C++: Tour 2

Plan

- Functions
- Arrays
- Basic I/O

But first…

- Errata
  - `float *f = 0x22222;`
- Not only a bad idea…but also illegal!

Functions

- Declaration vs definitions
  - Declaration is the function signature
    - Required before function can be used
    - Often included in a header file
    - Can exist in multiple files
    - `char *strcpy (char * to, char* from);`
  - Definition
    - Actual code
    - Must defined once and only once

Functions

- void and void *
  - void – indicates that a function does not return a value
  - void * -- function returns a "generic" pointer
    - Must be typecast to correct pointer type
    - Use with caution

Functions

- In C++ function arguments are pass by value:

```
aClass i(7);
foo (i);
```

```
void foo (aClass a)
{
    a.moo = 12;
}
```

Functions – Pass by value

- Get around this by changing the variable type of the arguments

```
aClass i(7);
foo (&i);
```

```
void foo (aClass *a)
{
    a->moo = 12;
}
```

Functions – Pass by value

- Yet another variable type:
  - Constant pointers
    - Once defined, the data/object pointed to by the pointer cannot be changed.
    - `const aClass *a (new aClass(7))`
    - `a->moo = 7;   // error`

Functions – Pass by value

- Works for function arguments as well.

```
aClass i(7);
foo (&i);
```

```
void foo (const aClass *a)
{
    a->moo = 12; // error
}
```

Functions – Pass by value

- Furthermore

```
const aClass *i
(new aClass(7));

foo (i); // error
```

```
void foo (aClass *a)
{
   a->moo = 13;
}
```

Functions – Pass by value

- Cleaner means is to use reference variables:

```
aClass i(7);
foo (i);
```

```
void foo (aClass &a)
{
    a.moo = 12;
}
```

Functions

- Question?

Arrays

- Ways of declaring arrays
  - If you know the number of elements in the array
    - `int myArray[7];`
  - If you want to initialize the array when declared
    - `int myArray[] = { 1, 2, 3, 4, 5, 6, 7 };`
  - Dynamic allocation
    - `int myArray[] = new int[3*n];`

Arrays

- Array variables can <u>almost always</u> be viewed as a pointer to to the first element of the array:
  - `int a[] = { 1, 2, 3, 4, 5};`
  - a == & (a[0])
- Where this fails
  - `int b = { 6, 7, 8, 9, 10);`
  - `a = b;  // Illegal`

Arrays

- Especially true when passing to function

```
int a[]= {1, 2, 3, 4};   void foo (int b[])
foo (i);                 {
                             b[2] = 12;
                         }
```

Arrays

- Works just as well
  - In fact, arrays are converted to * when given as arguments

```
int a[]= {1, 2, 3, 4};   void foo (int *b)
foo (i);                 {
                             b[2] = 12;
                         }
```

Arrays

- Unlike Java, C++ arrays have no bounds checking.

```
int a[]= {1, 2, 3, 4};   void foo (int *b)
foo (i);                 {
                             b[7] = 12;
                         }
```

Arrays and Strings

- C-style string
  - Strings are represented as array of char terminated by a '\0'
  - `char *myName = "Joe";`
  - `char myName[] = "Joe";`
    - myName[0] = 'J'
    - myName[1] = 'o'
    - myName[2] = 'e'
    - myName[3] = '\0'

Arrays and Strings

- Can manipulate C style strings using strings library:
  - `char *strcat (char *s1, const char *s2)`
  - `int strcmp (const char *s1, const char *s2)`
  - `char *strcpy (char *s1, const char *s2)`
  - `char *strchr (const char *s, char c)`
- Complete list can be found by:
  - `man -s 3S string`

Arrays and Strings

- Another look at main()
  - `main (int argc, char *argv[])`

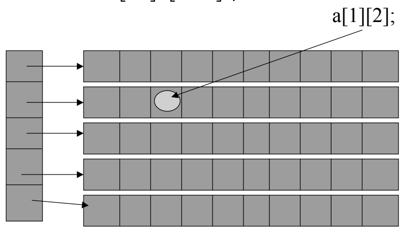
  - foo 1 3 fred
  - argc = 4
  - argv[0] = “foo”
  - argv[1] = “1”
  - argv [2] = “3”
  - argv[3] = “fred

Multidimensional Arrays

- C++ does support multidimensional arrays:
  - Interpreted as an array of arrays.
  - Or as an array of pointers to 1st element of arrays
  - Example:
    - `int a[5][10];`
    - a is an array of 5 arrays of 10 integers.

Multidimensional Arrays

- `int a[5][10];`



Arrays

- Questions?

Basic IO

- Two types of I/O
  - C-style (stdio)
    - `#include <stdio.h>`
    - `fprintf (FILE *f, const char *format, …);`
    - `fscanf (FILE *f, const char *format, …);`
    - `FILE *stdin;`
    - `FILE *stdout;`
    - `FILE *stderr;`
  - Only standard datatypes supported

  - `man -s 3C stdio`

Basic I/O

```
#include <stdio.h>
int a = 7;
float b = 6.4;
char *foo = "myString"
printf ("%d\t%f\t%s\n", a, b, foo);
```

```
7 6.4 myString
```

Basic I/O

```
#include <stdio.h>
int a;
float b;
char foo[10];
scanf ("%d\t%f\t%x\n", &a, &b, foo);
```

```
7 6.4 myString
```

Basic IO

- Two types of I/O
  - C++ style (I/O Streams)
    - << for output
    - >> for input
    - cout – standard input
    - cin – standard output
    - cerr – standard error
  - Classes can define << and >> operators

Basic I/O

```
#include <iostream>
using std

int a = 7;
float b = 6.4;
char *foo = "myString"
cout << a << '\t' << b << '\t' << foo < endl;
```

```
7 6.4  myString
```

Basic I/O

```
#include <iostream>
using std

int a;
float b;
char foo[10];
cin >> a >> b >> foo(10);
```

```
7 6.4 myString
```

Basic I/O

- Questions?

As a summary

- Let’s look at some code