CS 126 Lecture P1:
Introduction to C

Outline

• Administrivia
• Background
• Syntax
• Libraries
• Algorithms
To Get Started

1. Visit course web page:
   - [http://www.cs.princeton.edu/courses/cs126](http://www.cs.princeton.edu/courses/cs126)
   - Keep up with announcements
2. Get course packet from Pequod (ready now)
3. Makeup precept by Lisa (7pm, Wednesday)
4. Programming assignment 0 due Wednesday night
5. Get started on readings and exercises
6. Lab TA schedule on the web
7. PA1 in course packet has a typo (see web)

Learning C

1. No prior programming experience assumed!
2. Don’t expect to learn C solely from these lectures--they are just some examples
3. Readings for C programming
   - K&R: for people who have had C or other programming
   - D&D: for beginner programmers
     ~ first 170 pages for the first two weeks
     ~ next 100 pages for the third week
4. Experiment with code fragments on your own
Background

• Born along with Unix in the early 70s, one of the most popular languages today

• Features:
  - Exposes much of machine details
    (Remember “abstractions”? C exposes low level abstractions)
  - Terse syntax

• Consequences:
  - Positive: you can do whatever you want
    -- flexible and powerful
  - Negative: you can do whatever you want
    -- easy to shoot yourself in the foot!
Aspects of Learning to Program

• Syntax -- like learning English
• Algorithms -- like learning to tell a coherent story (not necessarily in English)
• Libraries -- like learning to reuse plots written by others
• These are quite different learning processes

Outline

• Administrivia
• Background
• Syntax
• Libraries
• Algorithms
### Functions

- A C program is a sequence of functions
- f: a C function is very much like a math function
- g: can have more diverse inputs than you have seen
  - example: numbers, strings, more complex data structures
- h: doesn’t have to have outputs
  - their purpose is “side effects”
  - like Pascal “procedures”

### Defining a Function

- First two lines: called “Prototype”, or the “interface”
- The rest (enclosed by {}: is the body, or the “implementation”
- Remember the concept of abstractions?
Functions consist of a sequence of DECLARATIONS followed by a sequence of STATEMENTS.

DECLARATIONS name data variables and specify their types:
- float: float h;
- integer: int i;

STATEMENTS manipulate data, control execution:
- assignment: inc = 0.0;
- control: while (inc < 2.0) { ... }
- function call: printf(...)

Sample program: print table of values of a function

```c
#include <stdio.h>
float f(float x) { return 2.0 - x*x*x; }
main() {
  float h;
  h = 0.0;
  while (h < 2.0) {
    printf("%4.1f %6.3f\n", h, f(h));
    h = h + 0.1;
  }
}
```

Your goals:
this week: understand programs like this
next week: write programs like this

Remember “abstractions”?
Running a program

- When you type commands, you are controlling an abstract machine (called the UNIX shell)

**COMPILE:** convert the program from "human's" language (C) to "machine's" language (stay tuned)

lcc function.c

Result of compilation:
- 1st try: errors in C program SYNTAX
- eventually: a file named a.out

**EXECUTE:** "start the machine"
- starts at machine language instruction corresponding to first statement of main

a.out

Result of execution:
- 1st try: errors in C program SEMANTICS
- eventually: desired "printf" output

```
% lcc function.c
% a.out
```

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>2.000</td>
</tr>
<tr>
<td>0.1</td>
<td>1.999</td>
</tr>
<tr>
<td>0.2</td>
<td>1.992</td>
</tr>
<tr>
<td>0.3</td>
<td>1.973</td>
</tr>
<tr>
<td>0.4</td>
<td>1.936</td>
</tr>
<tr>
<td>0.5</td>
<td>1.875</td>
</tr>
<tr>
<td>0.6</td>
<td>1.784</td>
</tr>
<tr>
<td>0.7</td>
<td>1.657</td>
</tr>
<tr>
<td>0.8</td>
<td>1.488</td>
</tr>
<tr>
<td>0.9</td>
<td>1.271</td>
</tr>
<tr>
<td>1.0</td>
<td>1.000</td>
</tr>
<tr>
<td>1.1</td>
<td>0.669</td>
</tr>
<tr>
<td>1.2</td>
<td>0.272</td>
</tr>
<tr>
<td>1.3</td>
<td>-0.197</td>
</tr>
<tr>
<td>1.4</td>
<td>-0.744</td>
</tr>
<tr>
<td>1.5</td>
<td>-1.375</td>
</tr>
<tr>
<td>1.6</td>
<td>-2.096</td>
</tr>
<tr>
<td>1.7</td>
<td>-2.913</td>
</tr>
<tr>
<td>1.8</td>
<td>-3.832</td>
</tr>
<tr>
<td>1.9</td>
<td>-4.859</td>
</tr>
</tbody>
</table>
Outline

- Administrivia
- Background
- Syntax
- **Libraries**
  - Commonly needed codes written for you already
  - Get an idea of what’s there (look at back of K&R)
  - When you see a possible use, understand the interface
  - Another application of abstractions
- Algorithms
Sometimes you don’t see a precise match in the library...

See if you can leverage what’s there to accomplish what you want.

---

**Example Program: Random Integers**

- Print 10 random integers
  
  library function `rand` (in `stdlib.h`) returns positive integers < RAND_MAX
  
  RAND_MAX is usually 32768 = 2^16

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    for (i = 0; i < 10; i++)
        printf("%d\n", rand());
}
```

Output:

```
16838
5758
10113
17515
31051
5627
23010
7419
16212
4086
```

---

**Random Reals**

- Print 10 random numbers between 0 and 1

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    for (i = 0; i < 10; i++)
        printf("%f\n", 1.0*rand()/RAND_MAX);
}
```

**Integer division [9/4 = 2]**

C has conversion conventions for mixed types [1.0*9/4 = 1.25]

```
Output:
0.513071
0.175726
0.308634
0.534532
0.947630
0.171728
0.702231
0.226417
0.494766
0.124699
```

---
Outline

- Administrivia
- Background
- Syntax
- Libraries
- Algorithms

Print 9-by-9 random patterns
**Top-down Design**

- Break down a big problem into smaller sub problems
- Break down small sub problems into smaller subsub ones
- Repeat until all details are filled out

```c
loop 9 times
    print a random row at a time

loop 9 times
    print a random element at a time

if head print "*
    else print " 
```

Random Numbers ??

Example: Print 9-by-9 random pattern

```c
#include <stdio.h>
#include <math.h>

main()
{
    int i, j;
    for (j = 0; j < 9; j++)
        for (i = 0; i < 9; i++)
            if ((rand()>>13)&1)
                printf("**");
            else printf(" ");
        printf("\n");
}
```

Q: Why not just use the following test?
    if (rand() % 2) ...
A: Random numbers are not random

Ex:
- often, rightmost bits alternate depends on implementation (see next slide)

Never can have all properties of random bits
Ex: sequence is always the same!

Moral: check assumptions about library functions
LFB5R? Cosmic Rays?
Reading Code

• Top-down is the use of abstractions
• Top-down is how programmers write code
• When we read code
  - First, we pretend to be the computer, and “trace” the execution
  - In the process of tracing, the goal is to discover/understand the top-down structures (abstractions)
Just like the last slide, except that it returns # of trials instead of printing stars

Loop for different starting amounts (rows)

Try 5 times for each amount (columns)

Print the result of each trial (a cell)

Experiment [main()]

Ruin sequence [doit()]

Random bit [rand()]