

CH-230-A

# **Programming in C and C++**

C/C++

## **Tutorial 1**

Dr. Kinga Lipskoch

Fall 2022

# Comments

- ▶ It is highly advisable to insert comments into your programs
- ▶ Comments start with the couple of characters  
`/* and end with the couple */` or  
`// this is also a comment`
- ▶ Everything between these two couples and after  
`// on the same line is considered to be a comment`
- ▶ Comments are ignored by the compiler

## Header Files/Libraries

- ▶ A header file is a file which contains the description of the resources provided by a library
- ▶ Technically it includes the prototypes of the provided functions
- ▶ Before using a library you must include the corresponding header file

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

- ▶ Issue

```
$> gcc -E -o hello.i hello.c  
look at the output file hello.i
```

# Functions

- ▶ A C program is a collection of functions
- ▶ A function is a piece of code which can be executed
- ▶ Every function has a name
- ▶ Functions may return back a value
- ▶ Calling a function means to execute the function
- ▶ Functions are a wide subject and will be in depth covered in a later lecture

## The main() Function

- ▶ Every C program must have a function called `main()`
- ▶ The main function is the logical starting point of the program
- ▶ Even if there are 200 functions before ...

```
1  int main() {  
2      ...  
3      statements...;  
4      ...  
5  }
```

## Escape Characters

- ▶ `printf` prints the characters in the string
- ▶ If a character is preceded by a `\` character, then it is called an escape character
- ▶ Escape characters are printed differently and are used to format the output
- ▶ Example: `\n` means new line
- ▶ Although you type in two characters, internally they is only one character

## Some Escape Characters

Character	Meaning
<code>\n</code>	Newline
<code>\a</code>	Bell
<code>\r</code>	Carriage Return
<code>\t</code>	Tabulator
<code>\b</code>	Backspace
<code>\\</code>	<code>\</code> (Backslash)
<code>\'</code>	<code>"</code> (Quote)
<code>%%</code> (in the <code>printf</code> control string)	<code>%</code> (Percent)

# The `return` Keyword

- ▶ When the `return` statement is executed, the current function terminates
- ▶ In the example the main function, and then the program, terminates
- ▶ `return` can provide a value to be returned
- ▶ This will be studied when learning functions in detail



# The Course of Semicolons

- ▶ Every statement must be terminated by a semicolon ;
- ▶ Statements:
  - ▶ Variable declarations
  - ▶ Function calls
  - ▶ Assignments
- ▶ Practice will help you . . .

## Indentation

- ▶ The layout of your program is not important for the compiler
- ▶ The program below is correct
- ▶ Semicolons are used to determine where a statement ends and where the next starts
- ▶ But not only machines will read your programs

```
1      #include <stdio.h>
2      int main(){ printf("Hello\n"); return 0; }
```

- ▶ Indent your programs to make them easier to read
- ▶ Choose one style and be coherent
- ▶ We will look at details of different styles later

# Data Types

- ▶ A program processes data
- ▶ Data can be combined by operators
- ▶ The type of a data defines
  - ▶ which values can be assumed
  - ▶ which operators can be applied
- ▶ C is a strongly typed language

# Characters

- ▶ `char` data type is used to store characters (ASCII code)
- ▶ A character is a symbol surrounded by a single quote like `'A'`, `'('` and so on
- ▶ C does not provide a string data type
- ▶ Strings are dealt as sequences of characters
- ▶ Details will follow in later lectures

# ASCII Table

32:	48: 0	64: @	80: P	96: `	112: p
33: !	49: 1	65: A	81: Q	97: a	113: q
34: "	50: 2	66: B	82: R	98: b	114: r
35: #	51: 3	67: C	83: S	99: c	115: s
36: \$	52: 4	68: D	84: T	100: d	116: t
37: %	53: 5	69: E	85: U	101: e	117: u
38: &	54: 6	70: F	86: V	102: f	118: v
39: '	55: 7	71: G	87: W	103: g	119: w
40: (	56: 8	72: H	88: X	104: h	120: x
41: )	57: 9	73: I	89: Y	105: i	121: y
42: *	58: :	74: J	90: Z	106: j	122: z
43: +	59: ;	75: K	91: [	107: k	123: {
44: ,	60: <	76: L	92: \	108: l	124:
45: -	61: =	77: M	93: ]	109: m	125: }
46: .	62: >	78: N	94: ^	110: n	126: ~
47: /	63: ?	79: O	95: _	111: o	127:

# Integers

- ▶ Positive or negative numbers without fractional part  
1, 2, 5, -999, 345302049
- ▶ Maximum and minimum values depend on the system
- ▶ The standard does not define this aspect

## Floating Point Numbers and Doubles

- ▶ Numbers with a fractional part  
2.3, 3.14, 0.293939
- ▶ `float` used to represent “real numbers”, they offer just a mere approximation
- ▶ `double` uses twice the number of bytes to represent numbers
  - ▶ Increased precision
  - ▶ Increased memory size
  - ▶ Increased time to process

## Variables (1)

- ▶ A variable is a named location in the computer's memory used to store a certain data type
- ▶ Variables content vary over time: they can be read or written
- ▶ Variables must be declared before use
- ▶ Every time you need to store some data in your program a variable is needed
- ▶ The type of a variable is fixed
- ▶ Variables are created and destroyed on the fly (more in the future)
- ▶ The content of a variable is retained as long as the variable is present



## Variables (2)

- ▶ Have a name
- ▶ Carry a type
- ▶ Hold a value
- ▶ Are located at a specific memory address

## Declaring Variables

- ▶ To declare a variable there is a fixed syntax
  - ▶ First data type and then variable name
  - ▶ Variable declarations are statements and must be terminated by a semicolon
- ▶ Consider the following:  
`int firstVariable;`
  - ▶ What is the value of `firstVariable`?
  - ▶ We do not know, and it cannot be known
  - ▶ Variable declaration just reserves enough space for the given type, and ties a name, but does not write anything to memory

# Initialization of Variables

- ▶ Using a non-initialized variable is a common error
- ▶ In C it is possible to declare and initialize a variable at the same time

```
1  int firstVariable = 23;  
2  float weight = 3.45;  
3  char first = 'A', second = 'B';
```

# Naming Variables

- ▶ Give variables meaningful names
  - ▶ Avoid too short or too long names
  - ▶ There are exceptions for loop variables (`i`, `j`, `k`, `m`)
- ▶ Rules:
  - ▶ First character must be a letter or the underscore
  - ▶ The remaining characters can be letters, numbers and underscores
  - ▶ No spaces are allowed
- ▶ A variable cannot have the same name as C keywords

## Naming Variables: Example

- ▶ These are valid identifiers

```
1      int firstVariable;  
2      float _startingWithUnderscore;  
3      char _99adsfq_743m_;
```

- ▶ These are not valid identifiers

```
1      int first Variable;  
2      float 945_temperature;  
3      char float;  
4      int some%data;
```

## Types Size: An Example

<b>Data type</b>	<b>Size (in bytes)</b>
<code>char</code>	1
<code>short int</code>	2
<code>int</code>	4
<code>long int</code>	4 or 8
<code>float</code>	4
<code>double</code>	8
<code>long double</code>	12 or 16

## Modifiers

- ▶ C provides some modifiers that apply to the basic data types
- ▶ The `long` modifier can be applied to the `int` and `double` data types
- ▶ The `signed` and `unsigned` modifiers can be applied to `int` and `char` data types
- ▶ The `short` modifier can be applied to the `int` data type
- ▶ Modifiers must be put before the data type
- ▶ If the data type is `int`, it may (but should not) be omitted

```
1    unsigned int modifiedVariable;  
2    long double somevar;  
3    unsigned unsignedVariable;
```

# Operators

- ▶ Operators perform mathematical or logical operations on data
- ▶ Can be roughly divided in arithmetic, relational and logical operators
- ▶ Apparently an easy subject, but there are many subtle details to know



## Arithmetic Operators

Operator	Integer	Floating point
+	$3 + 5 = 8$	$3.4 + 1.2 = 4.6$
-	$89 - 2 = 87$	$9.9 - 1.1 = 8.8$
*	$22 * 2 = 44$	$1.2 * 3.4 = 4.08$
/	$48 / 4 = 12$	$4.5 / 1.2 = 3.75$
% (modulo)	$49 \% 4 = 1$	n/a

# Assignment Operator

- ▶ The assignment operator = is used to write data to variables
- ▶ `lvalue = rvalue`
- ▶ `lvalue` is what is on the left
- ▶ `rvalue` is what is on the right
  - ▶ Could be a variable, a constant, or an expression

## Example

```
1  int main() {  
2      int first = 4, second = 5;  
3      int sum, difference;  
4      int product;  
5      product = first * second;  
6      sum = first + second;  
7      difference = first - second;  
8      return 0;  
9  }
```

- ▶ Note that this is a valid C program even if it does not print any of its results
- ▶ Modify the program, to print the results

## Shorthand Operators

- ▶ The following patterns are very common

```
1      int x = 32, y = 50;  
2      x = x + 10;  
3      y = y * 3;
```

- ▶ To shorten the notation, it is possible to use the following abbreviations

1	+=	x += 10;	x = x + 10;
2	-=	x -= 10;	x = x - 10;
3	*=	x *= 10;	x = x * 10;
4	/=	x /= 10;	x = x / 10;
5	%=	x %= 10;	x = x % 10;

## The Control String of printf

- ▶ The general syntax is the following  
`printf("control string", arg1, arg2, ...);`
- ▶ The control string specifies:
  - ▶ Which characters have to be printed,
  - ▶ How variables have to be formatted,
  - ▶ Number of decimal places, their type, etc.
- ▶ Note that `printf` accepts a variable number of arguments

## Specification of the base

- ▶ When printing integers it is possible to specify which base should be used for their representation

Specification	System
%o	Octal
%d	Decimal
%x	Hexadecimal

## Formatting Integer Numbers

- ▶ It is possible to specify how many digits should be used while printing an integer

```
1      int a = 145;  
2      printf("The value is %6d\n", a);
```

- ▶ This will print three spaces and then 145 (i.e., 6 places for a three digits number)
- ▶ If the number of digits is too small, it will be ignored

```
1      int a = 145;  
2      printf("The value is %2d\n", a);
```

- ▶ This will print 145 over 3 places

## The Precision Modifier

- ▶ The precision modifier is written `.number`
- ▶ For floating point numbers it controls the number of digits printed after the decimal point  
`printf("%.3f", 1.2);`  
will print 1.200
- ▶ If the number provided has more precision than is given, it will rounded  
`printf("%.3f", 1.2348);`  
will display as 1.235