Introduction
Introduction

- Survey and TA/SA Introductions
- Pause to Look Backwards and Forwards
- Course Objectives
- Course Operation/Expectations
- Course Plan and Syllabus
- Systems Concepts
- Higher Level Language History
  - 'Old' Development Environment
    - C and C++
Quick Look Backwards/Forwards

- Computing Devices
  - From mainframes to PCs to smart phones to ??
- Changes in WPI CS Curriculum
- Instructor
- Students
  - Expected Background
  - Going Forward
  - Your Future
CS2303 Course Objectives

- To expose students to the low level systems interface ‘grunge’ clearly visible in C.
- To learn to program in C++ by learning to program in C first.
- To further develop the ability to design programs with emphasis on the abstract view of data structures.
- To get experience with low level implementation of data structures in C.
CS2303 Course Objectives

- To learn the advantages of programming in an object-oriented language such as C++.
- To experience programming in the Large
Pointers!!
The course web page is an essential student asset.

*Students are responsible for ALL information on web pages!

- 5 Required Labs
- 5 Programming Assignments
- 2 Closed Book Exams
Course Plan and Syllabus

- To cover the details of C briskly.
  - Assumes students already have an understanding of iteration and conditional constructs.
  - Using only C I/O {grunge as promised!} at first.

- To introduce data structures in C by doing at least one program with structs and call by value.

- To finish up with as much C++ as possible.

{Note - reading of the textbook will require jumping around during the C portion of the course.}
The goal of this programming course is to expose the students to places where the software and hardware meet or where applications interface with the operating system (OS).

A systems viewpoint includes resource management (CPU and memory), process scheduling, concurrency and performance.

{But this is too much material for this instance of the course!}
The assignments include simulation and introduce two system performance concerns - **efficiency** and **fairness**.

The other important approach to appreciate is the **computer scientist abstraction concept** of insulating interfaces from ‘under-the-hood’ details (e.g., virtual memory and loaders).
1.5 Types of Programming Languages

- Although assembly-language code is clearer to humans, it’s incomprehensible to computers until translated to machine language.
- To speed the programming process even further, high-level languages were developed in which single statements could be written to accomplish substantial tasks.
- High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical expressions.
- Translator programs called compilers convert high-level language programs into machine language.
- Interpreter programs were developed to execute high-level language programs directly, although more slowly than compiled programs.
- Scripting languages such as JavaScript and PHP are processed by interpreters.
<table>
<thead>
<tr>
<th>Language</th>
<th>Year(s)</th>
<th>Year(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>1957</td>
<td>COBOL</td>
</tr>
<tr>
<td>Algol</td>
<td>1960 1968</td>
<td>Lisp</td>
</tr>
<tr>
<td>PL1</td>
<td>1964</td>
<td>APL</td>
</tr>
<tr>
<td>Pascal</td>
<td>1970</td>
<td>SNOBOL</td>
</tr>
<tr>
<td>C</td>
<td>1972</td>
<td>Prolog</td>
</tr>
<tr>
<td>Basic</td>
<td>1975</td>
<td>Scheme</td>
</tr>
<tr>
<td>C++</td>
<td>1986</td>
<td>ADA</td>
</tr>
<tr>
<td>Java</td>
<td>1995</td>
<td>Python</td>
</tr>
</tbody>
</table>
Standard Steps

1. **Edit source code**
2. **Preprocess**
3. **Compile**
4. **Link**
5. **Load**
6. **Execute**

---

Fig. 1.1 | Typical C development environment.

© 2007 Pearson Ed - All rights reserved.
User Memory Protection
Review of Introduction

- Course Objectives
- Course Operation/Expectations
- Course Plan and Syllabus
- C, data structures, C++
- Systems Viewpoint \{more later\}
- Program Development Environment