

## Outline

- 1 Course Goals
- 2 Introduction to Programming
- 3 Unix and C's History
- 4 Rudiments of C
  - Hello, world
    - Algorithm
    - Editing, compiling, running
    - The computer and Hello, world
- 5 Types and Variables
- 6 Input with `scanf`
- 7 Sample homework

# Beginning C Programming for Engineers

R. Lindsay Todd

Rensselaer Polytechnic Institute

Lecture 1: Introduction to C Programming

## Goals of this course

- Introduce general concepts of *programming*.
- Begin to *think like a programmer*.
- Learn to appreciate programming and *computer science*.
- Start programming in *C*.

## Evaluation Criteria for Beginning C

- Basic computer science
- Algorithm design
- C Programming
- Debugging
- Professional conduct

## What is programming?

Given a problem:

- 1 Make sure you understand the problem, i.e., it must be *well-defined!*
- 2 Find or develop an *algorithm* to solve a problem.
- 3 Express that algorithm in a way that the computer can execute it.

c

## Algorithms

In simple terms, an *algorithm* is a sequence of instructions to solve a problem, such that:

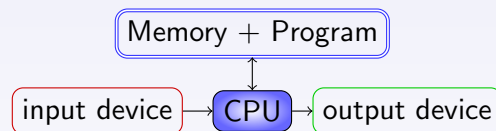
- Each instruction is unambiguous, and is something the computer can do.
- After an instruction is finished, there is no ambiguity about which instruction is to be executed next.
- Execution finishes in a finite number of steps.

Think of the computer as a meticulous moron.

c

## Von Neumann Machines

Von Neumann computer stores instructions in same memory as data.



*Input* is data passed into the computer.

The *CPU* processes a primitive machine language.

*Output* is data generated by executing the computer program.

c

## Compiled Programs

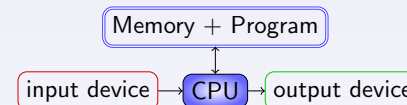


Figure: Adm. Grace Murray Hopper, pioneer of concept of “compilation.”



We can use programs to help us create programs.

- *Source code* is written by the programmer in a “high level language.”
- *Object code* is the *machine language* as translated by a *compiler*.

c

## Development cycle

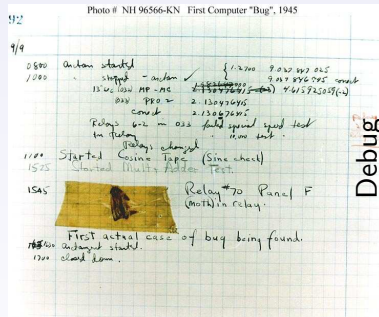
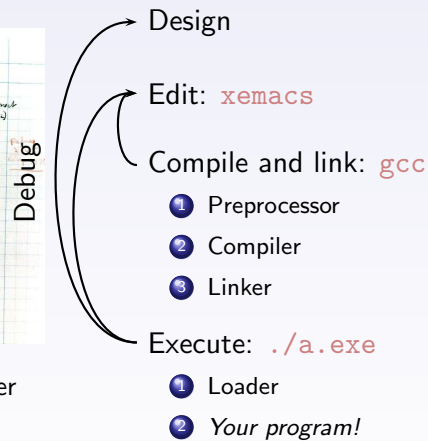


Figure: First identified computer bug.



C

## Unix and C



Figure: Ken Thompson and Dennis Ritchie

- Unix was written in C. C was created for Unix.
- Both Unix and C use a “standard I/O” concept that works well with “terminal emulator” windows, but not GUI-based operating system like Windows.
- The “Unix philosophy” is to build complex systems from simple, well-designed “little” tools. This is reflected in C.

C

## Cygwin/Unix Crash Course

From the Windows “Start” button, navigate to open a “Cygwin” shell. (RCS workstations have a “Unix window” that gives access to a command line “shell”.)

Command	Operation
<code>ls</code>	List files
<code>cp fromFile toFile</code>	Copy <i>fromFile</i> to <i>toFile</i>
<code>mv fromFile toFile</code>	Move <i>fromFile</i> to <i>toFile</i>
<code>rm deadFile</code>	Remove <i>deadFile</i>
<code>mkdir theDir</code>	Create <i>theDir</i>
<code>cd theDir</code>	Change directory to <i>theDir</i>
<code>more theFile</code>	Display <i>theFile</i>
<code>man aCommand</code>	Manual page for <i>aCommand</i>

C

## C Programs on the Computer

- C source code is stored in files on the computer. It must have the suffix “.c” (little “c”).
- Use **xemacs** or another text editor to create this file.
- The compiled program (object code) is named **a.exe**. (On Unix, it is named **a.out**.) Run it by typing:  
`./a.exe`

C

## Building Blocks in C

C gives us several “building blocks” for constructing programs:

- Variables and arithmetic expressions
- Conditional execution of statements
- Controlled repetition of statements
- Functions (subprograms)
- Printing, reading from keyboard
- Much more

C

## Our First Program

We want a program to print the message “Hello, world”.  
What is our algorithm?

The following steps are needed:

- 1 Print “Hello, world”.
- 2 Stop.

C

## Hello, world

Edit using: `xemacs hw.c &`

Program: **hw.c**

```
/* Hello, world program */
#include <stdio.h>

int
main()
{
    printf("hello, ");
    printf("world\n");
    return 0;
}
```

Results

hello, world

Compile using: `gcc -ansi -pedantic -Wall hw.c`

Execute as: `./a.exe`

C

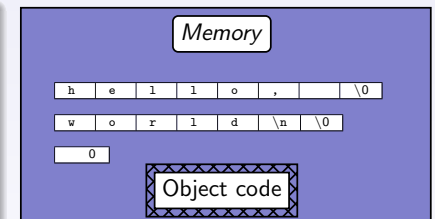
## Hello, world in memory

Character strings are stored in memory, along with the object code.

Program: **hw.c**

```
/* Hello, world program */
#include <stdio.h>

int
main()
{
    printf("hello, ");
    printf("world\n");
    return 0;
}
```



Results

hello, world

◀ Restart animation ▶ Skip animation

Constants, results of expressions, and *variables* are stored as “memory objects”.

C

## Introducing Types

- Memory is organized in units called *bytes*. Objects in memory use one or more bytes of memory.
- Each byte is (usually) eight *bits*. Values are represented as patterns of bits.
- How bit patterns are interpreted depends on the *type* of the data being stored.
- Common types in C are *integer*, *floating point*, *character*, and *character strings*.
- Constants and results of calculations are stored in memory objects.
- A *variable* is a named memory object.

c

## Type names

C deals with *integers*, *floating point*, *character strings*, and more complex objects.

int	integer
float	floating point
char	character
short or short int	short integer
long or long int	long integer
double	long floating point
long double	very long floating point

The `char` is treated as a very short integer. All the integer types may also be qualified as `unsigned`.

c

## Integer type

- The *integer* type, called `int`, can only represent integer values.
- An integer constant must not have a decimal point.
- Floating point values are truncated when assigned to an integer variable, or passed to an *int* argument of a function.
- An `unsigned int` value may not be negative.

```
int a = 4, b, c;           /* Declaration */
b = 9;                   /* Assignment */
c = a + b * 3;
```

c

## Float type

- The *floating point* type, called `float`, can represent real numbers. The *double precision* type, `double`, represents real numbers with more precision than `float`.
- A floating point constant must have a decimal point (otherwise it will be interpreted as an `int`).
- Floating point constants may be in scientific notation.
- Integer and other values are *promoted* to floating point when assigned to a *float* variable, or passed to a *float* argument of a function.

```
float a = 4., c;          /* Declaration */
c = a + 3.0e4;
```

c

## Choosing appropriate types

- The `int` (*integer*) type should be chosen to store quantities where only an integer or whole number is sensible.
- The `float` (*floating point*) type should be chosen to store quantities that might be fractional, such as physical dimensions.

c

## Formatted Output

### Program: `types.c`

```
#include <stdio.h>

int
main()
{
    printf("Ten: %d\n", 10);
    printf("Ten and 5 tenths: %f,\n %e, %g\n",
          10.5, 10.5, 10.5);
    printf("Ten: %s\n", "ten");
    printf("Oops! %d\n", 10.5);
    return 0;
}
```

### Results

```
Ten: 10
Ten and 5 tenths: 10.500000,
1.050000e+01, 10.5
Ten: ten
Oops! 0
```

<code>%d, %i</code>	integer
<code>%f, %e, %g</code>	float
<code>%s</code>	string
<code>%%</code>	%

c

## Review: Variables and Assignments

**Variable** Name for a memory object. Can be used in expressions. Variable names start with letters and may contain letters, digits, and underscores.

**Declaration** Tells compiler about variable and its type. Appears at the start of a block. Consists of type name followed by a comma-separated list of variable names.

**Assignment** Inserts a value into a memory object. Don't confuse assignments with equations.

c

## Sample: Variables

### Program: `vars.c`

```
/* Program using variables. */

#include <stdio.h>

int main()
{
    int a, b, c;

    a = 3;
    b = 4;
    c = a + b;
    printf("Sum: %d + %d -> %d\n",
          a, b, c);
    return 0;
}
```

### Results

```
Sum: 3 + 4 -> 7
```

c

## scanf

- Input can be done with `scanf`.
- Formats are like `printf`.
- Notice the *ampersand* (&) before each variable.

Program: `scanf.c`

```
#include <stdio.h>

int
main()
{
    int a1, a2, sum;
    printf("Enter a1, a2: ");
    scanf("%d, %d", &a1, &a2);

    sum = a1 + a2;
    printf("The sum is %d\n", sum);
    printf("I can lie: %d - %d = %d\n",
           a1, a2, sum);

    return 0;
}
```

C

## Formatted Input

Program: `scanf.c`

```
#include <stdio.h>

int
main()
{
    int a1, a2, sum;
    printf("Enter a1, a2: ");
    scanf("%d, %d", &a1, &a2);

    sum = a1 + a2;
    printf("The sum is %d\n", sum);
    printf("I can lie: %d - %d = %d\n",
           a1, a2, sum);

    return 0;
}
```

## Results

```
Enter a1, a2: 7, 3
The sum is 10
I can lie: 7 - 3 = 10
```

C

## Sample Homework

Program: `sample-homework.c`

```
/*
 * Homework #0, problem 0.
 * Programmers: April Showers, Mae Flowers, June Bugs
 *
 * This program prints out the circumference of a circle given the
 * radius.
 */

#include <stdio.h>

int
main()
{
    float r;    /* Radius */
    float C;    /* Circumference */

    /* Get the radius */
    printf("Enter the radius: ");
    scanf("%g", &r);

    /* Calculate and print the circumference. */
    C = 2. * 3.14159 * r;
    printf("Circumference = %g\n", C);
    return 0;
}
```

C

hw.c

```
/* Hello, world program */
#include <stdio.h>

int
main()
{
    printf("hello, ");
    printf("world\n");
    return 0;
}
```

types.c

```
#include <stdio.h>

int
main()
{
    printf("Ten: %d\n", 10);
    printf("Ten and 5 tenths: %f,\n %e, %g\n",
          10.5, 10.5, 10.5);
    printf("Ten: %s\n", "ten");
    printf("Oops! %d\n", 10.5);
    return 0;
}
```

**vars.c**

```
/* Program using variables. */
```

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

```
int main()
```

```
{
```

```
    int a, b, c;
```

```
    a = 3;
```

```
    b = 4;
```

```
    c = a + b;
```

```
    printf("Sum: %d + %d -> %d\n",
```

```
          a, b, c);
```

```
    return 0;
```

```
}
```

scanf.c

```
#include <stdio.h>

int
main()
{
    int a1, a2, sum;
    printf("Enter a1, a2: ");

    scanf("%d, %d", &a1, &a2);

    sum = a1 + a2;
    printf("The sum is %d\n", sum);
    printf("I can lie: %d - %d = %d\n",
          a1, a2, sum);

    return 0;
}
```

sample-homework.c

```
/*  
 * Homework #0, problem 0.  
 * Programmers: April Showers, Mae Flowers, June Bugs  
 *  
 * This program prints out the circumference of a circle given the  
 * radius.  
 */
```

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

```
int
```

```
main()
```

```
{
```

```
    float r;    /* Radius */
```

```
    float C;    /* Circumference */
```

```
    /* Get the radius */
```

```
    printf("Enter the radius: ");
```

```
    scanf("%g", &r);
```

```
    /* Calculate and print the circumference. */
```

```
    C = 2. * 3.14159 * r;
```

```
    printf("Circumference = %g\n", C);
```

```
    return 0;
```

```
}
```