

CMSC 216

Introduction to Computer Systems

Lecture 2

Introduction to C

Jan Plane & Alan Sussman
{jplane, als}@cs.umd.edu

Administrivia

- Project 1 will be posted next Wednesday, due 2 weeks later
- Read Chapters 2 and 3 of Reek

Chapter 1, Reek

A QUICK START WITH C

Comparison between C and Java

- C is procedural, not object-oriented
- C is fully compiled (to machine code), not to bytecode
- C allows direct manipulation of memory via pointers
- C does not have garbage collection
- Many of the basic language constructs in C act in similar ways to the way they work in Java
- C has many important, yet subtle, details

An example C program

- The following is C's version of the "Hello world" program:

```
#include <stdio.h>
int main() {
    printf("Hello world\n");
    return 0;
}
```
- How does it accomplish its goal?
 - includes library header file with function declarations (so call to **printf()** compiles OK)
 - provides definition of **main()** function, where all C programs begin
 - returns from **main()** to end program - the value returned can be checked by the program that invoked this one

Important caveats

- You will be writing C code that conforms to the C90 standard - this means a few things have to be kept in mind:
 - All comments must be of the **/* */** variety; C90 does not recognize **//** (single-line) comments
 - All variables must be declared at the beginning of a block (immediately after an opening brace); failure to do this will trigger "mixed declarations and code" error messages

Terrible style

```
if (condition) do_something();
    always_do_this();

if (x) y = j;

while (condition) x++;

if (c) {
    f();
    g();}

/* this funtion do stuf */

/* call f with value of 10 */
f(5);
```

- These examples are all bad style and you will lose credit for using them
 - statements executed conditionally should be put on separate lines from the condition
 - closing braces should be first thing on a line
 - comments should be spelled properly, and have correct and useful information

C function declarations

- Function prototype declarations provide information about a function's return type and parameters, but do not define the function
- Using function declarations allows the compiler to check your function calls for correctness
- Examples:

```
void foo(int, int, double);
int bar(double x, double y);
```

Declarations and the Preprocessor

- **#include**
 - provides ability to include declarations from other files
 - usually have the file name extension .h
 - generally only include function prototypes, and type and variable declarations
 - actual code is kept in a different file, usually with the extension .C
 - files with code are compiled individually to *object* code
 - and then linked together to create a file with *executable* code
- Two ways to use **#include**
 - #include <stdio.h>**
 - for standard system files
 - #include "swap.h"**
 - for user-written files

Compiling a C program

- C programs must be compiled to be executed
- Use the **gcc** program to build your programs
 - invocation: **gcc** options source-files
 - common options
 - g** enable debugging
 - Wall** warn about common things that may be problems
 - o filename** place output in filename
 - c** only compile to object file, don't link
 - a very simple compilation command
gcc file.c
 - a simple compilation command
gcc -g -Wall -o prog1 prog1.c tools.c
 - executable is run in UNIX by just typing its name (sometimes preceded by "./", like this: **./prog1**)
- There are programs to make the compiling process easier

Compilation stages

- Preprocessor
 - used to make sure the program parts see declarations they need (and other purposes too, e.g., macros)
 - directives begin with a # (pound sign)
 - do not end in a ; like C statements do
- Translation
 - makes sure individual parts are consistent within themselves
 - creates an object (.o) file
- Linking
 - joins one or more compiled object files together
 - includes matching function call to callee across separate files
 - and matching global variable use to where it's defined
 - makes sure caller/callee consistent with each other
 - creates an executable file (the convention in Unix is no filename extension is used for executable files, although sometimes .x or .exe are used)

Basic C syntax is similar to Java

- Variable declarations
 - format: *typename variable1, variable2;*
 - each variable can optionally have an initializer
 - examples:

```
int a;
float x, y;
int m = 8;
```
 - if appearing in a function, must appear before executable code
 - may be local to a function, or global (remainder of the same file)
- Legal variable names similar to Java
- Looping and conditionals
 - **if, switch**
 - **while, do/while, for**
- Function calls
 - format: *functionname(argument1, argument2, ...)*
- Compound statement
 - can go anywhere a single statement goes
 - surrounded by { }
 - can have block local variables declared at beginning

Output – the **printf()** function

- Definition
 - its declaration is included with **#include <stdio.h>**
 - then just use it because its definition is in the C standard I/O library that the compiler always links for you
- Syntax:
 - **printf("formatstring", expression1, expression2, expression3, ...);**
 - a function with a variable number of arguments!
- Within the format string, normal characters print as themselves, and:
 - to print values of expressions use format specifiers, for example:

%d	print an integer in decimal
%f	print a floating point value
%c	print a character
%s	print a character string
 - there are also special *escape sequences*, similar to Java:

\n	print a newline
\t	print a tab
 - each format specifier in the *formatstring* is substituted by the corresponding expression, and the *formatstring* is printed

printf() example

• Program:

```
#include <stdio.h>

int f(int);

int main() {
    int a = 42;
    printf("%d < %d\n", a, a+5);
    printf("%d\n", f(a));
    return 0;
}

int f(int i) {
    return i * 2;
}
```

• Output:

```
42 < 47
84
```

Input – the **scanf()** function

- Definition
 - its declaration is also included with **#include <stdio.h>** (it's also in the C standard I/O library)
- Syntax:
 - **scanf("formatstring", &variable1, &variable2, &variable3, ...);**
 - the format string has the same basic structure as in **printf()**
 - to match data entered in the input, use the same kinds of formatting expressions and format specifiers in the format string as for **printf()**:

normal characters	match themselves
%d	match a decimal integer
%f	match a floating point value
%c	match a character
 - arguments (variable names) indicate where converted input is to be stored, and their types should match the format specifiers
 - **be careful that the variable names are all preceded by & (we'll discuss why)**
- The format string matches against characters in the input stream until the last specifier is matched, or until matching fails
- Returns number of specifiers that are matched **and** assigns to variables

scanf() example

- Program:

```
#include <stdio.h>

int main() {
    int a, b;
    while (scanf("%d %d", &a, &b) == 2)
        printf("A: %d; B: %d\n", a, b);
    return 0;
}
```

- Input (in green) /
Output

```
12 23 39 11 5
20 42
A: 12; B: 23
A: 39; B: 11
A: 5; B: 20
```

Note: the space character in the format string causes all whitespace to be skipped before reading the next number.