# Contemporary C++: Learning Modern C++ in a Modern Way

## الماس فناوری ابری پاسارگاد- آلفا

مدرس: سعید امراللهی بیوکی

# Agenda 2/24



- Hello C++: The first program: Hello, world!
- Install and Setup C++ compilers: GCC and Clang
- Compile, Link and Execute chain
- 2nd program: Say a greeting to a specific person
- Basic concepts of Stream I/O
- Fundamental Types
- Fundamental Types I/O
- Expressions: Introduction to C++ operators
- Q & A

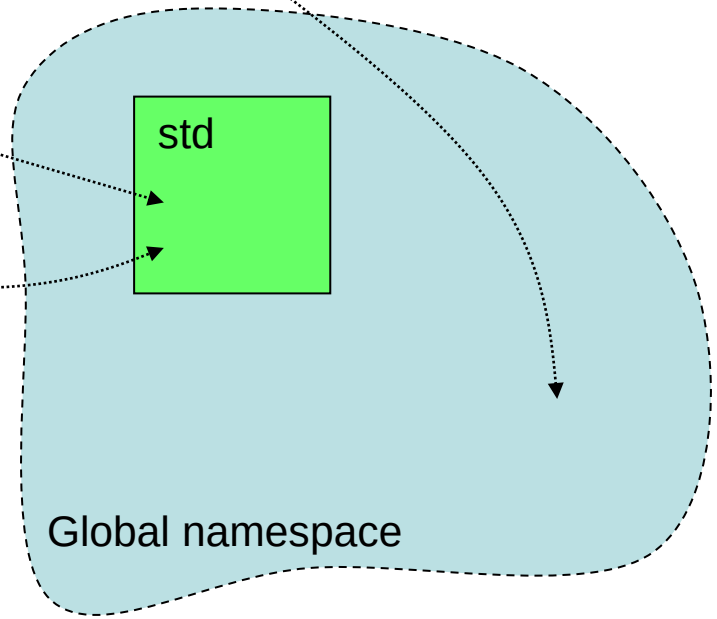
150 min (incl. Q & A)

# Hello C++

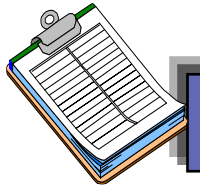
```
// Hello world in Standard C++
#include <iostream>
int main()
{
   std::cout << "Hello, world!" << std::endl;

   return 0;
}
```

main function

```
// Hello world in C++
#include <iostream.h>
main()
{
   cout << "Hello, world\n";
}
```




Remember that standard library facilities are defined in namespace std.



Standard C++ = C++ programming language (core language) + standard library

# The anatomy of Hello, world! program

- Comments
  - Single line comments
    - //
    - C++
  - Multi-line comments
    - /*, */

```
// Hello world! in Standard C++
```

```
// This is a single line comment.
```

```
/*
   This is a multi-line comment.
*/
/* single line comment */
```

- multi-line comments and comments that end before the end of a line.
  - C/C++

- The compiler ignore comments.

Angle brackets  Standard header

```
#include <iostream> // get the standard I/O facilities
```

- #include

- Programs ask for standard-library/non-standard library facilities by using #include directive.

☞ Remember to #include the headers for the facilities you use;

# main function

- Function: mathematical concept vs. Programming concept

$$y = f(x)$$

- Function: A function is a piece of program that has a name, and that another part of the program can *call*, or cause to run.

```
int  main()
```

Return type

Function name

Function parameters (no parameters)

- Every C++ program must contain a function called **`main`**.

- Curly braces
  - In C++, braces tell the implementation to treat whatever appears between them as a unit.

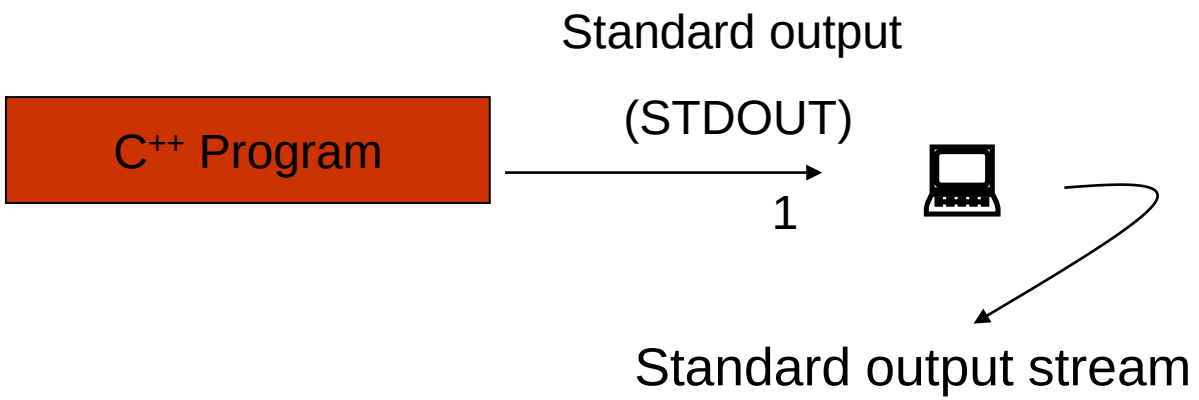
```
int main()
{ // the left brace
  // the statements go here
} // right brace
```

# Standard library and output

- The name iostream suggests support for sequential, or stream, input-output, rather than random-access or graphical input-output

```
std::cout << "Hello, world!" << std::endl;
```

- Output operation:
  - Output operator: <<
  - put to
  - Insertion operator (more technical sounding name)
  - `cout`, an `ostream` class object

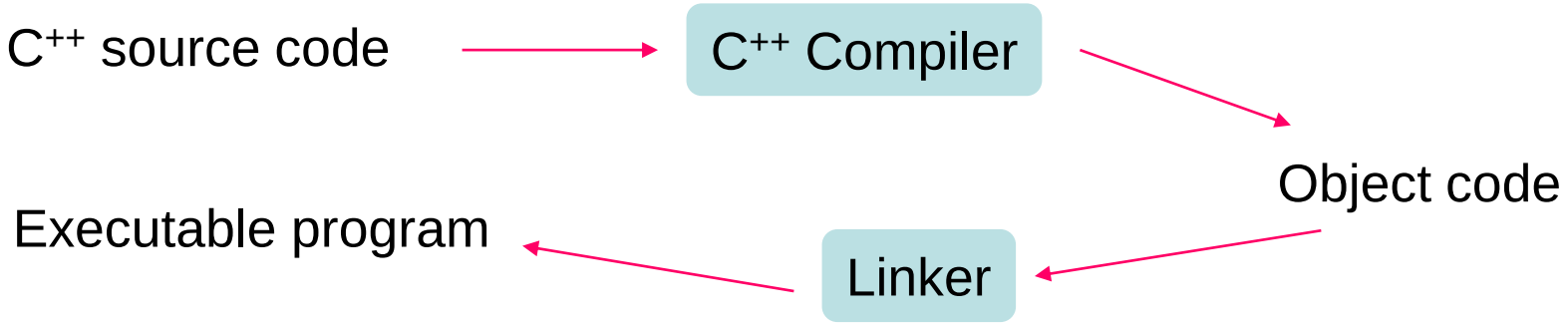



- namespace `std`
- `std::endl`

- return statement
- return pass the value to operating system

```
return 0; // 0 means success
```

# Compiling and Linking




- You write C++ source code
  - Source code is (in principle) human readable
- The compiler translates what you wrote into object code (sometimes called machine code)
  - Object code is simple enough for a computer to “understand”
- The linker links your code to system code needed to execute
  - E.g. input/output libraries, operating system code, and windowing code
- The result is an executable program
  - E.g. a .exe file on windows or an a.out file on Unix

```
$ g++ hello.c++
$ ./a.out
Hello, world!
```

# Hello, world!: A deeper look

- Expression

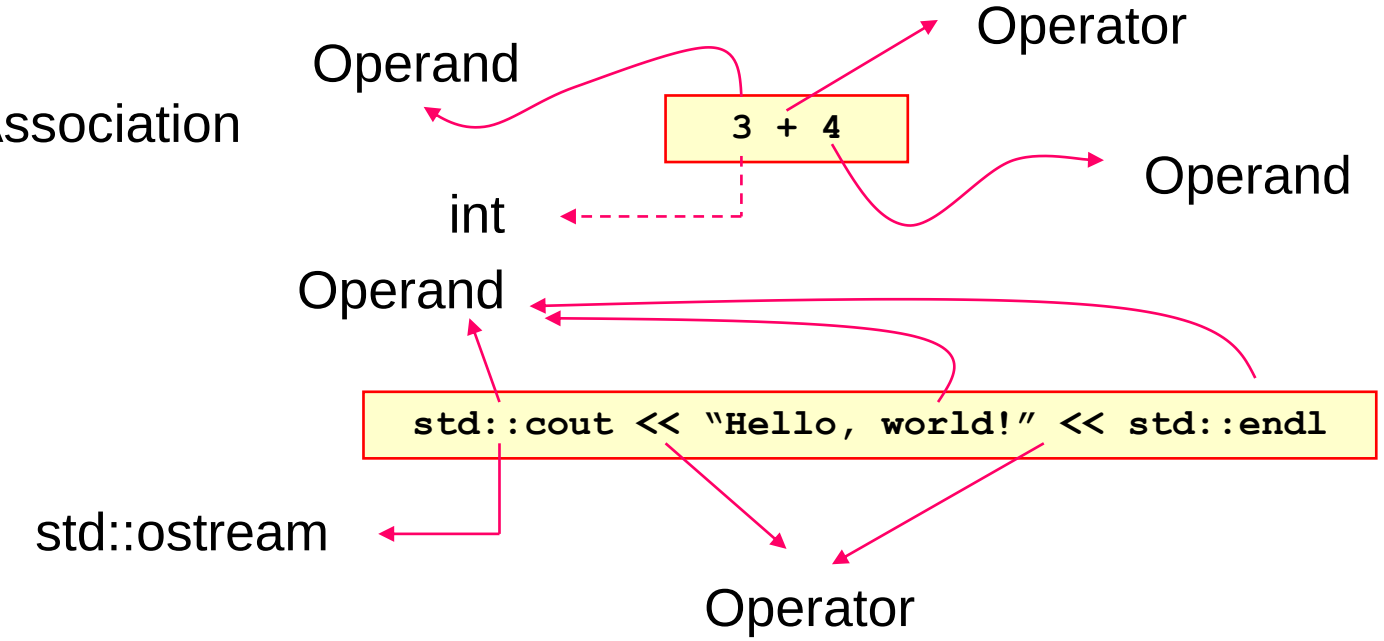
- An expression asks the implementation to compute something. The computation yields a result, and may also have side effects.

```
3 + 4 // result 7, no side effect
```

```
std::cout << "Hello, world!" << std::endl // no result, has side effect
```

- As its side effect, writes Hello, world! on the standard output streams and ends the current line.

- Precedence and Association




- Every operand has a type.

# Hello, world!: A deeper look cont.

- Precedence and Association
  - << is left-associative

```
std::cout << "Hello, world!" << std::endl
```

```
std::cout << "Hello, world!"; // return: nothing, side effect: output
std::cout << std::endl; // return: nothing, side effect: go to next line
```

Statement vs. expression

```
3 + 4  // expression
3 + 4; // statement
```

- std::endl is a manipulator. A manipulator, manipulates a stream.

• Scope

- The scope of a name is the part of a program in which that name has its meaning.

- Block
- Compound statement = Block

Left curly brace

Right curly brace

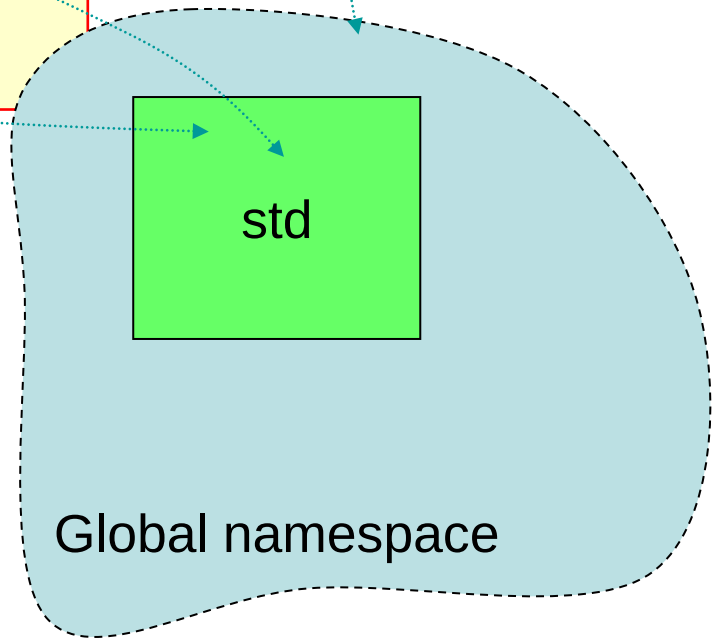
```
{
  /* … */
}
```

# Extending Hello, world!: Hello to specific person

- Once, we can write text, the logical next step is to read it.

```
#include <iostream>
#include <string>
int main()
{
  // ask for the person's name
  std::cout << "Please enter your first name: ";
  // read the name
  std::string name;        // define name
  std::cin >> name;        // read into name
  // write a greeting
  std::cout << "Hello, " << name << "!" << '\n';

  return 0;
}
```

```
$ g++ greeting.c++
Please enter your first name: Saeed
Hello, Saeed!
```



# Variable, object, type

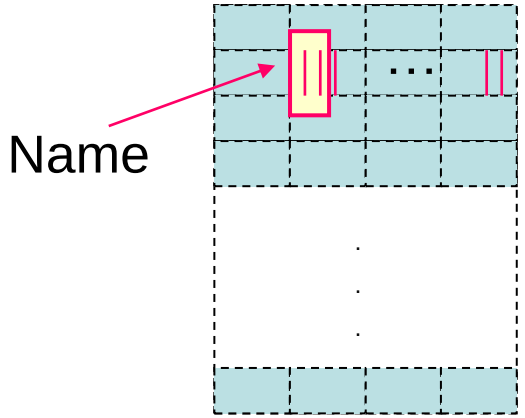
- Variable
  - Variable is a container.
  - Variable is an object that has a name. An object, in turn, is a part of the computer's memory that has a type.

First definition of object

Variable : name + type

```
std::string name;
```

type name




Memory

- Symbolic variables

```
#include <iostream>
int main()
{
  std::cout << "2 raised to the power of 10: ";
  std::cout << 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2;
  std::cout << std::endl;
}
```

# The anatomy of greeting program

```
std::cout << "Please enter your first name: "; // wait at the current line
```

```
std::string name; // a definition
```

- Because **`name`** is defined in a function (main) it is local variable.

```
int main()
{
  std::string name; // local variable

}
```

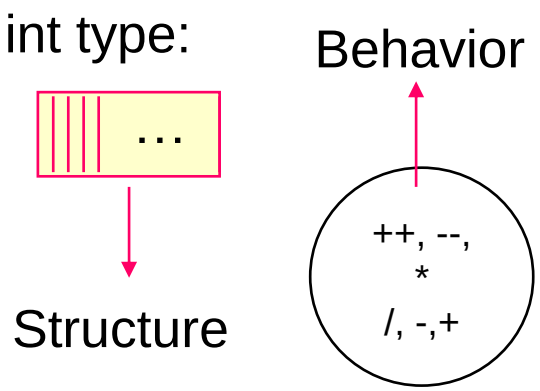
```
int main()
{
  std::string name;  // create here


} // destroy here
```

```
int i; // int is a built-in type
```

```
std::string name; // string is a user-define type
```

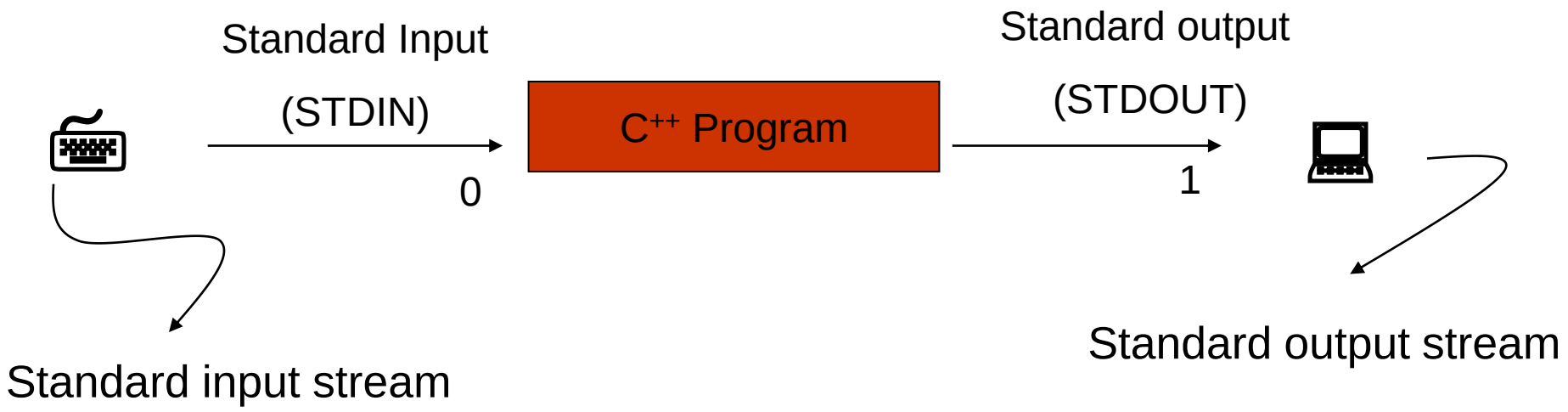
- Initialization

```
std::string name; // null or empty string
```



# Standard library and input

```
std::cin >> name; // read into name
```




- Input operation:
  - Input operator: >>
  - get from
  - Extraction operator (more technical sounding name)
  - cin, an istream class object

# Input operator

- >> operator skips whitespace.
- Whitespace characters: space, tab, backspace, or the end of the line.

```
std::string name;
std::cin >> name; // read a word: Saeed
```

- - - - - -Saeed- - - - - -

```
int i;
std::cin >> i; // read an int:12345
```

- - - - - -12345.6- - - - -

```
// write a greeting
 std::cout << "Hello, " << name << "!" << std::endl;
```

# Identifiers and Keywords cont.

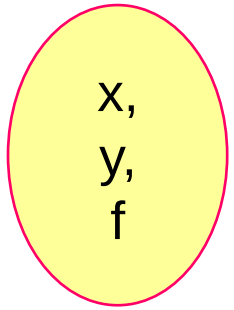
- C/C++ is *case-sensitive* language. Uppercase and lowercase letters are distinct.
- Count, count, COUNT, CounT: four different C++ identifiers.
- C++ Keywords:

  up until now: **`int`**, **`return`**, **`const`**

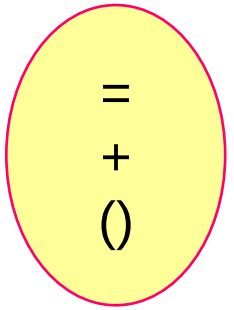
  bool, class, template, try
- C++ Reserved words: include, main, string, vector

# Types, identifiers and names

```
x = y + f(2);
```




Names




Operators

- Every name (identifier) has a type associated with it.

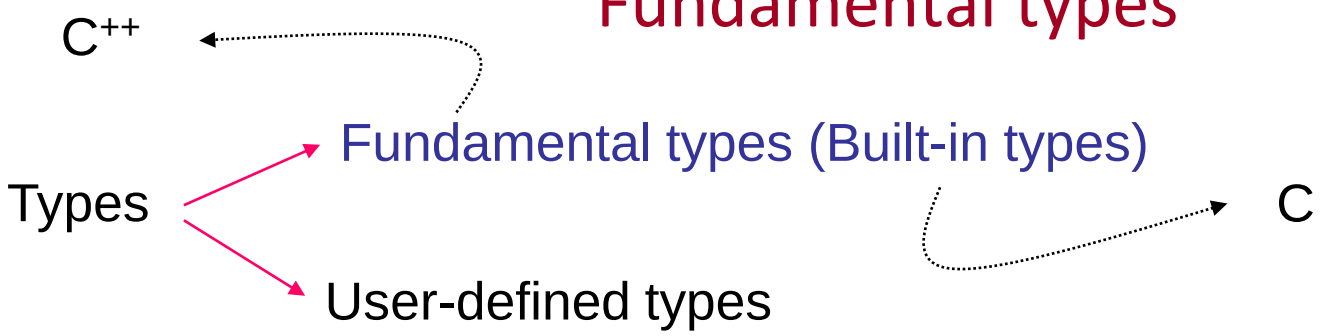
```
float x; // x is a floating point variable
int y = 7; // y is an integer variable with the initial value 7
float f(int); // f is a function taking an argument of type int and returning a
              // floating point number
```

- Name = Identifier
- An identifier is a sequence of letters and digits. The first character should be letter.

Letter: A-Z, a-z, _ Digit: 0-9

# Fundamental types?

# Fundamental types




- Boolean type *(bool)*

- Character types (such as *char*): A single byte, capable of holding one character in the local character set.

- Integer types (such as *int*): An integer, typically reflecting the natural size of integers on a machine.

- Floating-point types (such as *double*): Floating point numbers.

- User-defined types: enumerations, void, pointers, arrays, references, data structures and classes

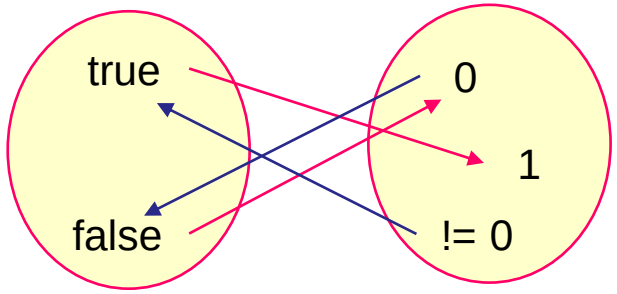
# Booleans

- A boolean, `bool`, can have one of two values *true* or *false*.

- boolean ←→ logical operations

```
bool accept()
{
  cout << "Do you want to proceed (y or n )? "; // write question
  char answer;
  cin >> answer; // read answer
  if (answer == ´y´) return true;
  return false;
}
```

- boolean conversion:




boolean values

integer values

```
bool greater(int a, int b)
{
  return a > b;
)
```

```
void f(int a, int b)
{
  bool b1 = (a == b);
}
```

```
bool b = 7; // bool(7) is true, so be becomes true.
int i = true; // int(true) is 1, so i becomes 1.
```

- boolean literals: *true*, *false.*

# Character types

- A variable of type **`char`** can hold a character of the implementation's character set.

```
char c = 'a';
```

- C++ character set consists of 96 characters:

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

_ { } [ ] # ( ) < > % : ; . ? * + - / ˆ & | ˜ ! = , \ " '

The space, Control characters: horizontal tab, vertical tab, form feed, and new-line.

- The size of character is implementation-defined. Almost universally, characters are 8 bits. So there can be 256 different characters

  - EBCDIC

  - ASCII

# Character types cont.

- Each character constant has an integer value.

ASCII character set

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 'A' | 'B' | | 'a' | 'b' | |
| 0 | 1 | … | 65 | 66 | … | 97 | … | 255 |

- The notation int (c) gives the integer value for a character c .

```
// read a character from input and write its integer value
#include <iostream>
int main()
{
  char c;
  std::cin >> c;
  std::cout << "The value of '" << c << "' is " << int(c) << std::endl;

  return 0;
}
```

- A type wchar_t is provided to hold characters of a larger character set such as *Unicode*. The size of wchar_t is implementation-defined.
- Character type is integral type: Arithmetic operation: +, - , ++, --, *, /, …
- character literals: 'a', '0', '!', ... The type of character literals are **`char`**.
- wide character literals: L'ab', L'ac'. The type of character literals are **`wchar_t`**.

# Integer types

- Integer types:
  - short int (short)
  - int
  - long int (long)

```
int i;
short si = 5;
long int li = 1000000000;
```

- The size of integer types are *implementation-defined*.

- Integer literals:
  - *decimal*
  - *octal*
  - *hexadecimal*
  - *character literals*

```
decimal:        0 2 6 3 8 3
octal:          00 02 077 0123
hexadecimal:    0x0 0x2 0x3f 0x53
```

- The letters a , b , c , d , e , and f , or their uppercase equivalents, are used to represent 10 , 11 , 12 , 13, 14, and 15, respectively.

- By default, a integer literal is of type *int*.

# C++14

Binary literals

digit separators

ATTRIBUTES

Function return type deduction

Global cbegin/cend

Generic lambda expressions

# C++14

Binary literals

digit separators

ATTRIBUTES

Function return type deduction

Global cbegin/cend

Generic lambda expressions

# Digit separators

- In C++14, the single-quote character may be used arbitrarily as a digit separator in numeric literals, both integer literals and floating point literals.

```
long long wp = 7'000'000'000; // World population
int shares = 500'000'000; // # shares at TSE
double price = 400.'00; // the closing price of each share in IRR
```

☞ Use digit separators to make large literals readable.

# Floating-point types

- The floating point types represent floating point numbers.
- floating-point types:
  - float (single precision)
  - double (double precision)
  - long double (extended precision)

```
float f; // uninitialized
double d2 = 1; // convert to 1.0
double pi = 3.14;
long double e = 2.7182818284590452354;
```

- The exact meaning of single, double, and extended precision is *implementation-defined*.
- By default, a floating-point literal is of type *double*.

```
1.23     .23     1.      6.02e23
```

# The sizes of fundamental types

- Sizes of C++ objects are expressed in terms of multiples of the size of a **`char`**.
- sizeof operator: size of type and size of object

```
1  ≡ sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)
sizeof(char) <= sizeof(wchar_t)
sizeof(float) <= sizeof(double) <= sizeof(long double)
```

- A **`char`** has at least 8 bits.
- A **`short`** has at least 16 bits.
- A **`long`** has at least 32 bits.

For most applications, one could simply use **`bool`** for logical values, **`char`** for characters, **`int`** for integer values, and **`double`** for floating-point values.

# The size of fundamental types

- Write a program that prints the sizes of the fundamental types, and a few variables of fundamental types.

```
#include <iostream>

int main()
{
  // fundamental types
  std::cout << "On this machine ..." << std::endl;
  std::cout << "Size of bool is " << sizeof(bool) << std::endl;
  std::cout << "Size of char is " << sizeof(char) << std::endl;
  std::cout << "Size of int is " << sizeof(int) << std::endl;
  std::cout << "Size of long int is " << sizeof(long) << std::endl;
  std::cout << "Size of double is " << sizeof(double) << std::endl;
  // some objects
  int i;
  long double d;
  bool b = false;
  std::cout << "Size of i is " << sizeof(i) << std::endl;
  std::cout << "Size of d is " << sizeof(d) << std::endl;
  std::cout << "Size of b is " << sizeof(b) << std::endl;
  return 0;
}
```

# Arithmetic operators

| Operator | Function | Use |
|---|---|---|
| + | addition | expr + expr |
| - | subtraction | expr - expr |
| * | multiplication | expr * expr |
| / | division | expr / expr |
| % | modulus (remainder) | expr % expr |

Additive

Multiplicative

Precedence: Multiplicative operators have higher precedence than additive operators.

```
int a, b, c;
a = a + b * c; // (a + b) + c
```

Association: All operators are left-to-right associative.

```
int a, b, c;
b = a + b + c; (a + b) + c
int d = a / b * c; // (a / b) * c
```

• Division between integers results in an integer. If the quotient contains a fraction part, it is truncated.

```
int q = 21 / 6; // q = 3;
q = 21 / 7; // q = 3;
```

# *Thanks for your patience …*

A man who asks a question is a fool for minute,
The man who does not ask, is a fool for a life.
- Confucius

Learning to ask the right (often hard) questions is an essential part of learning to think as a programmer.
- Bjarne Stroustrup *programming Principles and Practice Using C++, page 4.*

There is no stupid question, but there is stupid answer.
- Howard Hinnant