Welcome to...

CS113: Introduction to C

Instructor: Hubie Chen

E-mail: hubes+cs113@cs.cornell.edu

Course Website:

http://www.cs.cornell.edu/hubes/cs113-spring04

The website is linked to from the courses page of the CS department, www.cs.cornell.edu.

Assignments, lecture slides, etc. will be posted on the course website.

Administrivia and Mission

- Lecture slides and readings are posted to the course website, as are assignments, announcements, etc. Please check the site periodically for news.
- Class runs for four weeks, three meetings a week.
- Office hours: by appointment
- Prerequisite: CS100 or equivalent experience
- All auditors welcome
- Course goal: cover all major features of the C programming language, to the extent that students can subsequently learn about any features not discussed by reading a standard reference such as K & R.

Assignments, etc.

- Four assignments total. Assignments due once a week, on Fridays.
  - Turn in a printout of your program (no sample data necessary), AND e-mail the source code (the .c files) to hubes+assign?@cs.cornell.edu.
  - DO NOT just e-mail me your source code. YOU MUST turn in a printout of your source code as well.
- Solutions to (the first three) assignments will be discussed during the class meetings immediately following the due dates.
- ATTENDANCE IS MANDATORY. Please see me if you have any questions about this.
- We will have quizzes (which will not be announced ahead of time).
- Please do not take this class if you do not anticipate attending every lecture and being able to turn in all assignments on time.

More Course Info.

- Grades on Assignments: check plus (exceptional), check (acceptable), check minus, X (insufficient effort)
  - Turning in a program which compiles and is neatly formatted can help you avoid an X.
- Grading for course: S/U only.
  - You are guaranteed an S if all assignments completed with a grade of "check" AND all quizzes are passed
  - Failing to complete two assignments results in a U.
- Two textbooks: "Practical C Programming" by Steve Oualline and K&R. I recommend purchasing both.
- You may use any C compiler. Information on how to get started using CodeWarrior is on the web site.
- Collaboration: You may discuss ideas on a high level with others, but all code must be your own. You should understand everything that you turn in.

A Brief History of C

- 1969 - Ken Thompson, Bell Labs researcher, simplifies research language BCPL ("Basic Combined Programming Language", from London Univ. & Cambridge Univ. in England), giving rise to B.

- B unsuccessful; memory limitations (8K) allowed little other than experimentation. B also typeless.

- The ++, – operators made their first appearance in B!

- 1970 - PDP-11 introduced; featured hardware support for datatypes of different sizes. Dennis Ritchie creates compiler for "New B" on this machine. (Solves speed issue, introduces multiple datatypes.) Evolves into C.

"C is quirky, flawed, and an enormous success."

– Dennis Ritchie

Meet the C programming language

- Why learn C?
  - "Least common denominator": good building block for learning other languages. Subset of C++, similar to Java.
  - Closeness to machine allows (forces?) one to learn about system-level details (e.g. C only has call-by-value). Also allows one to write programs performing system-level tasks.
  - Portable - compilers available for most any platform!
- ANSI C standard - aim for ANSI C compliance.
- Compiled versus interpreted. Difference with Java. (C almost always compiled, Java often interpreted.)

The canonical "first C program"

```
#include <stdio.h>

void main()
{
   printf( "Hello, world!\n" );
}
```

Notice:

- All C programs must have a main() function; this is the first function invoked. The program terminates when this function terminates.

- printf is a function which prints formatted output

- void indicates that the program itself returns no value; won't worry about this too much now...

- But, note that some compilers insist on the declaration "int main()", in which case "return 0;" should be added to end of program.

Another example

```
#include <stdio.h>

void main()
{
   int x = 1, y;
   int sum;
   y = 3;
   sum = x + y; /* evaluates right hand side,
                   places value in variable sum */
   printf( "%d plus %d is %d\n", x, y, sum );
}
```

Some comments on comments and keywords

- Comments
  - Any string of symbols placed between the delimiters /* and */.
  - Can span multiple lines
  - Can't be nested (according to ANSI C standard)! Be careful.
    - Some development environments have an option that allows one to nest comments.
    - A curiosity: one can actually write a program which detects whether or not comments are nested or not!
  - Example: /* /* /* Hi, I'm a comment */

- Keywords
  - Reserved words that cannot be used as variable names
  - Examples: break, if, else, do, for, while, int, void (exhaustive list in K&R, p192)
  - Can be used within comments

A minor perturbation.

```
#include <stdio.h>

void main()
{
   int x, y;
   int product;

   printf( "Enter an integer: " );
   scanf( "%d", &x );
   printf( "Enter another integer: " );
   scanf( "%d", &y );

   product = x * y;

   printf( "%d times %d is %d\n", x, y, product );
}
```

Summing the numbers 1 through 10.

```
#include <stdio.h>

void main()
{
   int i = 1, sum = 0;

   while( i <= 10 )
   {
      sum = sum + i;  /* shortcut:  sum += i;  */
      i = i + 1;      /* shortcut:  i++;       */
   }

   printf( "The sum is %d\n", sum );
}
```

Summing the numbers 1 through 10.

```
#include <stdio.h>

void main()
{
   int i = 1, sum = 0;

   for( i = 1; i <= 10; i++ )
   {
      sum = sum + i;
   }

   printf( "The sum is %d\n", sum );
}
```

General form of a for loop:

```
for( initial-stmt; condition; iteration-stmt )
   body-stmt;
```

What happens?

1. Initialization is performed.
2. Condition is checked; if false, loop terminates. Otherwise...
3. Body is performed, followed by the iteration statement.
4. Then, the condition is checked again, and so forth.

Equality testing.

```
#include <stdio.h>

void main()
{
   int a, b;

   printf( "Enter a number: " );
   scanf( "%d", &a );
   printf( "Enter another: " );
   scanf( "%d", &b );

   if( a == b )
   {
      printf( "They're equal!\n" );
   }
   else
   {
      printf( "They're not equal.\n" );
   }
}
```

Note:

- Double equals used to compare ints
- "else" portion of "if" optional

Bracing Styles

Four widely used bracing styles:

- 1TBS: One True Bracing Style - used by K & R

  ```
  for( j = 0; j < 10; j++ ) {
      printf( "%d", j );
  }
  ```

- Allman - my personal preference

  ```
  for( j = 0; j < 10; j++ )
  {
      printf( "%d", j );
  }
  ```

- Whitesmith

  ```
  for( j = 0; j < 10; j++ )
      {
      printf( "%d", j );
      }
  ```

- GNU

  ```
  for( j = 0; j < 10; j++ )
      {
          printf( "%d", j );
      }
  ```

Most important rule of style: *Be consistent.*

Exponentiation

```
void main()
{
    int base, exponent, result;
    printf( "Enter the base:" );
    scanf( "%d", &base );
    printf( "Enter the exponent:" );
    scanf( "%d", &exponent );

    for( result = 1; exponent > 0; exponent-- )
    {
        result *= base;
    }
    printf( "%d\n", result );
}
```

Exponentiation, the sequel

```
void main()
{
    int base, exponent, result = 1;
    printf( "Enter the base:" );
    scanf( "%d", &base );
    printf( "Enter the exponent:" );
    scanf( "%d", &exponent );

    while( exponent > 0 )
    {
        if( exponent % 2 == 1 ) result *= base;
        base *= base;
        exponent /= 2;
    }
    printf( "%d\n", result );
}
```

Note:

- Trickier: maintain invariant that *result times base to the exponent power* is the desired value
- Faster algorithm