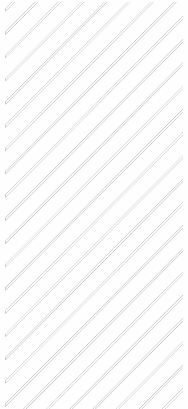

Lecture 1

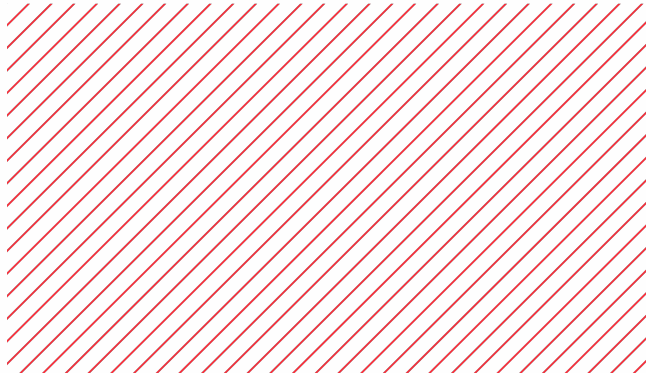
# Introduction to C++

Konstantin Leladze

C++ Basics

DIHT MIPT 2021





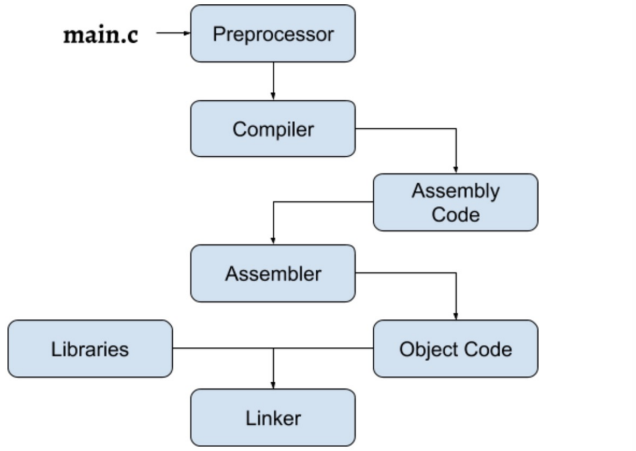
# C++ language

**Key features:**

- Compiled language
- Multi-paradigm programming language:
  - Procedural
  - Object-oriented
  - Functional
  - Generic
  - Modular
- Standardized

Language versions (according to standard):

C++98
C++03
C++07
C++11
C++14
C++17
C++20



# Program structure

- A program is a sequence of instructions
- A C++ program must have a main function.
- Before the program code, you can specify header files.
  Example: `#include <iostream>`
- Function body "is a sequence of instructions separated by a symbol ;
- Usually, instructions are of three types: declaration, expression, control structure

```cpp
#include <iostream>



int main () {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}

```

# Declarations

A declaration is a statement in two or three parts:

```
type identifier [= value or literal];
```

With a declaration, you can create a variable:

```
int x = 3;
```

**Type:**

- In C ++, the type system is strict. This means that every variable in your code has one well-defined type. In Python, for example, this is not the case.
- During the execution of your program, in the areas of RAM allocated for storing your local variables a new space of size corresponding to the size of the type of your variable is allocated in result of the declaration instruction.
- Each type has a numeric value called the size of the type, which indicates how much space variables of this type take up in RAM.

# Fundamental types

C ++ has a specific set of built-in types:

- **Integer** variables: store only integer (possibly negative) values
- **Real** variables: store an arbitrary real number
- **Boolean** variables: store one of two possible values: true or false.
- **Character** variables: store an arbitrary character from an ASCII table (will discuss that later)

| | |
|---|---|
| `int` | Size: 4 bytes. Stores integers in the range [-2 ^ 31, 2 ^ 31 - 1] |
| `unsigned` | Size: 4 bytes. Stores positive integers in the range [0, 2 ^ 32 - 1] |
| `long long` | Size: 8 bytes. Stores integers in the range [-2 ^ 63, 2 ^ 63 - 1] |
| `unsigned long long` | Size: 8 bytes. Stores positive integers in the range [0, 2 ^ 64 - 1] |
| `float` | Size: 4 bytes. Stores an approximate value of a real number |
| `double` | Size: 8 bytes. Stores an approximate value of the a number with double precision |
| `long double` | Size: 16 bytes. Stores an approximate value of the a number with 4x precision |
| `bool` | Size: 1 byte. Stores a Boolean value: true or false |
| `char` | Size: 1 byte. Stores a character from ASCII table |

# Identifiers

Any sequence of Latin letters, numbers and underscores that does not begin with a number and is not a **keyword** can be used as an identifier.

Example:

- `a_simple_identifier`
- `x_dir1`

Keywords:

- `if`
- `else`
- `for`
- `do`
- `while`
- `switch`
- `class`
- …

**NON**-Keywords:

- `vector`
- `map`
- `cout`
- `cin`
- `endl`
- `list`
- `array`
- `...`

# Literals

A literal is a typed special value that can be used to initialize a variable.
most primitive types have literals of their own.

Examples:

- `1` `int`
- `'a'` `char`
- `true` `bool`
- `false` `bool`
- `"abc"` `string` literal (will discuss later)
- `0.2f` `float` literal
- `0.3` `double` literal
- `nullptr` `pointer` literal (will discuss later)

# Variable definitions with literals

**Examples**:

```
int x = 5;
bool y = true;
float z = 5.1f;
double k = 5.0;
long long p = 5ll;
unsigned long long q = 4ull;
```

**Implicit type conversions**

if you try to initialize a variable of a certain type with a certain literal of another type, in some cases it will work (and in some it will be an error).

**Examples**:

```
int x = 5.8;
bool y = 1;
float z = 5.1;
double k = 5;
```

# Other examples of implicit conversions

**`bool` <–> `int` implicit conversions:**

`bool` –> `int`:
`false` –> `0`
`true` –> `1`

`int` –> `bool`:
`0` –> `false`
Every other number –> `true`

**Rule**: when converting between types of different sizes, it is important to follow the rule: a number that can be converted to a specific type should not exceed the range of values of this type. Otherwise, a certain complex and ambiguous situation will occur, which is called overflow.

**Some more examples of implicit type conversions**:

| | |
|---|---|
| `'a' + 'b'` –> `int` | `char` –> `int` |
| `5/3` –> `int` | division operator (will be discussed later) results in integer |
| `long long x = 5` | `int` –> `long long` |

# Scopes

Each name that appears in a C++ program is only visible in some possibly discontiguous portion of the source code called its *scope*.

Within a scope, unqualified name lookup can be used to associate the name with its declaration. There are different kinds of scopes: block scope, function arguments scope, namespace scope, enumeration scope, class scope, etc. But today we will only talk about one of them: **block scope**.

**Block scope**

The potential scope of a name declared in a block (compound statement) begins at the point of declaration and ends at the end of the block. Actual scope is the same as potential scope unless an identical name is declared in a nested block, in which case the potential scope of the name in the nested block is excluded from the actual scope of the name in the enclosing block.

```
int main() {
    int i = 0; // scope of outer i begins
    ++i; // outer i is in scope
    {
        int i = 1; // scope of inner i begins,
                   // scope of outer i pauses
        i = 42; // inner i is in scope
    } // block ends, scope of inner i ends,
      // scope of outer i resumes
} // block ends, scope of outer i ends
//int j = i; // error: i is not in scope
```

# Scopes

**One definition rule**:
A variable with a certain identifier can only be defined once in a scope.

Error:

```
int main () {
    int x = 3;
    int x = 4;
}
```

BUT this code is ok:

```
int main () {
    int x = 3;
    {
        int x = 4;
    }
}
```


Lecture 1

# Introduction to C++

Konstantin Leladze

C++ Basics

DIHT MIPT 2021