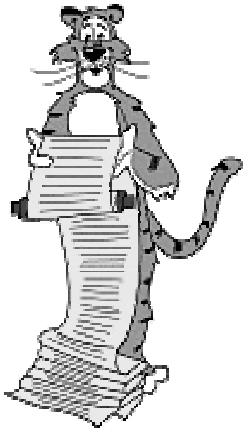
Lecture P1: Introduction to C



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

Learning to Program

Programming is learned with practice and patience.

- Don't expect to learn solely from these lectures.
- Do exercises.
- Experiment and write lots of code.

Do reading.

- Finish King Chapters 1-6 today!

Aspects of learning to program.

- Language syntax.
- Algorithms.
- Libraries.
- These are different skills and learning processes.

C Background

Born along with Unix in the early 1970s.

- One of most popular languages today.

C Features.

- Concise.
- Widespread usage.
- Exposes low-level details of machine.

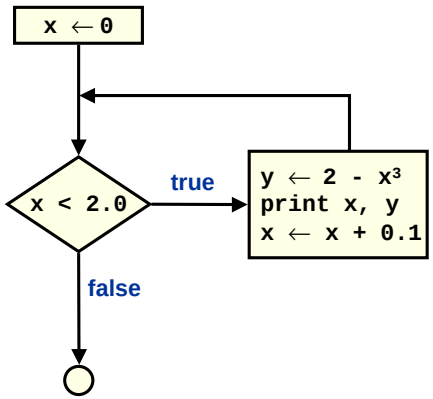
Consequences.

- Positive: you can do whatever you want.
- Negative: you can do whatever you want.

Language Syntax: Loops

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

```
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
```



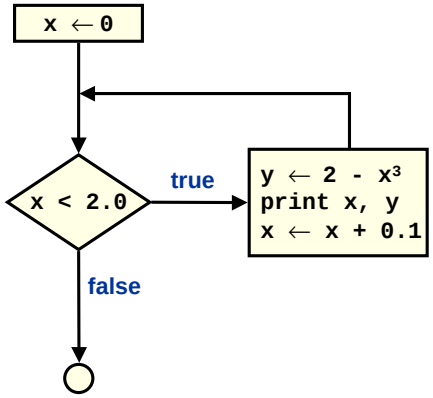
Anatomy of a While Loop

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

- Use `while` loop to perform repetitive tasks.

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

C code



Language Syntax: Loops

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

- Use `while` loop to perform repetitive tasks.

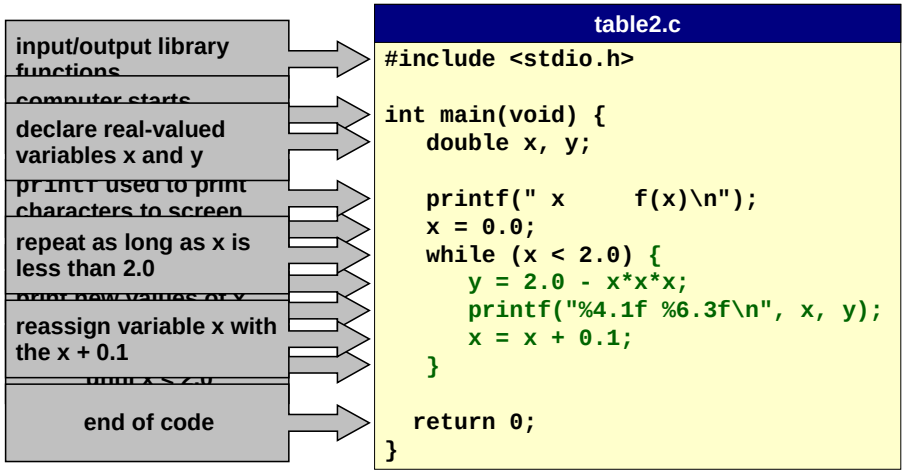


table2.c

```
#include <stdio.h>

int main(void) {
   double 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;
}
```


Debugging 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.
- Syntax error: illegal C program.
- Semantic error: wrong C program.
- Debugging: cyclic process of editing, compiling, and fixing errors.
  - always a logical explanation
  - enjoy the satisfaction of a working program!

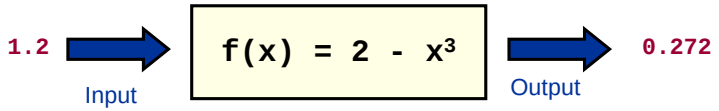
Unix

```
% gcc table.c
% a.out
```

Language Syntax: Functions

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.




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

C function

Anatomy of a Function

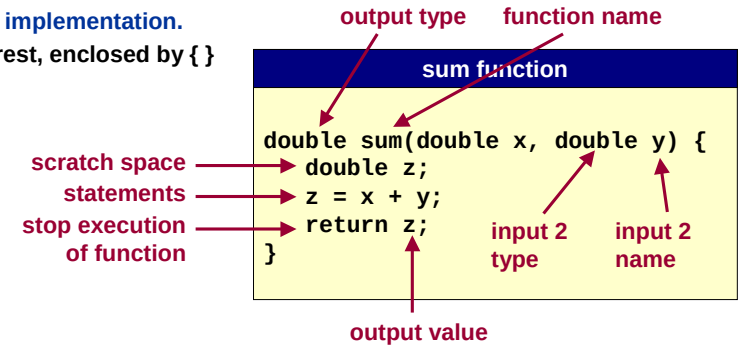
C function similar to mathematical function.

Prototype or interface is first line of C function.

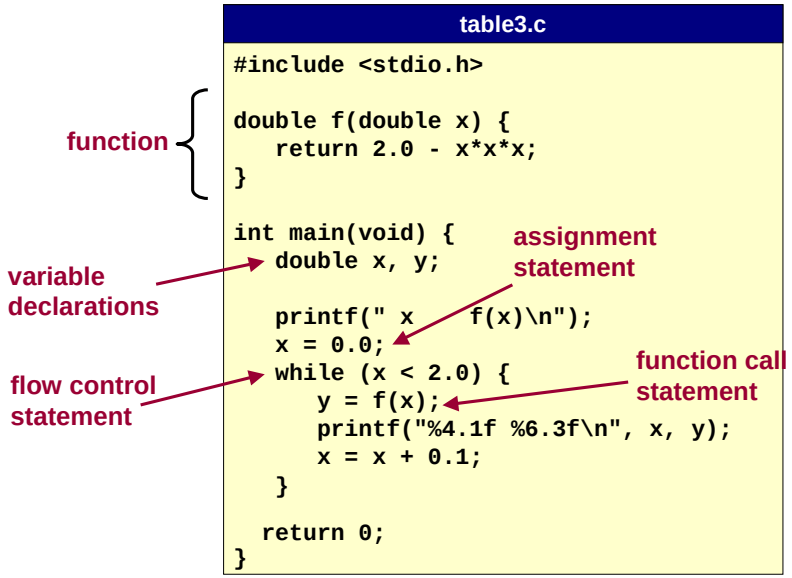
- specifies input argument(s) and their types
  - can be integers, real numbers, strings, vectors, user-defined
- specifies return value

Body or implementation.

- The rest, enclosed by { }



Anatomy of a C Program



Library Functions: `printf()`

Library functions.

- Functions provided as part of C implementation.

Example: `printf()`.

- Contact between your C program and outside world.
- Puts characters on "standard output."
- By default, stdout is the "terminal" that you're typing at.

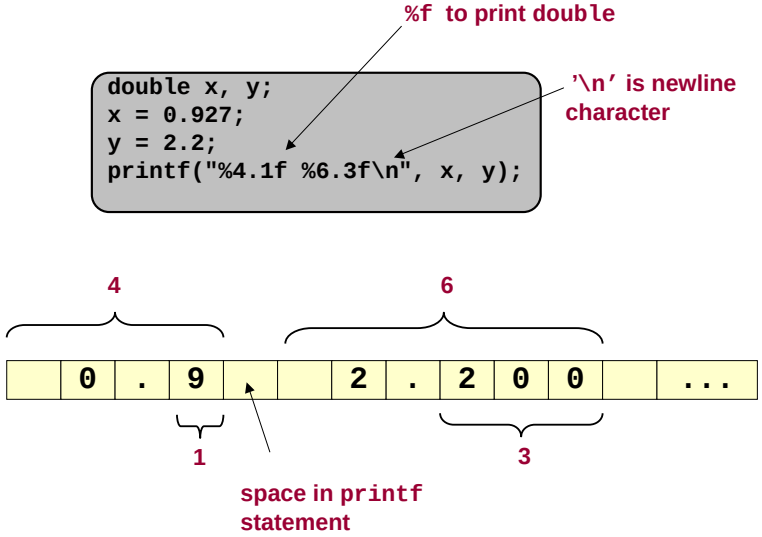
Internally, all numbers represented in BINARY (0's and 1's).

- `printf()` displays more useful representation (`int`, `double`).

Formatted output.

- How do you want the numbers to look?
  - integers, how many digits?
  - real numbers, how many digits after decimal place?
- Very flexible.

Library Functions: `printf()`



Library Functions: `printf()`

How is library function `printf()` implemented?

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

Library Functions: `rand()`

Print 10 "random" integers.

- Library function `rand()` in `stdlib.h` returns integer between 0 and `RAND_MAX` (32,767 = 2^{16} - 1 on arizona).

int.c

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

int main(void) {
   int i = 0;
   while (i < 10) {
      printf("%d\n", rand());
      i = i + 1;
   }
   return 0;
}
```

i++;

Unix

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


Library Functions: `rand()`

Print 10 "random" integers between 0 and 599.

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

int600.c

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

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

int main(void) {
   int i = 0;
   while (I < 10) {
      printf("%d\n", randomInteger(600));
      i++;
   }
   return 0;
}
```

p % q gives remainder of p divided by q

Unix

```
% gcc int600.c
% a.out
168
5
1
175
310
562
230
341
16
386
```


Library Functions: `rand()`

How is library function `rand()` implemented?

- Linear feedback shift register? Cosmic rays?
- Depends on compiler and operating system.
- Caveat 1: "random" numbers are not really random.

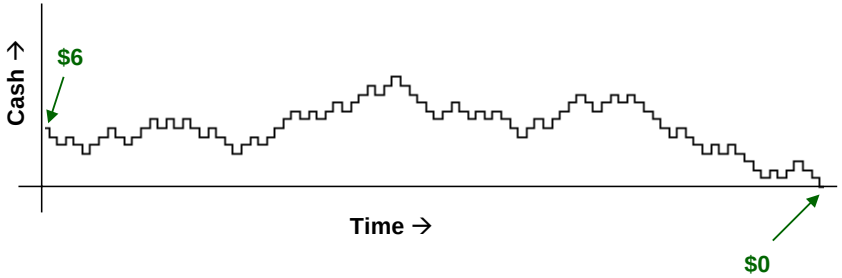
- Caveat 2: on many systems, our `randomInteger()`is very poor.

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

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

int randomInteger(int n) { ... }

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

  while (cash > 0) {
     if (randomInteger(2) == 1)
        cash++;
     else
        cash--;
     printf("%d\n", cash);
  }
  return 0;
}
```

scanf() takes input from terminal

srand() sets random seed

while I still have money left, repeat

make a bet

print money left

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
4 543            4 1234
3                3
4                2
5                3
4                4
3                3
4                4
3                5
2                6
1                7
0                6
                 7
                 8
                 9
```

Hmmm.

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
4 543            4 1234
***              ***
****             **
*****            ***
****             ****
***              ***
****             ****
***              *****
**               ******
*                *******
                 ******
                 *******
                 ********
                 *********
```

To print plot, replace:

```
printf("%d\n", cash);
```

with

```
i = cash;
while (i > 0) {
   printf("*");
   i--;
}
printf("\n");
```

Gambler's Ruin Numerical Experiment

Goal: run experiment to see how long it takes to go broke.

- Do for different values of starting cash values c.

Unix

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

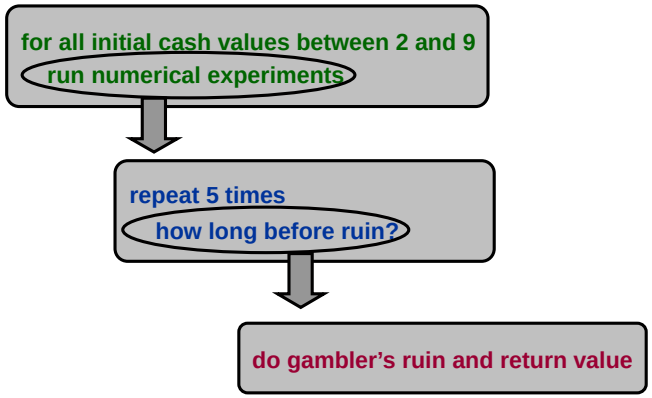
initial cash | # bets

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

Top-Down Design of Numerical Experiment

Goal: run experiment to see how long it takes to go broke.

- Do for different values of starting cash values c.



Gambler's Ruin Numerical Experiment

single experiment (code as before)

return # of times instead of printing each trial

gexperiment.c

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

int randomInteger(int n) { ... }

int doit(int cash) {
   int cnt = 0;
   while (cash > 0) {
      if (randomInteger(2) == 1) cash++;
      else cash--;
      cnt++;
   }
   return cnt;
}
```


Gambler's Ruin Numerical Experiment

repeat for all initial cash values 2 to 9

repeat 5 times

gexperiment.c (cont)

```
int main(void) {
   int cash, t;

   cash = 2;
   while (cash < 10) {
      printf("%2d ", cash);
      t = 0;
      while (t < 5) {
         printf("%7d", doit(cash));
         t++;
      }
      printf("\n");
      cash++;
   }

   return 0;
}
```

Do one experiment

Gambler's Ruin Numerical Experiment

Unix

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

initial cash / # bets

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 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

Programming Advice

Understand your program.

- What would the machine do?

Read, understand, and borrow from similar code.

" Good artists borrow.
Great arists steal. "

Develop programs incrementally.

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

Summary

Lots of material.

C is a structured programming language.

- Functions, loops.
- Simple, but powerful 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.

For Loop Example

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

use a for loop

table4.c

```
#include <stdio.h>

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

int main(void) {
   double x, y;

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

   return 0;
}
```

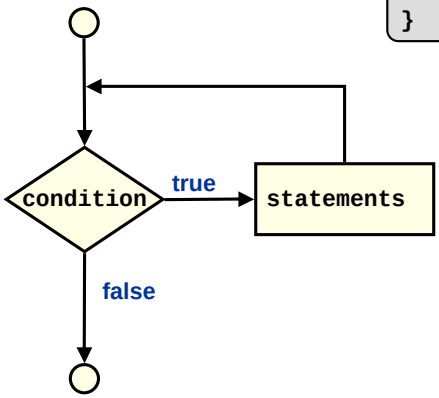
x += 0.1 is shorthand in C for x = x+ 0.1

Anatomy of a While Loop

The while loop is a common repetition structure.

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

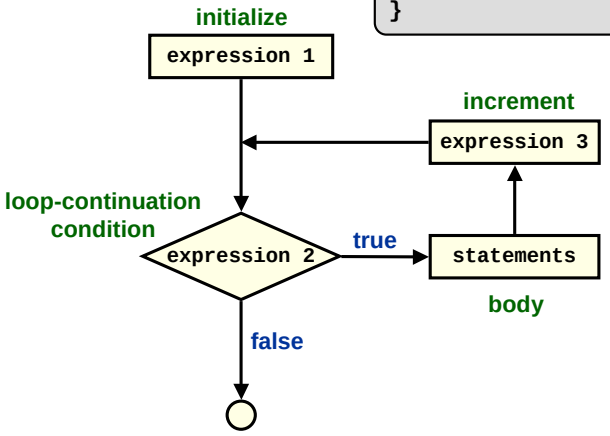
while loop



Anatomy of a For Loop

The for loop is another common repetition structure.

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

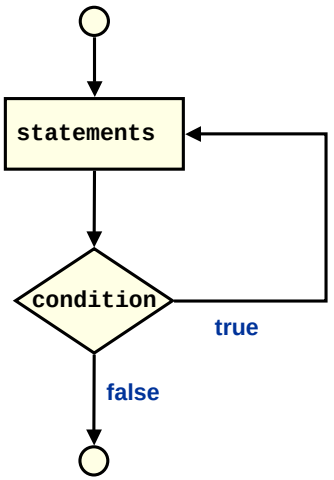


Anatomy of a Do-While Loop

The do-while loop is not-so-common repetition structure.

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

do-while loop



What is a C Program?

C PROGRAM: a sequence of FUNCTIONS that manipulate data.

- `main()` function executed first.

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(x)` or `sum(x, y)`.

A DECLARATION names variables and defines type.

- double `double x;`
- integer `int i;`

A STATEMENT manipulate data or controls execution.

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