# Lecture P1: Introduction to C

```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 \( f(x) = 2 - x^3 \).

<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

Print a table of values of function \( f(x) = 2 - x^3 \).
- Use while loop to perform repetitive tasks.

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

Language Syntax: Loops

Print a table of values of function \( f(x) = 2 - x^3 \).
- Use while loop to perform repetitive tasks.

```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;
}
```

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.

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

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!
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 {}
Library Functions: `printf()`

How is library function `printf()` implemented?
- User doesn't need to know details.  \( \text{see COS 217} \)
- User doesn't want to know details. \( \text{abstraction} \)

Library Functions: `rand()`

Print 10 "random" integers.
- Library function `rand()` in `stdlib.h` returns integer between 0 and `RAND_MAX` \( (32,767 = 2^{16} - 1 \text{ on arizona}) \).

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

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

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.

```
#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
101175
310
5622
341
16
386
```

Moral: check assumptions about library function.
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

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

int randomInteger(int n) { ... }

int main(void) {
    int cash, seed;
s...
Gambler's Ruin Numerical Experiment

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

1. Do for different values of starting cash values c.

```
#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;
}
```

gexperiment.c (cont)

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

Top-Down Design of Numerical Experiment

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

1. Do for different values of starting cash values 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 Numerical Experiment

<table>
<thead>
<tr>
<th>Unix</th>
</tr>
</thead>
<tbody>
<tr>
<td>% gcc gexperiment.c</td>
</tr>
<tr>
<td>% a.out</td>
</tr>
<tr>
<td># bets</td>
</tr>
<tr>
<td>2 2 6 304 2 2</td>
</tr>
<tr>
<td>3 33 17 15 53 29</td>
</tr>
<tr>
<td>4 22 1024 7820 22 54</td>
</tr>
<tr>
<td>5 243 25 41 7 249</td>
</tr>
<tr>
<td>6 494 14 124 152 14</td>
</tr>
<tr>
<td>7 299 33 531 49 93</td>
</tr>
<tr>
<td>8 218 10650 36 42048 248</td>
</tr>
<tr>
<td>9 174090315 83579 299 759 69</td>
</tr>
</tbody>
</table>

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 artists 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.

```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;
}
```

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.
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(...);