Ramona Lea
Università degli studi di Trieste
Laurea Magistrale in Fisica
A.A. 2019/2020

Mail : ramona.lea@ts.infn.it

https://www.ts.infn.it/~lea/cpp2020.html

Moodle: https://moodle2.units.it/course/view.php?id=5049
Corsi 2019/2020- ***455SM-2 - MODULO 2N 2019***

References

- Slides and other material:

  http://www.ts.infn.it/~lea/cpp2020.html

  - Moodle UniTs
- On line resources:
  - http://www.learncpp.com
  - http://www.cplusplus.com
  - http://root.cern.ch
- Book
  - "Programming with C++" John R. Hubbard, Schaum's outlines
  - "C++ How to Program- Fourth Edition", by H. M. Deitel, P. J. Deitel, Prentice Hall, New Jersey, 2003, ISBN: 0-13-038474.
  - "The C++ programming language" Bjarne Stroustrup, Addison-Wesley Professional, 3 edition (1997), ISBN: 978-0201889543
  - "Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples", John J. Barton, Lee R. Nackam, Addison Wesley (1994), ISBN: 978-0201533934

Timetable and final examination

Place:

Monday: Aula T21

Friday : Aula T21

- Timetable:

Moday from 14.00 to ~17.30

Friday from 09.00 to ~12.30

- **Monday 02/12 no lesson**
- Lectures structure: (a bit of) theory and (a lot of) programming will be mixed during the afternoon
- Examination, two steps:
  - "written part" coding an analysis program (at home)
  - "oral part": running and discussion of the code
- To register the vote the Module1 of the course has to be passed

C++ programming

Compilers

- The essential tools needed to do follow this course are a computer and a compiler tool-chain able to compile C++ code and build the programs to run on it
- Computers understand only one language and that language consists of sets of instructions made of ones and zeros. This computer language is appropriately called *machine language*.

  Example: A single instruction to a computer could look like this:

00000	10011110

- A particular computer's machine language program that allows a user to input two numbers, adds the two numbers together, and displays the total could include these machine code instructions:

00000	10011110
00001	10011110
00010	11110100
00011	11010100
00100	10011110

Compilers

- As you can imagine, programming a computer directly in machine language using only ones and zeros is very tedious and error prone. To make programming easier, high level languages have been developed. High level programs also make it easier for programmers to inspect and understand each other's programs easier.
- This is a portion of code written in C++ that accomplishes the exact same purpose:

```
1 int a, b, sum;
2
3 cin >> a;
4 cin >> b;
5
6 sum = a + b;
7 cout << sum << endl;
```

00000	10011110
00001	10011110
00010	11110100
00011	11010100
00100	10011110

- Even if you cannot really understand the code above, you should be able to appreciate how much easier it will be to program in the C++ language as opposed to machine language.

Compilers

- Because a computer can only understand machine language and humans wish to write in high level languages high level languages have to be re-written (translated) into machine language at some point. This is done by special programs called **compilers**, **interpreters**, or **assemblers** that are built into the various programming applications.

- C++ is designed to be a compiled language, meaning that it is generally translated into machine language that can be understood directly by the system, making the generated program highly efficient. For that, a set of tools are needed, known as the development toolchain, whose core are a compiler and its linker.

Console programs

- Console programs are programs that use text to communicate with the user and the environment, such as printing text to the screen or reading input from a keyboard
- Console programs are easy to interact with, and generally have a predictable behavior that is identical across all platforms. They are also simple to implement and thus are very useful to learn the basics of a programming language
- The way to compile console programs depends on the particular tool you are using.
- If you happen to have a Linux or Mac environment with development features, you should be able to compile any program directly from a terminal

Console programs

- Console programs are programs that use text to communicate with the user and the environment, such as printing text to the screen or reading input from a keyboard
- Console programs are easy to interact with, and generally have a predictable behavior that is identical across all platforms. They are also simple to implement and thus are very useful to learn the basics of a programming language
- The way to compile console programs depends on the particular tool you are using.
- If you happen to have a Linux or Mac environment with development features, you should be able to compile any program directly from a terminal

Compiler	Platform	Command
GCC	Linux, among others…	`g++ -std=c++0x example.cpp -o example_program`
Clang	OS X, among others…	`clang++ -std=c++11 -stdlib=libc++ example.cpp -o example_program`

History of C and C++

- History of C:
  - Evolved from two other programming languages (BCPL and B “Typeless” languages)
  - Dennis Ritchie (Bell Laboratories): added data typing, other features
  - Development language of UNIX
  - Hardware independent (Portable programs)
- 1989: ANSI standard
- 1990: ANSI and ISO standard published
  - ANSI/ISO 9899: 1990
- History of C++
  - Extension of C: Early 1980s: Bjarne Stroustrup (Bell Laboratories), “Spruces up” C
- Provides capabilities for object-oriented programming:
  - Objects: reusable software components (Model items in real world)
- Object-oriented programs :Easy to understand, correct and modify
- Hybrid language
  - C-like style
  - Object-oriented style
  - Both

History of C and C++

- History of C:
  - Evolved from two other programming languages (BCPL and B “Typeless” languages)
  - Dennis Ritchie (Bell Laboratories): added data typing, other features
  - Development language of UNIX
  - Hardware independent (Portable programs)
- 1989: ANSI standard
- 1990: ANSI and ISO standard published
  - ANSI/ISO 9899: 1990
- History of C++
  - Extension of C: Early 1980s: Bjarne Stroustrup (Bell Laboratories), “Spruces up” C
- Provides capabilities for object-oriented programming:
  - Objects: reusable software components (Model items in real world)
- Object-oriented programs :Easy to understand, correct and modify
- Hybrid language
  - C-like style
  - Object-oriented style
  - Both

In this course C++ and C differences will NOT be pointed out, but you should already be master of C

C++ Standard Library

- C++ programs
  - Built from pieces called classes and functions
- C++ standard library
  - Rich collections of existing classes and functions
- “Building block approach” to creating programs
  - “Software reuse”

The Key Software Trend: Object Technology

- Objects
  - Reusable software components that model real world items
  - Meaningful software units
    - Date objects, time objects, paycheck objects, invoice objects, audio objects, video objects, file objects, record objects, etc.
    - Any noun can be represented as an object
  - More understandable, better organized and easier to maintain than procedural programming
  - Favor modularity
    - Software reuse
      - Libraries
        - MFC (Microsoft Foundation Classes)
        - Rogue Wave

Basics of a Typical C++ Environment

Phases of C++ Programs:

1)Edit

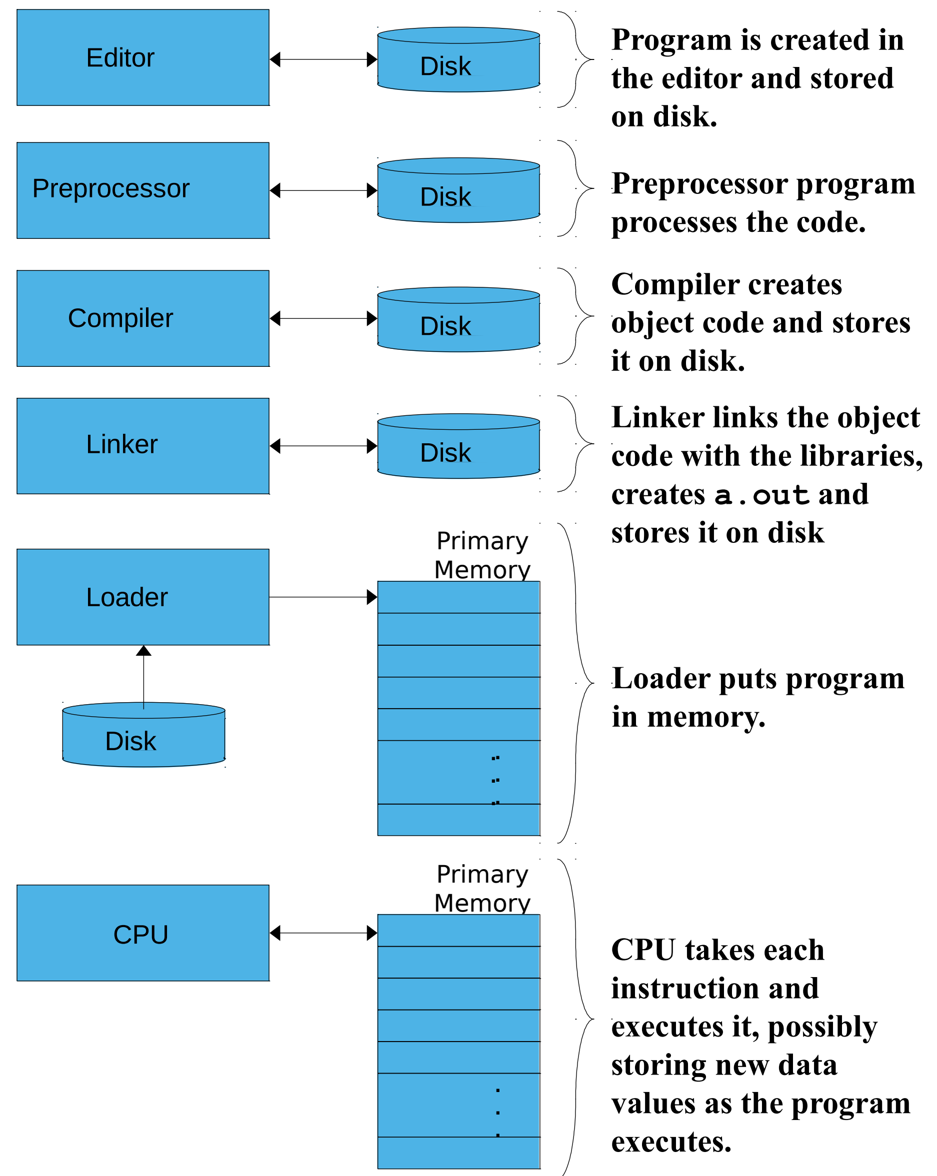
2)Preprocess

3)Compile

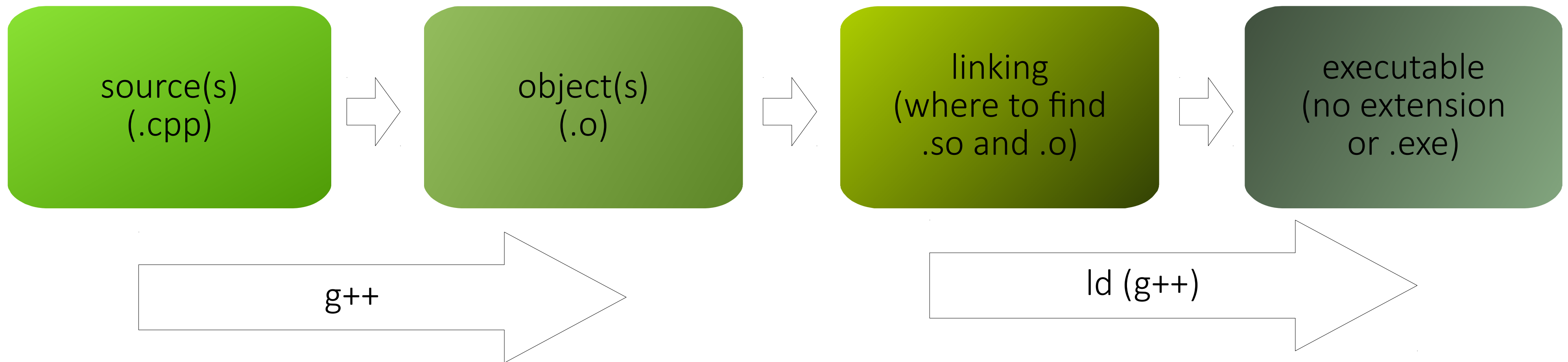
4)Link

5)Load

6)Execute

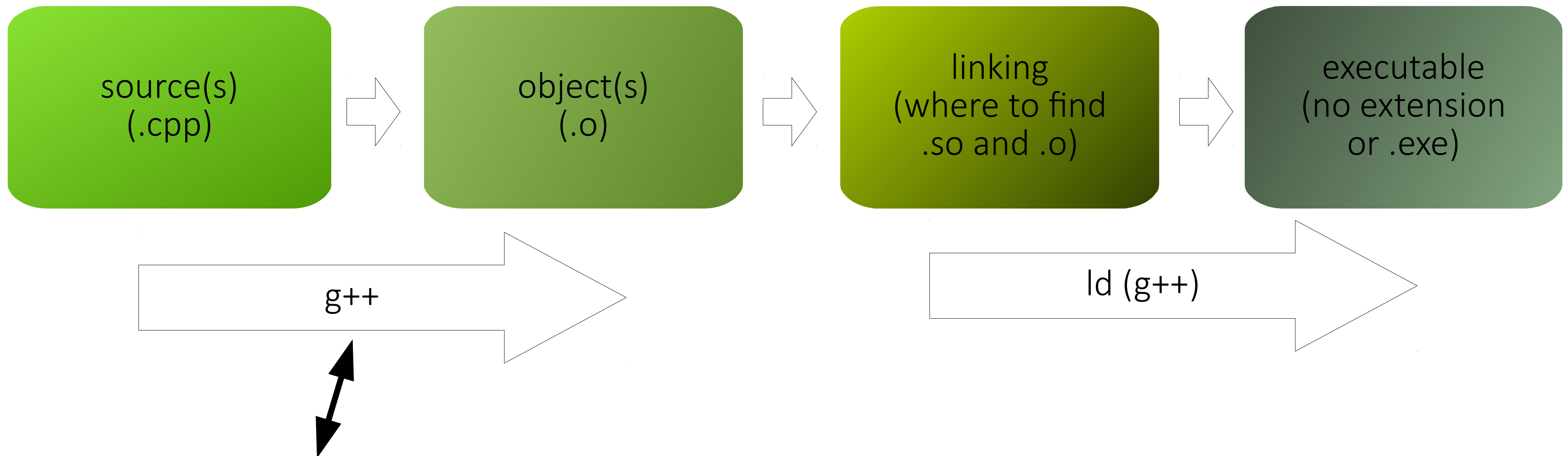


Executables, compiling and running



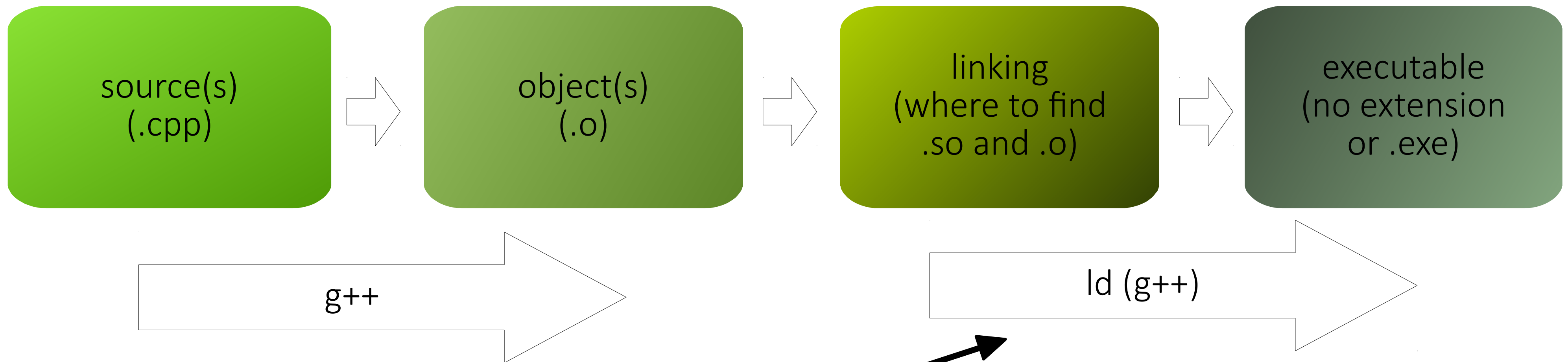

```
|prompt> g++ -c hello.cpp
|prompt> g++ -o hello hello.o
|prompt> ls -lrt hello*
-rw-r--r-- 1 ramona ramona  320 feb 14 17:44 hello.cpp
-rw-r--r-- 1 ramona ramona 2,5K feb 14 17:45 hello.o
-rwxr-xr-x 1 ramona ramona 8,8K feb 14 17:45 hello
```

Executables, compiling and running



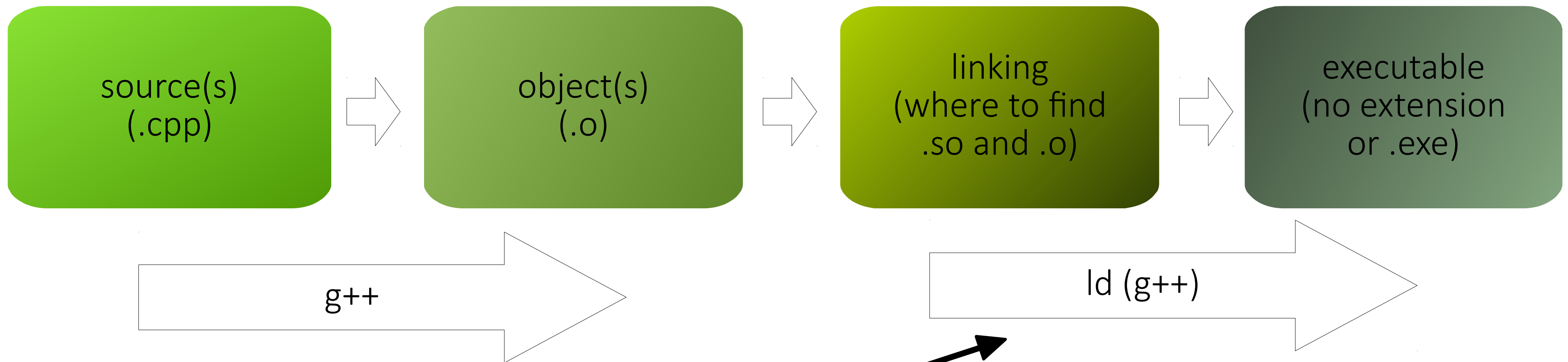

```
|prompt> g++ -c hello.cpp
|prompt> g++ -o hello hello.o
|prompt> ls -lrt hello*
-rw-r--r-- 1 ramona ramona  320 feb 14 17:44 hello.cpp
-rw-r--r-- 1 ramona ramona 2,5K feb 14 17:45 hello.o
-rwxr-xr-x 1 ramona ramona 8,8K feb 14 17:45 hello
```

Executables, compiling and running




```
|prompt> g++ -c hello.cpp
|prompt> g++ -o hello hello.o
|prompt> ls -lrt hello*
-rw-r--r-- 1 ramona ramona  320 feb 14 17:44 hello.cpp
-rw-r--r-- 1 ramona ramona 2,5K feb 14 17:45 hello.o
-rwxr-xr-x 1 ramona ramona 8,8K feb 14 17:45 hello
```

Executables, compiling and running




```
|prompt> g++ -c hello.cpp
|prompt> g++ -o hello hello.o
|prompt> ls -lrt hello*
-rw-r--r-- 1 ramona ramona  320 feb 14 17:44 hello.cpp
-rw-r--r-- 1 ramona ramona 2,5K feb 14 17:45 hello.o
-rwxr-xr-x 1 ramona ramona 8,8K feb 14 17:45 hello
|prompt> g++ -o hello hello.cpp
```

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8          std::cout << "Hello world!\n";
9
10        return 0;   // indicate that program ended successfully
11
12     } // end function main
```

hello.cpp

The simplest C++ program: printing a line of text

```
1      // A first program in C++
2      // Filename: hello.cpp
3      #include <iostream>
4
5      // function main begins program execution
6      int main()
7      {
8         std::cout << "Hello world!\n";
9
10        return 0;   // indicate that program ended successfully
11
12    } // end function main
```

hello.cpp

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program exec
6       int main()
7       {
8          std::cout << "Hello world!\n";
9
10        return 0;    // indicate that program ended successfully
11
12     } // end function main
```

Preprocessor directive to include input/output stream header file **`<iostream>`**

hello.cpp

The simplest C++ program: printing a line of text

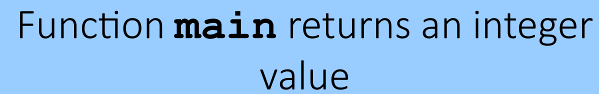
```
1      // A first program in C++
2      // Filename: hello.cpp
3      #include <iostream>
4
5      // function main begins program execution
6      int main()
7      {
8         std::cout << "Hello world!\n";
9
10        return 0;   // indicate that program ended successfully
11
12    } // end function main
```

Function **main** appears exactly once in every C++ program

hello.cpp

The simplest C++ program: printing a line of text

```
1      // A first program in C++
2      // Filename: hello.cpp
3      #include <iostream>
4
5      // function main begins program execution
6      int main()
7      {
8         std::cout << "Hello world!\n";
9
10        return 0;    // indicate that program ended successfully
11
12     } // end function main
```



hello.cpp

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8          std::cout << "Hello world!\n";
9
10        return 0;    // indicate that program ended successfully
11
12     } // end function main
```

hello.cpp

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8          std::cout << "Hello world!\n";
9
10        return 0;    // inc
11
12     } // end function main
```

Corresponding right brace **}** ends function body

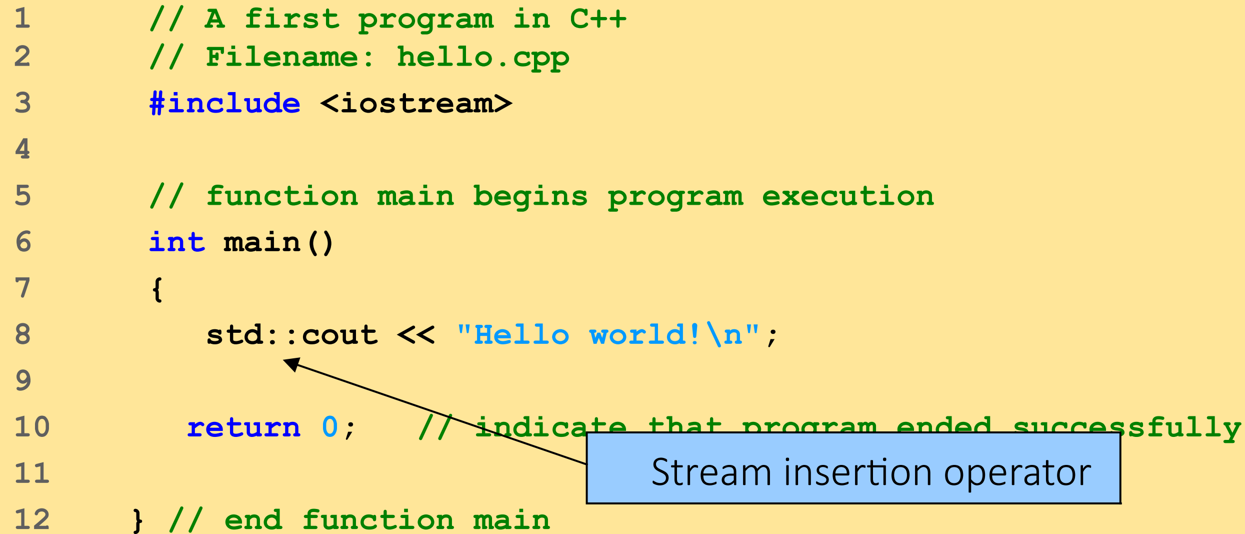
hello.cpp

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8          std::cout << "Hello world!\n";
9
10        return 0;   // indicate that program ended successfully
11
12     } // end function main
```

hello.cpp

The simplest C++ program: printing a line of text



hello.cpp

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8          std::cout << "Hello world!\n";
9
10        return 0;    // indicate that program ended successfully
11
12     } // end function main
```

Keyword **return** is one of several means to exit function; value **0** indicates program terminated successfully

hello.cpp

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8          std::cout << "Hello world!\n";
9
10        return 0;    // indicate that program ended successfully
11
12     } // end function main
```

Compile it with:

```
|prompt> g++ hello.cpp -o hello
```

Execute the program with:

```
|prompt> ./hello
```

hello.cpp

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8          std::cout << "Hello world!\n";
9
10        return 0;    // indicate that program ended successfully
11
12     } // end function main
```

Compile it with:

```
|prompt> g++ hello.cpp -o hello
```

Execute the program with:

```
|prompt> ./hello
```

```
Hello world!
```

hello.cpp

The simplest C++ program: printing a line of text

- Standard output stream object
  - **`std::cout`**
  - “Connected” to screen
  - **`<<`**
    - Stream insertion operator
    - Value to right (right operand) inserted into output stream
- Namespace
  - **`std::`** specifies using name that belongs to “namespace” **`std`**
  - **`std::`** removed through use of **`using`** statements
- Escape characters
  - **`\`**
  - Indicates “special” character output

The simplest C++ program: printing a line of text

Escape Sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
****	Backslash. Used to print a backslash character.
\"	Double quote. Used to print a double quote character.

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8          std::cout << "Hello";
9          std::cout << "world!\n";
10        return 0;   // indicate that program ended successfully
11
12    } // end function main
```

Multiple stream insertion statements produce one line of output

Compile it with:

|prompt> g++ hello.cpp -o hello

Execute the program with:

|prompt> ./hello

```
Hello world!
```

The simplest C++ program: printing a line of text

```
1       // A first program in C++
2       // Filename: hello.cpp
3       #include <iostream>
4
5       // function main begins program ex
6       int main()
7       {
8          std::cout << "Hello\n \nworld!\n";
9
10        return 0;   // indicate that program ended successfully
11
12     } // end function main
```

Using newline characters to print on multiple lines.

Compile it with:

|prompt> g++ hello.cpp -o hello

Execute the program with:

|prompt> ./hello

```
Hello

world!
```

Proper use of comments- what

Typically, comments should be used for three things. At the library, program, or function level, comments should be used to describe what the library, program, or function, does. For example:

```
// This program calculate the student's final grade
// based on his test and homework scores.
// This function uses newton's method to
// approximate the root of a given equation.
// The following lines generate a random item based
// on rarity, level, and a weight factor.
```

All of these comments give the reader a good idea of what the program is trying to accomplish without having to look at the actual code. The user (possibly someone else, or you if you're trying to reuse code you've already written in the future) can tell at a glance whether the code is relevant to what he or she is trying to accomplish. This is particularly important when working as part of a team, where not everybody will be familiar with all of the code.

Proper use of comments- how

Second, within the library, program, or function described above, comments should be used to describe how the code is going to accomplish it’s goal.

```
/* To calculate the final grade, we sum all the
weighted midterm and homework scores and then divide
by the number of scores to assign a percentage. This
percentage is used to calculate a letter grade. */
```

```
// To generate a random item, we're going to do the following:
//1) Put all of the items of the desired rarity on a list
//2) Calculate a probability for each item based on level and weight factor
//3) Choose a random number
//4) Figure out which item that random number corresponds to
//5) Return the appropriate item
```

These comments give the user an idea of how the code is going to accomplish it’s goal without going into too much detail.

Proper use of comments- why

At the statement level, comments should be used to describe why the code is doing something. A bad statement comment explains what the code is doing. If you ever write code that is so complex that needs a comment to explain what a statement is doing, you probably need to rewrite your code, not comment it.

- Bad comment:

```
// Set sight range to 0
sight = 0;
```

(yes, we already can see that sight is being set to 0 by looking at the statement)

- Good comment:

```
// The player just drank a potion of blindness and can not see anything
```

sight = 0; (now we know WHY the player's sight is being set to 0)

Proper use of comments

- Bad comment:

```
// Calculate the cost of the items
cost = items / 2 * storePrice;
```

(yes, we can see that this is a cost calculation, but why is items divided by 2?)

- Good comment:

```
// We need to divide items by 2 here because they are bought in pairs
cost = items / 2 * storePrice;
```

(now we know!)

Proper use of comments

Programmers often have to make a tough decision between solving a problem one way, or solving it another way.

Comments are a great way to remind yourself (or tell somebody else) the reason you made a one decision instead of another.

Good comments:

```
// We decided to use a linked list instead of an array because
// arrays do insertion too slowly.
// We're going to use newton's method to find the root of a
// number because there is no deterministic way to solve these
// equations.
```

Proper use of comments

- Finally, comment should be written in a way that makes sense to someone who has no idea what the code does. It is often the case that a programmer will say “It’s obvious what this does! There’s no way I’ll forget about this”. Guess what? It’s not obvious, and you will be amazed how quickly you forget. :)
- You (or someone else) will thank you later for writing down the what, how, and why of your code in human language.
- Reading individual lines of code is easy. Understanding what goal they are meant to accomplish is not.

(http://www.learncpp.com/cpp-tutorial/12-comments/)

- To summarize:
  - At the library, program, or function level, describe **what**
  - Inside the library, program, or function, describe **how**
  - At the statement level, describe **why**

Esercitazione 1 (A)

1) Write a program which print a greeting (`cheers.cpp`)

```
Hi!
```

Variables

- Variables: Location in memory where value can be stored
  - Common data types:
    - **`int`** : integer numbers
    - **`char`** : characters
    - **`double`** : floating point numbers
  - Declare variables with <u>name</u> and <u>data</u> type **before** use
    - **`int integer1;`**
    - **`int integer2;`**
    - **`int sum;`**
  - Can declare several variables of same type in one declaration Comma-separated list: **`int integer1, integer2, sum;`**

Variables

- Variables
  - Variable names
  - Valid identifier
    - Series of characters (letters, digits, underscores)
    - Cannot begin with digit
    - Case sensitive

Here is the complete list of fundamental types in C++:

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

Two more operators

- Input stream object

  >> (stream extraction operator)

  - Used with **`std::cin`**
  - Waits for user to input value, then press *Enter* (Return) key
  - Stores value in variable to right of operator
  - Converts value to variable data type

  = (assignment operator)

  - Assigns value to variable
  - Binary operator (two operands)
  - Example:

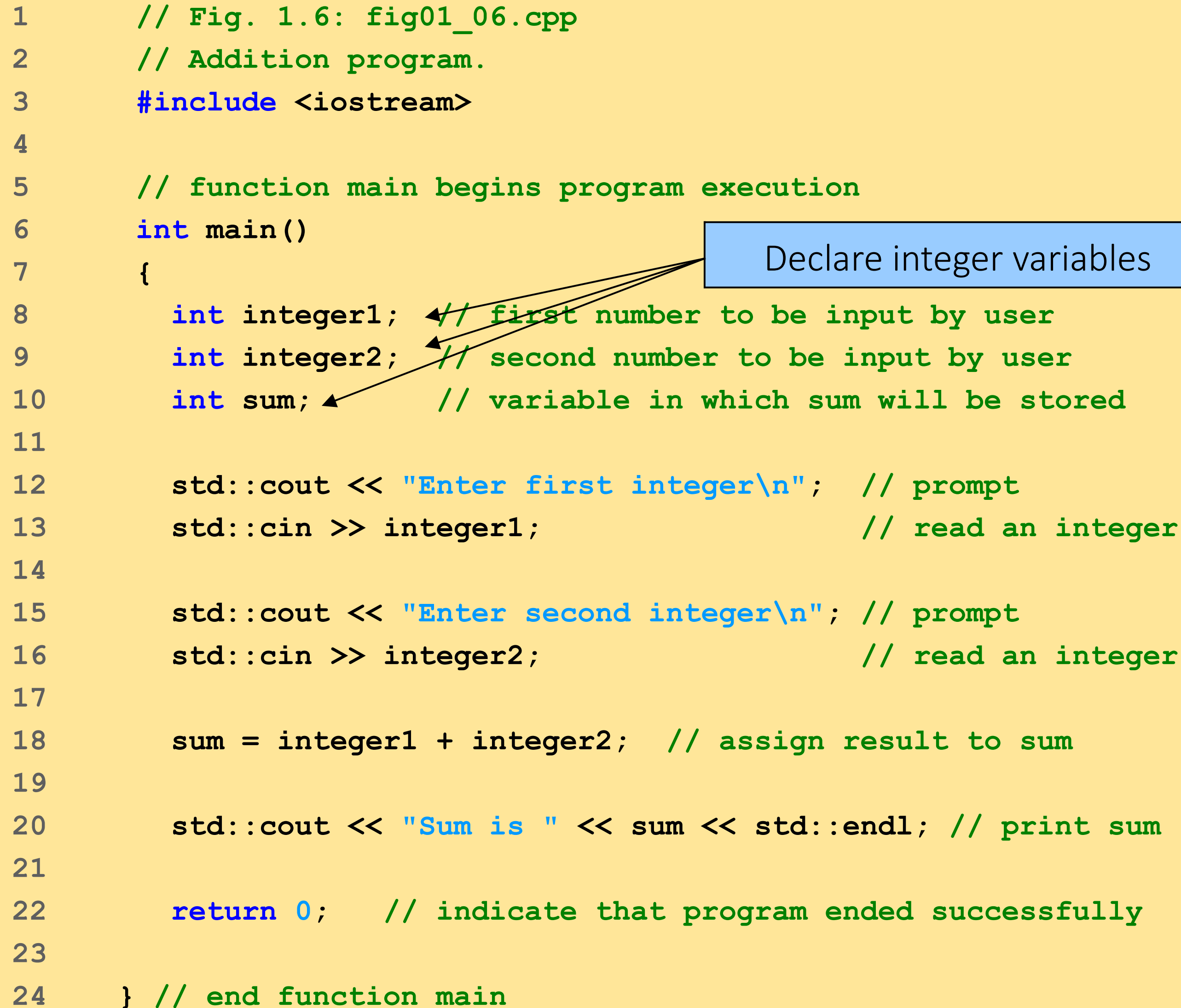
```
sum = variable1 + variable2;
```

Another Simple Program: Adding Two Integers

```
1      // Fig. 1.6: fig01_06.cpp
2      // Addition program.
3      #include <iostream>
4
5      // function main begins program execution
6      int main()
7      {
8        int integer1;  // first number to be input by user
9        int integer2;  // second number to be input by user
10       int sum;       // variable in which sum will be stored
11
12       std::cout << "Enter first integer\n";  // prompt
13       std::cin >> integer1;                  // read an integer
14
15       std::cout << "Enter second integer\n"; // prompt
16       std::cin >> integer2;                  // read an integer
17
18       sum = integer1 + integer2;  // assign result to sum
19
20       std::cout << "Sum is " << sum << std::endl; // print sum
21
22       return 0;   // indicate that program ended successfully
23
24    } // end function main
```

sumInteger.cpp

Another Simple Program: Adding Two Integers



sumInteger.cpp

Another Simple Program: Adding Two Integers

```
1       // Fig. 1.6: fig01_06.cpp
2       // Addition program.
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8         int integer1;  // first number to be input by user
9         int integer2;  // second num
10        int sum;       // variable i
11
12        std::cout << "Enter first in
13        std::cin >> integer1;                  // read an integer
14
15        std::cout << "Enter second integer\n"; // prompt
16        std::cin >> integer2;                  // read an integer
17
18        sum = integer1 + integer2;  // assign result to sum
19
20        std::cout << "Sum is " << sum << std::endl; // print sum
21
22        return 0;   // indicate that program ended successfully
23
24     } // end function main
```

Use stream extraction operator with standard input stream to obtain user input

sumInteger.cpp

Another Simple Program: Adding Two Integers

```
1      // Fig. 1.6: fig01_06.cpp
2      // Addition program.
3      #include <iostream>
4
5      // function main begins program execution
6      int main()
7      {
8        int integer1;  // first number to be input by user
9        int integer2;  // second number to be input by user
10       int sum;       // variable in which sum will be stored
11
12       std::cout << "Enter first integer\n";  // prompt
13       std::cin >> integer1;                  // read an integer
14
15       std::cout << "Enter second integer\n"; // prompt
16       std::cin >> integer2;                  // read an integer
17
18       sum = integer1 + integer2;  // assign result to sum
19
20       std::cout << "Sum is " << sum << std::endl; // print sum
21
22       return 0;   // indicate that program ended successfully
23
24    } // end function main
```

Stream manipulator **std::endl** outputs a newline, then "flushes output buffer"

sumInteger.cpp

Another Simple Program: Adding Two Integers

```
1     // Fig. 1.6: fig01_06.cpp
2     // Addition program.
3     #include <iostream>
4
5     // function main begins program execution
6     int main()
7     {
8       int integer1;  // first number to be input by user
9       int integer2;  // second number to be input by user
10      int sum;       // variable in which sum will be stored
11
12      std::cout << "Enter first integer\n";  // prompt
13      std::cin >> integer1;                   // read an integer
14
15      std::cout << "Enter second integer\n"; // prompt
16      std::cin >> integer2;                   // read an integer
17
18      sum = integer1 + integer2;  // assign result to sum
19
20      std::cout << "Sum is " << sum << std::endl; // print sum
21
22      return 0;   // indicate that program ended successfully
23
24   } // end function main
```

Concatenating, chaining or cascading stream insertion operations

Another Simple Program: Adding Two Integers

```
1       // Fig. 1.6: fig01_06.cpp
2       // Addition program.
3       #include <iostream>
4
5       // function main begins program execution
6       int main()
7       {
8         int integer1;  // first number to be input by user
9         int integer2;  // second number to be input by user
10        int sum;       // variable in which sum will be stored
11
12        std::cout << "Enter first integer\n";  // prompt
13        std::cin >> integer1;                  // read an integer
14
15        std::cout << "Enter second i
16        std::cin >> integer2;
17
18        sum = integer1 + integer2;
19
20        std::cout << "Sum is " << sum << std::endl; // print sum
21
22        return 0;   // indicate that program ended successfully
23
24    } // end function main
```

Calculations can be performed in output statements: alternative for lines 18 and 20:

```
std::cout << "Sum is " << integer1 + integer2 << std::endl;
```

sumInteger.cpp

Another Simple Program: Adding Two Integers

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

sumInteger.cpp

Memory Concepts

- Variable names
  - Correspond to actual locations in computer's memory
  - Every variable has name, type, size and value
  - When new value placed into variable, overwrites previous value
  - Reading variables from memory nondestructive

```
std::cin >> integer1;
```

Assume user entered 45

integer1	45

```
std::cin >> integer2;
```

Assume user entered 72

integer1	45
integer2	72

```
sum = integer1 + integer2;
```

integer1	45
integer2	72
sum	117

Arithmetic

- Arithmetic calculations
  - `*` Multiplication
  - `/` Division
    - Remember: Integer division truncates remainder, so **`7 / 5`** evaluates to 1
  - `%` Modulus operator returns remainder
    - **`7 % 5`** evaluates to 2
- Rules of operator precedence:
  - Operators in parentheses evaluated first
    - Nested/embedded parentheses
    - Operators in innermost pair first
  - Multiplication, division, modulus applied next
    - Operators applied from left to right
  - Addition, subtraction applied last
    - Operators applied from left to right

Esercitazione 1 (B)

1) Write a program which asks your name and prints a greeting (`name.cpp`)

2) Write a program which asks for the density and the radius of a sphere and prints out its volume and mass (`sphere.cpp`)

Decision Making: Equality and Relational Operators

- **`if`** structure
  - Make decision based on truth or falsity of condition;
  - If condition met, body executed
  - Else, body not executed
- Equality and relational operators
  - Equality operators
    - Same level of precedence
  - Relational operators
    - Same level of precedence
  - Associate left to right

Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
$>$	**>**	**x > y**	**x** is greater than **y**
$<$	**<**	**x < y**	**x** is less than **y**
$\geq$	**>=**	**x >= y**	**x** is greater than or equal to **y**
$\leq$	**<=**	**x <= y**	**x** is less than or equal to **y**
Equality operators			
$=$	**==**	**x == y**	**x** is equal to **y**
$\neq$	**!=**	**x != y**	**x** is not equal to **y**

`using` statements

- **`using`** statements
  - Eliminate use of **`std::`** prefix
  - Write **`cout`** instead of **`std::cout`**

`if` Selection Structure

- Selection structure: *Choose among alternative courses of action*

  Pseudocode example:

  If student's grade is greater than or equal to 60 Print "Passed"

- If the condition is **`true`** : print statement executed, program continues to next statement
- If the condition is **`false`** :print statement ignored, program continues
- Indenting makes programs easier to read (C++ ignores whitespace characters (tabs, spaces, etc.))
  - Example:

```
if ( grade >= 60 )
   cout << "Passed";
```

Decision Making: Equality and Relational Operators

```
1       // Fig. 1.14: fig01_14.cpp
2       // Using if statements, relational
3       // operators, and equality operators.
4       #include <iostream>
5
6       using std::cout;  // program uses cout
7       using std::cin;   // program uses cin
8       using std::endl;  // program uses endl
9
10     // function main begins program execution
11     int main()
12     {
13        int num1;  // first number to be read from user
14        int num2;  // second number to be read from user
15
16        cout << "Enter two integers, and I will tell you\n"
17             << "the relationships they satisfy: ";
18        cin >> num1 >> num2;   // read two integers
19
20        if ( num1 == num2 )
21           cout << num1 << " is equal to " << num2 << endl;
22
23        if ( num1 != num2 )
24           cout << num1 << " is not equal to " << num2 << endl;
25
```

Decision Making: Equality and Relational Operators

```
1      // Fig. 1.14: fig01_14.cpp
2      // Using if statements, relational
3      // operators, and equality operators.
4      #include <iostream>
5
6      using std::cout;  // program uses cout
7      using std::cin;   // program uses cin
8      using std::endl;  // program uses endl
9
10    // function main begins program execution
11    int main()
12    {
13       int num1;  // first number to be read from user
14       int num2;  // second number to be read from user
15
16       cout << "Enter two integers, and I will tell you\n"
17            << "the relationships they satisfy: ";
18       cin >> num1 >> num2;   // read two integers
19
20       if ( num1 == num2 )
21          cout << num1 << " is equal to " << num2 << endl;
22
23       if ( num1 != num2 )
24          cout << num1 << " is not equal to " << num2 << endl;
25
```

using statements eliminate need for **std::** prefix

firstIf.cpp

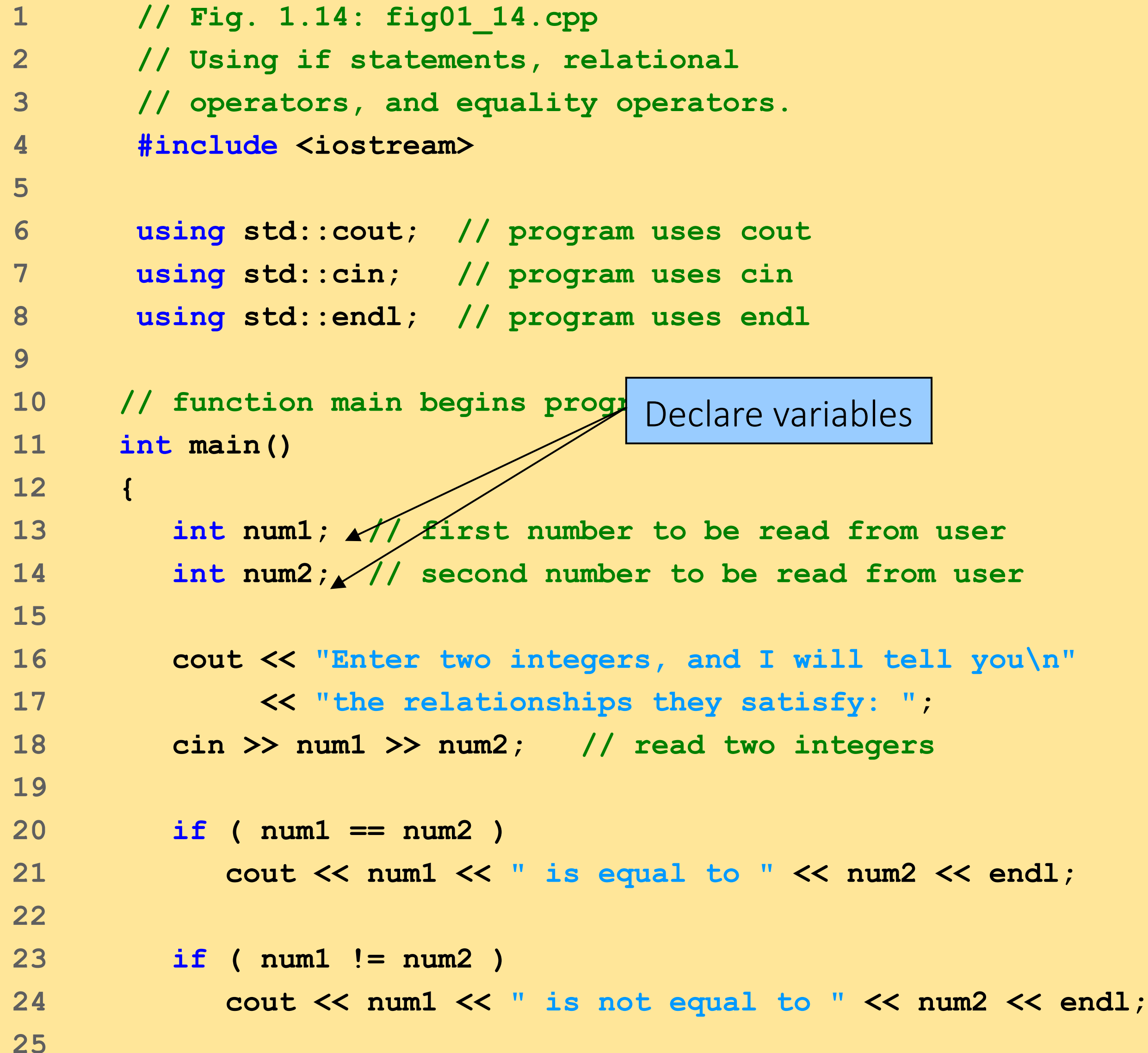
Decision Making: Equality and Relational Operators

```
1       // Fig. 1.14: fig01_14.cpp
2       // Using if statements, relational
3       // operators, and equality operators.
4       #include <iostream>
5
6       using std::cout;  // program uses cout
7       using std::cin;   // program uses cin
8       using std::endl;  // program uses endl
9
10     // function main begins program execution
11     int main()
12     {
13        int num1;  // first num
14        int num2;  // second nu
15
16        cout << "Enter two integers, and I will tell you\n"
17             << "the relationships they satisfy: ";
18        cin >> num1 >> num2;   // read two integers
19
20        if ( num1 == num2 )
21           cout << num1 << " is equal to " << num2 << endl;
22
23        if ( num1 != num2 )
24           cout << num1 << " is not equal to " << num2 << endl;
25
```

Can write **cout** and **cin** without **std::** prefix

firstIf.cpp

Decision Making: Equality and Relational Operators



```
1      // Fig. 1.14: fig01_14.cpp
2      // Using if statements, relational
3      // operators, and equality operators.
4      #include <iostream>
5
6      using std::cout;  // program uses cout
7      using std::cin;   // program uses cin
8      using std::endl;  // program uses endl
9
10    // function main begins program execution
11    int main()
12    {
13       int num1; // first number to be read from user
14       int num2; // second number to be read from user
15
16       cout << "Enter two integers, and I will tell you\n"
17            << "the relationships they satisfy: ";
18       cin >> num1 >> num2;   // read two integers
19
20       if ( num1 == num2 )
21          cout << num1 << " is equal to " << num2 << endl;
22
23       if ( num1 != num2 )
24          cout << num1 << " is not equal to " << num2 << endl;
25
```


firstIf.cpp

Decision Making: Equality and Relational Operators

```
1      // Fig. 1.14: fig01_14.cpp
2      // Using if statements, relational
3      // operators, and equality operators.
4      #include <iostream>
5
6      using std::cout;  // program uses cout
7      using std::cin;   // program uses cin
8      using std::endl;  // program uses endl
9
10    // function main begins program execution
11    int main()
12    {
13       int num1;  // first number to be read from user
14       int num2;  // second number to be read from user
15
16       cout << "Enter two integers, and I will tell you\n"
17            << "the relationships they
18       cin >> num1 >> num2;   // read
19
20       if ( num1 == num2 )
21          cout << num1 << " is equal to " << num2 << endl;
22
23       if ( num1 != num2 )
24          cout << num1 << " is not equal to " << num2 << endl;
25
```

if structure compares values of **num1** and **num2** to test for equality

firstIf.cpp

Decision Making: Equality and Relational Operators

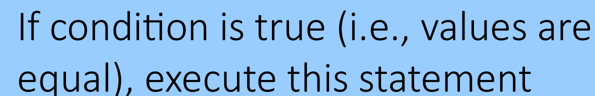
```
1      // Fig. 1.14: fig01_14.cpp
2      // Using if statements, relational
3      // operators, and equality operators.
4      #include <iostream>
5
6      using std::cout;  // program uses cout
7      using std::cin;   // program uses cin
8      using std::endl;  // program uses endl
9
10    // function main begins program execution
11    int main()
12    {
13       int num1;  // first number to be read from user
14       int num2;  // second number to be read from user
15
16       cout << "Enter two integers, and I will tell you\n"
17            << "the relationships they satisfy: ";
18       cin >> num1 >> num2;   // read two integers
19
20       if ( num1 == num2 )
21          cout << num1 << " is equal to
22
23       if ( num1 != num2 )
24          cout << num1 << " is not equal to " << num2 << endl;
25
```

if structure compares values of **num1** and **num2** to test for inequality

firstIf.cpp

Decision Making: Equality and Relational Operators

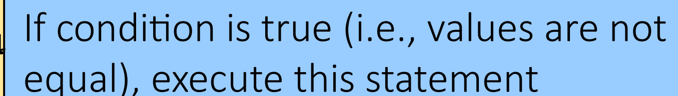
```
1       // Fig. 1.14: fig01_14.cpp
2       // Using if statements, relational
3       // operators, and equality operators.
4       #include <iostream>
5
6       using std::cout;  // program uses cout
7       using std::cin;   // program uses cin
8       using std::endl;  // program uses endl
9
10     // function main begins program execution
11     int main()
12     {
13        int num1;  // first number to be read from user
14        int num2;  // second number to be read from user
15
16        cout << "Enter two integers, and I will tell you\n"
17             << "the relationships they satisfy:
18        cin >> num1 >> num2;   // read two intege
19
20        if ( num1 == num2 )
21           cout << num1 << " is equal to " << num2 << endl;
22
23        if ( num1 != num2 )
24           cout << num1 << " is not equal to " << num2 << endl;
25
```



firstIf.cpp

Decision Making: Equality and Relational Operators

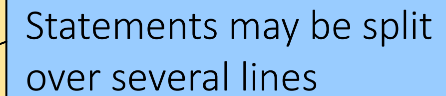
```
1       // Fig. 1.14: fig01_14.cpp
2       // Using if statements, relational
3       // operators, and equality operators.
4       #include <iostream>
5
6       using std::cout;  // program uses cout
7       using std::cin;   // program uses cin
8       using std::endl;  // program uses endl
9
10     // function main begins program execution
11     int main()
12     {
13        int num1;  // first number to be read from user
14        int num2;  // second number to be read from user
15
16        cout << "Enter two integers, and I will tell you\n"
17             << "the relationships they satisfy: ";
18        cin >> num1 >> num2;   // read two integers
19
20        if ( num1 == num2 )
21           cout << num1 << " is equal to " << num2 << endl;
22
23        if ( num1 != num2 )
24           cout << num1 << " is not equal to " << num2 << endl;
25
```



firstIf.cpp

Decision Making: Equality and Relational Operators

```
26       if ( num1 < num2 )
27          cout << num1 << " is less than " << num2 << endl;
28
29       if ( num1 > num2 )
30          cout << num1 << " is greater than " << num2 << endl;
31
32       if ( num1 <= num2 )
33          cout << num1 << " is less than or equal to "
34               << num2 << endl;
35
36       if ( num1 >= num2 )
37          cout << num1 << " is greater than or equal to "
38               << num2 << endl;
39
40       return 0;   // indicate that program ended successfully
41
42    } // end function main
```



firstIf.cpp

Decision Making: Equality and Relational Operators

```
26        if ( num1 < num2 )
27           cout << num1 << " is less than " << num2 << endl;
28
29        if ( num1 > num2 )
30           cout << num1 << " is greater than " << num2 << endl;
31
32        if ( num1 <= num2 )
33           cout << num1 << " is less than or equal to "
34                << num2 << endl;
35
36        if ( num1 >= num2 )
37           cout << num1 << " is greater than or equal to "
38                << num2 << endl;
39
40        return 0;   // indicate that program ended successfully
41
42     } // end function main
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

`if/else` Selection Structure

`if` : Performs action if condition true

`if/else`: Different actions if conditions true or false

Example:

```
if ( grade >= 60 )
   cout << "Passed";
else
   cout << "Failed";
```

- Compound statement : Set of statements within a pair of braces

```
if ( grade >= 60 )
   cout << "Passed.\n";
else {
   cout << "Failed.\n";
   cout << "You must take this course again.\n";
}
```

Without braces, `cout << "Failed.\nYou must take this course again.\n";`

always executed Block Set of statements **within braces**

`while` Repetition Structure

- Repetition structure: Action repeated while some condition remains true
- **`while`** loop repeated until condition becomes false

  Example

```
int product = 2;
while ( product <= 1000 )
   product = 2 * product;
```

while Repetition Structure

```
1      // Fig. 2.7: fig02_07.cpp
2      // Class average program with counter-controlled repetition.
3      #include <iostream>
4
5      using std::cout;
6      using std::cin;
7      using std::endl;
8
9      // function main begins program execution
10    int main()
11    {
12       int total;        // sum of grades input by user
13       int gradeCounter; // number of grade to be entered next
14       int grade;        // grade value
15       int average;      // average of grades
16
17       // initialization phase
18       total = 0;          // initialize total
19       gradeCounter = 1;   // initialize loop counter
20
```

firstWhile.cpp

while Repetition Structure

```
21        // processing phase
22        while ( gradeCounter <= 10 ) {        // loop 10 times
23           cout << "Enter grade: ";           // prompt for input
24           cin >> grade;                      // read grade from user
25           total = total + grade;             // add grade to total
26           gradeCounter = gradeCounter + 1;   // increment counter
27        }
28
29        // termination phase
30        average = total / 10;                 // integer division
31
32        // display result
33        cout << "Class averag
34
35        return 0;    // indica
36
37     } // end function main
```

The counter gets incremented each time the loop executes. Eventually, the counter causes the loop to end.

firstWhile.cpp

`while` Repetition Structure

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

Formulating Algorithms (Sentinel-Controlled Repetition)

- Suppose problem becomes:
- *Develop a class-averaging program that will process an arbitrary number of grades each time the program is run*

  → Unknown number of students
- How will program know when to end? Sentinel value, indicates “end of data entry”
- Loop ends when sentinel input
- Sentinel chosen so it cannot be confused with regular input (e.g.-1 in this example)

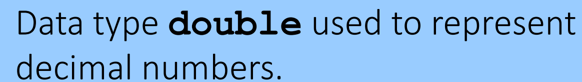
Sentinel-Controlled Repetition

```
1      // Fig. 2.9: fig02_09.cpp
2      // Class average program with sentinel-controlled repetition.
3      #include <iostream>
4
5      using std::cout;
6      using std::cin;
7      using std::endl;
8      using std::fixed;
9
10    #include <iomanip>        // parameterized stream manipulators
11
12    using std::setprecision;  // sets numeric output precision
13
14    // function main begins program execution
15    int main()
16    {
17       int total;          // sum of grades
18       int gradeCounter;   // number of grades entered
19       int grade;          // grade value
20
21       double average;     // number with decimal point for average
22
23       // initialization phase
24       total = 0;          // initialize total
25       gradeCounter = 0;   // initialize loop counter
```

sentinel1.cpp

Sentinel-Controlled Repetition

```
1      // Fig. 2.9: fig02_09.cpp
2      // Class average program with sentinel-controlled repetition.
3      #include <iostream>
4
5      using std::cout;
6      using std::cin;
7      using std::endl;
8      using std::fixed;
9
10    #include <iomanip>        // parameterized stream manipulators
11
12    using std::setprecision;  // sets numeric output precision
13
14    // function main begins program execution
15    int main()
16    {
17       int total;          // sum of grades
18       int gradeCounter;   // number of grades entered
19       int grade;          // grade value
20
21       double average;     // number with decimal point for average
22
23       // initialization phase
24       total = 0;          // initialize total
25       gradeCounter = 0;   // initialize loop counter
```



sentinel1.cpp

Sentinel-Controlled Repetition

```
26
27        // processing phase
28        // get first grade from user
29        cout << "Enter grade, -1 to end: ";  // prompt for input
30        cin >> grade;                        // read grade from user
31
32        // loop until sentinel value read from user
33        while ( grade != -1 ) {
34           total = total + grade;            // add grade to total
35           gradeCounter = gradeCounter + 1;  // increment counter
36
37           cout << "Enter grade, -1 to end: ";  // prompt for input
38           cin >> grade;                        // read next grade
39
40        } // end while
41
42        // termination phase
43        // if user entered at least one grade ...
44        if ( gradeCounter != 0 ) {
45
46           // calculate average of all grades entered
47           average = static_cast< double >( total ) / gradeCounter;
48
```

sentinel1.cpp

Sentinel-Controlled Repetition

```
26
27        // processing phase
28        // get first grade from user
29        cout << "Enter grade, -1 to end: ";  // prompt for input
30        cin >> grade;                         // read grade from user
31
32        // loop until sentinel value read from user
33        while ( grade != -1 ) {
34           total = total + grade;
35           gradeCounter = gradeCount
36
37           cout << "Enter grade, -1
38           cin >> grade;
39
40        } // end while
41
42        // termination phase
43        // if user entered at least one grade ...
44        if ( gradeCounter != 0 ) {
45
46           // calculate average of all grades entered
47           average = static_cast< double >( total ) / gradeCounter;
48
```

`static_cast<double>()` treats **`total`** as a **`double`** temporarily (casting).

Required because dividing two integers truncates the remainder.

`gradeCounter` is an **`int`**, but it gets *promoted* to **`double`**.

sentinel1.cpp

Sentinel-Controlled Repetition

```
49          // display average with two digits of precision
50          cout << "Class average is " << setprecision( 2 )
51               << fixed << average << endl;
52
53       } // end if part of if/else
54
55       else // if no grades were entered, output appropriate message
56          cout << "No grades were entered" << endl;
57
58       return 0;   // indicate program ended successfully
59
60    } // end function main
```

sentinel1.cpp

Sentinel-Controlled Repetition

```
49          // display average with two digits of precision
50          cout << "Class average is " << setprecision( 2 )
51               << fixed << average << endl;
52
53       } // end if part of if/else
54
55       else // if no grades were entered, output appropriate message
56          cout << "No grades were entered" << endl;
57
58       return 0;   // indicate program ended successfully
59
60    } // end function main
```

`setprecision(2)` prints two digits past decimal point (rounded to fit precision).

Programs that use this must include **`<iomanip>`**

sentinel1.cpp

Sentinel-Controlled Repetition

```
49         // display average with two digits of precision
50         cout << "Class average is " << setprecision( 2 )
51              << fixed << average << endl;
52
53      } // end if part of if/else
54
55      else // if no grades were entered, output appropriate message
56         cout << "No grades were entered" << endl;
57
58      return 0;   // indicate program ended successfully
59
60   } // end function main
```

`fixed` forces output to print in fixed point format (not scientific notation). Also, forces trailing zeros and decimal point to print.

Include **`<iostream>`**

sentinel1.cpp

Sentinel-Controlled Repetition

```
49          // display average with two digits of precision
50          cout << "Class average is " << setprecision( 2 )
51               << fixed << average << endl;
52
53       } // end if part of if/else
54
55       else // if no grades were entered, output appropriate message
56          cout << "No grades were entered" << endl;
57
58       return 0;   // indicate program ended successfully
59
60    } // end function main
```

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

sentinel1.cpp

Nested Control Structures

- Problem statement
  - *A college has a list of test results (1 = pass, 2 = fail) for 10 students. Write a program that analyzes the results. If more than 8 students pass, print "Raise Tuition".*
- Notice that
  - the program processes 10 results (Fixed number, use counter-controlled loop)
  - Two counters can be used: one counts number that passed another counts number that fail
  - Each test result is 1 or 2
  - If not 1, assume 2

Nested Control Structures

```
1      // Fig. 2.11: fig02_11.cpp
2      // Analysis of examination results.
3      #include <iostream>
4
5      using std::cout;
6      using std::cin;
7      using std::endl;
8
9      // function main begins program execution
10    int main()
11    {
12       // initialize variables in declarations
13       int passes = 0;            // number of passes
14       int failures = 0;          // number of failures
15       int studentCounter = 1;    // student counter
16       int result;                // one exam result
17
18       // process 10 students using counter-controlled loop
19       while ( studentCounter <= 10 ) {
20
21          // prompt user for input and obtain value from user
22          cout << "Enter result (1 = pass, 2 = fail): ";
23          cin >> result;
24
```

nested.cpp

Nested Control Structures

```
25          // if result 1, increment passes; if/else nested in while
26          if ( result == 1 )          // if/else nested in while
27             passes = passes + 1;
28
29          else  // if result not 1, increment failures
30             failures = failures + 1;
31
32          // increment studentCounter so loop eventually terminates
33          studentCounter = studentCounter + 1;
34
35       } // end while
36
37       // termination phase; display number of passes and failures
38       cout << "Passed " << passes << endl;
39       cout << "Failed " << failures << endl;
40
41       // if more than eight students passed, print "raise tuition"
42       if ( passes > 8 )
43          cout << "Raise tuition " << endl;
44
45       return 0;   // successful termination
46
47    } // end function main
```

nested.cpp

Nested Control Structures

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Passed 6
Failed 4

Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed 9
Failed 1
Raise tuition
```

nested.cpp

Assignment expression abbreviations

- Addition assignment operator

  `c = c + 3;` abbreviated to

  `c += 3;`

- Statements of the form

  `variable = variable operator expression;`

can be rewritten as

  `variable operator= expression;`

- Other assignment operators:

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

(Pre)(Post) Increment and decrement operator

- Increment operator (**++**): can be used instead of **`c += 1`**
- Decrement operator (**--**): can be used instead of **`c -= 1`**
- Pre-increment (decrement): the operator is used before the variable (**`++c`** or **`--c`**). Variable is changed, then the expression it is in is evaluated.
- Post-increment (decrement): operator is used after the variable (**`c++`** or **`c--`**). Expression the variable is in executes, then the variable is changed.

Pre(Post)-increment

- Operator after variable (**`c++, c--`**):

  If **`c = 5`**, then

  ```
  cout << ++c;
  ```

  ⇒ **`c`** is changed to **6**, then printed out

  ```
  cout << c++;
  ```

  ⇒ Prints out **5** (**`cout`** is executed before the increment), **`c`** then becomes **6**

Pre(Post)-increment

```
1	// Fig. 2.14: fig02_14.cpp
2	// Preincrementing and postincrementing.
3	#include <iostream>
4
5	using std::cout;
6	using std::endl;
7
8	// function main begins program execution
9	int main()
10	{
11	   int c;                        // declare variable
12
13	   // demonstrate postincrement
14	   c = 5;                        // assign 5 to c
15	   cout << c << endl;            // print 5
16	   cout << c++ << endl;          // print 5 then postincrement
17	   cout << c << endl << endl; // print 6
18
19	   // demonstrate preincrement
20	   c = 5;                        // assign 5 to c
21	   cout << c << endl;            // print 5
22	   cout << ++c << endl;          // preincrement then print 6
23	   cout << c << endl;            // print 6
24
25	   return 0;    // indicate successful termination
26
27	} // end function main
```

increment.cpp

Pre(Post)-increment

```
1     // Fig. 2.14: fig02_14.cpp
2     // Preincrementing and postincrementing.
3     #include <iostream>
4
5     using std::cout;
6     using std::endl;
7
8     // function main begins program execution
9     int main()
10    {
11       int c;                        // declare variable
12
13       // demonstrate postincrement
14       c = 5;                        // assign 5 to c
15       cout << c << endl;            // print 5
16       cout << c++ << endl;          // print 5 then postincrement
17       cout << c << endl << endl; // print 6
18
19       // demonstrate preincrement
20       c = 5;                        // assign 5 to c
21       cout << c << endl;            // print 5
22       cout << ++c << endl;          // preincrement then print 6
23       cout << c << endl;            // print 6
24
25       return 0;   // indicate successful termination
26
27    } // end function main
```

increment.cpp

`for` Repetition Structure

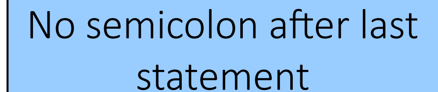
- General format when using **`for`** loops

```
for ( initialization; LoopContinuationTest;increment )
  statement
```

- Example:

```
for( int counter = 1; counter <= 10; counter++ )
cout << counter << endl;
```

Prints integers from one to ten



`for` Repetition Structure

```
1      // Fig. 2.17: fig02_17.cpp
2      // Counter-controlled repetition with the for structure.
3      #include <iostream>
4
5      using std::cout;
6      using std::endl;
7
8      // function main begins program execution
9      int main()
10    {
11       // Initialization, repetition condition and incrementing
12       // are all included in the for structure header.
13
14       for ( int counter = 1; counter <= 10; counter++ )
15          cout << counter << endl;
16
17       return 0;   // indicate successful termination
18
19    } // end function main
```

firstFor.cpp

for Repetition Structure

```
1	// Fig. 2.17: fig02_17.cpp
2	// Counter-controlled repetition with the for structure.
3	#include <iostream>
4
5	using std::cout;
6	using std::endl;
7
8	// function main begins program execution
9	int main()
10	{
11	   // Initialization, repetition condition and incrementing
12	   // are all included in the for structure header.
13
14	   for ( int counter = 1; counter <= 10; counter++ )
15	      cout << counter << endl;
16
17	   return 0;   // indicate successful termination
18
19	} // end function main
```

```
1
2
3
4
5
6
7
8
9
10
```

firstFor.cpp

`for` Repetition Structure

- **`for`** loops can usually be rewritten as **`while`** loops

```
initialization;
while ( loopContinuationTest){
   statement
   increment;
}
```

- Initialization and increment
- For multiple variables, use comma-separated lists

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
  cout << j + i << endl;
```

`for` Repetition Structure

```
1      // Fig. 2.20: fig02_20.cpp
2      // Summation with for.
3      #include <iostream>
4
5      using std::cout;
6      using std::endl;
7
8      // function main begins program execution
9      int main()
10     {
11        int sum = 0;                         // initialize sum
12
13        // sum even integers from 2 through 100
14        for ( int number = 2; number <= 100; number += 2 )
15           sum += number;                    // add number to sum
16
17        cout << "Sum is " << sum << endl;  // output sum
18        return 0;                            // successful termination
19
20     } // end function main
```

```
Sum is 2550
```

secondFor.cpp

for Repetition Structure

```
1       // Fig. 2.20: fig02_20.cpp
2       // Summation with for.
3       #include <iostream>
4
5       using std::cout;
6       using std::endl;
7
8       // function main begins program execution
9       int main()
10     {
11          // Example with multiple variables
12              for ( int i = 0, j = 0; j+1 <= 10; j++, i++ )
13               cout << "i: "<<i<<" j: "<<j<<" i+j: " << i+j << endl;
14        return 0;                          // successful termination
15
16     } // end function main
```

```
i: 0 j: 0 i+j: 0
i: 1 j: 1 i+j: 2
i: 2 j: 2 i+j: 4
i: 3 j: 3 i+j: 6
i: 4 j: 4 i+j: 8
i: 5 j: 5 i+j: 10
i: 6 j: 6 i+j: 12
i: 7 j: 7 i+j: 14
i: 8 j: 8 i+j: 16
i: 9 j: 9 i+j: 18
```

secondFor.cpp

Esercitazione 2

1) Compute the sum of the first n integer numbers. n is arbitrary and is given by the user.

   Use a while loop to calculate the sum. (`SumNumbers.cpp`)

2) Write a program which, given an arbitrary set of positive integer numbers, finds how many are odd numbers and how many are even numbers. Use a while loop. (`EvenOdd.cpp`)

3) Write a program which reads an integer numbers and prints as many as "*" as the input number (`histo.cpp`)

```
bash$ ./histo
Enter a positive integer number, -1 to exit 3
***
Enter a positive integer number, -1 to exit 7
*******
Enter a positive integer number, -1 to exit 4
****
Enter a positive integer number, -1 to exit -1
```

Esercitazione 2

4) Write a program which draws a right-angled triangle with sides equal to the input number (`Triangular.cpp`)

```
bash$ ./triangular
Side length: 6
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

5) Modify the program of point 4) in order to obtain a triangle as shown below (`ReflectedTriangular.cpp`)

```
bash$ ./reflected
Side length: 5
        *
      * *
    * * *
  * * * *
* * * * *
```

`switch` Multiple-Selection Structure

- **`switch`** Test variable for multiple values
- Series of **`case`** labels and optional default case

```
switch ( variable ) {
    case value1:          // taken if variable == value1
    statements
    break;                // necessary to exit switch
    case value2:
    case value3:    // taken if variable == value2 or == value3
    statements
    break;
    default:        // taken if variable matches no other cases
    statements
    break;
}
```

`switch` Multiple-Selection Structure

- Example upcoming:
  - Program to read grades (A-F) and display number of each grade entered
  - Single characters typically stored in a **`char`** data type;
  - **`char`** a 1-byte integer, so **`char`**s can be stored as **`int`**s
  - Can treat character as **`int`** or **`char`**
  - 97 is the numerical representation of lowercase ‘a’ (ASCII)
  - Use single quotes to get numerical representation of character

```
cout << "The character (" << 'a' << ") has the value "
     << static_cast< int > ( 'a' ) << endl;
```

  - Prints

```
The character (a) has the value 97
```

switch Multiple-Selection Structure

```
1       // Fig. 2.22: fig02_22.cpp
2       // Counting letter grades.
3       #include <iostream>
4       #include <stdio.h>
5       using std::cout;
6       using std::cin;
7       using std::endl;
8
9       // function main begins program execution
10     int main()
11     {
12        int grade;       // one grade
13        int aCount = 0;  // number of As
14        int bCount = 0;  // number of Bs
15        int cCount = 0;  // number of Cs
16        int dCount = 0;  // number of Ds
17        int fCount = 0;  // number of Fs
18
19        cout << "Enter the letter grades." << endl
20             << "Enter the EOF character to end input." << endl;
21
```

firstSwitch.cpp

switch Multiple-Selection Structure

```
22       // loop until user types end-of-file key sequence
23       while ( ( grade = cin.get() ) != EOF ) {
24
25          // determine which grade was input
26          switch ( grade ) {  // switch structure nested in while
27
28             case 'A':        // grade was uppercase A
29             case 'a':        // or lowercase a
30                ++aCount;     // increment aCount
31                break;        // necessary to exit switch
32
33             case 'B':        // grade was uppercase B
34             case 'b':        // or lowercase b
35                ++bCount;     // increment bCount
36                break;        // exit switch
37
38             case 'C':        // grade was uppercase C
39             case 'c':        // or lowercase c
40                ++cCount;     // increment cCount
41                break;        // exit switch
42
```

firstSwitch.cpp

`switch` Multiple-Selection Structure

```
22        // loop until user types end-of-file key sequence
23        while ( ( grade = cin.get() ) != EOF ) {
24
25           // determine which grade was input
26           switch ( grade ) {  // switch structure nested in while
27
28              case 'A':        // grade was uppercase A
29              case 'a':        // or lowercase a
30                 ++aCount;     // increment aCount
31                 break;        // necessary to exit switch
32
33              case 'B':        // grade was uppercase B
34              case 'b':        // or lowercase b
35                 ++bCount;     // increment bCount
36                 break;        // exit switch
37
38              case 'C':        // grade was uppercase C
39              case 'c':        // or lowercase c
40                 ++cCount;     // increment cCount
41                 break;        // exit switch
42
```

`cin.get()` uses dot notation (explained later in the course). This function gets 1 character from the keyboard (after *Enter* pressed), and it is assigned to **`grade`**.

- **`cin.get()`** returns EOF (end-of-file) after the EOF character is input, to indicate the end of data. EOF may be **ctrl-d** or ctrl-z, depending on your OS.

firstSwitch.cpp

switch Multiple-Selection Structure

```
22       // loop until user types end-of-file key sequence
23       while ( ( grade = cin.get() ) != EOF ) {
24
25          // determine which grade was input
26          switch ( grade ) {  // switch structure nested in while
27
28             case 'A':        // grade was uppercase A
29             case 'a':        // or lowercase a
30                ++aCount;     // increment aCount
31                break;        // necessary to exit switch
32
33             case 'B':        // grade
34             case 'b':        // or low
35                ++bCount;     // incre
36                break;        // exit s
37
38             case 'C':        // grade
39             case 'c':        // or low
40                ++cCount;     // increment cCount
41                break;        // exit switch
42
```

Assignment statements have a value, which is the same as the variable on the left of the **=**. The value of this statement is the same as the value returned by **cin.get()**.
This can also be used to initialize multiple variables: **a = b = c = 0;**

firstSwitch.cpp

switch Multiple-Selection Structure

```
22       // loop until user types end-of-file key sequence
23       while ( ( grade = cin.get() ) != EOF ) {
24
25          // determine which grade was input
26          switch ( grade ) {  // switch structure nested in while
27
28             case 'A':        // grade was uppercase A
29             case 'a':        // or lowercase a
30                ++aCount;     // increment aCount
31                break;        // necessary to exit switch
32
33             case 'B':        // grade was uppercase B
34             case 'b':        // or lowercase b
                                // increment bCount
                                // exit switch

                                // grade was uppercase C
                                // or lowercase c
40                ++cCount;     // increment cCount
41                break;        // exit switch
42
```

Compares **grade** (an **int**) to the numerical representations of **A** and **a**.

firstSwitch.cpp

switch Multiple-Selection Structure

```
22        // loop until user types end-of-file key
23        while ( ( grade = cin.get() ) != EOF ) {
24
25           // determine which grade was input
26           switch ( grade ) {  // switch structu
27
28              case 'A':        // grade was uppercase A
29              case 'a':        // or lowercase a
30                 ++aCount;     // increment aCount
31                 break;        // necessary to exit switch
32
33              case 'B':        // grade was uppercase B
34              case 'b':        // or lowercase b
35                 ++bCount;     // increment bCount
36                 break;        // exit switch
37
38              case 'C':        // grade was uppercase C
39              case 'c':        // or lowercase c
40                 ++cCount;     // increment cCount
41                 break;        // exit switch
42
```

break causes **switch** to end and the program continues with the first statement after the **switch** structure.

firstSwitch.cpp

switch Multiple-Selection Structure

```
43            case 'D':        // grade was uppercase D
44            case 'd':        // or lowercase d
45               ++dCount;     // increment dCount
46               break;        // exit switch
47
48            case 'F':        // grade was
49            case 'f':        // or lowerca
50               ++fCount;     // increment
51               break;        // exit switc
52
53            case '\n':       // ignore new
54            case '\t':       // tabs,
55            case ' ':        // and spaces
56               break;        // exit switch
57
58            default:         // catch all other characters
59               cout << "Incorrect letter grade entered."
60                    << " Enter a new grade." << endl;
61               break;        // optional; will exit switch anyway
62
63         } // end switch
64
65      } // end while
66
```

This test is necessary because *Enter* is pressed after each letter grade is input. This adds a newline character that must be removed. Likewise, we want to ignore any whitespace.

firstSwitch.cpp

switch Multiple-Selection Structure

```
43             case 'D':        // grade was uppercase D
44             case 'd':        // or lowercase d
45                ++dCount;     // increment dCount
46                break;        // exit switch
47
48             case 'F':        // grade was uppercase F
49             case 'f':        // or lowercase f
50                ++fCount;     // increment fCount
51                break;        // exit switch
52
53             case '\n':       // ignore newlines,
54             case '\t':       // tabs,
55             case ' ':        // and space
56                break;        // exit swit
57
58             default:         // catch all other characters
59                cout << "Incorrect letter grade entered."
60                     << " Enter a new grade." << endl;
61                break;        // optional; will exit switch anyway
62
63          } // end switch
64
65       } // end while
66
```

Notice the **default** statement, which catches all other cases.

firstSwitch.cpp

switch Multiple-Selection Structure

```
67      // output summary of results
68      cout << "\n\nTotals for each letter grade are:"
69           << "\nA: " << aCount   // display number of A grades
70           << "\nB: " << bCount   // display number of B grades
71           << "\nC: " << cCount   // display number of C grades
72           << "\nD: " << dCount   // display number of D grades
73           << "\nF: " << fCount   // display number of F grades
74           << endl;
75
76      return 0;  // indicate successful termination
77
78   } // end function main
```

firstSwitch.cpp

`switch` Multiple-Selection Structure

```
Enter the letter grades.
Enter the EOF character to end input.
a
B
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z


Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

do/while Repetition Structure

```
1     // Fig. 2.24: fig02_24.cpp
2     // Using the do/while repetition structure.
3     #include <iostream>
4
5     using std::cout;
6     using std::endl;
7
8     // function main begins program execution
9     int main()
10    {
11       int counter = 1;              // initialize counter
12
13       do {
14          cout << counter << "  ";   // display counter
15       } while ( ++counter <= 10 );  // end do/while
16
17       cout << endl;
18
19       return 0;   // indicate successful termination
20
21    } // end function main
```

firstDoWhile.cpp

do/while Repetition Structure

```
1      // Fig. 2.24: fig02_24.cpp
2      // Using the do/while repetition structure.
3      #include <iostream>
4
5      using std::cout;
6      using std::endl;
7
8      // function main begins program execution
9      int main()
10     {
11        int counter = 1;              // initialize counter
12
13        do {
14           cout << counter << "  ";   // display counter
15        } while ( ++counter <= 10 );  // end do/while
16
17        cout << endl;
18
19        return 0;   // indicate successful termination
20
21     } // end function main
```

Notice the preincrement in loop-continuation test.

```
1  2  3  4  5  6  7  8  9  10
```

firstDoWhile.cpp

break Statements

- **break** statement:

  Immediate exit from **while**, **for**, **do/while**, **switch**

  Program continues with first statement after structure
- Common uses: Escape early from a loop
- Skip the remainder of **switch**

break Statements

```
1      // Fig. 2.26: fig02_26.cpp
2      // Using the break statement in a for structure.
3      #include <iostream>
4
5      using std::cout;
6      using std::endl;
7
8      // function main begins program execution
9      int main()
10     {
11
12        int x;  // x declared here so it can be used after the loop
13
14        // loop 10 times
15        for ( x = 1; x <= 10; x++ ) {
16
17           // if x is 5, terminate loop
18           if ( x == 5 )
19              break;           // break loop only if x is 5
20
21           cout << x << " ";   // display value of x
22
23        } // end for
24
25        cout << "\nBroke out of loop when x became " << x << endl;
```

Exits **for** structure when **break** executed

breakExample.cpp

break Statements

```
26
27        return 0;    // indicate successful termination
28
29    } // end function main
```

```
1 2 3 4
Broke out of loop when x became 5
```

breakExample.cpp

`continue` Statements

- **`continue`** statement:

  Used in **`while`**, **`for`**, **`do/while`**

  Skips remainder of loop body

  Proceeds with next iteration of loop

- **`while`** and **`do/while`** structure: Loop-continuation test evaluated immediately after the **`continue`** statement

- **`for`** structure: Increment expression executed; Next, loop-continuation test evaluated

continue Statements

```
1    // Fig. 2.27: fig02_27.cpp
2    // Using the continue statement in a for structure.
3    #include <iostream>
4
5    using std::cout;
6    using std::endl;
7
8    // function main begins program execution
9    int main()
10   {
11      // loop 10 times
12      for ( int x = 1; x <= 10; x++ ) {
13
14         // if x is 5, continue with next iteration of loop
15         if ( x == 5 )
16            continue;        // skip remaining code in loop body
17
18         cout << x << " ";   // display value of x
19
20      } // end for structure
21
22      cout << "\nUsed continue to skip printing the value 5"
23           << endl;
24
25      return 0;              // indicate successful termination
26
27   } // end function main
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

continueExample.cpp

Logical Operators

- Used as conditions in loops and in if statements:
- **`&&`** (logical **AND**): **`true`** if both conditions are **`true`**

```
if ( gender == 1 && age >= 65 )
   ++seniorFemales;
```

- **`||`** (logical **OR**): **`true`** if either of condition is **`true`**

```
if ( semesterAverage >= 90 || finalExam >= 90 )
   cout << "Student grade is A" << endl;
```

- **`! (logical NOT, logical negation):`** Returns **`true`** when its condition is **`false`**, & vice versa

```
if ( !( grade == sentinelValue ) )
   cout << "The next grade is " << grade << endl;
```

Alternative:

```
if ( grade != sentinelValue )
   cout << "The next grade is " << grade << endl;
```

Confusing Equality (==) and Assignment (=) Operators

- Common error. Does not typically cause syntax errors
- Aspects of problem: Expressions that have a value can be used for decision Zero = false, nonzero = true
- Assignment statements produce a value (the value to be assigned)
- Example

```
if ( payCode == 4 )
  cout << "You get a bonus!" << endl;
```

  - If paycode is 4, bonus given

- If **==** was replaced with **=**

```
if ( payCode = 4 )
 cout << "You get a bonus!" << endl;
```

  - Paycode set to 4 (no matter what it was before), Statement is true (since 4 is non-zero) Bonus given in every case

Esercitazione 3

1) Write a program which evaluate the factorial of a given number n. Use a for loop. (`factorial.cpp`)

```
bash$ ./factorial
Give me an integer: 12
12! = 479001600
```

2) Write a program which determines the number of digits of a given number using a while loop. (`numersOfDigits.cpp`)

(Tip: using the rules of divisions between integers, divide the number by 10, until the results is 0. The number of divisions is the number of digits of the integer.)

```
bash$ ./numerofDigits
Give me an integer: 12345
12345 has 5 digits
```

Esercitazione 3

3) Write a program that prints the position of a body moving with a uniformly accelerated motion every deltaT seconds for n times. (`motion.cpp`)

```
bash$ ./motion
Print the position of a body moving with a uniformly
accelerated
motion every deltaT seconds for n times

Give me acceleration, velocity and x0 4 6 8
How many times do you want to print the position ? 10
Delta T ? 2

 x(t): 8  t= 0 seconds
 x(t): 28 t= 2 seconds
 ...........
 x(t): 928 t= 20 seconds
```

4) A ball, dropped from a given height, rebounds reaching at every rebound half of the height of the previous rebound. Write a program that prints the ball rebounds until the height of the rebound is less than a pre-set tolerance (`rebound.cpp`)

```
bash$ ./rebound
Initial height: 10
Rebound # 1: height 5 meters
Rebound # 2: height 2.5 meters
   ..............
Rebound # 14: height 0.000610352 meters
```

5) Write a program that determines whether a given number is prime or not (`PrimeNumber.cpp`)

```
bash$ ./primenum
Give me an integer 8
Number 8 is not prime

bash$ ./primenum
Give me an integer 7
Number 7 is prime
```