
02393 Programming in C++

# Module 2: Basic C++ and Data Types

**Alceste Scalas**
<alcsc@dtu.dk>

7 September 2021

# Course plan

| Module no. | Date | Topic | Book chapter* |
|---|---|---|---|
| 0 and 1 | 31.08 | Welcome & C++ Overview | 1 |
| 2 | 07.09 | Basic C++ and Data Types | 1, 2.2 – 2.5 |
| 3 | 14.09 | *LAB DAY* | *C++ Practice* |
| 4 | 21.09 | Data Types | 2 |
| 5 | 28.09 | Libraries and Interfaces | 3 |
| 6 | 05.10 | Classes and Objects | 4.1, 4.2 and 9.1, 9.2 |
| 7 | 12.10 | Templates | 4.1, 11.1 |
| | | *Autumn break* | |
| 8 | 26.10 | Inheritance | 14.3, 14.4, 14.5 |
| 9 | 02.11 | **Guest lecture** & *LAB DAY* | *Previous exams* |
| 10 | 09.11 | Recursive Programming | 5 |
| 11 | 16.11 | Linked Lists | 10.5 |
| 12 | 23.11 | Trees | 13 |
| 13 | 30.11 | Conclusion & *LAB DAY* | *Exam preparation* |
| | **05.12** | **Exam** | |

* Recall that the book uses some ad-hoc libraries (e.g., for strings and vectors). We will use standard libraries

# Outline

# A recap from the first lecture

- **The structure of a C++ program**
  - `#include` directives, the `main` function, user-defined functions
- **Simple input/output**
  - `cin`, `cout`
- **Variables, values, and types**
  - `string`, `int`, `double`, `float`
- **Expressions**
  - Some numeric and boolean operators and math functions
- **Statements**
  - `if`, `while`, `for`

# Compiling C++ programs

You should learn to **use the C++ compiler and run programs on a terminal**

1. Edit the `.cpp` source code file (e.g., `program.cpp`) and save it
   - Recommended editor: Microsoft Visual Studio Code
2. Compile `program.cpp` producing the executable `program.exe`:
   ```
   g++ program.cpp -o program.exe
   ```
3. If you don't get compilation errors, run the resulting executable:
   ```
   ./program.exe
   ```

The example above uses g++, but other compilers work similarly

# Arrays in C++

An **array** is a collection of values of a same type

We can easily define arrays when their size is **known at compile-time**

```
int    a[] = {1, 2, 3}; // Array with 3 initialised elements of type int
double b[3];            // Array with 3 uninitialised elements of type double
```

To access array elements, we use **indexing, starting with 0**

```
cout << a[0] + 1 << endl; // Access the value of the first element of a
b[2] = 3.14;              // Overwrite the third (last) element of b
```

**You can index an array beyond its last elements** and get **unspecified results or crashes!**

# Arrays in C++ (cont'd)

How do we **create an array whose size is only known at runtime**?

One might try the following, but **this is not standard C++**!

```
1 int n;
2 cout << "How many elements?" << endl;
3 cin >> n;
4 double b[n]; // Array with n uninitialised doubles.  Valid in C, not in C++!
```

# Arrays in C++ (cont'd)

How do we **create an array whose size is only known at runtime**?

One might try the following, but **this is not standard C++**!

```
1 int n;
2 cout << "How many elements?" << endl;
3 cin >> n;
4 double b[n]; // Array with n uninitialised doubles.  Valid in C, not in C++!
```

Instead, we need to **manually allocate and deallocate the array** using `new` and `delete[]`

```
1 int n;
2 cout << "How many elements?" << endl;
3 cin >> n;
4 double *b = new double[n]; // Pointer to memory with n uninitialised doubles
5
6 // Use b as a normal array...
7
8 delete[] b; // Deallocate the array memory when not needed anymore
```

# Pointers: a first overview

A variable of type pointer contains a **memory address**

The pointer type tells what type of value is stored at that address

```
1  int x = 5;
2  int y = 42;
3  int *p; // p can contain the address of an int value (but is uninitialised)
4
5  p = &x; // Now p contains the memory address of x (written &x)
6
7  // Difference: p (memory address) vs. *p (value stored at that address)
8  cout << "The value of p is " << p << " and points to value " << *p << endl;
9
10 p = &y; // Now p contains the memory address of y (written &y)
11
12 cout << "The value of p is " << p << " and points to value " << *p << endl;
```

# Pointers vs. arrays

A variable of type array is just a **pointer to the beginning of the array**!

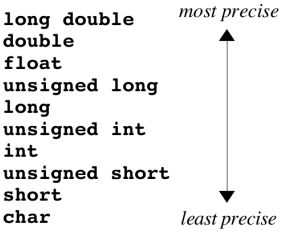
```
int a[] = {100, 101, 102};
int *p; // p can contain the address of an int value (but is uninitialised)

p = a; // Now p is equal to a, so it points to the beginning of the array

cout << "The value of a is " << a << " and points to value " << a[0] << endl;
cout << "The value of p is " << p << " and points to value " << *p << endl;

// We can also index a pointer as if it was an array!
cout << "p[0]=" << p[0] << " p[1]=" << p[1] << " p[2]=" << p[2] << endl;
```

# Data types precision and conversions




C++ numeric data types differ in **memory usage** and **precision**

- They all have **minumum and maximum values**
- The compiler may perform **automatic conversions** toward the "largest" type
- We can explicitly **cast** (convert) values across types (but **be careful**!)
- `float`s and `double`s are sometimes **approximated**, leading to **rounding errors**

Similar issues can be found in many programming languages!

# Miscellaneous features

Here is a code snippet with **some commonly used features and operators**

- preprocessor definitions `#define`
- (postfix) increment operator `++`
- conditional expression `condition ? whenTrue : whenFalse`

```
// We define a preprocessor identifier: MAXIMUM will be replaced by 10
#define MAXIMUM 10

int main() {
  // Below we use 'i++' as shorthand for 'i = i+1'
  for (int i = 0; i < MAXIMUM; i++) {

    // Conditional expression: if i is even, it yields "even", otherwise "odd"
    string oddOrEven = ((i % 2) == 0) ? "even" : "odd";

    cout << "The number " << i << " is " << oddOrEven << endl;
  }
  return 0;
}
```

# Lab (today)

**Today's lab begins now**. Tasks:

- make sure C++ works on your computer, request help if it doesn't
- begin working on **Assignment 2**
- ask questions if something is unclear (including on Assignment 1)

# Lab (today) and LAB DAY (next week)

**Today's lab begins now**. Tasks:

- make sure C++ works on your computer, request help if it doesn't
- begin working on **Assignment 2**
- ask questions if something is unclear (including on Assignment 1)

**LAB DAY on 14 September**. There will be **no new topics** — just programming practice:

- make sure C++ works on your computer, request help if it doesn't
- begin working on **Assignment 3**
- ask questions if something is unclear (including Assignments 1 and 2)