CS 113: Introduction to

Lecture 1
CS 113 – Spring 2008

Course information

- MWF 12:20 - 1:10pm 1/21-2/15, 306 Hollister Hall
  - Add/drop deadline: 1/28
- Instructor: David Crandall
  - See website for office hours and contact information
- Prerequisites
  - CS 100 or similar experience with programming
  - Interest in C
  - Patience
    - C can be frustrating, especially at first
    - But it gets easier with time and practice!

Grading

- CS 113 is 1 credit, S/U grade option only
- To earn an S, you need to earn the equivalent of a C-
  - A final course score of 70% *guarantees* an S
- Grades will be based on:
  - Three programming assignments (75% of grade)
  - One quiz (25% of grade), announced ahead of time

Assignments

- Three programming assignments
  - Graded based on completeness, correctness, style of code
  - Submit assignments via the Course Management System (CMS)
  - Late submissions generally not accepted
- You may use any C compiler that you wish, e.g.
  - Dev-C++ in CIT labs: 318 Phillips and B7 Upson
  - Eclipse
  - gcc under Windows (Cygwin), Linux, Mac OS
  - Microsoft Visual Studio
  - See website for more information

Course resources

- Course website
  - http://www.cs.cornell.edu/courses/cs113
- Course newsgroup: cornell.class.cs113
  - Great place to post questions, discuss assignments, etc.
- Textbooks
  - Oualline, Practical C Programming.
    - available electronically (for free!) via Cornell Libraries website
    - Reading assignments listed on course website
  - "K&R": Kernighan and Ritchie, The C Programming Language.
    - not required, but highly recommended

Academic integrity

- Read and understand AI code on course website
- We will look for and prosecute AI violations
- Be especially diligent about assignments

  "You may discuss homework problems with other students at a high level. That is, you may discuss general approaches to a problem, high-level algorithm design, and the general structure of code. However, you may not share written code with any other students except your own partner, nor may you possess code written by another student who is not your partner, either in whole or in part, regardless of format." -- CS 113 website
- It's pointless to cheat in an S/U course!

Course objectives

- Cover major features of C, to the extent that students can subsequently learn about any features not discussed by reading a standard reference.
- Topics:
  - History, motivation behind C
  - Basic C syntax
  - Variables, data types, operators
  - Functions
  - Pointers
  - Memory management
  - C programming style
  - Debugging
  - The preprocessor
  - Arrays and strings
  - Complex types
  - File I/O
  - Threads

Machine Language

- Used with the earliest electronic computers (1940s)
  - Machines use vacuum tubes instead of transistors
- Programs are entered by setting switches or reading punch cards
- All instructions are numbers
- Example code

```
0110 0001 0000 0110
Add  Reg1      6
```

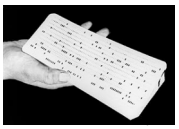
- An idea for improvement
  - Use "words" instead of numbers
  - Result: Assembly Language

Assembly Language

- Idea: Use a program (an *assembler*) to convert assembly language into machine code
- Early assemblers were some of the most complicated code of the time (1950s)
- Example code

```
ADD R1 6
MOV R1 COST
SET R1 0
JMP TOP
```

  - Typically, an assembler uses *2 passes*



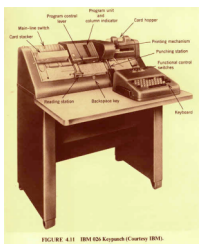
- Idea for improvement
  - Let's make it easier for humans
  - Result: high-level languages

High-Level Language

- Idea: Use a program (a *compiler* or an *interpreter*) to convert high-level code into machine code
- Pro
  - Easier for humans to write, read, and maintain code
- Con
  - The resulting program will never be as efficient as good assembly-code
    - Waste of memory
    - Waste of time
- The whole concept was initially controversial
  - FORTRAN (mathematical FORmula TRANslating system) was designed with efficiency very much in mind

FORTRAN

- Initial version developed in 1957 by IBM



- Example code

```
C     SUM OF SQUARES
      ISUM = 0
      DO 100 I=1,10
      ISUM = ISUM + I*I
  100 CONTINUE
```

- FORTRAN introduced many high-level language constructs still in use today
  - Variables & assignment
  - Loops
  - Conditionals
  - Subroutines

ALGOL

- ALGOL = ALGOrithmic Language
- Developed by an international committee
- First version in 1958 (not widely used)
- Second version in 1960 (widely used)
- ALGOL 60 included *recursion*
  - Pro: easier to design clear, succinct algorithms
  - Con: too hard to implement; too inefficient
- Sample code

```
comment Sum of squares
begin
   integer i, sum;
   for i:=1 until 10 do
      sum := sum + i*i;
end
```

COBOL

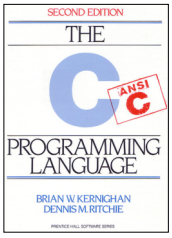
- COBOL = COmmon Business Oriented Language
- Developed by the US government (about 1960)
  - Design was greatly influenced by Grace Hopper
- Goal: Programs should look like English
- Sample code

```
MULTIPLY B BY B GIVING B-SQUARED.
MULTIPLY 4 BY A GIVING FOUR-A.
MULTIPLY FOUR-A BY C GIVING FOUR-A-C.
SUBTRACT FOUR-A-C FROM B-SQUARED GIVING RESULT-1.
COMPUTE RESULT-2 = RESULT-1 ** .5.
```

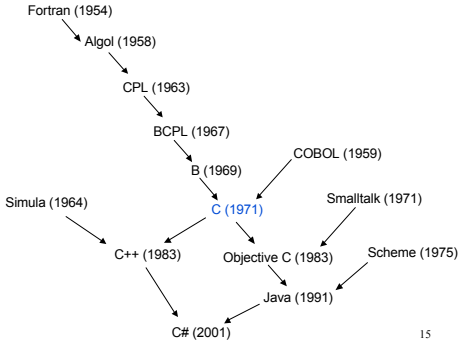
- COBOL included the idea of *records* (a single data structure with multiple *fields*, each field holding a value)
- *Immensely* popular language

C

- Developed in 1972 by Dennis Ritchie
  - at Bell Labs
- Idea was a higher-level language with nearly the power of assembly language
  - Simple, terse syntax for use on machines with little memory
  - Originally used to write UNIX
- Very widely used
  - Descendents: C++, C#, Objective C, etc.



C family tree



Advantages of C

- Very good for writing *fast* code
  - Usually much faster than Java, Matlab, etc.
- Simple syntax
- Portable and ubiquitous
  - C compilers available on almost *any* platform
- Very powerful. Programmer has complete control.
  - C allows operations that other languages don't (e.g. direct access to raw memory addresses)

Writing in C or C++ is like running a chain saw with all the safety guards removed. -- Bob Gray

Disadvantages of C

- Difficult to learn
  - Memory management is particularly frustrating
- Difficult to write bug-free, maintainable C code
  - Memory errors are difficult to detect and locate
  - Poor C code is responsible for most of the security holes that worms and viruses exploit
  - C does not enforce good software engineering principles. Hard to use for large programming projects.

C/C++ is an atrocity, the bletcherous scab of the computing world, responsible for more buffer overflows, more security breaches, more blue screens of death, more mysterious failures than any other computer language in the history of the planet Earth.
-- Eric Lee Green

Why learn C?

- C is *the* language for systems programming
- Also very popular for writing applications
  - Although higher-level languages (Java, C#, etc.) are also popular, and often more appropriate
- Learning C gives you more insight into how programs actually work
  - Pointers and memory addresses
  - Dynamic memory allocation
  - Function calls
  - Other languages (e.g. Java) hide these details

First C program

```
#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```


First C program

```
#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

- **`stdio.h`** is the *header file* for the standard I/O library
  - It contains declarations for many useful functions
  - **`#include`** instructs the compiler to read this file
  - **`#include`** is similar to **`import`** in Java
- C standard library includes many other header files

First C program

```
#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

- Any C program declares exactly one **`main`** function
  - **`main`** returns an integer and doesn't take arguments
    - We'll talk about the return value and arguments to main later

First C program

```
#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

- **`printf`** prints to the console
  - Stands for print formatted
  - Declared in **`stdio.h`**
  - The **`\n`** in the string constant is a *newline* character

First C program

```
#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

- As in Java, **`return`** causes the function to end
  - By convention, C programs return 0 if successful, non-zero if an error occurred

Another example

```
#include <stdio.h>

int main(void)
{
    int x = 1, y;
    int sum;
    y = 3;
    sum = x + y; /* compute sum */
    printf("%d plus %d is %d\n", x, y, sum);
}
```

Another example

```
#include <stdio.h>

int main(void)
{
    int x = 1, y;
    int sum;
    y = 3;
    sum = x + y; /* compute sum */
    printf("%d plus %d is %d\n", x, y, sum);
}
```

- Declare integer variables named **x** and **y**
- **y** is declared but not initialized.
  - Q: What is its initial value?

A tangent: C standards

- C has evolved over time
  - There are three common versions:
    - Original C (1978) "K&R C"
    - ANSI Standard C (1989) "Ansi C"
    - Revised ANSI Standard (1999) "C99"
  - Example: K&R and ANSI C required variable declarations to appear *before* any other statements in a function

Legal in K&R, Ansi C, C99:

```
int main() {
    int x = 1, y;
    int sum;
    y = 3;
    return 0;
}
```

Legal in C99 only:

```
int main() {
    int x = 1, y;
    y = 3;
    int sum;
    return 0;
}
```

  - Most modern compilers support C99 extensions

Another example

```
#include <stdio.h>

int main(void)
{
    int x = 1, y;
    int sum;
    y = 3;
    sum = x + y; /* compute sum */
    printf("%d plus %d is %d\n", x, y, sum);
}
```

- Comments begin with **/*** and end with ***/**
  - Caution: you can't nest comments

```
/* /* /* this works ok */
/* /* /* but this causes an error */ */ */
```


Another example

```
#include <stdio.h>

int main(void)
{
    int x = 1, y;
    int sum;
    y = 3;
    sum = x + y; /* compute sum */
    printf("%d plus %d is %d\n", x, y, sum);
}
```

- **printf** can format and print out values of variables
  - **%d** is a special *placeholder*
  - **printf** substitutes the values of the variables specified as arguments into the placeholders
  - **printf** has many other options and features

Getting to know you…

- On the index card, write:
  - Your name and Cornell NetId
  - What you like to be called (nickname?)
  - Your year, college, major
  - Why are you taking 113?
  - CS courses you have taken (+ where and when)
  - Programming languages you have used
  - Any concerns you have about the course