *not* include a next pointer.

Adding A Value To A Collection-Based List

Student aStudent;
... // set the field values for aStudent
student.push_back(aStudent); // add at the end
...or...
student.push_front(aStudent); // add at the front

Retrieving A Value From A Collection-Based List

student[i]

- **i** is an index, as in an array.
- The expression **student[i]** can be used on either side of an assignment operator (that is, an equal sign), to retrieve or set the value in the collection.

Removing A Value From A Collection-Based List

```
   ... // collection-based list exists with one or more values
If (!student.empty())
{
 student.pop_back(); // remove value from end of list
...or...
student.pop_front(); // remove value from front of list
}
```

Example using Collections

studentCollection.cpp

Example using Collections

- Note that the function parameter list includes a specification for the collection.
- The ampersand symbol **&** in the deque variable specification is *not* a typo -- it tells the program to share the original deque variable with the function instead of sharing a copy.

Functions that call themselves

Recursion

Recursion

- Recursion is when the code block of a function contains a statement with a call to itself!
- This actually is a form of a loop, which would seem to be infinite.
- What makes it work is that the function must also contain logic to decide whether to continue calling itself or not -- similar to the "if-break" in a **while (true)** loop.

Divide problem in parts and ask:

- which parts *do* you know how to solve
- which parts *don't* you know how to solve

What you need to know for Recursion

- Recursion is very useful in solving problems: actually simplifies many solutions
- Need to know how to solve a base case
- Need to reduce problem complexity *towards a base case*
- Need to identify a "base case", and use that in a logic statement to stop the recursion.

Recursion

- If you are not using recursion, you are using "iteration" -- it's what we have been using all along, but since it was the only thing we used, it really did not need a name.
- Iteration uses loops; recursion uses functions.
- Recursion is usually an upper-level computer science topic.
- It is very useful in solving problems, and actually *simplifies* many solutions.

Recursive countdown example

countDown.cpp

Caution about Recursion

- Do not get too attached to recursion.
- There is always an iterative solution to everything -- it's just that recursion is sometimes simpler to code, once you get familiar enough with it.
- But recursion is always less efficient than iteration, so use iteration if you can.
- Use recursion as a convenience, realizing that there is a cost involved.
- And sometimes that cost is too much, even for the easiest of recursive solutions -- try counting down from 100000 and see!

What did we learn?

- This introduction to programming used C++ to make the concepts real.
- These concepts, such as variables, code blocks, logic, subprograms, loops, arrays, and lists, are common to all programming languages.
- What you learned here actually serves as the introduction that you would need to go on to the study of *any* specific language -- not just C++.

Course Goals

- Purpose of this course is to turn *you* into a programmer
- Prepare for COMSC-265 (and 266, 267, and 210)
- Prepare for transfer to UC/CSU COMSC program
- Prepare you for using programming in the real world!

Course goals

- Think of this course as an opportunity to have explored a job as a programmer.
- You were treated like as if you were working on a job as a programmer.
- You had to THINK like the user.
- You developed and thought in terms of algorithms.
- The idea was to apply what you learn and grow!
- You hopefully learned how to “debug” your programs, and how to fix your mistakes!

Student Learning Objectives

- SLO#1. write programs that use constructs of sequence, selection (ifs), and iteration (loops).
- SLO#2. write programs that have subprograms (C++ "functions" or Java "methods") with parameter (or "argument") lists.
- SLO#3. write programs that apply arrays.

Where do you go from here?

- If this is where you end your study of programming, then hopefully you found this interesting and enlightening, and you have a better understanding of how computers work and what computer professionals do.
- If this is just the beginning for you, then you are ready to master one or more languages and then learn techniques of problem solving.

Where do you go from here?

- You will likely eventually learn to develop user interfaces and write GUI (graphical user interface) programs.
- You may learn to be a solo programmer or a member of a team of programmers.
- You may write scientific simulations, business databases, or interactive games.
- Whatever you do, it's a great profession with an unlimited future, and you will always be glad you found your way into it!

References

- INTRODUCTION TO PROGRAMMING USING C++ by Robert Burns