

Basic C programming

Fundamentals of Computer and Programming

Spring 2019

Bahador Bakhshi

CE & IT Department, Amirkabir University of Technology



What We Will Learn

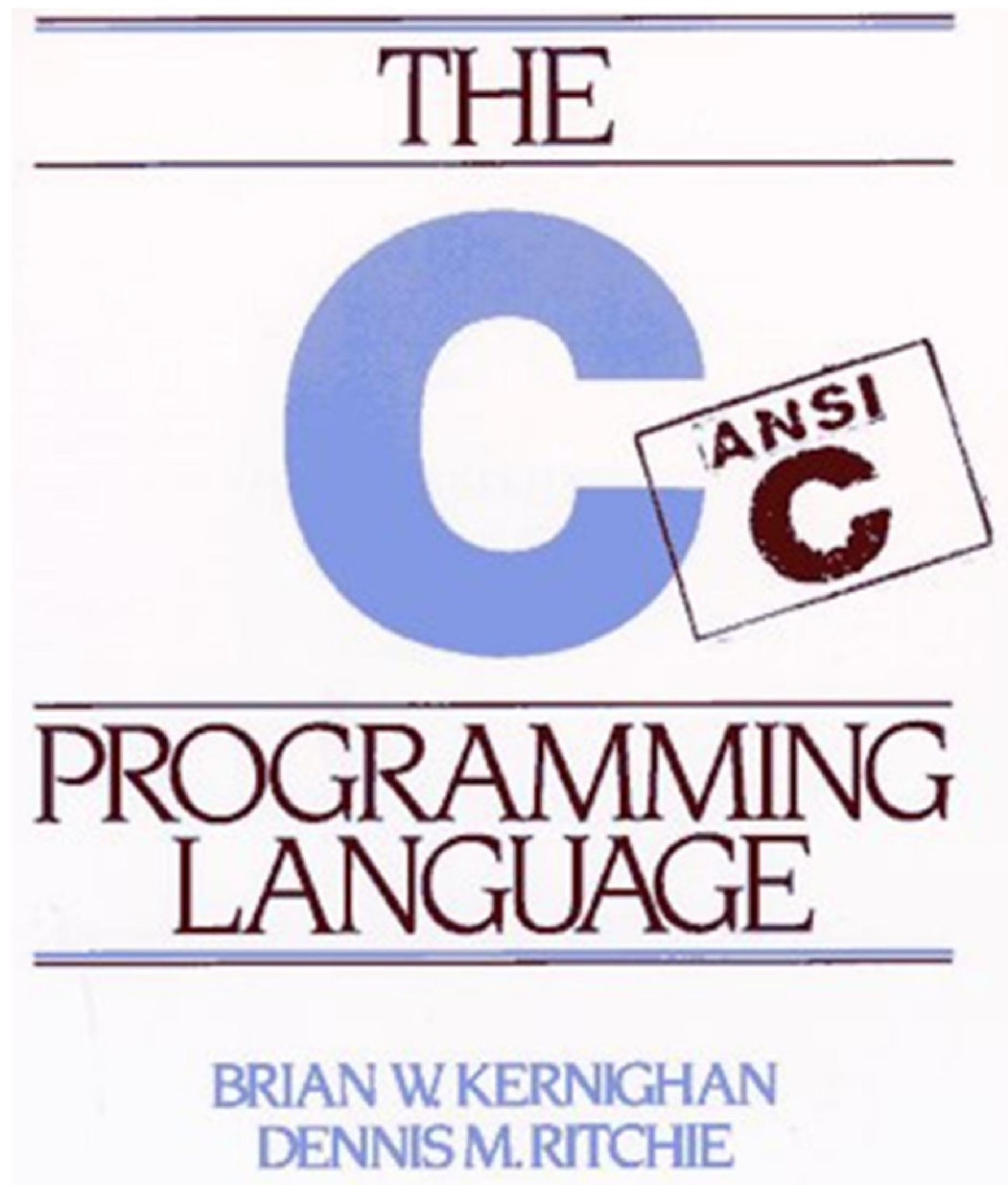
- What is the C
- Variables
 - Types
- Values
- Casting
- Constants & Definition



What We Will Learn

- What is the C
- Variables
 - Types
- Values
- Casting
- Constants & Definition





The C Language

- *C* is a *general-purpose* programming language
- *C* is developed by *Dennis Ritchie* at *Bell Laboratories*
- *C* is one of the widely used languages
 - Application development
 - System programs, most operating systems are developed in C: Unix, Linux
 - Many other languages are based on it



Programming in C Language

- **C** programming language
 - A set of notations for representing programs
- **C** standard libraries
 - A set of developed programs (functions)
- **C** programming environment
 - A set of tools to aid program development



The First Example

➤ Write a program that prints

“Hello the CE juniors :-)”



The First C Program

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

```
int main(void) {
```

```
    printf("Hello the CE juniors :-) \n");
```

```
    return 0;
```

```
}
```



General Rules

- C is case sensitive: **main** is not **MaIn**
- A “;” is required after each statement
- Each program should have a **main** function

```
int main(void) {...
```

```
void main(void) {...
```

```
main() {...
```

```
int main(int argc, char ** argv) {...
```

- Program starts running from the main
- You should follow coding style (**beautiful code**)



General Rules: Spaces

Equal Statements

<code>int main(void) {</code>	<code>int main (void) {</code>
<code>printf("abc"); return 0;</code>	<code>printf ("abc"); return 0;</code>
<code>return 0;</code>	<code>return 0;</code>



General Rules: Spaces

Not Equal Statements

<code>int main(void) {</code>	<code>intmain(void) {</code>
<code>printf("abc def");</code>	<code>printf("abcdef");</code>



Comments

```
/* Our first  
  
C program */  
  
#include <stdio.h>  
  
int main(void) {  
  
    //This program prints a simple message  
  
    printf("Hello the CE juniors :-) \n");  
  
    return 0;  
  
}
```



The First C Program

- You should
 - Develop the source code of program
 - Compile
 - Run
 - Debug
- All of them can be done in IDE
 - Code::Blocks
 - CLion
 - VS Code



What We Will Learn

- What is the C
- Variables
 - Types
- Values
- Casting
- Constants & Definition



Variables

- “write a program to calculate the sum of two numbers given by user”
- Solving problems
 - Input data → Algorithm → Output data
- What we need
 - Implementing the algorithm
 - Named **Functions**
 - We will discuss later
 - Storing the input/output data
 - **Variables**



Variables (cont'd)

- Data is stored in the main memory
- Variables
 - Are the **name** of locations in the main memory
 - We use names instead of physical addresses
 - Specify the **coding** of the location
 - What do the “01”s means?
 - What is the **type** of data?



Variables

➤ Variables in the C

<Qualifier> <Type> <Identifier>;

➤ <Qualifier>

- Is optional
- We will discuss later

➤ <Type>

- Specifies the coding

➤ <Identifier>

- Is the name



Types: Integers

➤ Integer numbers

➤ Different types, different sizes, different ranges

Type	Size	Unsigned	Signed
<code>short</code>	16Bits	$[0, 2^{16} - 1]$	$[- 2^{15}, 2^{15} - 1]$
<code>int</code>	32Bits	$[0, 2^{32} - 1]$	$[- 2^{31}, 2^{31} - 1]$
<code>long</code> يا	32/64	$[0, 2^{32 64} - 1]$	$[- 2^{31 63}, 2^{32 63} - 1]$
<code>long int</code>	Bits		
<code>long long</code> يا	64	$[0, 2^{64} - 1]$	$[- 2^{63}, 2^{63} - 1]$
<code>long long int</code>	Bits		



Types: Float & Double

➤ Floating point number

- float 32 bits
- double 64 bits
- long double 96 bits

➤ Limited precision

- float: 8 digits precision
 - $1.0 == 1.00000001$
- double: 16 digits precision
 - $1.0 == 1.000000000000000001$



Overflow & Underflow

- All types have limited number of bits
 - Limited range of number are supported
 - Limited precision
- Overflow
 - Assign a very big number to a variable that is larger than the limit of the variable
- Underflow
 - Assign a very small number to a variable that is smaller than the limit of the variable

Example



Types: Char

- Character
 - Type: `char`
- Single letters of the alphabet, punctuation symbols
- Should be single quotation
 - `'a', '^', 'z', '0', '1', '\n', '\', '\0'`



Types: Booleans

- `#include <stdbool.h>`
- Logics (Boolean): `bool`
- Only two values: `false` , `true`



Variables: Identifier

- The name of variables: **identifier**
- Identifier is string (**single word**) of
 - Alphabet
 - Numbers
 - “ _ ”
- But
 - Can**not** start with digits
 - Can**not** be the key-words (reserved words)
 - Can**not** be duplicated
 - Should **not** be library function names: printf



Variables: Identifier

- Use readable identifiers:
 - Do **not** use `memorystartaddress`
 - Use `memory_start_address`
 - Do **not** use `xyz`, `abc`, `z`, `x`, `t`
 - Use `counter`, `sum`, `average`, `result`, `parameter`, ...
 - Do **not** be lazy
 - Use meaningful names



C reserved words (cannot use for identifiers)

<code>_Bool</code>	<code>default</code>	<code>if</code>	<code>sizeof</code>	<code>while</code>
<code>_Complex</code>	<code>do</code>	<code>inline</code>	<code>static</code>	
<code>_Imaginary</code>	<code>double</code>	<code>int</code>	<code>struct</code>	
<code>auto</code>	<code>else</code>	<code>long</code>	<code>switch</code>	
<code>break</code>	<code>enum</code>	<code>register</code>	<code>typedef</code>	
<code>case</code>	<code>extern</code>	<code>restrict</code>	<code>union</code>	
<code>char</code>	<code>float</code>	<code>return</code>	<code>unsigned</code>	
<code>const</code>	<code>for</code>	<code>short</code>	<code>void</code>	
<code>continue</code>	<code>goto</code>	<code>signed</code>	<code>volatile</code>	



C++ reserved words (cannot use for identifiers)

asm	bool	catch	class
const_cast	delete	dynamic_cast	explicit
export	false	friend	inline
mutable	namespace	new	operator
private	protected	public	reinterpret_cast
static_cast	template	this	throw
true	try	typeid	typename
using	virtual	wchar_t	



Variable Identifiers

➤ Valid identifiers

student grade sum
all_students average_grade_1

➤ Invalid identifiers

if 32_test wrong* \$sds\$



Variables: Declaration (اعلان)

- Reserve memory for variable: **declaration**
 - <type> <identifier>;
- A variable must be declared **before** use

```
char test_char;
```

```
int sample_int;
```

```
long my_long;
```

```
double sum, average, total;
```

```
int id, counter, value;
```



Variable Type Effect (in complied Lang.)

- Important note: the type of variable is *NOT* stored in the main memory
 - After compiling the program → NO type is associated to memory locations!!!
- So, what does do the type?!
 - It determines the “*operations*” that work with the memory location

➤ E.g.

➤ **int** x, y, z;

z = x + y;

Integer + and =
Performed by ALU

➤ **float** a, b, c;

c = a + b;

Float + and =
Performed by FPU



Variables: Initial Values

- What is the initial value of a variable?
 - In C: we do **not** know.
 - In C: it is **not** 0.

We need to assign a value to each variable before use it.



What We Will Learn

- What is the C
- Variables
 - Types
- **Values**
- Casting
- Constants & Definition



Constants in C

➤ Values

➤ Numeric

- Integer numbers

- Float numbers

➤ Char

➤ Strings

➤ Symbolic constant

➤ Constant variables



Values

➤ Variables

- Save/restore data (value) to/from memory
- Declaration specifies the type and name (identifier) of variable
- Assigning value to the variable: **assignment**
 - `<identifier> = <value>;`
 - Compute the `<value>` and save result in memory location specified by `<identifier>`



Values: Examples

```
int i, j;
```

```
long l;
```

```
float f;
```

```
double d;
```

```
i = 10;
```

```
j = 20;
```

```
f = 20.0;
```

```
l = 218;
```

```
d = 19.9;
```



Value Types

- Where are the values stored?!

```
int x = 20;  
x = 30 + 40;
```

- In main memory
 - There is a logical section for these constant values
- So, we need to specify the type of the value
 - The coding of 01s of the value
- The type of value is determined from the value itself



Values: Integers

➤ Valid integer values

10; -20; +400; 0x12A; 011; 5000L

➤ Invalid integer values

10.0; -+20; -40 0; 600,000; 5000 L



Values: Float & Double

➤ Valid numbers:

0.2; .5; -.67; 20.0; 60e10; 7e-2

➤ Invalid numbers:

0. 2; 20. 0; 20 .0; 7 e; 6e; e12



Values: Chars

➤ Char values

➤ Should be enclosed in single quotation

➤ 'a', '^', 'z', '0', '1', '\n', '\'', '\0'

➤ Each character has a code: ASCII code

➤ 'A': 65; 'a': 97; '1': 49; '2': 50; '\0' : 0

➤ Character vs. Integer

➤ '1' != 1 ; '2' != 2

➤ '1' == 49 But 1 == 1



Values: Strings

- String is a set of characters
 - Starts and ends with "
- Examples

`"This is a simple string"`

`"This is a cryptic string #$56* (#"`



Effect of Value Types

- The type of values have the same effect of the type of variables
 - It determines the “*operations*” that work on the values

➤ E.g.

➤ `int z;`

➤ `float c;`

`z = 10 + 20;`

`c = 1.1 + 2.2;`

Integer + and =
Performed by ALU

Float + and =
Performed by FPU



Values: Initialization

```
int i = 20;
```

```
int j = 0x20FE, k = 90;
```

```
int i, j = 40;
```

```
char c1 = 'a', c2 = '0';
```

```
bool b1 = true;
```

```
float f1 = 50e4;
```

```
double d = 50e-8;
```



Values: From memory to memory

```
int i, j = 20;
```

```
i = j;           // i = 20
```

```
double d = 65536; // d = 65536.0
```

```
double b = d;     // b = 65536.0
```

```
d = b = i = j = 0;
```

```
// j = 0, i = 0, b = 0.0, d = 0.0
```



Basic Input Output

To read something: **scanf**

Integer: **scanf**("%d", &int_variable);

Float: **scanf**("%f", &float_variable);

Double: **scanf**("%lf", &double_variable);

To print something: **printf**

Integer: **printf**("%d", int_variable);

Float: **printf**("%f", float_variable);

Message: **printf**("message");



What We Will Learn

- What is the C
- Variables
 - Types
- Values
- **Casting**
- Constants & Definition



Casting

- What is the casting?
 - When the type of variable and value **are not the same**
 - Example: Assigning double value to integer variable
- It is **not** a syntax error in C (only warning)
 - But can cause **runtime errors**
- It is useful (in special situations)
 - But we should be very very careful



Implicit casting

➤ Implicit (ضمنی)

➤ We don't say it

➤ But we do it

```
char f2 = 50e6; /* cast from double to char */
```

```
int i = 98.01; /* cast from double to int */
```



Explicit casting

➤ Explicit (صریح)

➤ We say it

➤ And we do it

```
int i = (int) 98.1; /* cast from double to int */
```

```
char c = (char) 90; /* cast from int to char */
```



Casting effects

- Casting from small types to large types
 - There is not any problem
 - No loss of data

```
int i;  
short s;  
float f;  
double d;  
  
s = 'A';    // s = 65  
i = 'B';    // i = 66  
f = 4566;   // f = 4566.0  
d = 5666;   // d = 5666.0
```



Casting effects (cont'd)

- Casting from large types to small types
 - Data loss is possible
 - Depends on the values

```
float f = 65536;           // 65536.0
```

```
double d = 65536;          // 65536.0
```

```
short s = 720;              // 720
```

```
char c = (char) 65536;      // c = 0
```

```
short s = (short) 65536;    // s = 0
```

```
int i = 1.22;               // i = 1
```

```
int j = 1e23;               // j = ???
```



Casting effects (cont'd)

- Casting to Boolean
 - If value is zero → false
 - If values is not zero → true

```
bool b2 = 'a', b3 = -9, b4 = 4.5;    //true
```

```
bool b5 = 0, b6 = false; b7 = '\0'; //false
```



What We Will Learn

- What is the C
- Variables
 - Types
- Values
- Casting
- Constants & Definition



Constant Variables!!!

- Constants
 - Do not want to change the value
 - Example: $\pi = 3.14$
- We can only *initialize* a constant variable
 - We MUST initialize the constant variables (why?!)
- **const** is a qualifier

```
const int STUDENTS = 38;  
const long int MAX_GRADE = 20;  
int i;  
i = MAX_GRADE;  
STUDENT = 39; //ERROR
```



Definitions

- Another tool to define constants
 - Definition is not variable
 - We define definition, don't declare them
 - Pre-processor replaces them by their values before compiling

```
#define STUDENTS 38
```

```
int main(void) {
```

```
    int i;
```

```
    i = STUDENTS;
```

```
    STUDENTS = 90; //ERROR! What compiler sees: 38 = 90
```



Summary

- Simple programs in C
- Two basics
 - Variables
 - Types
 - Values
 - Types
- Casting
 - The type mismatch
- Constant variables & definitions



Reference

- **Reading Assignment:** Chapter 2 of “C How to Program”

