Basic Syntax

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

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg/

Table of Contents

Have a Question?

Software University

sli.do

#cpp-fundamentals

What is C++?

What is C++?

Software University

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

Example: Hello World

- A classic "Hello World" example


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

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

Entry Point and Termination

Software University

- The `main` function – entry point of the program
  - No other function can be named "`main`"
  - 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

Software University

- Has a lot of functionality in its standard code libraries
- 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' and conditionals' code is in blocks

`main()` code block

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

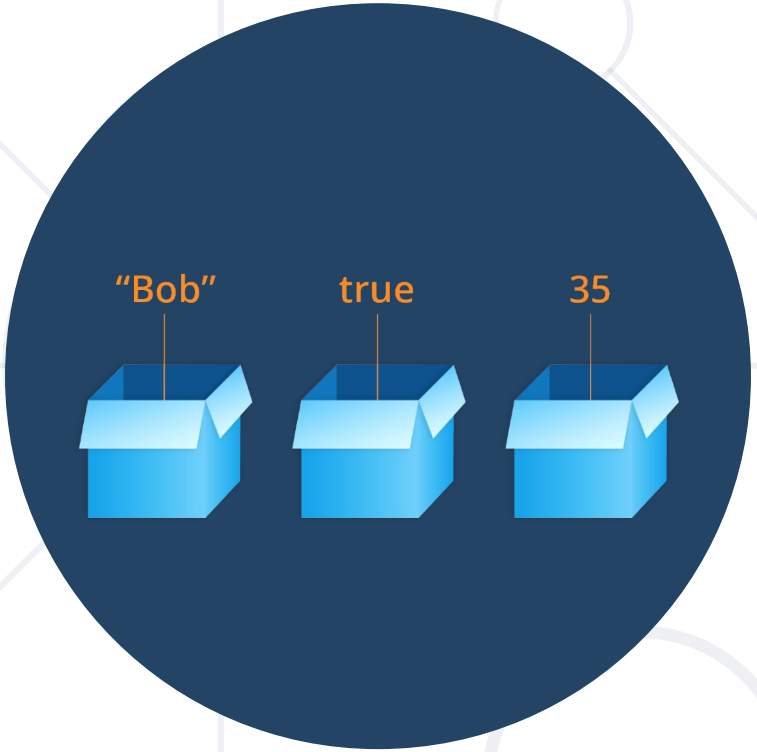
Program Structure: Statements and Comments

Software University

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

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

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



Data Types and Variables

Declaring and Initializing Variables

Software University

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

- Declaring: `int num;`
- Initializing: `num = 5;`
- Combined: `int num = 5`,
  and additionally `int num(5); int num{5};`
- Can declare multiple of same type by separating with comma `,`
  - `int trappist1BMassPct=85, trappist1CMassPct=80;`

Local and Global Variables

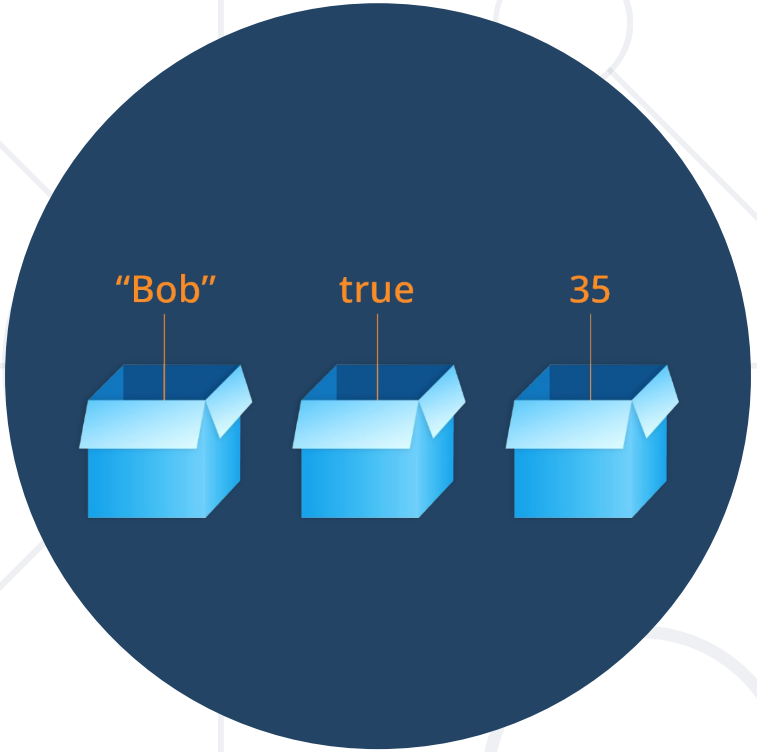
- **Global**
  - defined **outside blocks**, usable from all code
- **Local**
  - defined **inside blocks**, usable only from code in their block
  - DO NOT get **initialized automatically**

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;
}
```



Primitive Data Types

Integer Types – `int`

- 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** 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

Software University

- Represent real numbers (approximations)
  - **2.3, 0.7, -Infinity, -1452342.2313, NaN**
- **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

- **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)

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

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);
}
```

Implicit and Explicit Casting

Software University

- 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 and 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;
```

- Can lose accuracy if value can't be represented in a **smaller** type

Conditionals and Loops

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 cppIsCool = true;
```

Expressions and Operators

Expressions and Operators

Software University

- **Operators**: perform actions on one or more variables / literals
  - Can be customized for different behavior **based on data type**
  - Operator precedence and associativity table: http://en.cppreference.com/w/cpp/language/operator_precedence
- **Expressions**: literals / variables **combined** with operators / functions

Commonly Used Operators

Software University

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

Conditionals

- 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

- 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 Switch-Case Statement

Switch-case

Software University

- Example of 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

Software University

- The 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
- 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

- **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

Loops

Code Block Repetition

For Loop

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

While Loop

Software University

```
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;
```

Do - While Loop

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

Loop Control Keywords

- 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

Streams

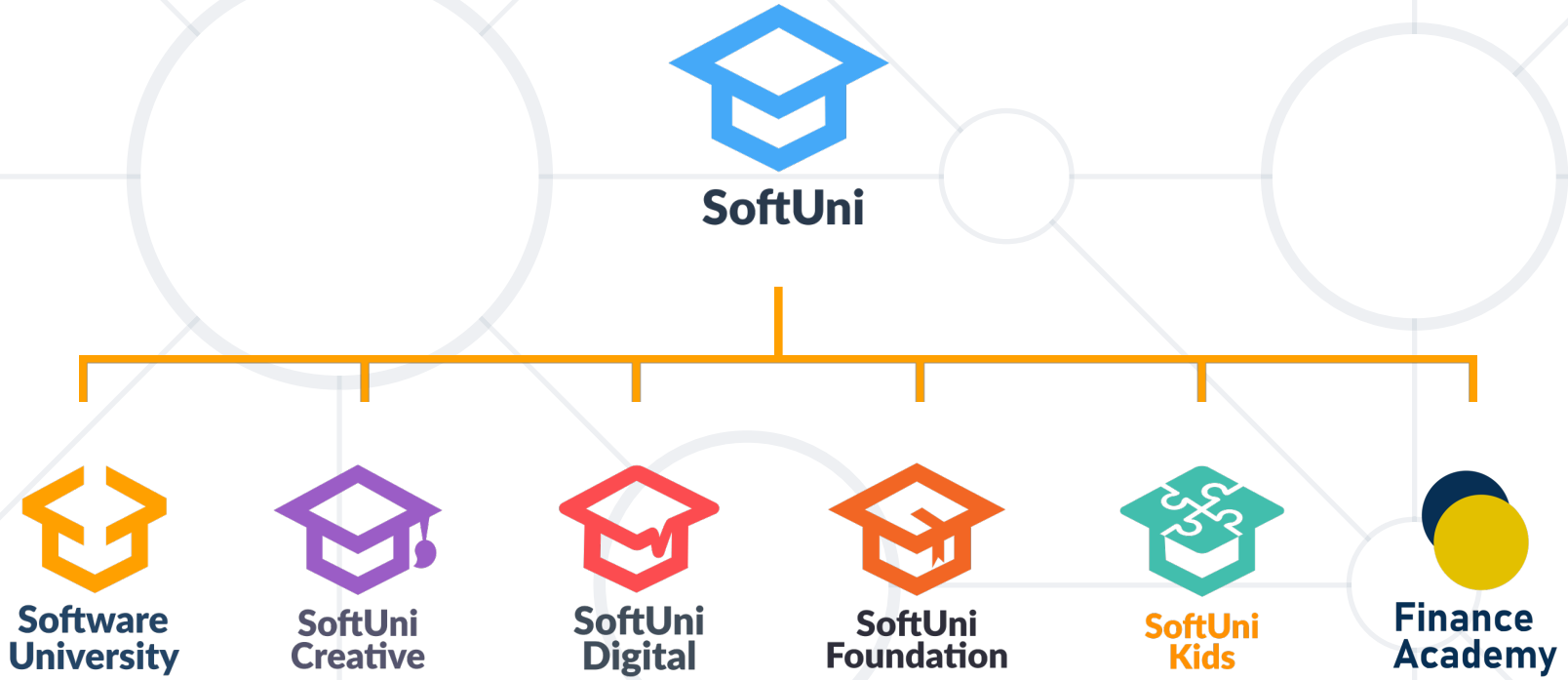
- Classes that either read or write data piece by piece
- `cout`
  - writes data to the console
  - standard output
  - uses the `<<` operator to write
- `cin`
  - reads data to the console
  - standard input
  - uses the `<<` operator to write

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

Summary

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

Questions?



SoftUni Diamond Partners

Software University

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

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