

CS 60

Introduction to C, C++, and Unix/Linux

Review session

What we've learned

- Basics of Unix/Linux
 - Commands, editing, compiling, debugging, shell scripting, version control...
- The C language
 - Preprocessor, types, operators, memory and pointers...
- The C++ language
 - Classes, function overloading, exceptions, templates

Basics of Unix/Linux

- See slides from last week's discussion sessions
- In addition:
 - Emacs
 - Makefiles
 - Version control (CVS)
- Be able to:
 - Give the Unix commands to perform a certain task
 - Describe what a particular Unix command does
- Do not need to memorize all the flag possibilities
 - Exceptions:
 - ◆ gcc (and g++) -cgoESIL -Wall
 - ◆ ls -arlt
 - ◆ cp -r
 - ◆ rm -r

The C language

- Basic C syntax, types, operators, control flow, etc.
 - Program structure, **argc** and **argv**
 - Expressions, statements
 - Operator precedence
 - Typedef
 - Struct, union
 - Variables: Local, global, static, extern, const
- Functions
- The compiling process
 - Preprocessor, compiler, assembler, linker
 - Various files generated
- The preprocessor and macros

The C language (cont.)

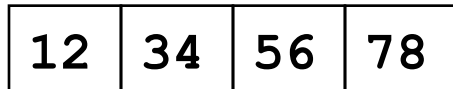
- Source and header files
 - File structure, scope of variables and functions
 - Linking, libraries
 - Declaration vs. definition/initialization
- Arrays, multidimensional arrays
- Memory structure: heap, stack, little-endian
 - Malloc/free, sizeof()
- Pointers
 - *****, **&**
 - Pointers and arrays
 - Pointers to pointers, pointers to functions
- File I/O
 - f/s/printf, f/s/scanf, fopen, fclose, fread, fwrite, stdout/stdin/stderr

What order are bytes stored?

- **Big-endian** – the most significant byte has the lowest address (“big end first”)
- **Little-endian** – The least significant byte has the lowest address (“little end first”)
- Historically, most mainframes have been big-endian, and PCs are little-endian. This *does* affect portability!
 - CSIL machines (all Intel-based computers) are little-endian
 - Motorola processors (Macs) are big-endian

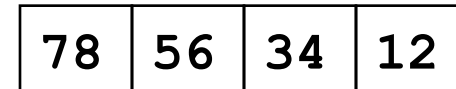
`int x = 0x12345678;`
MSB LSB

Big-endian



lower address → higher address

Little-endian



lower address → higher address

The C++ language

- Differences between C and C++
- Inheritance, data encapsulation, polymorphism
- Namespace
- Standard library (cin/cout/cerr, string class)
- Reference declarations, parameters, and return types (&)
- Function overloading
 - Overloading operators
- Default arguments
- new, new[], delete, delete[]

The C++ language (cont.)

- Classes
 - constructors, destructor, private/protected/public
 - static member variables and functions
 - Base and derived classes – what gets called when?
- Virtual class functions
- Abstract classes
- Calling C functions from C++
- Exception handling
 - Assert
 - Try/throw/catch
- Function templates, class templates
 - Function specialization, class specialization
- STL – what is it?

The Standard Template Library (STL)

- The STL is a general purpose library of data structures and algorithms, using the C++ template mechanism
- Its components are heavily parameterized – almost every component in the STL is a template

Main STL components

- At its core are *container classes* – classes whose purpose is to contain other objects
 - These are templates that can be instantiated to contain any type of object
- The STL also includes a large collection of *algorithms* that manipulate the data stored in containers
 - These are global functions, decoupled from the classes
- The STL defines *iterators* – a generalization of pointers – that allow access to the data inside containers
 - Forward, reverse, random