Basic Syntax

Basic Syntax , I/O, Conditions, Loops and Debugging

SoftUni Team

Technical Trainers

Software University

https://softuni.bg/

Table of Contents

Have a Question?

sli.do

#cpp-fundamentals

What is C++?

What is C++

- General purpose programming language
- Compiles to binary – i.e. multi-platform
- Statically typed – data types, classes, etc.
- Multi-paradigm
- Fast

Philosophy, Compilers and IDEs

Program Structure

Entry Point, Building and Running

Hello World

- A classic C++ "Hello World" example

```
#include <iostream>
using namespace std;

int main(int argc, char * argv[]) {
   cout << "Hello World!" << endl;
   return 0;
}
```

Include the input-output library

Say we're working with the std namespace

These are optional

Print to the console

"main" function – our entry point

For main, 0 means everything went ok

C++ Entry Point & Termination

- The `main` function – entry point of the program
  - No other function can be named "`main`"
  - C++ needs specific function to start from
  - Everything else is free-form – code ordering, namings, etc.
  - Can receive command line parameters
- Termination – `main` finishes (returns), the program stops
  - The return value of main is the "exit code"
  - `0` means no errors – informative, not obligatory

Program Structure: Including Libraries

- C++ has a lot of functionality in its standard code libraries
- C++ can also use functionality from user-built code libraries
- Say what libraries to use with the **#include** syntax
- For now, for standard libraries: put the library name in **<>**

```
#include <iostream>
using namespace std;

int main(int argc, char * argv[])
```

iostream contains console I/O functionality

Program Structure: Blocks

- Basic building block (pun intended) of a program
- Most actual program code is in blocks (bodies)
- Start with { and end with }, can be nested
- Functions' (`main()`), loops' & conditionals' code is in blocks

`main()` code block

```
int main(int argc, char * argv[])
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Program Structure: Statements & Comments

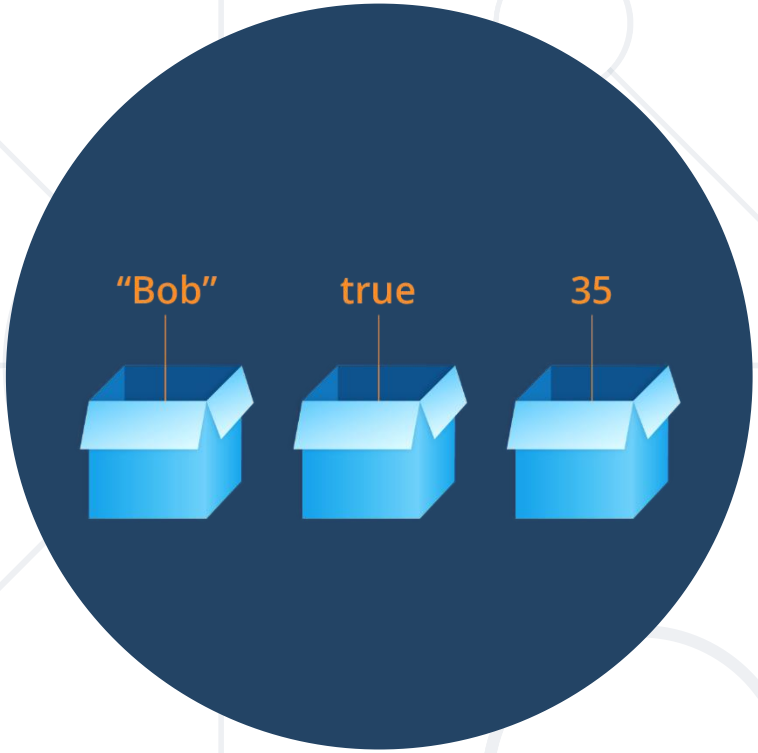
- Statement: a piece of code to be executed
  - Blocks consist of statements
- Statements contain C++ code and end with a **;**

```
int main(int argc, char * argv[])
{
     cout << "Hello World!" << endl;
     return 0;
 }
```

- C++ has comments (parts of the code ignored by compiler)
  - **//** comments a line, **/*** starts a multi-line comment, ***/** ends it

C++ Hello World

Live Demo



Data Types and Variables

Types, Declaration, Initialization, Scope

Declaring and Initializing Variables

- `<data_type> <identifier> [= <initialization>];`
- Declaring: `int num;`
- Initializing: `num = 5;`
- Combined: `int num = 5`,
  and additionally `int num(5);` or `int num{5};` (C++11)
- Can declare multiple of same type by separating with comma `,`
  - `int trappist1BMassPct=85, trappist1CMassPct=80;`

Declaring & Initializing Variables

LIVE DEMO

Uninitialized Locals

LIVE DEMO

Local & Global Variables

- Global: defined outside blocks, usable from all code
- Local: defined inside blocks, usable only from code in their block
- Locals DO NOT get initialized automatically

Local & Global Variables

- Globals get initialized to their "default" value (0 for numerics)

```
int secondsInMinute = 60;
int minutesInHour = 60;
int hoursInDay = 24;
int secondsInHour = secondsInMinute * minutesInHour;

int main() {
    int days = 3;
    int totalSeconds = days * hoursInDay
*secondsInHour;
```

Global & Local Variables

LIVE DEMO

const Variables

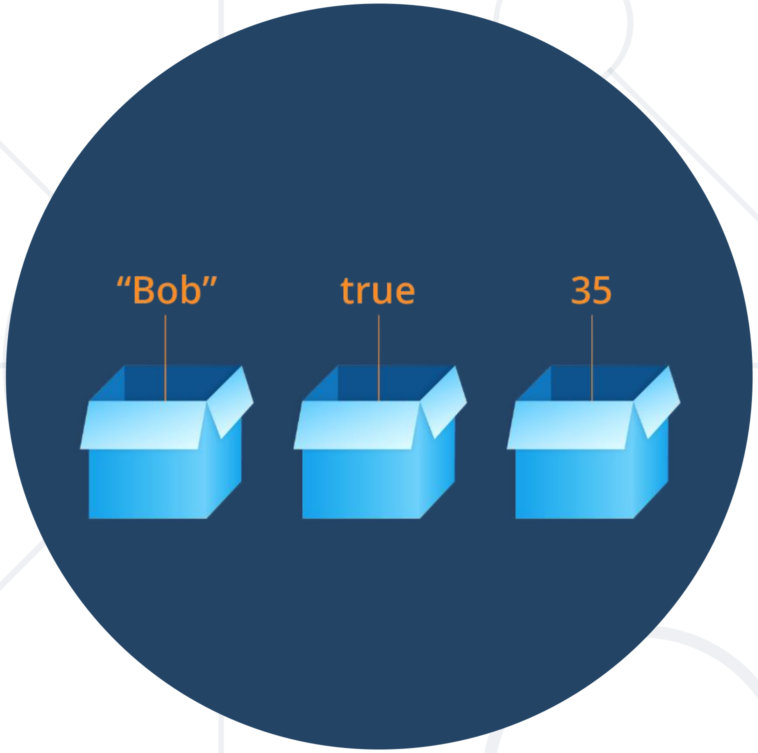
- C++ supports constants – "variables" that can't change value
- Can and MUST receive a value at initialization, nowhere else
- Can be local, can be global
- **secondsInMinute**, **minutesInHour**, etc., are not things that normally change in the real world – the following will not compile:

```
const int secondsInMinute = 60;
int main() {
    secondsInMinute = 13; //compilation error
```

const Variables
LIVE DEMO

Other variable modifiers

- `static` variables initialize once and exist throughout program
  - Can be used to make a local variable that acts like a global one
  - Can be used on a global variable, but has no real effect
- `extern` tells the compiler a variable exists somewhere in a multi-file project (to avoid multi-declaration)



Primitive Data Types

Integer Types – `int`

- C++ has "only one" integer type – **int**
- "Width" modifiers control the type's size and sign
  - **short** – at least 16 bits; **long** – at least 32 bits
  - **long long** – 64 bits
- **signed** & **unsigned** – use or not use memory for sign data
- Modifiers can be written in any order
- **int** can be omitted if any modifier is present
- Defaults: **int** "usually" means **signed long int**

Integer Types

LIVE DEMO

Floating-Point Types

- Represent real numbers (approximations)
  - 2.3, 0.7, -Infinity, -1452342.2313, NaN, etc.
- **float**: single-precision floating point, usually IEEE-754 32-bit
- **double**: double-precision, usually IEEE-754 64-bit

Name	Description	Size*	Range*
float	Floating point number.	4bytes	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ (~7 digits)
double	Double precision floating point number.	8bytes	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ (~15 digits)
long double	Long double precision floating point number.	8bytes	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ (~15 digits)

Using Floating-Point Types

LIVE DEMO

Character Types – char

- char is the basic character type in C++
- Basically an integer interpreted as a symbol from ASCII
- Guaranteed to be 1 byte – a range of 256 values
- Initialized by either a character literal or a number (ASCII code)

```cpp
int main() {
    char letter = 'a';
    char sameLetter = 97;
    char sameLetterAgain = 'b' - 1;
    cout << letter << sameLetter << sameLetterAgain << endl;
    return 0;
}
```

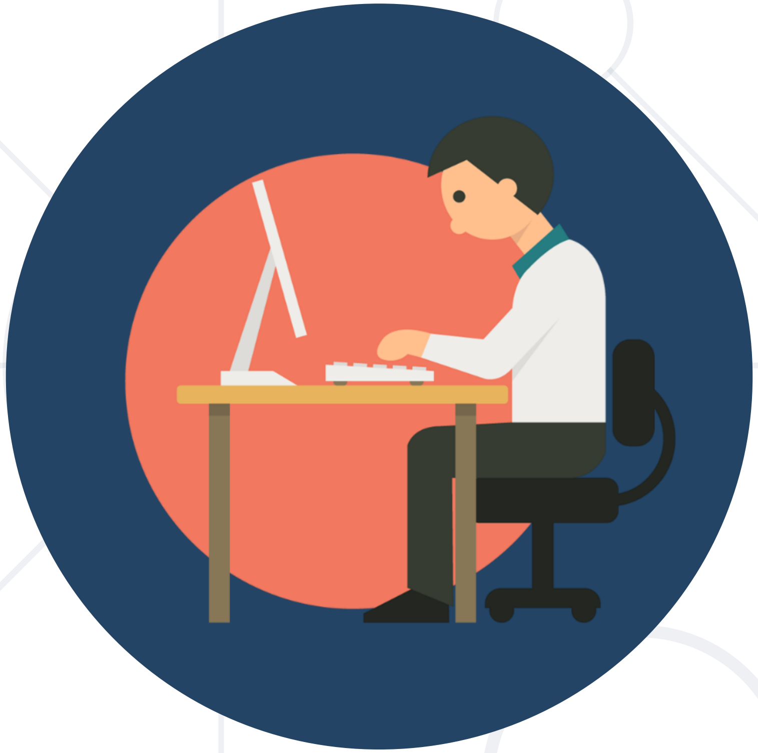
Using Character Types

LIVE DEMO

Boolean Type – bool

- **bool** – a value which is either **true** or **false**, takes up 1 byte
- Takes **true**, **false**, or numeric values
  - Any non-zero numeric value is interpreted as **true**
  - Zero is interpreted as **false**

```
int main() {
    bool initializedWithKeyword = true;
    bool initializedWithKeywordCtor(false);
    bool initializedWithZero = 0;
    bool initializedWithNegativeNumber(-13);
```



Using Boolean Types

LIVE DEMO

Implicit & Explicit Casting (1)

- Types which "fit" into others can be assigned to them implicitly
- For integer types, "fit" usually means requiring less bytes
  - Valid: **char a = 'a'; int i = a;**
  - NOT VALID: **int i = 97; char a = i;**
  - For floating point, **float** fits into **double**

Implicit & Explicit Casting (2)

- If you really want to store a **bigger** type in a **smaller** type
  - Explicitly cast the **bigger** type to the **smaller** type:

    smallType smallVar = (smallType) bigVar;
- Can lose accuracy if value can't be represented in a **smaller** type

Operators, Expressions, Conditionals, Loops

C++ Numeric Literals

- Represent values in code, match the primitive data types
- Integer literals – value in a numeral system

```
unsigned long long num;
num = 5; num = -5; num = 5L; num = 5ULL; num = 0xF;
```

- Floating-point literals – decimal **or** exponential notation
  - Suffix to describe precision (single or double-precision)

```
double num;
num = .42; num = 0.42; num = 42e-2;
float floatNum;
floatNum = .42f; floatNum = 0.42f; floatNum = 42e-2f;
```

Non-Numeric Literals

- Character literals – letters surrounded by apostrophe (')

```
char letter = 'a';
```

- String literals – a sequence of letters surrounded by quotes (")

```
cout << "Hello World!" << endl;
```

- Boolean literals – `true` and `false`

```
bool cppIsCool = true;
```

C++ Literals

LIVE DEMO

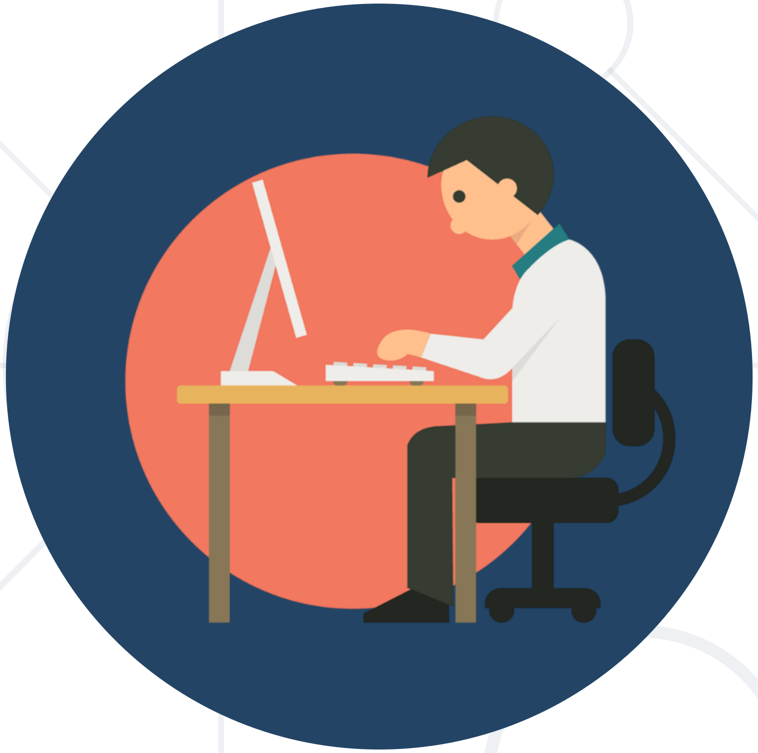
Expressions and Operators

Expressions and Operators

- Operators perform actions on one or more variables/literals
  - Can be customized for different behavior based on data type
- C++ operator precedence and associativity table: http://en.cppreference.com/w/cpp/language/operator_precedence
  - Don’t memorize. Use brackets or check precedence when needed
- Expressions: literals/variables combined with operators/functions

Commonly Used C++ Operators

Category	Operators											
Arithmetic	+	-	*	/	%	++	--					
Logical	&&	||	^	!								
Binary	&	|	^	~	<<	>>						
Comparison	==	!=	<	>	<=	>=						
Assignment	=	+=	-=	*=	/=	%=	&=	|=	^=	<<=	>>=	
String concatenation	+											
Other	.	[]	()	a?b:c	new	delete	*	->	::	(*type*)	<<	>>



Expressions and Operators

LIVE DEMO

Conditionals (1)

- The **if-else** statement takes in a Boolean expression
  - If the expression evaluates to **true**, the **if** block is executed
  - If the expression evaluates to **false**, the **else** block is executed
  - The **else** block is optional

Conditionals (2)

- Block { } brackets can be omitted if only 1 statement

```
double value1 = 5 * 5 / 2.f, value2 = 5 * 5 / 2;
if (value1 > value2) {
    cout << "value1 is larger" << endl;
}
else {
    cout << "value2 is larger" << endl;
}
```

"Chaining" if-else

- Can **chain** several checks one after the other

```
if (value1 > value2) {
    cout << "value1 is larger";
}
else if (value1 == value2) {
    cout << "values are equal";
}
else {
    cout << "value2 is larger";
}
```

- The code below is equivalent. Each else block contains 1 **if** statement, so they do not need brackets. The left variant skips the brackets

```
if (value1 > value2) {
    cout << "value1 is larger";
}
else {
    if (value1 == value2) {
        cout << "values are equal";
    }
else {
        cout << "value2 is larger";
    }
}
```

`if` and `if-else`

LIVE DEMO

The Switch-Case Statement

Simplified If-else-if-else

switch-case

- Example of C++ switch-case usage

```
switch (day)
{
    case 1: cout << "Monday"; break;
    case 2: cout << "Tuesday"; break;
    case 3: cout << "Wednesday"; break;
    case 4: cout << "Thursday"; break;
    case 5: cout << "Friday"; break;
    case 6: cout << "Saturday"; break;
    case 7: cout << "Sunday"; break;
    default: cout << "Error!"; break;
}
```

`switch-case` structure (1)

- The C++ switch statement takes in
  - An integer expression or an enumeration type
  - Or something which converts to an int (like char)
- The case block can contain case labels and any other code

`switch-case` structure (2)

- Each label has an expression of the same type as the **`switch`**
- The **`case`** block can also contain the **`break`** statement
  - If reached, code continues from after the **`case`** block
- There is a special **`default`** label (without an expression)

`switch-case` execution

- **`switch`** evaluates the expression and finds the matching **`case`**
- Any code before the matching **`case`** is skipped
- Any code after the matching **`case`** is executed
  - Until **`break`** or the end of the block is reached
- If there is no matching **`case`**
  - If the block contains the special **`default`** label, it is executed
  - Otherwise the case block is skipped

switch-case

LIVE DEMO

Loops

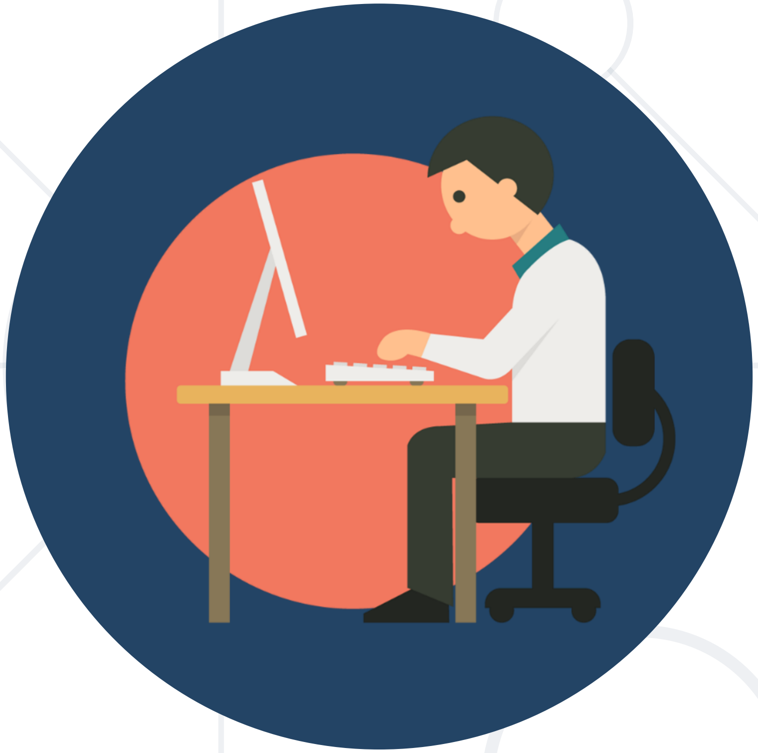
Code Block Repetition

`for` Loop (1)

- `for([init]; [condition]; [increment]) {…}`
- The **`init`** statement can declare and initialize variables
  - Declared variables are usable only IN the **`for`**’s body
- The loop runs while the **`condition`** statement is **`true`**

`for` Loop (2)

- **`increment`** is executed AFTER the **`for`**'s body
  - Can execute any expression
- Expressions inside **`init`** and **`increment`** are separated by comma (**,**)



for Loop

LIVE DEMO

while and do-while Loops (1)

- **while (condition) { body code; }**
  - Executes until **condition** becomes **false**, may never execute

```
int age = 0;
while (age < 18) {
    cout << "can't drink at age " << age << endl;
    age++;
}
cout << "age " << age << ", can finally drink!" << endl;
```

`while` and `do-while` Loops (2)

- `do { body code; } while (condition);`
  - First executes body, then checks condition
  - Guaranteed to execute at least once

while and do-while Loops

LIVE DEMO

Loops

- C++ loop control keywords:
  - `break` – interrupts the loop and continues after its block
  - `continue` – the current iteration skips the remaining part of the loop block
- Range-based for loop

Basic Console I/O

Writing to and Reading from the Console

C++ Streams (1)

- Classes that either read or write data piece by piece
- `cout`
  - Writes data to the console (standard output)
  - `cout` has a counterpart – `cin`

C++ Streams (2)

- Reads data from the console (standard input)
- **cin** uses the **>>** operator
- **cout** uses the **<<** operator to write

```cpp
#include<iostream>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
    return 0;
}
```

Input and Output

LIVE DEMO

Summary

- Structure, Specifics, Compilers & IDEs
- Data Types and Variables
- Declaration and Initialization
- Operators and Expressions
- Conditional Statements
  - `if`, `if-else`, `switch-case`
- Loops
  - `for`, `while`
- Input and Output

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

Software University

DraftKings

INDEAVR
Serving the high achievers

Coca-Cola HBC Bulgaria

SCHWARZ

LIDL

Kaufland

Postbank
Решения за твоето утре

POKERSTARS | CAREERS

SCHWARZ Produktion

prezero

SUPER HOSTING .BG

SOFTWARE GROUP

taulia®

MOTION SOFTWARE

PHAR VISION

SmartIT

createX

Educational Partners

Software University

CODE IT UP

VIRTUAL RACING SCHOOL

VRS

License

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg