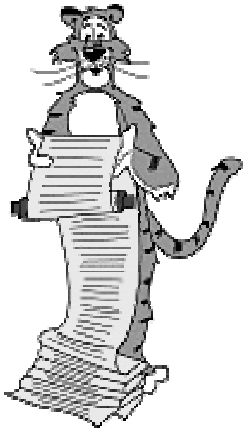
Lecture P1: Introduction to C



```
#include <stdio.h>
int main(void) {
  printf("This is a C program\n");
  return 0;
}
```


Learning C

No prior programming experience assumed.

- Although it will make things easier.

Programming is learned with practice.

- Don't expect to learn solely from these lectures.
- Do exercises.
- Experiment with code on your own.

Do reading.

- K&R for people with programming experience.
- Deitel & Deitel for beginners.
  - first 170 pages first two weeks
  - next 100 pages third week

C Background

Born along with Unix in the early 1970's.

- One of most popular languages today.

Features.

- Exposes much of machine detail.
  - remember abstractions?
  - C exposes low-level abstractions
- Concise language.

Consequences.

- Positive: you can do whatever you want.
  - flexible and powerful
- Negative: you can do whatever you want.
  - shoot yourself in the foot

Aspects of Learning to Program

C Syntax

- Learning English.

Algorithms

- Learning to tell a coherent story (not necessarily in English).

Libraries

- Learning to reuse plots written by others.

These are different skills and learning processes.

An Example

Print a table of values of function f(x) = 2 - x^3 . A first attempt:

input/output library functions

declare real-valued

printf used to print characters to screen

print the values of x and y to the screen

print new values of x and

36 lines of similar code is omitted

end of code

table1.c

```
#include <stdio.h>

int main(void) {
  float x, y;

  printf("x     f(x)\n");
  x = 0.0;
  y = 2.0 - x*x*x;
  printf("%4.1f %6.3f\n", x, y);
  x = 0.1;
  y = 2.0 - x*x*x;
  printf("%4.1f %6.3f\n", x, y);
  . . .
  x = 1.9;
  y = 2.0 - x*x*x;
  printf("%4.1f %6.3f\n", x, y);
  return 0;
}
```


Printf Library Function

Contact between your C program and outside world.

- Puts characters on "standard output."
- By default, stdout is the "terminal" that you're typing at.

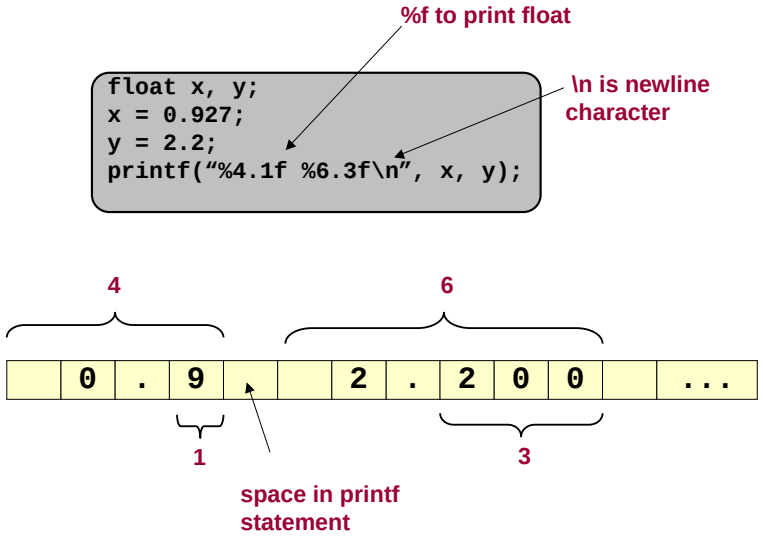
Internally, all numbers and characters represented in BINARY (0's, 1's).

- `printf` converts from binary to more useful form (`int`, `float`).

Formatted output.

- How do you want the numbers to look?
  - integers, how many digits?
  - real numbers, how many digits after decimal place?
- Very flexible, see K&R pp. 13, 154.

Anatomy of Printf



Running a Program in Unix

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).
  - 1st try: syntax errors in C program
  - eventually, a file named a.out
- Execute: start the machine (at first instruction corresponding to first statement of `main`).
  - 1st try: semantic errors in C program
  - eventually, desired "printf" output

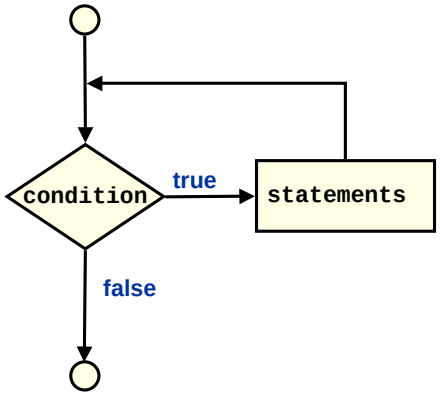
Unix

```
% gcc table.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
```

Anatomy of a While Loop

Previous program repeats the same code over and over.

- Repetitive code boring to write and hard to debug.
- Use while loop to repeat code.



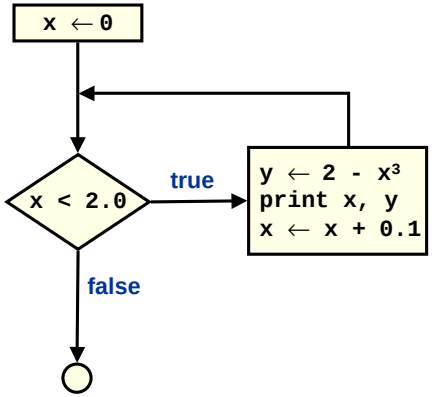

```
while (condition) {
  statements;
}
```

while loop

Anatomy of a While Loop

Previous program repeats the same code over and over.

- Repetitive code boring to write and hard to debug.
- Use while loop to repeat code.




```
x = 0.0;
while (x < 2.0) {
  y = 2 - x*x*x;
  printf("%f %f", x, y);
  x = x + 0.1;
}
```

C code

While Loop Example

Print a table of values of function f(x) = 2 - x^3. A second attempt.

uses while loop

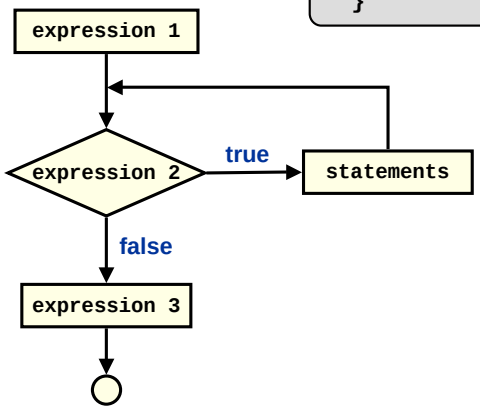
table2.c

```
#include <stdio.h>

int main(void) {
  float x, y;

  printf(" x     f(x)\n");
  x = 0.0;
  while (x < 2.0) {
    y = 2.0 - x*x*x;
    printf("%4.1f %6.3f\n", x, y);
    x = x + 0.1;
  }
  return 0;
}
```


Anatomy of a For Loop

The for loop is another common repetition structure.

```
for (expr1; expr2; expr3) {
  statements;
  }
```



For Loop Example

Print a table of values of function f(x) = 2 - x^3. A third attempt.

uses for loop

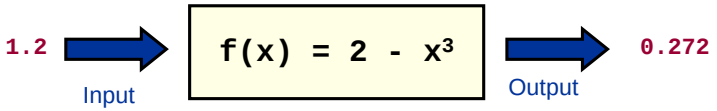
table3.c

```
#include <stdio.h>

int main(void) {
  float x, y;

  printf(" x     f(x)\n");
  for (x = 0.0; x < 2.0; x = x + 0.1) {
    y = f(x);
    printf("%4.1f %6.3f\n", x, y);
  }
  return 0;
}
```


Anatomy of a Function

Convenient to break up programs into smaller modules or functions.

- Layers of abstraction.
- Makes code easier to understand.
- Makes code easier to debug.
- Makes code easier to change later on.




```
float f (float x) {
  return 2 - x*x*x;
  }
```

function in C

Anatomy of a Function

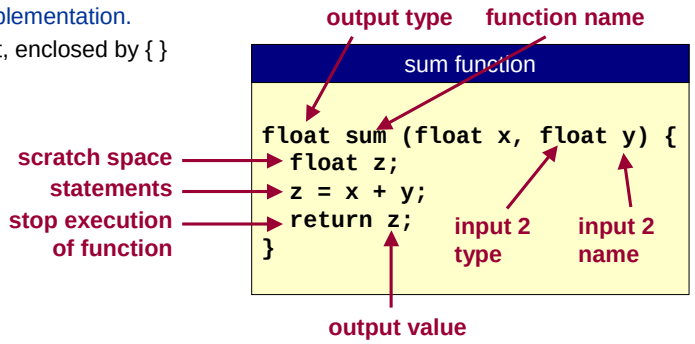
C function similar to mathematical function.

Prototype or interface is first line of C function.

- specifies input argument(s) and their types
  - can be integers, real numbers, strings, vectors, user-defined
- specifies return value

Body or implementation.

- The rest, enclosed by { }



Function Example

Print a table of values of function f(x) = 2 - x^3. A fourth attempt.

table4.c

```
#include <stdio.h>

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

int main(void) {
  float x;

  printf(" x     f(x)\n");
  for (x = 0.0; x < 2.0; x += 0.1) {
    printf("%4.1f %6.3f\n", x, f(x));
  }
  return 0;
}
```

x += 0.1 is shorthand in C for x = x+ 0.1

no need for { } if only one statement

What is a C Program?

C PROGRAM: a sequence of FUNCTIONS that manipulate data.

- `main` function is first one executed.

A FUNCTION consists of a sequence of DECLARATIONS followed by a sequence of STATEMENTS.

- Can be built-in like `printf`.
- Or user-defined like `f` or `sum`.

A DECLARATION names variables and defines type.

- float `float x;`
- integer `int i;`

A STATEMENT manipulate data or controls execution.

- assignment: `x = 0.0;`
- control: `while (x < 2.0) {...}`
- function call: `printf(...);`

Anatomy of a C Program

table3.c

```
#include <stdio.h>

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

int main() {
  float x;

  x = 0.0;
  while (x < 2.0) {
    printf("%4.1f %6.3f\n", x, f(x));
    x = x + 0.1;
  }

  return 0;
}
```

function

declaration

assignment statement

function call statement

flow control statement

Random Integers

Print 10 "random" integers.

- Library function `rand()` in `stdlib.h` returns integer between 0 and `RAND_MAX - 1` (usually 32767).

int.c

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

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

Unix

```
% gcc int.c
% a.out
16838
5758
10113
17515
31051
5627
23010
7419
16212
4086
```


Random Integers

Print 10 "random" integers between 0 and 599.

- No precise match in library.
- Try to leverage what's there to accomplish what you want.

int.c

```
#include <stdio.h>
#include <stdlib.h>
#define N 600

int randomInteger(int n) {
  return rand() % n;
}

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

p % q gives remainder of p divided by q

Unix

```
% gcc int.c
% a.out
168
575
310
562
230
341
16
386
```

Random Real Numbers

Print 10 “random” real numbers between 0.0 and 1.0.

- No precise match in library.
- Try to leverage what’s there to accomplish what you want.

real.c

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

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

Integer division: 16838 / 32767 = 0.
C has conversions for mixed types:
1.0 * 15838 / 32767 = 0.513871.

Unix

```
% gcc real.c
% a.out
0.513871
0.175726
0.308634
0.534532
0.947630
0.171728
0.702231
0.226417
0.494766
0.124699
```

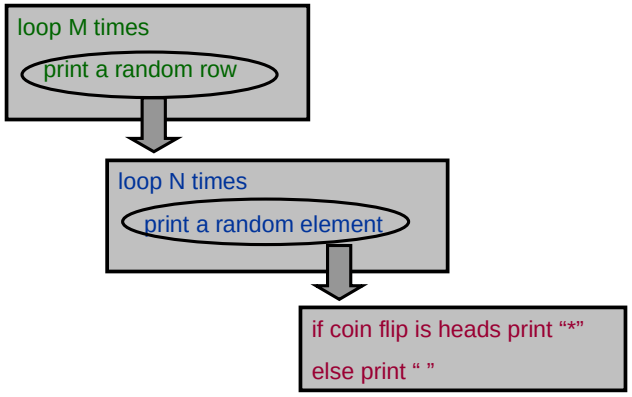

Random M x N Pattern

```
 * * * *          ** ***          * **   **
* * * * *           **** *        ***    **
 * * * *             *   *         * * *
* * * * *          * ***          **  * **
 * * * *             **** **       *  *
* * * * *             *   **       *** *** *
 * * * *            ***   *       **     *
* * * * *           **** ***            *  **
 * * * *             *   ***       *     ***

  **  **           *     ***       *   * ***
   **  **            * *             ** *   *
*  **  **           **   *           **** * *
**  **  *          *****  *          * **  **
 **  **            ** * ****          *
  **  **           **  *** *           ***
*  **  **             ***             *      **
**  **  *           * *               *  *  *
 **  **             **   * *           *     **
```


Random M x N Pattern

Top-down design.

- Break a big problem into smaller subproblems.
- Break down subproblems into sub-subproblems.
- Repeat until all details filled in.



Random M x N Pattern

randpattern.c

```
#include <stdio.h>
#define M 9
#define N 9

int randomInteger(int n) {...}

int main(void) {
  int i, j;

  for (i = 0; i < M; j++) {
    for (j = 0; j < N; j++) {
      if (randomInteger(2) == 1) printf("*");
      else printf(" ");
    }
    printf("\n");
  }

  return 0;
}
```

Print a random element

Print a random M x N pattern.

Libraries

How is library function `printf()` created?

- User doesn't need to know details (see COS 217).
- User doesn't want to know details (abstraction).

How is library function `rand()` created?

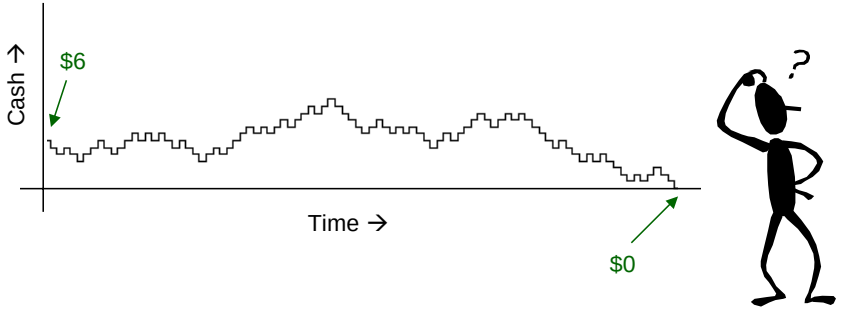
- Linear feedback shift register? Cosmic rays?
- Depends on compiler and operating system.
- Caveat: "random" numbers are not really random.
  - can never have all properties of random bits
  - computers do exactly what we tell them to do!
- Note: on many systems, our `randomInteger` is very bad.

Moral: check assumptions about library function.

Gambler's Ruin

Simulate gambler placing $1 even bets.

- Will gambler always go broke.
- If so, how long will it take if gambler starts with $c?



Gambler's Ruin

gambler.c

```
int main(void) {
   int i, cash, seed;
   scanf("%d %d", &cash, &seed);
   srand(seed);

   while (cash > 0) {
      if (randomInteger(2) == 1)
         cash++;
      else
         cash--;
      for (i = 0; i < cash; i++)
         printf(" ");
      printf("*\n");
   }
   return 0;
}
```

scanf takes input from terminal

`srand` sets random seed

while I still have money left, repeat

make a bet

print a * at the right place

Gambler's Ruin

Simulate gambler placing $1 even bets.
Q. How long does the game last if we start with $c ?

Unix

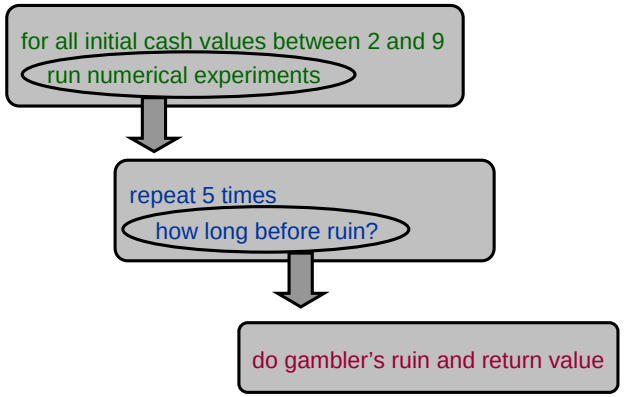
```
% gcc gambler.c

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

Hmmm.

Top-Down Design of Numerical Experiment

Goal: run an experiment to determine how long does it take to go broke.

- Find out how this changes for different values of c.



Gambler's Ruin Experiment

gexperiment.c

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

int doit (int cash) {
  int cnt;
  for (cnt = 0; cash > 0; cnt++)
    if (randomInteger(2)) cash++;
    else cash--;
  return cnt;
}
int main(void) {
  int cash, t;
  for (j = 2; j < 10; j++) {
    printf("%2d  ", j);
    for (t = 0; t < 5; t++)
      printf("%7d", doit(cash));
    printf("\n");
  }
  return 0;
}
```

single experiment (code as before)

repeat for all initial cash values 2 to 9

repeat 5 times

Gambler's Ruin Experiment

Unix

```
% gcc gexperiment.c
% a.out
```

initial cash

bets

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

How long will it take to go broke?

Layers of abstraction.

- Random bit → gambler's ruin sequence → experiment.

Programming Advice

Understand your program.

- What would the machine do?

Read, understand, and borrow from similar code.

Develop programs incrementally.

- Test each piece separately before continuing.
- Plan multiple lab sessions.

Debugging

Find the FIRST bug and fix it.

Syntax error - illegal C program.

- Compiler error messages are good - tell you what you need to change.

Semantic error - wrong C program.

- Use “printf” method.

Always a logical explanation.

Enjoy the satisfaction of a fully functional program!

Programming Style

Concise programs are the norm in C.

Your goal: write READABLE and EFFICIENT programs.

- Use consistent indenting.
  - automatic indenting in emacs
- Choose descriptive variable names.
- Use comments as needed.

“Pick a style that suits you, then use it consistently.”

-Kernighan and Ritchie

Summary

Lots of material.

C is a structured programming language.

- Function, while loop, for loop.
- Can design large robust programs with these simple tools.

Programming maturity comes with practice.

- Everything seems simpler in lecture and textbooks.
- Always more difficult when you do it yourself!
- Learn main ideas from lecture, learn to program by writing code.