C Basics

2020 Spring: Introduction to C

April 22nd, 2020

Today

- **General Concepts**
- **C Program Structure**
- **C Strings**
  - `printf()`
  - Escape sequences
- **Comments**
- **C Data Types**
  - Declaration and assignment
  - Representation of numbers
- **C Operators**

Programming – General Concepts

- **Program**
  - A **set of instructions** to be carried out by a computer
- **Programming**
  - Creating an ordered set of instructions to solve a problem with a computer
- **Programming language**
  - A systematic set of rules used to describe computations in a format that is editable by humans
  - Ex) C, C++, Java, Python ...

Programming – General Concepts

- **Syntax**
  - Set of *legal structures and commands* that can be used in a language
  - **Every basic C statement ends with a semicolon ;**
  - If you violate this, you will get...

- **Syntax Error (Compile Error)**
  - A problem in the structure of a program that causes compilation failure
    - Missing semicolon
    - Mismatching { } braces
    - Illegal variable names
    - ...
  - When error occurs, read the error messages carefully!

Programming – General Concepts

1. *Write it*

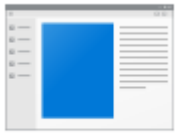
- **Code** or **source code**: the set of instructions in a program

hello.c

```c
#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}
```

2. *Compile it*

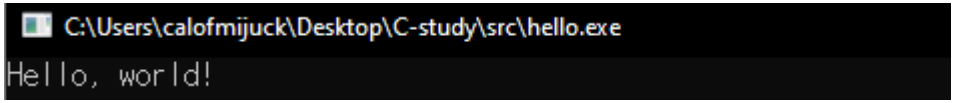
- **compile**: translate a program from one language to another



hello.exe

3. *Execute it*

- The messages printed to the user by a program
- **console**: Text box where the program's output is printed



C Program Structure

```
#include <stdio.h>   … ①

int main() { … ②
    printf("Hello, world!");   … ③
    return 0;
}
```

1. **header:** code to include, usually libraries
2. **method:** a named group of statements
3. **statement:** a command to be executed

- ***Statements inside main will be executed!***

Strings

- **String**: ***a sequence of characters***
  - Starts and ends with a " (quote) character
  - The quotes do not appear in the output
  - Examples:
    - `"hello"`
    - `"This is a string. It's very long!"`
  - May not span multiple lines
  - May not contain a " character

C Output

- **`printf(...)`**
  - Defined in `stdio.h` - `#include <stdio.h>` to use it!
  - Prints output on the *console*
  - More about this later!

- **How to use `printf(...)`**
  - `printf("message");`
    - Prints the given string `("message")` as output

```c
#include <stdio.h>

int main() {
    printf("message");
    return 0;
}
```

```
C:\Users\calofmijuck\Desktop\C-study\src\hello.exe
message
--------------------------------
Process exited after 0.01731 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

Escape Sequences

- **Escape Sequence**
  - A special sequence of characters used to represent special characters in a string
    - `\t` tab character
    - `\n` new line character
    - `\"` quotation mark character
    - `\\` backslash character

- **Example:**
  - `printf("\\hello\nhow\tare \"you\"?\\\\");`
  - Output:

```
\hello
how     are "you"?\\
```

Exercise

- **Print the following text using C**

```
Welcome to C study!
We are learning how to use printf()!
```

- **Print the following shape using C**

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

Exercise

- **Write a `printf` statement to produce this output**
  - All blanks are spaces

```
/ \ // \\ /// \\\
```

- **Use a <u>single</u> `printf` statement to produce this output**
  - All blanks are tabs

```
a	b	c
"\'"
Escape
sequences
```

C Comments

- **Comment**
  - A note written in source code by the programmer to describe or clarify the code
  - Comments are ignored when your program runs

- **Examples**
  - `// This is a one-line comment`
  -
```
/* This is a
   multi-line comment */
```

- **Comments are useful for:**
  - Explaining complex pieces of code or complex programs
  - Multiple programmers working together

Data Types

- **Type:** A category or set of data values
  - Used to represent real-world objects
  - Constrains the operations that can be performed on data
  - C programmers must specify types
  - Ex) Integers, real numbers, character, ...

- **Primitive Types:** Built-in types
  - **`int`** Integers (2, -26, 3000)
  - **`double`** Real numbers (3.1, -0.25, 0.001)
  - **`char`** Single characters ('a', 'b', 'c')
  - And more ...

Variables

- ***We want to use these data for computation***
  - *Can we **store** data ?*
  - *Can we perform **operations** on them ?*

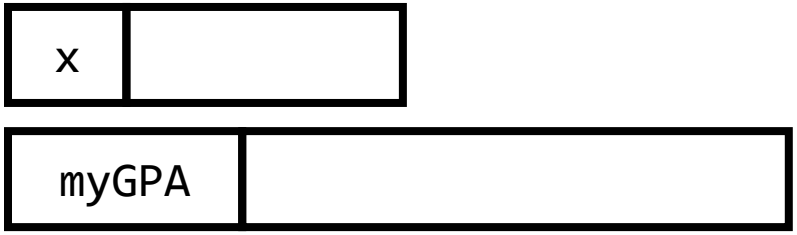
- **Variable:** A piece of computer memory that is given a **name** and **type**, and can **store a value**
  - Steps for using a variable
    - ***Declare*** it - State its name and type
    - ***Initialize*** it - Store a value into it
    - ***Use*** it - Print it or use in operation

Variable Declaration

- **Variable Declaration**
  - Sets aside memory for storing a value
  - **Variables must be declared before usage**

- **Syntax**
  - `type name;`
  - The name is called an ***identifier***

- **Examples**
  - `int x;`
  - `double myGPA;`

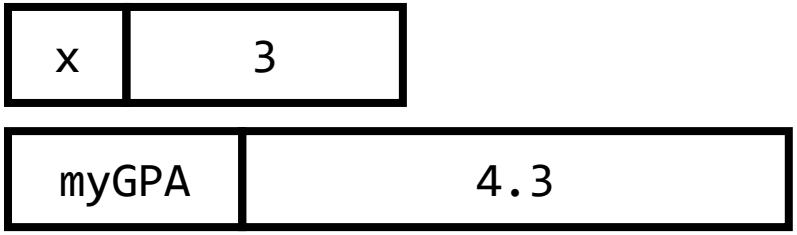


Variable Declaration

- **Assignment**
  - Stores a value into a variable
  - **= does not mean equals!**

- **Syntax**
  - `name = expression;`

- **Examples**
  - `x = 3;`
  - `myGPA = 3.1 + 1.2;`



Expressions

- **Expression:** A value or operation that computes a value
- **Examples**
  - `1 + 4 * 5` `// 21`
  - `(7 + 2) * 6 / 3` `// 18`
  - `42` `// 42`

  - The simplest expression is a *literal value*
  - A complex expression can use operators and parentheses

- **As a program runs, its expressions are *evaluated***
  - `1 + 1` evaluates to 2

Using Variables

- Once given a value, a variable can be used in expressions
- You can assign a value more than once

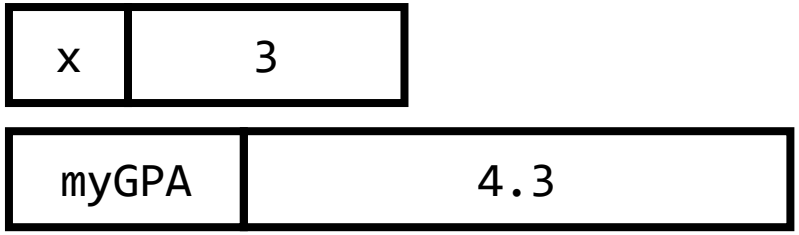
```c
#include <stdio.h>

int main() {
    int x;                  // declare x
    x = 15;                 // x is 15
    int y;                  // declare y
    y = 2 * x - 1;          // y is 29
    x = 4 + 7;              // x is now 11
}
```

Variable Declaration/Assignment

- **A variable can be declared and initialized in one statement.**

- **Syntax**
  - `type name = value;`

- **Examples**
  - `int x = 3;`
  - `double myGPA = 4.3;`



Assignment

- **= is called an assignment operator**
  - Does not mean equals!
  - Means: *"Store the value at right in variable at left"*

- **The right-side expression is evaluated first, and the result is stored in the variable at left**

- **Example**
  - `int x = 3;`

x	3

  - `x = x + 2;`

x	5

  - x + 2 is evaluated and stored in x

Common Mistakes

- **A variable *should* only store a value of its own type**
  - No errors will be shown, so be careful!
  - `int x = 2.5;  // Not good`
  - `double x = 2;  // OK. 2 is a real number`
- **A variable *shouldn't* be used until it is assigned a value**
  - No errors will be shown, so be careful!
  - `int x;  // ???`
  - `int y = x + 1;  // What is x? Then what is the value of y?`
- **You may not declare the same variable twice**
  - `int x;`
  - `int x;  // error: variable x is already defined`

Identifiers and Keywords

- **Identifier**: A name given to an item in your program
  - Must start with a letter or _ or $
  - Subsequent characters can be any of those or a number
  - Legal identifiers
    - `_myName, TheCure, ANSWER_IS_42, $bling$`
  - Illegal identifiers
    - `me+u, 49ers, side-swipe, Ph.D's`

- **Keyword**: An identifier that you cannot use because it already has a reserved meaning in C
  - **`int, double, float`** …

Format Strings and `printf`

- **How to print the value of variables?**
  - Values must be formatted before printing!
  - *Format strings* are used to format when
- `printf("format string", parameters);`
  - There can be many parameters

- **A format string can contain placeholders to insert parameters**
  - `%d` integer
  - `%lf` real numbers (double)
  - `%.Dlf` real numbers, with D digits precision
  - `%c` character
  - `%s` string

Format Strings and printf

- **Example**

```c
#include <stdio.h>

int main() {
    int x = 3;
    double y = 3.1415;
    printf("My name is %s.\n", "Guardian");
    printf("The value of %c is %d\n", 'x', x);
    printf("y is %lf.\n", y);
    printf("3 digit precision: %.3lf\n", y);
}
```

```
C:\Users\calofmijuck\Desktop\C-study\src\hello.exe
My name is Guardian.
The value of x is 3
y is 3.141500.
3 digit precision: 3.142
```

Representation of Numbers

- Digital devices have two stable states, 0 and 1
- The binary number system has two digits, 0 and 1
- **A single digit (0 or 1) is called a *bit*, short for binary digit**
- **1 byte = 8 bits**

- **Decimal Integers (Base 10)**
  - Uses ten digits (0 ~ 9)
  - Position values are powers of 10
  - n decimal digits can represent 10^n unique values

- **Binary Integers (Base 2)**
  - Uses two digits (0, 1)
  - Position values are powers of 2
  - n binary digits can represent 2^n unique values

Representation of Numbers

- **How to count in binary**

	128	64	32	16	8	4	2	1
117	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**

- **C `int` uses 4 bytes (32 bits)**
  - 1 bit is used for sign
  - Stores numbers from $-2^{31} \sim 2^{31} - 1$
  - Generally, n bit integer can store $-2^{n-1} \sim 2^{n-1} - 1$

Representation of Numbers

- **C double**

sign	*exponent*	*mantissa*
1 bit	*11 bits*	*52 bits*

  - Uses scientific notation
  - $(-1)^{sign} * mantissa \ * 2^{exponent}$

- **Mantissa has *finitely* many digits**
  - Causes *round-off errors*
  - Somewhat different from real numbers

C Operators

- **Operator:** Computation that combines multiple values or expressions
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Assignment Operators
  - Increment and Decrement Operators

Arithmetic Operators

- **Used for calculation involving numbers**

Operator	Meaning	Example
+	Addition	`3 + x`
-	Subtraction	`p - q`
*	Multiplication	`6 * i`
/	Division	`10 / 4`
%	Remainder (mod)	`11 % 8`

- **Can be applied to numerical types**
  - **`int, double,`** `(long long, float)`

Integer Arithmetic

- **Integer division returns the *quotient*!**
  - Ex. `14 / 4` is 3, not `3.5`
  - Division by 0 causes an error

- **% operator computes the remainder from integer division**
  - Ex. `14 % 4` is 2
  - Check if x is odd: `x % 2`
  - Obtain last digit of x : `x % 10`

- ***Subtle when handling negative integers***
  - `-4 / 3` is `-1`
  - `-5 % 3` is `-2`

Real Number Arithmetic

- **Examples:** `6.022, -42.0, 3.1415`
  - Placing `.0` or . after an integer makes it a **`double`**

- **/ produces an *exact answer***
  - `15.0 / 2.0` is `7.5`

- **When `int` and `double` are mixed, the result is a `double`**
  - `4.2 * 3` is `12.6`
  - `7.2 / 3` is `2.4`

Exercise

- **Calculate the answer of the following expression**
  - `123 + 456 * 789 / 3 % 2`

- **Follow the steps.**
  - Declare a variable x and assign 30
  - Declare a variable y and assign 15
  - Print `x + y, x - y, x * y, x / y, x % y`, respectively on each line

Relational Operators

- **Determine relations between values**

Operator	Meaning	Example
`==`	Equal to	`x == 10`
`!=`	Not equal to	`10 != 11`
`>`	Greater than	`3 > x`
`<`	Less than	`2 < 10.0`
`>=`	Greater than or equal to	`3.14 >= 3.1`
`<=`	Less than or equal to	`2.718 <= e`

- **Relational operators are used in *boolean expressions***
  - Boolean expressions will evaluate to **`true`** or **`false`**
  - Ex. 2 > 3 will evaluate to **`false`**

Logical Operators

- **Logical operators are applied to *boolean expressions* to form *compound boolean expression* that evaluate to `true` or `false`**

Operator	Meaning	Example
!	Logical NOT	`!x`
&&	Logical AND	`3 < x && x < 5`
||	Logical OR	`x > 5 || x < -2`

- **Truth Tables**

!	
T	F
F	T

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

Assignment Operators

- **Provides compact form**

Operator	Example	Meaning
`=`	`x = 2`	Simple assignment
`+=`	`x += 4`	`x = x + 4`
`-=`	`y -= 6`	`y = y - 6`
`*=`	`p *= 5`	`p = p * 5`
`/=`	`n /= 10`	`n = n / 10`
`%=`	`n %= 10`	`n = n % 10`

- **Chaining assignment is allowed, with evaluation from right to left**
  - `next = prev = sum = 0;`
  - Initializes `sum` to `0`, `prev` to `sum`, `next` to `prev`

Increment/Decrement Operators

Operator		Example
++	Pre-increment	++i
++	Post-increment	i++
--	Pre-decrement	--i
--	Post-decrement	i--

- **Increase or decrease the value in variable by 1**
  - Pre-in/decrement - Calculated ***on*** evaluation
  - Post-in/decrement - Calculated ***after*** evaluation

- **Example**

```c
#include <stdio.h>

int main() {
    int i = 5, j = 3;
    printf("%d\n", ++i);     // prints 6
    printf("%d\n", j++);     // prints 3, j in incremented to 4
    printf("%d\n", j);       // prints 4
}
```

Operator Precedence

- **Operator precedence**: Order of operator evaluation in expression
- Parentheses are always evaluated first
- **Associativity**: Order of evaluation on operators with same precedence

Precedence	Operator	Description	Associativity
1	++ -- ( ) [ ] . ->{} (*type*){*list*}	Suffix/postfix increment and decrement Function call Array subscripting Structure and union member access Structure and union member access through pointer Compound literal(C99)	Left-to-right
2	++ -- + - ! ~ (*type*) * & sizeof _Alignof	Prefix increment and decrement[note 1] Unary plus and minus Logical NOT and bitwise NOT Cast Indirection (dereference) Address-of Size-of[note 2] Alignment requirement(C11)	Right-to-left
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	? :	Ternary conditional[note 3]	Right-to-Left
14[note 4]	= += -= *= /= %= <<= >>= &= ^= |=	Simple assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Exercise

- **Guess the output/value without running the code!**
  - `5 + 3 < 6 - 1`
  - `"asdf" + 1 + 2`
  - `1 + 2 + "asdf"`
  - `!(3 >= 4) && (4 != 3)`

  - `int i = 5;`
  - `int x = i++;`
  - `x > i;`
  - `x += i;`

BOJ

- **Register on** https://acmicpc.net
- We will solve a lot of problems as homework!