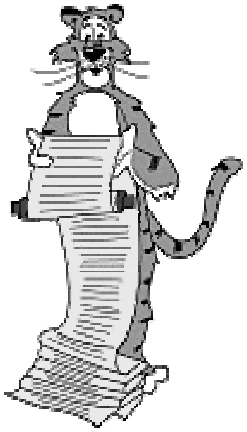
Lecture P1: Introduction to C



```
#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 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:

```
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
```

table1.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 = 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.

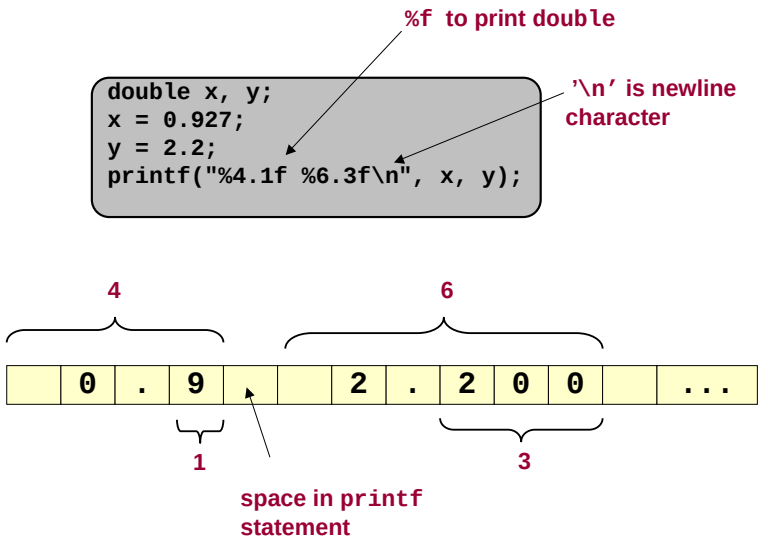
Internally, all numbers represented in BINARY (0's and 1's).

- `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



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

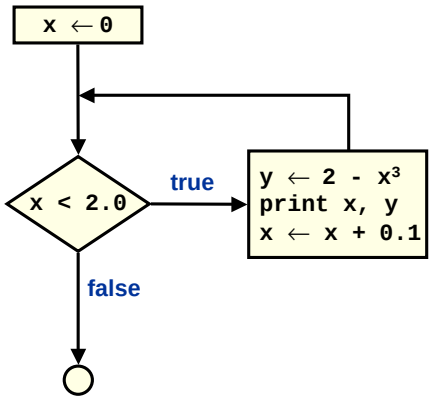
Unix

```
% 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.




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

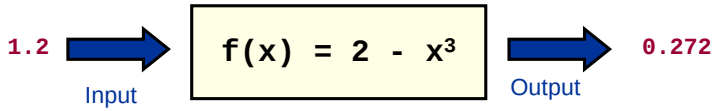
table2.c

```
#include <stdio.h>

int main(void) {
  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.




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

C function

Anatomy of a Function

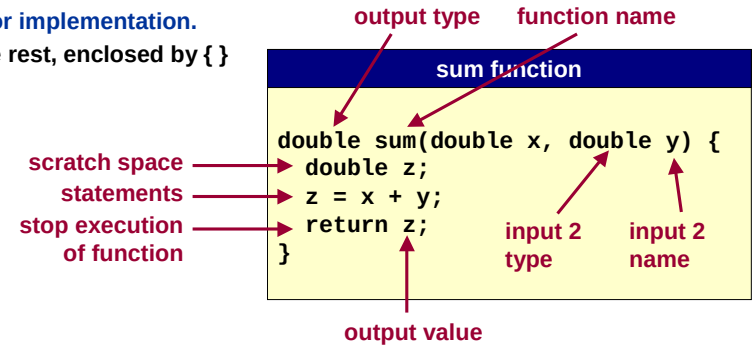
C function similar to mathematical function.

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

table3.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");
  x = 0.0;
  while (x < 2.0) {
    y = f(x);
    printf("%4.1f %6.3f\n", x, y);
    x = x + 0.1;
  }

  return 0;
}
```

function

declaration

assignment statement

flow control statement

function call statement

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).

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
```


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.

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
```

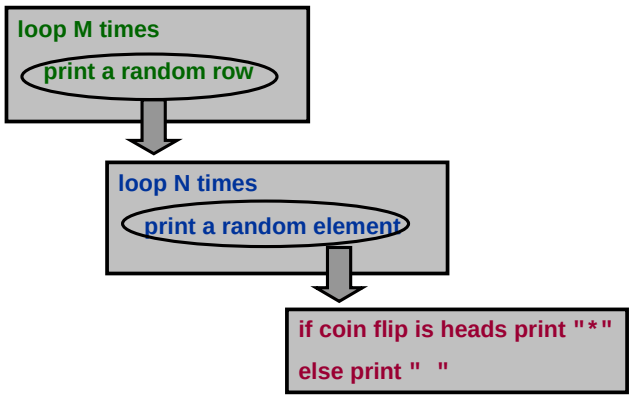

Random M x N Pattern

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

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


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.



randpattern.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.

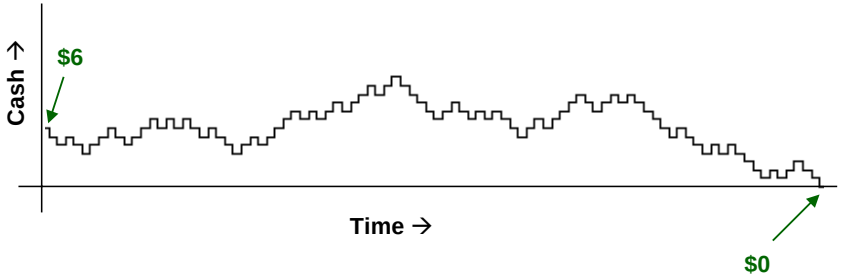
- Caveat 2: on many systems, our `randomInteger()` is very poor.

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

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");
```


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

single experiment (code as before)

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 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;
}
```


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.

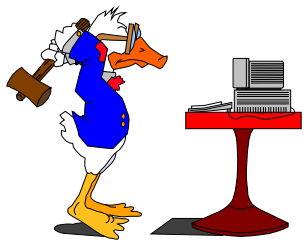
Syntax error - illegal C program.

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

Semantic error - wrong C program.

- Use "printf" method.

Always a logical explanation.



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

Poor Indentation

fact.c

```
#include <stdio.h>

#define l11l 0xFFFF
#define ll1 for
#define llll1 if
#define l1l1 unsigned
#define llll struct
#define lll11 short
#define llll1 long
#define l1ll putchar
#define l1l1l(l) l=malloc(sizeof(llll llll1));l->lll1l=1-1;l->ll1l1=1-1;
#define l1ll1 *lllll++=l1ll%10000;l1ll/=10000;
#define l1lll llll1(!l1->llll1){l1ll1(l1->lll1l);l1->lll1l->ll1l1=l1;}\
lllll=(l1=l1->lll1l)->lll;ll=1-1;
#define llll 1000

                                                  llll llll1 {
                                                  llll llll1 *
      lll1l,*ll1l1       ;l1l1                    llll1 lll [
      llll];};main      (){llll llll1             *ll11,*l1l,*
      l1, *lll1, *    malloc ( ) ; l1l1           llll1 l1ll ;
      ll1l1 ll1,ll  ,l;l1l1 llll1 *lll1,*         lllll; ll1(l
      =1-1 ;l< 14; l1ll1("\t\"8)>l\"9!.)>vl"      [l]^'L'),++l
      );scanf("%d",&l);l1l1l(l1l) l1l1l(llll      ) (l1=l1l)->
      lll[l1l->lll[1-1]     =1]=llll;lll(lll      =1+1;l1l<=l;
      ++ll1){l1=llll;         lll1 = (llll=(      llll=l1l))->
      lll; llll =(              l1l=l1)->lll;     ll=(l1ll=1-1
      );l1l(;l1ll->             lll1l||l11l!=     *lll1;){l1ll
      +=ll1**llll++             ;l1ll1 llll       (++ll>llll){
      l1lll llll=(              l1ll =llll->      llll1)->lll;
      }}ll1(;l1ll;              ){llll1 llll      (++ll>=llll)
      { llll} } *               lllll=ll1l;}
      ll1(l=(ll=1-              1);(l<llll)&&
      (l1->lll[ l]              !=ll1l);++l);     ll1 (;l1;l1=
      l1->ll1l1,l=              llll){llll(--l    ;l>=1-1;--l,
      ++ll)printf(              (ll)?((ll%19)     ?"%04d":(ll=
      19,"\n%04d")              ):"%4d",l1->      lll[l] ) ; }
                                                  llll1(10); }
```


Summary

Lots of material.

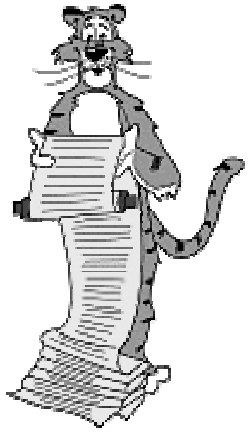
C is a structured programming language.

- Functions, loops.
- Simple but powerful tools.

Programming maturity comes with practice.

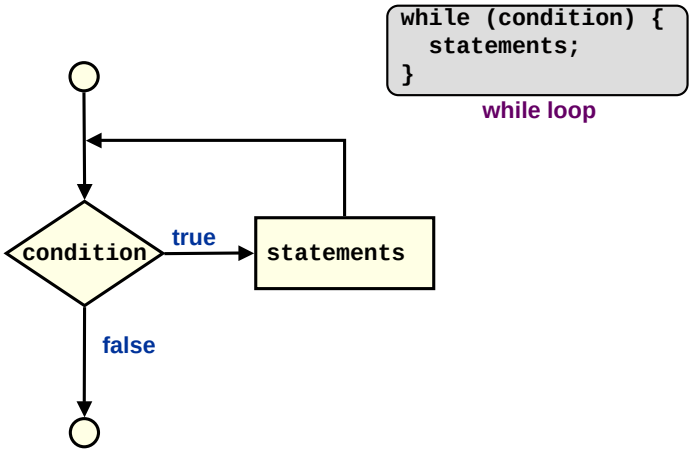
- Everything seems simpler in lecture and textbooks.
- Always more difficult when you do it yourself!
- Learn main ideas from lecture, learn to program by writing code.

Lecture P1: Supplemental Notes



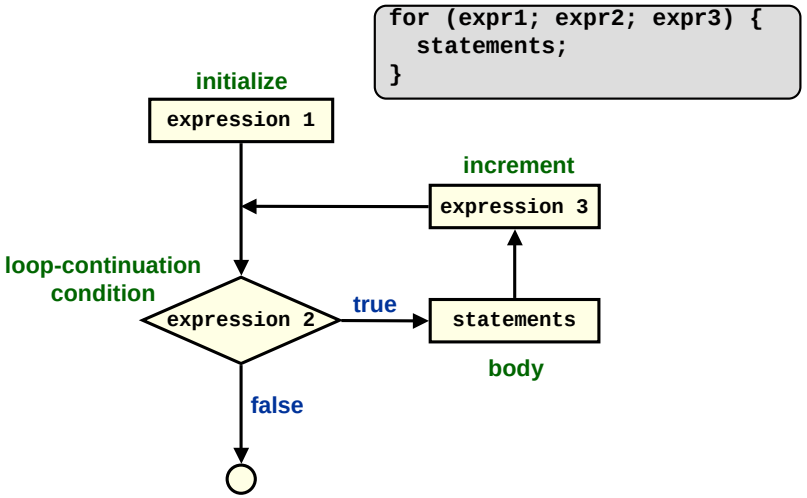
Anatomy of a While Loop

The while loop is a common repetition structure.



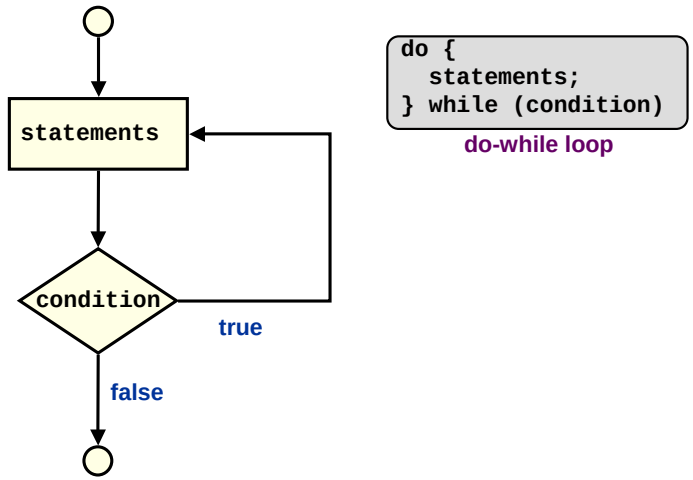
Anatomy of a For Loop

The for loop is another common repetition structure.



Anatomy of a Do-While Loop

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



For Loop Example

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

uses for loop

table4.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;
}
```

Final Attempt

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

table5.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;
}
```

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 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).

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.

int600.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;
}
```

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
```

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
```