CSE 1311

Introductory Programming for Engineers & Scientists

Darin Brezeale

The University of Texas at Arlington

Goals of Course

Goals of the course:

- Introduction to the programming language C
- Learn how to program
- Learn ‘good’ programming practices

C Language

The C language was created in the early 1970s. The version we will learn is C89 (sometimes referred to as C90), which is based on the 1989 ANSI standard.

Why learn a 20 year old version of the language? Because it is still the most common version.

Programming

What is computer programming?
Representation of a task or algorithm in a computer language.

What is an algorithm?
A set of directions for accomplishing a task.

Example of an Algorithm

The algorithm for calculating the arithmetic mean (i.e., the average) of a set of numbers is

1. add all the numbers together
2. divide this sum by the quantity of numbers

In mathematical terms this is written as

$$\text{average} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Levels of Abstraction

Suppose a student is asked to come to the front of the class

- high-level – stand, walk to front

Levels of Abstraction

Suppose a student is asked to come to the front of the class

- high-level – stand, walk to front
- mid-level – stand, turn 90° to the right, walk 15 feet, turn 90° to the left, walk 10 feet

Levels of Abstraction

Suppose a student is asked to come to the front of the class

- high-level – stand, walk to front
- mid-level – stand, turn 90° to the right, walk 15 feet, turn 90° to the left, walk 10 feet
- low-level – contract specific muscles in a specific order

Levels of Abstraction

Suppose a student is asked to come to the front of the class

- high-level – stand, walk to front
- mid-level – stand, turn 90° to the right, walk 15 feet, turn 90° to the left, walk 10 feet
- low-level – contract specific muscles in a specific order
- lowest-level – you think about walking, initiating many electrochemical reactions

Levels of Abstraction cont.

Computer languages that correspond to these levels of abstraction:

- high-level – perl, python, matlab
- mid-level – C
- low-level – assembly language
- lowest-level – machine code

Analyzing a Problem

Computers understand concrete steps, not abstract concepts.

To determine the concrete steps involved in solving a problem, we may

- Represent the problem using pseudocode
- Work through the process using a simpler or smaller version of the problem

Analyzing a Problem cont.

Example: You must sort 1000 numbers in ascending order.
Simpler version: Sort 3, 2, 4, 1 in ascending order

Pseudocode:

Basic Concepts

Much of programming consists of the following:

- Storing and updating information – Variables and Statements
  Example: store the answer given by a user
- Making decisions – Conditionals
  Example:

```
if temperature > 150 degrees
    print ‘‘It’s too hot!’’
```

- Repeating certain tasks – Loops
  Example: calculate the first 1000 prime numbers

Variables

C requires the programmer to specify the type of variable. Some examples are:

- `int` – used for integers, e.g., 1, 208, -19
- `double` – used for floating point numbers, e.g., 1.98, 10.0
- `char` – used for characters, e.g., ‘A’, ‘7’, ‘?’

Different variable types use different amounts of memory.

Variables cont.

C requires that variables be declared with the variable type before use:

```
int age;
age = 45;
```

or we could declare the variable and initialize it simultaneously

```
int age = 45;
```

Note how we end each statement with a semicolon.

Variables Names

Variable names

- must begin with a letter or an underscore
- can include letters, numbers, and underscores, e.g., `age, first_name, answer12`
- can't be the same as keywords, e.g., `int, for`
- are case sensitive, e.g., `name, Name`

Operators

The basic operators that you have in math are also available in C: +, -, *, /, =

There are many more operators available that we will introduce over time.

Note: The inside of the back cover of the textbook lists the operators available in C and the order of precedence

Operators cont.

One place where operators in C (and some other languages) differ from their math use is integer division.

WARNING:

$$\frac{\text{integer}}{\text{integer}} = \text{integer}$$

Example:

```
int topnum = 9, bottomnum = 4;
int answer;
answer = topnum / bottomnum;
```

`answer` has a value of 2, not 2.25.

Operators cont.

One operator that we will use on many occasions is the **modulus** operator: %. It returns the integer remainder from integer division.

Example:

```
int topnum = 9, bottomnum = 4;
int answer;
answer = topnum % bottomnum;
```

`answer` has a value of 1.

Basic Structure of Program

```
int main( void )
{
    /* your code goes here */
}
```

Example Program

```
#include <stdio.h>

int main( void )
{
    int age = 41, old;
    int weight;

    weight = 180;
    old = 2*age;

    printf("You weigh %d pounds.\n", weight);
    printf("You are %d years old.\n", age);
    printf("People twice your age are %d years old.\n", old);
}
```

The output of this is

```
You weigh 180 pounds.
You are 41 years old.
People twice your age are 82 years old.
```

Compilation Process

A C program must be compiled in order to make an executable. That is, the program written in the C language must be translated to something the computer understands before you can run it.

The entire compilation process consists of the following:

source code ⇒ preprocessor ⇒ compiler ⇒ assembler ⇒ linker ⇒ executable file

A ‘Good’ Program

There are different criteria by which one program may be considered better than another. Some examples are:

- Readability
- Maintainability
- Portability
- Scalability
- Performance (e.g., how fast it runs or how much memory it uses)

Programming Style

In this course, we care about readability, which is related to maintainability. You want other people (or you in the future) to be able to understand how the program works. This can be done through the use of

- Comments
- White space
- Indentation
- Meaningful variable names

Programming Style cont.

Comments are enclosed by the characters `/*` and `*/`.

Some examples are:

```
/* this is a comment */

int some_variable;
```

or

```
/*
    this is a comment
*/

int another_variable;
```

Programming Style cont.

Comments should assist the reader, not waste the time of the programmer.

Good comment

```
/*
    this function calculates pi using
    the fast Fourier transform
*/
```

Useless comment

```
int age; /* this variable is the age */
```

Programming Style cont.

Version 1: no indentation or use of whitespace

```
#include <stdio.h>
int main(void){int age=41;printf("You are %d years old.\n",age);}
```

Version 1 is not a problem for the compiler, but it is for the programmer.

Programming Style cont.

The C compiler ignores white space, so use it to improve readability.

Version 2: uses whitespace and indentation

```
#include <stdio.h>

int main( void )
{
    int age = 41;

    printf("You are %d years old.\n", age);
}
```

Programming Style cont.

Meaningful variable names aid in reading and debugging; they also eliminate the need for some comments.

Version 1: variable names are not meaningful

```
int a = 10;
int b = 5;
int c;

c = a + b;
```

Version 2: meaningful variable names

```
int labor = 10;
int materials = 5;
int total_cost;

total_cost = labor + materials;
```

Source of Bugs

When a program does not function as intended, or won't compile, the cause is referred to as a 'bug'. The act of tracking down and correcting bugs is debugging.

Bugs can be placed into two (broad) categories:

- code that is syntactically wrong
- logic errors

Types of Bugs

- syntax error

```
int int;
```

  This attempts to create a variable called `int`, which is a keyword.

- logic error

  you wrote

```
answer = 2 * 3;
```

  when you meant

```
answer = 2 + 3;
```

Success in this course

There are several things you can do to improve your chances of being successful in this course:

- run code that you know works
- modify code to see how it affects the results (including breaking the code)
- write small programs to test concepts