Basic Syntax

Basic Syntax , Conditions and Loops

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg/

Have a Question?

Software University

sli.do

#cpp-fundamentals

Table of Contents

What is C++?

What is C++?

- General purpose **programming language**
- Designed in **1979** to be an **extension of the C language**
- Compiles to **binary**
- Statically **typed**
- **Multi-paradigm**
- Close to **low-level language**
  - **Fast**
  - Used in **embedded systems**

Program Structure

Example: 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

"main" function: our entry point

Print new line to the console

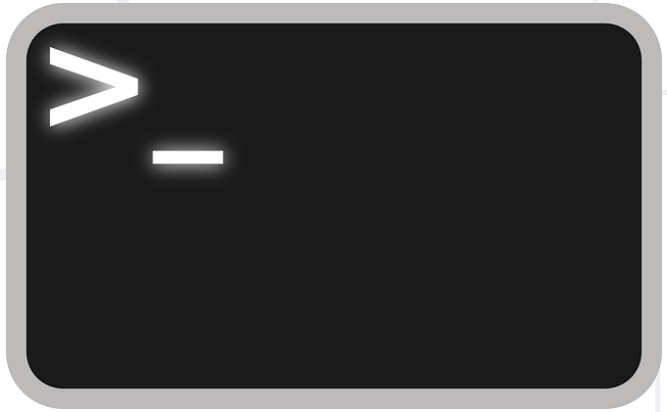
0 means everything went ok

Entry Point

- 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**
  - Can receive **command line** parameters
  - Cannot be used **anywhere** in the program

Termination

- Termination – `main` finishes (returns) and the program **stops**
  - The return value of main is the **"exit code"**
  - `0` means no errors



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
- Organizing C++ code into **.h** and **.cpp files**
- Say what libraries to use with the **#include** syntax
- For standard libraries: put the library name in **<>**

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

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

iostream contains console Input / Output functionality

Program Structure: Blocks

- Basic **building block** of a program
- Most actual program code is in **blocks**
- Start with **{** and end with **}**
- Loops' and conditionals' code is in blocks

main() code block

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

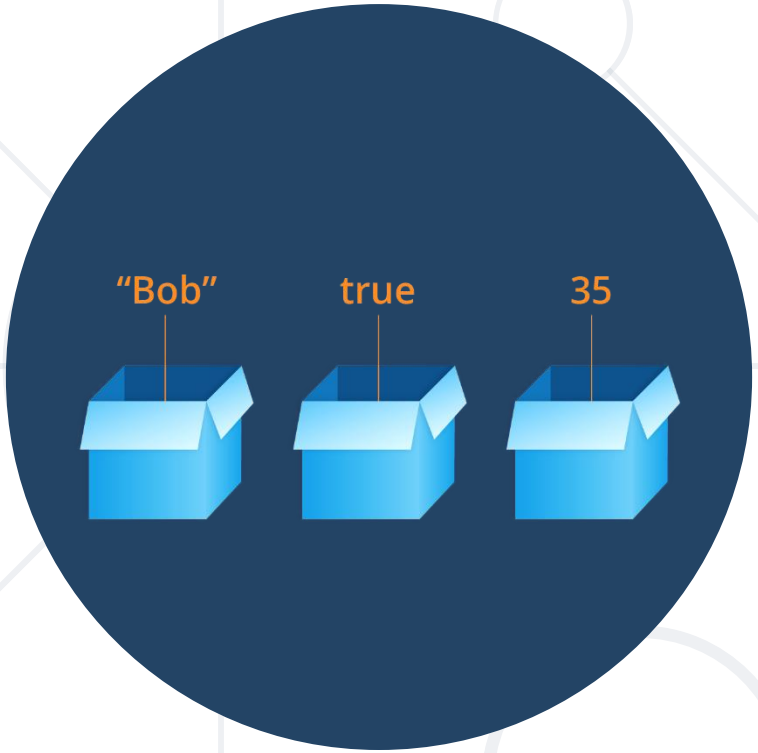
Program Structure: Statements and Comments

Software University

- Statement: **a piece of code to be executed**
- 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 and ***/** ends it



Data Types and Variables

Declaring and Initializing Variables

```
<data_type> <identifier> [= <initialization>];
```

- **Declaring**: `int num;`
- **Initializing**: `num = 5;`
- **Combined**: `int num = 5;`

- Can declare multiple of same type by separating with comma `,`

```
int count = 1, money = 10;
```

- What if you don't provide initialization value: **default initialization**

Local Variables

- Defined **inside blocks**
- Usable only from **code in their block**
- Locals get initialized to **indeterminate values**
- This is dangerous as it can cause **undefined behavior** if we use them later in the program

Global Variables

- Defined **outside blocks**
- Usable from **all code**
- **Variable scope**: good practice to keep the scope of variables as tight as possible

Example: Local and Global Variables

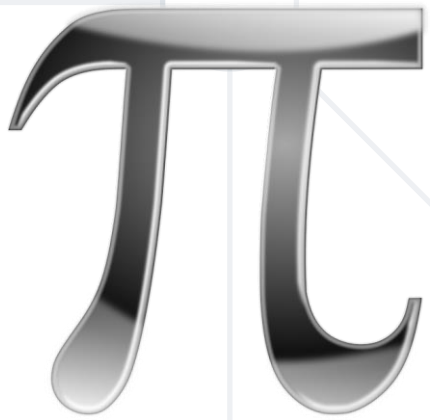
```
//GLOBAL
int secondsInMinute = 60;
int minutesInHour = 60;
int hoursInDay = 24;
int secondsInHour = secondsInMinute * minutesInHour;

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

Constant Variables

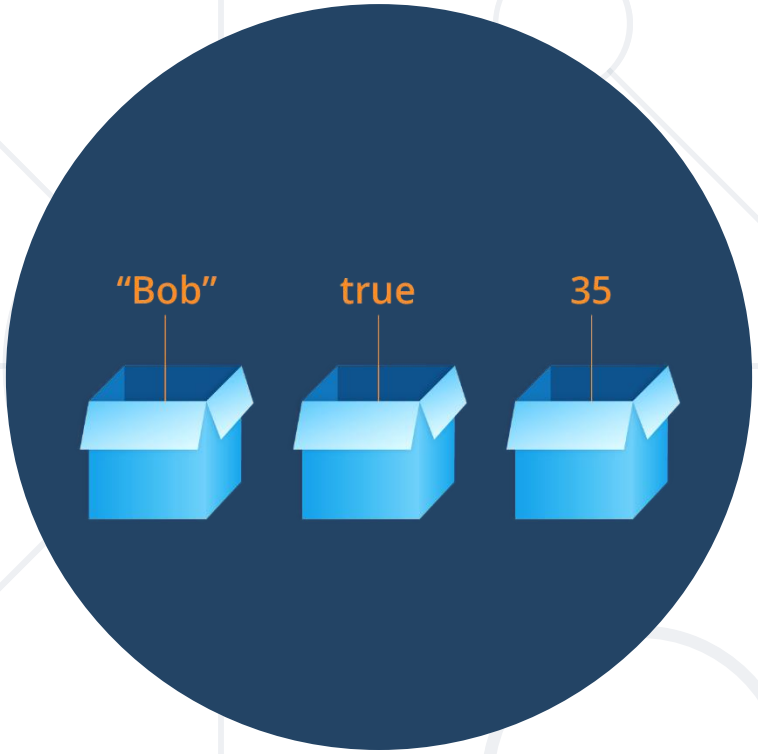
- Variables that **can't change value**
- Must receive a **value at initialization**, nowhere else
- Can be **local**, can be **global**
- Good practice to make **any variable possible** a const variable
- Example:

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



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` variables
  - Tells the compiler a **variable exists somewhere** in a multi-file project



Primitive Data Types

Integer Types

- **short** – at least 16 bits
- **long** – at least 32 bits
- **long long** – 64 bits

- **signed** and **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**

Floating-Point Types

- Represent **real numbers**
  - **Examples**: 2.3, 0.7, -Infinity, -1452342.2313
- **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.	4 bytes	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ (~7 digits)
double	Double precision floating point number.	8 bytes	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ (~15 digits)
long double	Long double precision floating point number.	8 bytes	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ (~15 digits)

Character Types

- **char** is the basic character type
- Basically an **integer** interpreted as a **symbol** from ASCII
- Guaranteed to be **1 byte**
- Initialized by either a character literal or a number (ASCII code)

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

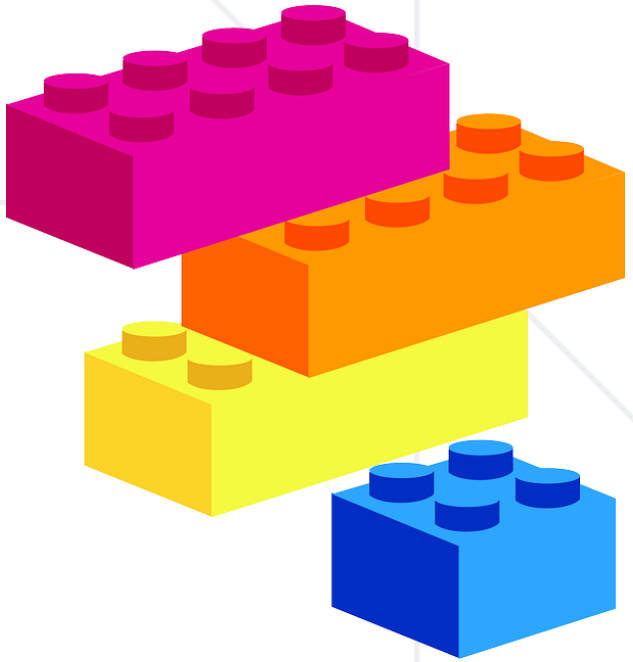
Boolean Type

- **bool** is 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**

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

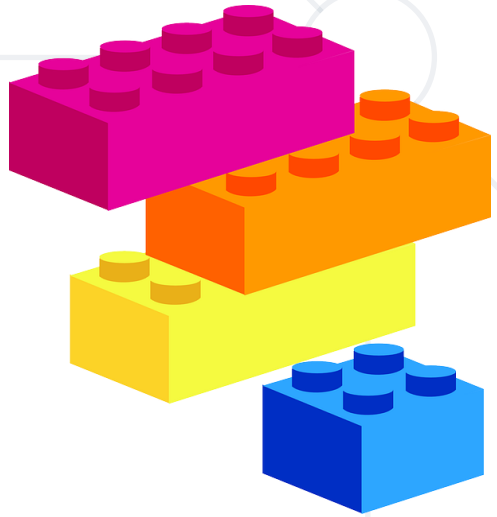
Implicit Casting

- 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**



Explicit Casting

- 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;`
    - `smallType smallVar = static_cast<smallType> bigVar;`
- Can lose accuracy if value can't be represented in a **smaller** type



Literals

Numeric Literals

Software University

- 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 isValid = true;
```

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**
  - Don't **memorize**
  - Use **brackets** or check precedence when needed
- **Expressions:** literals/variables combined with operators/functions

Commonly Used C++ Operators

Software University

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

Conditionals

If - else

- 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
- **If statement with initializer** (C++ 17)

```
if (init, condition)
```

Example: if - else

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

```
double value1 = 10;
double value2 = 20;
if (value1 > value2)
{
    cout << "value1 is larger" << endl;
}
else
{
    cout << "value2 is larger" << endl;
}
```

Ternary Operator

- **Conditional** operator
- Provides a concise way to **perform a simple if-else**
- Syntax: **condition ? value_if_true : value_if_false;**
- Example:

```
int age = 34;
string status = (age >= 18) ? "adult" : "child";
cout << "You are an " << status << "." << endl;
```

The Switch-Case Statement

Switch - Case Structure

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

```
switch (number)
{
    case 1: cout << "one"; break;
    case 2: cout << "two"; break;
    case 3: cout << "three"; break;
}
```

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**

Example: 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;
}
```

Loops

For Loop

```
for([init]; [condition]; [increment])
{
    //body
}
```

- The **init** statement can declare and initialize variables
- The loop runs while the **condition** statement is **true**
- **Increment** is executed AFTER the body

While Loop

- `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;
```

Keywords: Break and Continue

- 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

C++ Streams

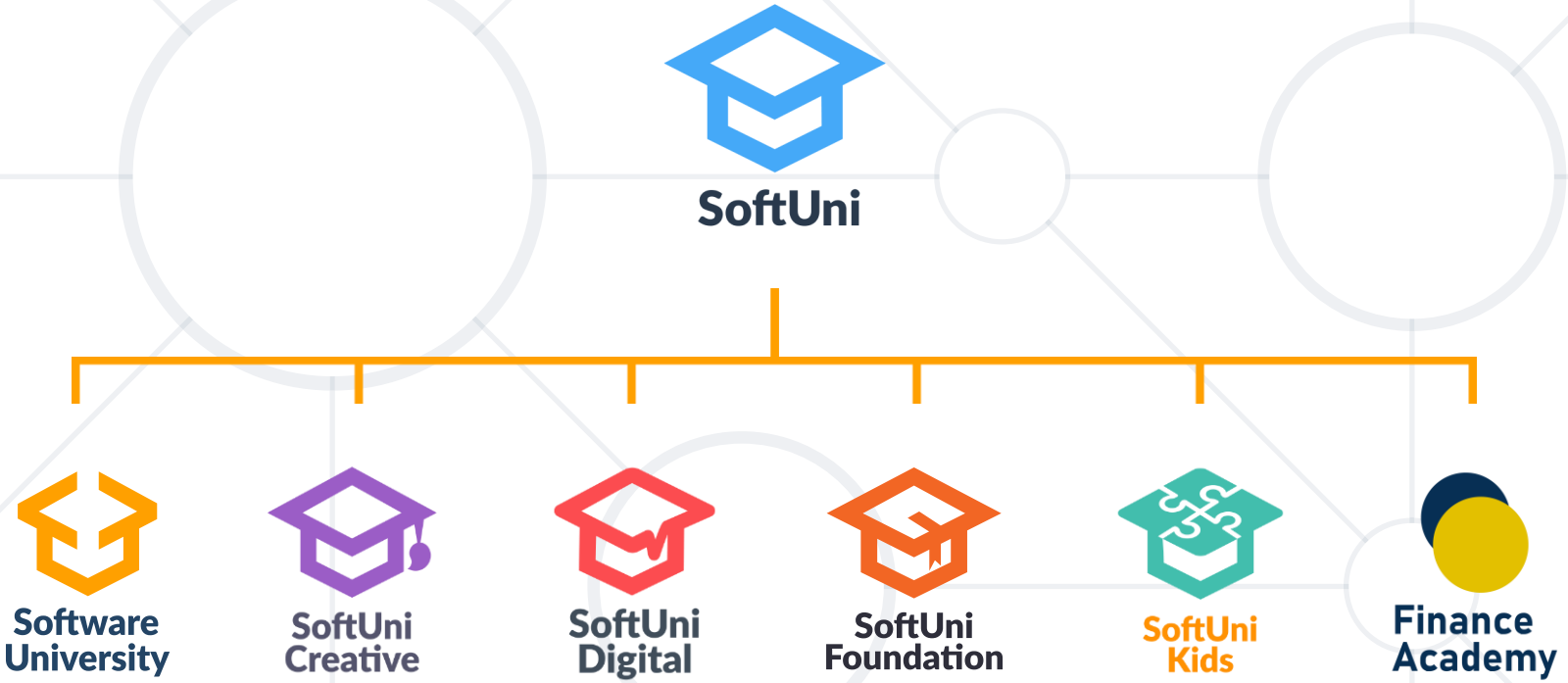
- `cin`
  - uses the `>>` operator
  - read data from the console
- `cout`
  - uses the `<<` operator
  - write data to the console

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

Summary

- 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 Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank
Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

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