CSE 303
Lecture 8

Intro to C programming

read *C Reference Manual*
pp. Ch. 1, 2.2 - 2.4, 2.6, 3.1, 5.1, 7.1 - 7.2, 7.5.1 - 7.5.4, 7.6 - 7.9, Ch. 8;

*Programming in C* Ch. 1-6

slides created by Marty Stepp

http://www.cs.washington.edu/303/
Lecture summary

• History and characteristics of C

• major C language features
  ▪ differences between C and Java

• basic console input and output  (printf and scanf)

• Our learning objectives in C:
  ▪ procedural programming
  ▪ deeper understanding of program compilation and execution
  ▪ learn details of memory management
  ▪ debugging skills
  ▪ software development strategies
History

• created in 1972 by Dennis Ritchie of Bell Labs to accompany the Unix operating system

• designed for creating system software (programs close to the OS that talk directly to hardware)
  ▪ also designed to be hardware-independent (portable)
  ▪ C is also used to develop high-level applications

• currently the 1st or 2nd most widely used language worldwide

• based on ALGOL; has influenced the designs of many languages
  ▪ C++, Java, C#, Perl, PHP, JavaScript, Objective-C, D, ...
Characteristics of C

• fairly similar basic syntax and semantics to Java
  - if/else, for, while, int, double, {}, [], (), ; +- *%/ ++

• much smaller provided standard library / API than Java

• more low-level (more work for programmer, less for compiler)

• procedural (not object-oriented)
  - C (essentially) does not have objects as we know them
  - verb(noun); rather than noun.verb();

• more unsafe (an incorrect program can cause more damage)
  - C programs have more direct access to the system / hardware
First C program

```c
#include <stdio.h>

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

- Kernighan and Ritchie started the convention that the first program you show in a new language should be one that prints "Hello, world!"
Dissecting Hello World

```c
#include <stdio.h>

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

/* Computes greatest common divisor (GCD) with Euclid's algorithm. */
#include <stdio.h>

int main(int argc, char** argv) {
    int a, b, temp, r;

    printf("Please enter two positive integers: ");
    scanf("%d %d", &a, &b);

    if (b > a) {
        temp = a;
        a = b;
        b = temp;
    }

    while ((r = a % b) != 0) {
        a = b;
        b = r;
    }

    printf("The GCD is %d.\n", b);
    return 0;
}
Compiling/running

<table>
<thead>
<tr>
<th>command</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>GNU C compiler</td>
</tr>
</tbody>
</table>

• to compile a program, type:
  
gcc -o target source.c

  (where target is the name of the executable program to build)
  - the compiler builds an actual executable file, not a .class like Java
  - example: gcc -o hi hello.c

• to run your program, just execute that file
  - example: ./hi
### gcc options (partial)

<table>
<thead>
<tr>
<th>option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-W</td>
<td>level of warnings to display (common usage: -Wall for all warnings)</td>
</tr>
<tr>
<td>-o</td>
<td>output executable file name (if omitted, compiles to file a.out)</td>
</tr>
<tr>
<td>-g</td>
<td>generates information for debugger tools</td>
</tr>
</tbody>
</table>

- most common usage for this course:

  ```
gcc -g -Wall -o target source.c
  ```

  - the warnings from -Wall will protect us from unwise idioms
printf

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printf</td>
<td>prints formatted output to console</td>
</tr>
</tbody>
</table>

printf("format string", parameters);

- A format string contains *placeholders* to insert parameters into it:
  - `%d` or `%i` an integer
  - `%lf` a double ('long floating-point')
  - `%s` a string
  - `%p` a pointer (seen later)

```c
int x = 3;
int y = 2;
printf("(%d, %d)\n", x, y);    // (3, 2)
```
printf continued

- A placeholder can specify the parameter's *width* or *precision*:
  - `%8d` an integer, 8 characters wide, right-aligned
  - `% -8d` an integer, 8 characters wide, left-aligned
  - `%.4f` a real number, 4 digits after decimal
  - `%6.2f` a real number, 6 total characters wide, 2 after decimal

- Examples:
  ```c
  int age = 45;
  double gpa = 1.2345678;
  printf("%8d %7.3f\n", age, gpa);
  printf("%8.2f %.1f %10.5f", gpa, gpa, gpa);
  ```

- Output:
  ```
  45   1.234
  1.23 1.2    1.23457
  ```
Same as Java

- general syntax for statements, control structures, function calls

- types int, double, char, long
  - type-casting syntax

- expressions, operators, precedence
  
  +  -  *  /  %  ++  -- 
  
  =  +=  -=  *=  /=  %=
  
  <  <=  ==  !=  >  >=  &&  ||  !

- scope (within set of {   } braces)

- comments: /* ... */ ,  //
  ( // not officially legal until C99)
Mostly the same as Java

• variables
  ▪ can be used without being initialized (!)
  ▪ must be declared at the start of a function or block \textit{(changed in C99)}

• for loops
  ▪ variable cannot be declared in the loop header

• if/else statements, while and do/while loops
  ▪ there is no boolean type \textit{(changed in C99)}
  ▪ any type of value can be used as a test
  ▪ 0 means \texttt{false}, every other number means \texttt{true}

• parameters / returns
  ▪ C has certain features for values vs. references \textit{("pointers")}
Very different from Java

• Strings
  ▪ very clunky to use in C; essentially just arrays of characters
  ▪ are not objects; do not contain methods (external string functions)

• I/O to console and files
  ▪ no Scanner; must use input functions such as scanf
  ▪ console I/O different than file I/O

• errors and exceptions
  ▪ C has no try/catch and does not represent errors as objects
  ▪ errors are usually returned as integer error codes from functions
  ▪ crashes are mostly called "segmentation faults" and are evil
Also very different

• arrays
  ▪ are just bare contiguous blocks of memory
  ▪ have no methods and do not know their own length (!)

• objects
  ▪ C doesn't have them
  ▪ closest similar feature: struct (a set of fields; no methods)

• memory management
  ▪ most memory that you consume, you must explicitly free afterward

• API and provided libraries
  ▪ C doesn't have very many, compared to Java
  ▪ you must write many things yourself (even data structures)
# scanf

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanf</td>
<td>reads formatted input from console</td>
</tr>
</tbody>
</table>

\[
\text{scanf("format string", variables);}\
\]

- uses same syntax for formatted strings, placeholders as `printf`
  - doubles use `%lf` ('long float')

- must precede each variable with an & (address-of operator)

```c
int x;
int y;
printf("Type your x and y values: ");
scanf("%d %d", &x, &y);
```
scanf continued

- scanf returns the number of values successfully read
  - can be examined to see whether the reading was successful

- if # of variables listed doesn't match # of format placeholders:
  - too many variables: later ones ignored
  - too few variables: program crashes!

- string can be complex to match a specific input pattern

```c
int x;
int y;
printf("What is your (x, y) point?\n");
scanf("My point is (%d, %d)", &x, &y);
```
Exercise

- Write a C program that makes change:
  - prompts the user for an amount of money
  - reports the number of pennies, nickels, dimes, quarters, and dollars

- Example:

  **Amount of money? 17.93**
  Pennies: 2
  Nickels: 1
  Dimes: 1
  Quarters: 3
  Dollars: 17
#include <stdio.h>

int main(void) {
    int pennies = 0, nickels = 0, dimes = 0, quarters = 0, dollars;
    double money;

    printf("Amount of money? ");
    scanf("%lf", &money);
    dollars = (int) money;
    pennies = (int) (money * 100) % 100;

    while (pennies >= 25) {
        pennies -= 25;
        quarters++;
    }
    while (pennies >= 10) {
        pennies -= 10;
        dimes++;
    }
    while (pennies >= 5) {
        pennies -= 5;
        nickels++;
    }

    printf("Pennies : %3d\n", pennies);
    printf("Nickels : %3d\n", nickels);
    printf("Dimes : %3d\n", dimes);
    printf("Quarters: %3d\n", quarters);
    printf("Dollars : %3d\n", dollars);
    return 0;
}