CS 126 Lecture P1: Introduction to C

Outline

- **Administrivia**
- Background
- Syntax
- Libraries
- Algorithms

To Get Started

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

Learning C

- No prior programming experience assumed!
- Don't expect to learn C solely from these lectures--
  they are just some examples
- 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
- Experiment with code fragments on your own

Outline

- ~~Administrivia~~
- **Background**
- Syntax
- Libraries
- Algorithms

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!

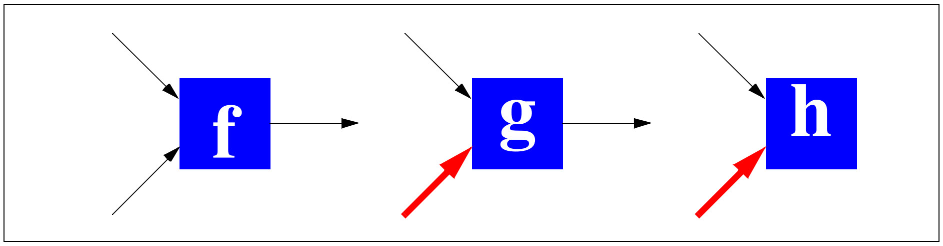
Aspects of Learning to Program

- 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

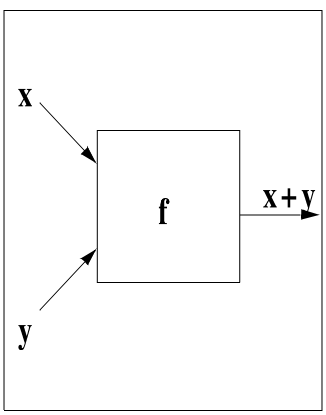
- ~~Administrivia~~
- ~~Background~~
- **Syntax**
- Libraries
- Algorithms

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




```
float
f(float x, float y)
{
   float z;
   z = x + y;
   return z;
}
```

output type
input type
input name
temporary scratch space
output value

- First two lines: called "Prototype", or the "interface"
- The rest (enclosed by { }): is the body, or the "implementation"
- Remember the concept of abstractions?

COS 126 Lecture P1: Introduction to C

- Programs are a sequence of FUNCTIONS
  that manipulate DATA

FUNCTIONS are built-in or defined by users

- library
- user-defined

```
#include <stdio.h>
float f(float x)
  { return 2.0 - x*x*x; }
```

function PROTOTYPE specifies types
of ARGUMENTS and RETURN VALUE

Functions consist of
a sequence of DECLARATIONS
followed by
a sequence of STATEMENTS

function body

DECLARATIONS name data variables
and specify their types

- float
- integer

```
float h;
int i;
```

STATEMENTS manipulate data, control execution

- assignment
- control
- function call

```
inc = 0.0;
while (inc < 2.0) { ... }
printf(...)
```

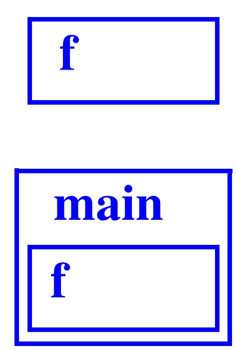

Anatomy of a C program

- Sample program:
  print table of values of a function

```
#include <stdio.h>

float f(float x)                    function
  { return 2.0 - x*x*x; }

main()
  {
     float h;     declaration
     h = 0.0;
     while (h < 2.0)
        {
           printf("%4.1f %6.3f\n", h, f(h));
           h = h + 0.1;
        }                statement
  }
```



- Remember "abstractions"?

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

#include <stdio.h>: "system" declarations
[for printf]

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

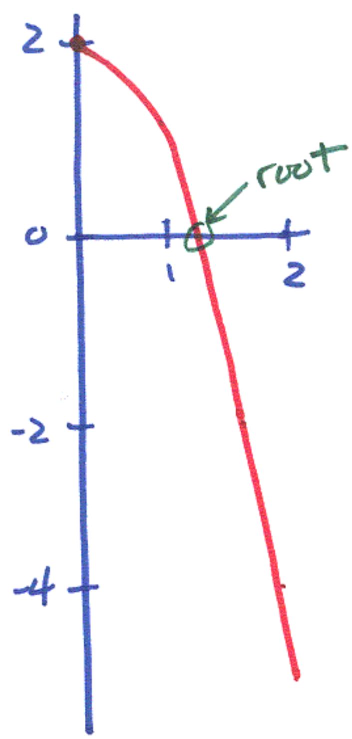
EXECUTE: "start the machine"
starts at machine language instruction
corresponding to first statement of main

```
a.out
```

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

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

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



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

printf library function

- Contact between your C program and the outside world
- printf puts characters on "standard output" (default: "terminal" that you're typing at)

Ex:

```
printf("Hello, world\n");
```

All numbers are represented internally in BINARY (0s and 1s), but we rarely wish to see them in that form

- printf also converts numbers to character sequences that represent them

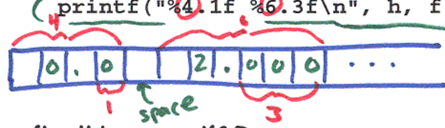
How do you want the numbers to look?

ints: how many digits?
floats: how many digits after decimal point?

Ex:

space

```
printf("%4.1f %6.3f\n", h, f(h));
```



- Very flexible: see K&R pp. 13, 154.

Ex: Could print out title line for table with

```
printf(" x     f(x)\n");
```

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}

```
#include <stdio.h>  ← for printf
#include <stdlib.h> ← for rand
main()
  {
    int i;
    for (i = 0; i < 10; i++)
      printf("%d\n", rand());
  }
```

"new line"

Output:

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

all positive, $< 2^{16}$

Random Reals

- Print 10 random numbers between 0 and 1

```
#include <stdio.h>
#include <stdlib.h>
main()
  {
    int i;
    for (i = 0; i < 10; i++)
      printf("%f\n", 1.0*rand()/RAND_MAX);
  }
```

between 0 and 1

Integer division [9/4 = 2]
C has conversion conventions for
mixed types [1.0*9/4 = 2.25]

Output:

```
0.513871
0.175726
0.308634
0.534532
0.947630
0.171728
0.702231
0.226417
0.494766
0.124699
```

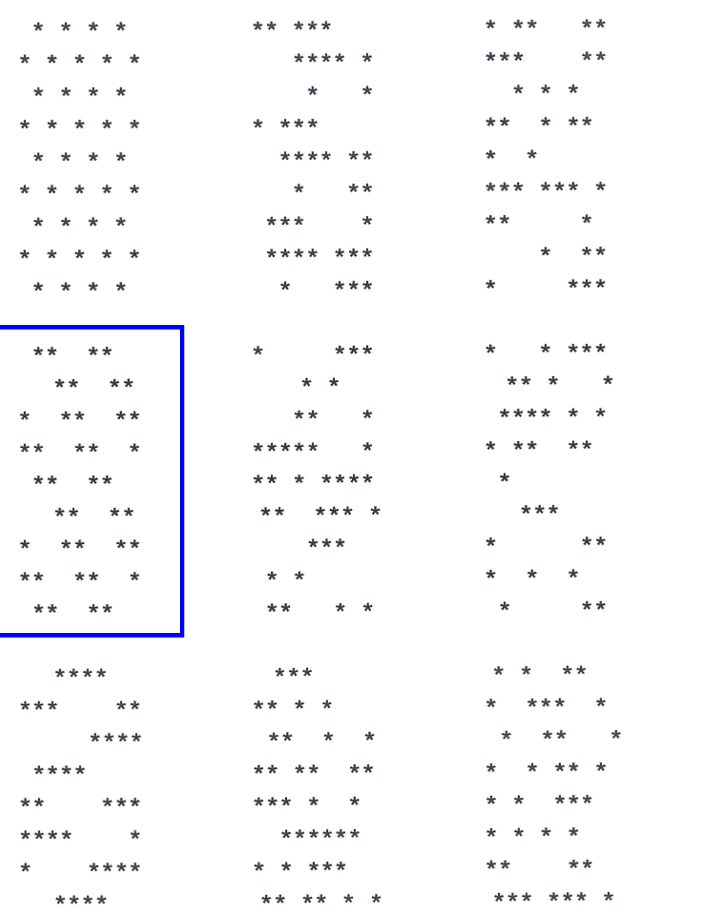
all between 0 and 1

- Sometimes you don't see a precise match in the library...
- See if you can leverage what's there to accomplish what you want.

Outline

- ~~Administrivia~~
- ~~Background~~
- ~~Syntax~~
- ~~Libraries~~
- **Algorithms**

Print 9-by-9 random patterns



P1.10

Top-down Design

```
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 " "
```

- 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

Random Numbers ??

Example: Print 9-by-9 random pattern

```
#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");
     }
}
```

Print one element

Print one row

Print all rows

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
LFBSR? 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)

Another Example: Gambler's Ruin

- Simulate gambler placing $1 even bets
- How long does the game last?

```
#include <stdio.h>
#include <stdlib.h>
main()
  { int i, cash, seed;
     scanf("%d %d", &cash, &seed);
     srand(seed);
     while (cash > 0)
       {
         if ((rand()>>13)&1) cash++; else cash--;
         for (i = 0; i < cash; i++) printf(" ");
         printf("*\n");
       }
  }
```

while I still have money, repeat:

make a bet

print a star at the "right" place

scanf function takes input from terminal
srand initializes random number generator

```
.     % a.out              % a.out
.     4 1231               4 1234
.        *                      *
.         *                    *
.        *                    *
.         *                    *
.        *                    *
.       *                      *
.      *                        *
.                              *
.                               *
.                                *
.                               *
.                                *
```

Hmmmmm.

Gambler's Ruin Experiment

(marrying previous two programs)

```
#include <stdio.h>
#include <stdlib.h>
int doit(int cash)
  { int cnt;
    for (cnt = 0; cash != 0; cnt++)
      if ((rand()>>13)&1) cash++; else cash--;
    return cnt;
  }
main()
  { int j, t;
    for ( j = 2; j < 10; j++)
      {
        printf("%2d  ", j);
        for (t = 0; t < 5; t++)
          printf(" %7d", doit(j));
        printf("\n");
      }
  }
```

function returns # bets

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)

Output:

2	2	6	304	2	2
3	33	17	15	53	29
4	22	1024	7820	22	54
5	243	25	41	7	249
6	494	14	124	152	14
7	299	33	531	49	93
8	218	10650	36	42048	248
9	174090315	83579	299	759	69

!

- Well-studied random process in mathematics
  Ref: Feller. Probability Theory and Applications

LEVELS OF ABSTRACTION
- random bit, ruin sequence, experiment

Experiment [main()]

Ruin sequence [doit()]

Random bit [rand()]