Programming Techniques

Week 2
Topic: Data abstraction and ADTs

01/2014
What is in today?

- Programming paradigms in C++
- Data Abstraction and Abstract Data Types
Programming Paradigms

- The most important aspect of C++ is its ability to support many different programming paradigms
- We will cover this term
  - procedural abstraction
  - modular abstraction
  - data abstraction
    - as ways or techniques used to solve problems
Procedural Abstraction

- This is where you build a “fence” around program segments, preventing some parts of the program from “seeing” how tasks are being accomplished.

- Any use of globals causes **side effects** that may not be predictable, reducing the viability of procedural abstraction.
Procedural Abstraction

- This may be the approach taken with stage #1...where the major tasks are broken into functions.
- You can test your functions separately before the entire program is written and debugged.
Modular Abstraction

- With modular abstraction, we build a “screen” surrounding the internal structure of our program prohibiting programmers from accessing the data except through specified functions.

- Many times data structures (e.g., structures) common to a module are placed in a header files along with prototypes (allows external references).
Modular Abstraction

- The corresponding functions that manipulate the data are then placed in an implementation file.
- Modules (files) can be compiled separately, allowing users access only to the object (.o) files.
- We progress one small step toward OOP by thinking about the actions that need to take place on data...
Modular Abstraction

- Later this term we will be implementing modular abstraction by separating out various functions/structures/classes into multiple .cpp and .h files.

- .cpp files contain the implementation of our functions

- .h files contain the prototypes, class and structure definitions.
Modular Abstraction

☐ We then include the .h files in modules that need access to the prototypes, structures, or class declarations:
  ■ `#include "myfile.h"`
  ■ (Notice the double quotes!)

☐ We then compile the programs
Data Abstraction

- Data Abstraction is one of the most powerful programming paradigms.
- It allows us to create our own user defined data types (using the class construct) and then define variables (i.e., objects) of those new data types.
Data Abstraction

With data abstraction we think about what operations can be performed on a particular type of data and not how it does it.

Here we are one step closer to object oriented programming.
Data Abstraction

- Data abstraction is used as a tool to increase the modularity of a program
- It is used to build walls between a program and its data structures
  - what is a data structure?
  - talk about some examples of data structures
- We use it to build new abstract data types
Data Abstraction

- An abstract data type (ADT) is a data type that we create
  - consists of data and operations that can be performed on that data

- Think about an char type
  - it consists of 1 byte of memory and operations such as assignment, input, output, arithmetic operations can be performed on the data
Data Abstraction

- An abstract data type is any type you want to add to the language over and above the fundamental types.
- For example, you might want to add a new type called: list
  - which maintains a list of data
  - the data structure might be an array of structures
  - operations might be to add to, remove, display all, display some items in the list
Data Abstraction

- Once defined, we can create lists without worrying about how the data is stored.
- We “hide” the data structure used for the data within the data type -- so it is transparent to the program using the data type.
- We call the program using this new data type: the client program (or client).
Data Abstraction

- Once we have defined what data and operations make sense for a new data type, we can define them using the class construct in C++
- Once you have defined a class, you can create as many instances of that class as you want
- Each “instance” of the class is considered to be an “object” (variable)
Data Abstraction

- Think of a class as similar to a data type and an object as a variable
- And, just as we can have zero or more variables of any data type...
  - we can have zero or more objects of a class!
- Then, we can perform operations on an object in the same way that we can access members of a struct...
Example

- For a list of videos, we might start with a struct defining what a video is:

```c
struct video {
    char title[100];
    char category[5];
    int quantity;
};
```
Example

- For a list of videos data type:

```cpp
class list {
    public:
        list();
        int add (const video &);
        int remove (char title[]);
        int display_all();
    private:
        video my_list[CONST_SIZE];
        int num_of_videos;
};
```
Example

For a client to create a list object:

```cpp
main() {
    list home_videos;  //has an array of 100 videos
    list kids_shows;   //another 100 videos here...
    ...

    video out_of_site;
    cin.get(out_of_site.title,100,'
    cin.ignore(100,'
    ...

    home_videos.add(out_of_site);  //use operation
```
For Next Time

- Study classes...we’ll look at terminology

- Next time we will discuss:
  - class constructors
  - where to place the class “interface” we saw previously and
  - where to place the implementation of the “member functions”