#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 1970’s.
- 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.
An Example

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

<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>

```c
#include <stdio.h>

int main(void) {
    double 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 = 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 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.
Anatomy of Printf

double x, y;
x = 0.927;
y = 2.2;
printf("%4.1f %6.3f\n", x, y);

%f to print double

'\n' is newline character

Space in printf statement

4

6

...
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.
  - 1st try: syntax errors in C program
  - eventually, a file named a.out

- **Execute**: start the machine.
  (at first instruction in `main`)
  - 1st try: semantic errors in C program
  - eventually, desired "printf" output

```
% gcc table.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>
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.

C code:

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

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

```
#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;
}
```
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.

\[ f(x) = 2 - x^3 \]

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>0.272</td>
</tr>
</tbody>
</table>

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 { }
#include <stdio.h>

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

int main(void) {
    double x, y;
    printf(" x    f(x)\n");
    x = 0.0;
    while (x < 2.0) {
        y = f(x);
        printf("%4.1f %6.3f\n", x, y);
        x = x + 0.1;
    }
    return 0;
}
Random Integers

Print 10 "random" integers.

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

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

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

```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;
}
```

Unix

```
% gcc int600.c
% a.out
168
575
101
1175
310
562
230
341
16
386
```

\[ p \mod q \text{ gives remainder of } p \text{ divided by } q \]
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.

```python
if coin flip is heads print "*"
else print "   "
```

```plaintext
loop M times
    print a random row
```

```plaintext
loop N times
    print a random element
```

```plaintext
```
```c
#include <stdio.h>
#define M 9
#define N 9
int randomInteger(int n) {...}

int main(void) {
    int i, j;
    i = 0;
    while (i < M) {
        j = 0;
        while (j < N) {
            if (randomInteger(2) == 1) printf("*");
            else printf(" ");
            j++;
        }
        printf("\n");
        i++;
    }
    return 0;
}
```

Print random M x N pattern.

Print a random element.

Print a random row.
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 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

```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
- Print money left
- Make a bet

---

23
Gambler’s Ruin

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

<table>
<thead>
<tr>
<th>Unix</th>
</tr>
</thead>
<tbody>
<tr>
<td>% gcc gambler.c</td>
</tr>
<tr>
<td>% a.out % a.out</td>
</tr>
<tr>
<td>4 543 4 1234</td>
</tr>
<tr>
<td>3 3</td>
</tr>
<tr>
<td>4 2</td>
</tr>
<tr>
<td>5 3</td>
</tr>
<tr>
<td>4 4</td>
</tr>
<tr>
<td>3 3</td>
</tr>
<tr>
<td>4 4</td>
</tr>
<tr>
<td>3 5</td>
</tr>
<tr>
<td>2 6</td>
</tr>
<tr>
<td>1 7</td>
</tr>
<tr>
<td>0 6</td>
</tr>
<tr>
<td>7 7</td>
</tr>
<tr>
<td>8 8</td>
</tr>
<tr>
<td>9 9</td>
</tr>
</tbody>
</table>

Hmmm.
Gambler’s Ruin

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

Unix

```c
% gcc gambler.c
% a.out
4 543
***
****
*****
*****
***
***
***
**
*

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

To print plot, replace:

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

with

```c
i = cash;
while (i > 0) {
    printf("*"卓越);
    i--;
}
printf("\n");
```
Top-Down Design of Numerical Experiment

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

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

for all initial cash values between 2 and 9
run numerical experiments

repeat 5 times
how long before ruin?

do gambler’s ruin and return value
Gambler’s Ruin Experiment

```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;
}
```

single experiment (code as before)
Gambler’s Ruin Experiment

```c
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;
}
```

repeat for all initial cash values 2 to 9

repeat 5 times
# Gambler’s Ruin Experiment

## Unix

<table>
<thead>
<tr>
<th>initial cash</th>
<th># bets</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>243</td>
</tr>
<tr>
<td>6</td>
<td>494</td>
</tr>
<tr>
<td>7</td>
<td>299</td>
</tr>
<tr>
<td>8</td>
<td>218</td>
</tr>
<tr>
<td>9</td>
<td>174090315</td>
</tr>
</tbody>
</table>

How long will it take to go broke?

Layers of abstraction.
- Random bit $\rightarrow$ gambler’s ruin sequence $\rightarrow$ 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.
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
#include <stdio.h>
#define l111 0xFFFF
#define l11 for
#define l111 if
#define l1111 unsigned
#define l111 struct
#define l1111 short
#define l1111 long
#define l1111 putchar
#define l11111 (l) l=malloc(sizeof(l111 lll1l1));l->lll1l=1-1;l->ll1l1=1-1;
#define l1l1l *lllll++=l1ll%10000;l1ll/=10000;
#define l1ll1 ll111(!l1->lll1l){l1l1l(l1->lll1l);l1->lll1l->ll1l1=l1;}
lllll=(l1=l1->lll1l)->lll;ll=1-1;
define l1ll1 1000

1111, *11111 ;1111
111111;);main () {1111 11111
11111, * malloc ( ) ; 1111
1111 1111, 1;1111 11111 *1111, *
11111; 11111 =1-1 ;1< 14 ;11111("\"t\"8>&1"9!.)>v1" [1]^'L'),++1
);scanf("%d",&1) ;11111(111) 11111(1111)
) (11=111) ->
111[111->ll1l11[1-1] =1]=l1ll1;111(111
=1+1;111<=1;
++111)(11=1111; 1111 = (1111=(
1111=111)) ->
111; 11111 = (1111=111)->111;
11=1111=1-1
);111 (;1111-> 11111|11111=! 1111;)(1111
++)111++1111++ ;11111 11111
(1111 =11111-> 111111)->1111;
})111 (;1111;
)(11111 11111
(1111) ) * 11111=11111;
)
111 (l=(11=1=1)
1); (1<11111) &&
(11->ll111[ 1] !=11111);++1); 111 (;11;11=
11->ll1111,1= 11111(111(--1 1)>=1=1--;=1,
++111)printf( (11)?((11%19) ?"%04d": (11=
19,"\n%04d") ):"%4d",11-> 11111[ 1] ); }
11111(10); }
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.
Lecture P1: Supplemental Notes
Anatomy of a While Loop

The while loop is a common repetition structure.

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

while loop
The for loop is another common repetition structure.

```java
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)
For Loop Example

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

```c
#include <stdio.h>

int main(void) {
    double x, y;

    printf(" x     f(x)\n");
    for (x = 0.0; x < 2.0; x = x + 0.1) {
        y = 2 - x*x*x;
        printf("%4.1f %6.3f\n", x, y);
    }

    return 0;
}
```
Print a table of values of function $f(x) = 2 - x^3$. A fifth attempt.

```c
#include <stdio.h>

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

int main(void) {
    double 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;
}

no need for { } if only one statement

x += 0.1 is shorthand in C for $x = x + 0.1$
```
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(...);
Random Integers

Print 10 "random" integers.

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

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

int main(void) {
    int i;
    for (i = 0; i < 10; i++)
        printf("%d
", 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.

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

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

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

Unix

```
gcc int600.c
gcc int600.c
```

```
168
575
101
75
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.

```
#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
```