CS 213 -- Lecture #13

"Late Night Guide to C++"
Chapter 7 pg 184 - 191
Chapter 8 pg 217 - 221
Enumerations, Pointers to Functions

Administrative...

- Prelims are graded

Enumerations

- Sometimes, when programming, we need to deal with a limited range of constants:
  - A finite set of colors
  - Result codes
- It is useful to define a set of constants which can be used in place of the actual integer values:
  - increases readability of code
  - protects you against integer values changing
- We can do this with constants

```
// Define error codes
const short cNoError          = 0;
const short cBadArg     = 1;
const short cBadResult  = 2;
const short cUnknownErr = 3;
```

Enumerations (cont)

- This is fine, but it means that all functions which deal with these errors simply return (or take as arguments) a `short` type.
- If we used an enumeration, we can define a new data type as well as defining constant names.
- The syntax would look like this:

```
// Define error codes
enum RonsError
{
  cNoError = 0,   // Values are optional, default is 0
  cBadArg,        // If a value is not present,
  cBadResult,     //   assign previous value + 1
  cUnknownErr
};
```

Enumerations (cont)

- Consider the following:

```
enum RonsError
{
  cNoError = 0,   // Values are optional, default initial
  cBadArg,        //          value is 0
  cBadResult,     // If a value is not present,
  cUnknownErr     //   assign previous value + 1
};

int main()
{
  RonsError rerr = RonsFunction();  // An arbitrary function
  if (rerr != cNoError)
    cerr << "Ooooops, Error:" << rerr << endl;
  else
    cout << "No error" << endl;
}
```

Enumerations (cont)

```
int main()
{
  RonsError rerr = RonsFunction();  // An arbitrary function
  if (rerr != cNoError)
    cerr << "Ooooops, Error:" << rerr << endl;
  else
    cout << "No error" << endl;
}
```

- The variable `rerr` is not treated as an integer type.
- If I try to assign an integer value directly to it, I will get a compile time error.
- Although I could use a cast to force a value into the enumeration variable.
- This gives us some protection against accidentally assigning raw integer values to a variable of type `RonsError`.

Pointers to Functions

- What is a pointer to a function?
  - A pointer just like any other
  - Data pointed at by the pointer is actually machine code for the function pointed at.
- How is it declared?

```
// Define a pointer to a function
int (*f)(int start,int stop);
```

- This declares a variable `f` which is a pointer to a function that returns an `int` and takes two `ints` as parameters.
- Now, just like any other pointer, the declaration does no allocation.
- So, in this case, `f` points at nothing and any attempt to dereference it will have very spectacular side effects!
- You *cannot* dynamically allocate memory for function pointers.

Pointers to Functions (cont)

- You can only set pointer-to-function variables equal to pointers to existing functions.
- How do you do that?
- Consider the following code:

```
int SimpleAdd(int arg1,int arg2)
{
  return arg1 + arg2;
}
int main()
{
  int (*f)(int start,int stop);
  f = SimpleAdd;
  // f now points at the function “SimpleAdd”
  // What can we do with it now?
}
```

Pointers to Functions (cont)

- We can call it!
- How do you call a function when you have a pointer to it?
- Consider the following:

```
int SimpleAdd(int arg1,int arg2)
{
  return arg1 + arg2;
}

int main()
{
  int (*f)(int start,int stop);
  f = SimpleAdd;
  int x = (*f)(3,4);  // Call the function pointed at by f
  cout << “x is : “ << x << endl;
}
```

Demonstration

Simple Function Pointer

Pointers to Functions (cont)

- OK, interesting concept. But what use is it?
- Most frequently used to allow a programmer to pass a function to another function.
- Suppose I am writing a function which contains variables which need to be acted on.
- Suppose that I want to be able to have multiple ways to act on those variables.
- A function pointer as a parameter is a good solution.
- As you’ll all remember from that riveting lecture on Project SALSA, a client in the Project SALSA environment has multiple layers:
  - VCSAPI (server communication, file downloads)
  - File Delivery Layer (versioning logic, GUI code)
  - Runway (front end)

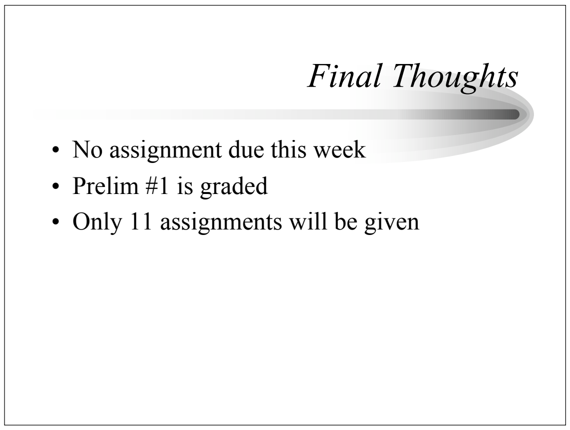
Pointers to Functions (cont)

- Whenever we download a file we want to provide a friendly progress bar.
- The problem is that the file download actually happens in the VCSAPI which has no GUI code in it at all!
- The solution is that the VCSAPI call which actually downloads a file take a *function pointer* as a parameter :

```
typedef short (*VCSAPI_ProgressCallback)(short);

VCSAPI_Error VCSAPI_FileGet(VCSAPI_Server server,
                            VCSAPI_FileRecord *fileRecPtr,
                            VCSAPI_ProgressCallback prog,
                            short callbackInterval,
                            char *localPath,
                            int *serverRC,char **serverMSG)
```

- The FDL (which manages all of the GUI code for versioning dialogs, etc.) can pass a pointer to a function which updates a graphical progress bar.



Final Thoughts

- No assignment due this week
- Prelim #1 is graded
- Only 11 assignments will be given