Fundamentals of Programming I

2 Types and Instructions I

Grado en Ingeniería Informática

Luis Hernández Yáñez
Facultad de Informática
Universidad Complutense

Index

A Programming Example

Luis Hernández Yáñez

A computer in a car...

Instructions the computer understands:

<instruction> ::= <inst> ;

<inst> ::= `Start` | `Stop` | <run>

<run> ::= `Go` <direction> <num> `Blocks`

<direction> ::= `North` | `East` | `South` | `West`

<num> ::= `1` | `2` | `3` | `4` | `5`

Examples:

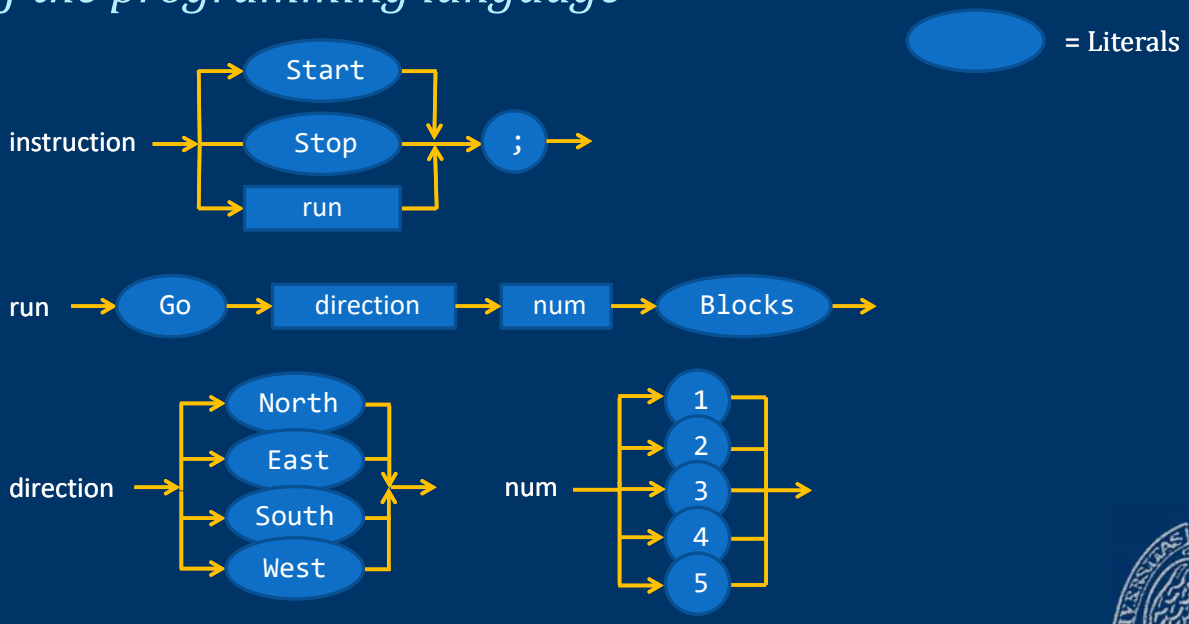
```
Start;
Go North 3 Blocks;
Stop;
```

pxhere.com

Luis Hernández Yáñez

A Programming Example

Syntax of the programming language

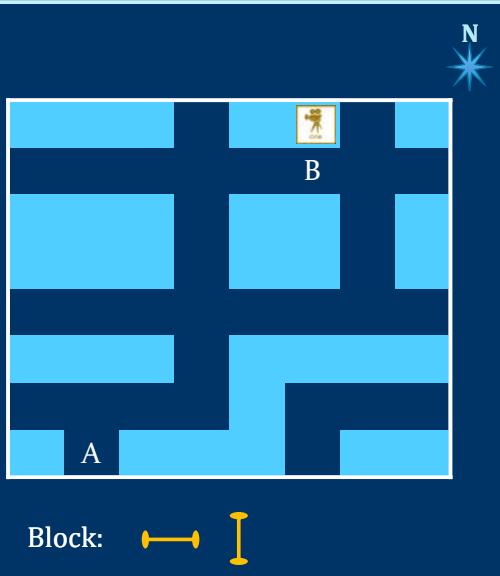


Luis Hernández Yáñez

A Programming Example

Problem to be solved

The car being in position A,
to reach the Tivoli Cinema (B)



What are the steps to be followed?

Start
Go North one block
Go East two blocks
Go North five blocks
Go East two blocks
Stop

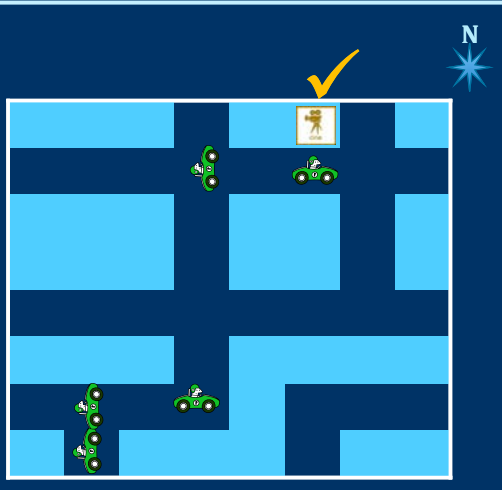
Luis Hernández Yáñez

A Programming Example

Algorithm

Sequence of steps to be followed to solve the problem:

1.- Start
2.- Go North one block
3.- Go East two blocks
4.- Go North five blocks
5.- Go East two blocks
6.- Stop



Same steps for a person or a computer

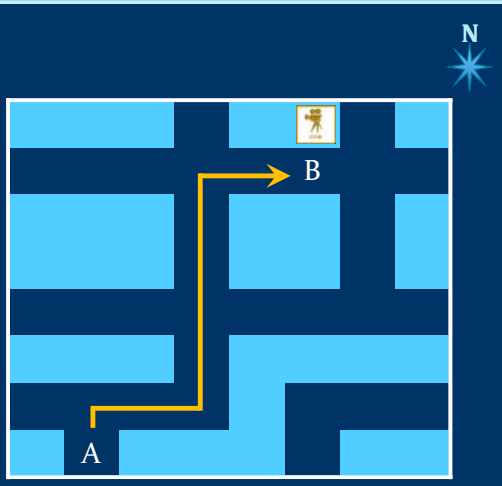
Luis Hernández Yáñez

A Programming Example

Program

Instructions written in the programming language:

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go East 2 Blocks;
Stop;
```

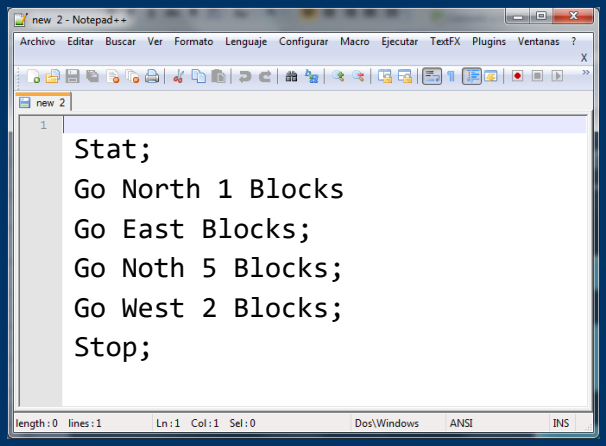


Luis Hernández Yáñez

A Programming Example

Program

We write the program code with an editor, and save it in a file:



We copy the file, and take it to the car

Luis Hernández Yáñez

A Programming Example

Compilation

We download the file into the car, and press the execute button:

```
Stat;
----^ Unknown word.
Go North 1 Blocks
-----------------^ ; missing.
Go East Blocks;
--------^ Number missing.
Go Noth 5 Blocks;
-------^ Unknown word.
Go West 2 Blocks;
Stop;
There are errors. Impossible to run the program.
```

Syntax errors

Luis Hernández Yáñez

A Programming Example

Debugging

We edit the code, correcting syntactic errors:

```
Stat;                          Start;
Go North 1 Blocks              Go North 1 Blocks;
Go East Blocks;        →       Go East 3 Blocks;
Go Noth 5 Blocks;              Go North 5 Blocks;
Go West 2 Blocks;              Go West 2 Blocks;
Stop;                          Stop;
```

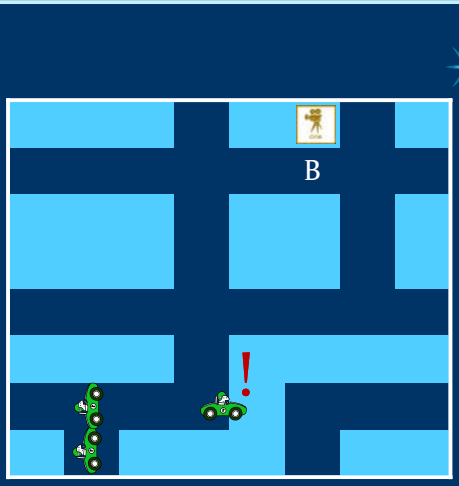
Luis Hernández Yáñez

A Programming Example

Execution

It does what it is asked to:

```
Start;
Go North 1 Blocks;
Go East 3 Blocks;
```



Execution error

Impossible to execute one instruction!

Luis Hernández Yáñez

A Programming Example

Debugging

We edit the code, correcting execution errors:

```
Start;
Go North 1 Blocks;
Go East 3 Blocks;
Go North 5 Blocks;
Go West 2 Blocks;
Stop;
```

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go West 2 Blocks;
Stop;
```

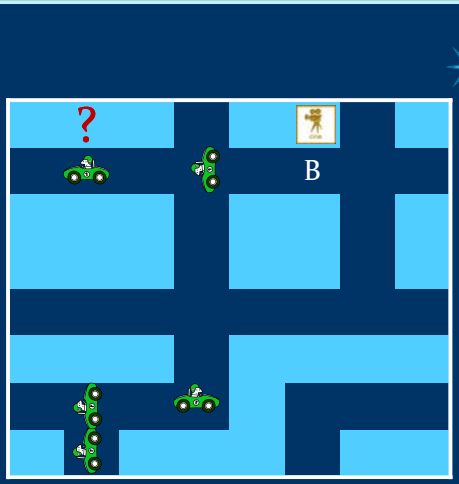
Luis Hernández Yáñez

A Programming Example

Execution

It does what it is asked to:

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go West 2 Blocks;
Stop;
```



Logic error

The program does not solve the problem!

Luis Hernández Yáñez

A Programming Example

Debugging

We edit the code, correcting logical errors:

```
Start;                          Start;
Go North 1 Blocks;              Go North 1 Blocks;
Go East 2 Blocks;       →       Go East 2 Blocks;
Go North 5 Blocks;              Go North 5 Blocks;
Go West 2 Blocks;               Go East 2 Blocks;
Stop;                           Stop;
```

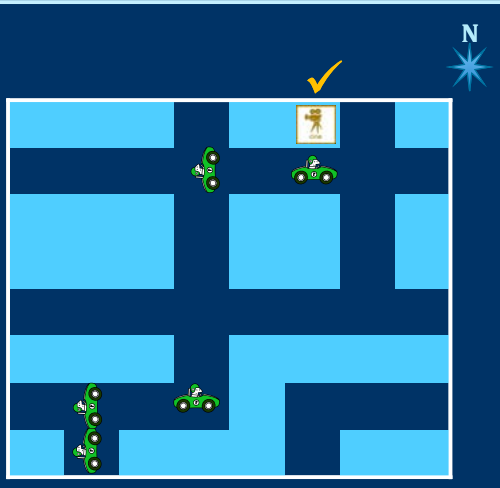
Luis Hernández Yáñez

A Programming Example

Execution

It does what it is asked to:

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go East 2 Blocks;
Stop;
```



Success!

Luis Hernández Yáñez

First C++ Program

Luis Hernández Yáñez

Hello world!

Again with the greeting program:

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

int main() // main() is where execution starts
{
   cout << "Hello world!" << endl;
   // Outputs Hello world!

   return 0;
}
```

Luis Hernández Yáñez

First C++ Program

Hello world!

Only one instruction produces something tangible:

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

int main() // main() is where execution starts
{
   cout << "Hello world!" << endl;
   // Outputs Hello world!

   return 0;
}
```

Luis Hernández Yáñez

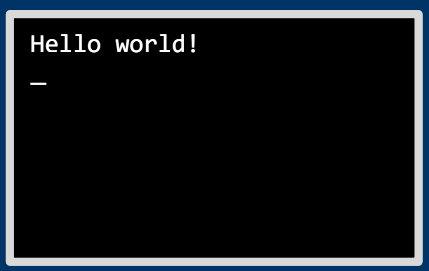
First C++ Program

`cout` (`iostream`) — *character output stream*

Output to screen: `<<` operator (*insertor*)

```
cout << "Hello world!" << endl;
```

`<< "Hello world!"<< endl;`



`endl` → *end line*

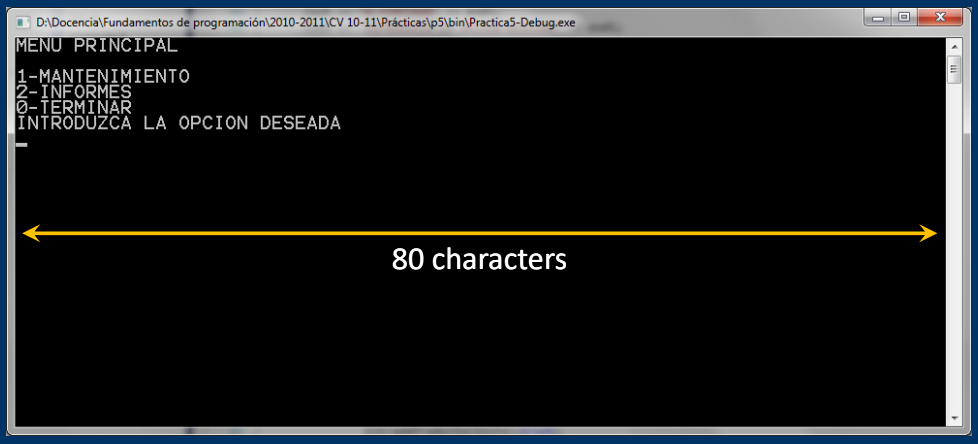
Luis Hernández Yáñez

Output Device

Screen in text mode

Typically 80-character lines (more or less, depending on the window's width)

Application in text mode

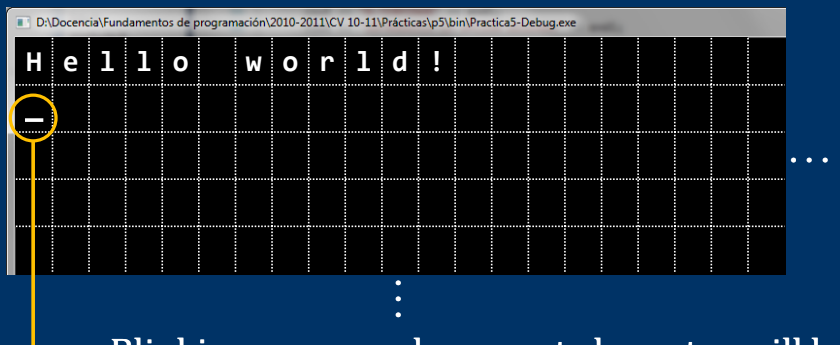


Output Device

Console windows (Terminals)

Applications in text mode execute into text mode windows:

- ✓ MS Windows: console windows (*command line*)
- ✓ Linux: terminal windows



Data Visualization

Insertor (<<)

```
cout << ...;
```

Inserts texts in text mode screen

Text representation of numerical data

Starts at cursor position

Line wrap (will continue in next line if needed)

Can be chained:

```
cout << ... << ... << ...;
```

Remember: instructions end with ;

Luis Hernández Yáñez

Data Visualization

With << insertor we can display...

- ✓ Literal character strings
  Text enclosed between double quotes: `"..."`
  `cout << "Hello world!";`
  Quotes are not displayed!

- ✓ Literal numbers
  With or without decimals and sign: `123`, `-37`, `3.1416`, ...
  `cout << "Pi = " << 3.1416;`
  The characters that make up the number are displayed

- ✓ `endl`

Luis Hernández Yáñez

First C++ Program

Main program

`main()` function: where execution starts...

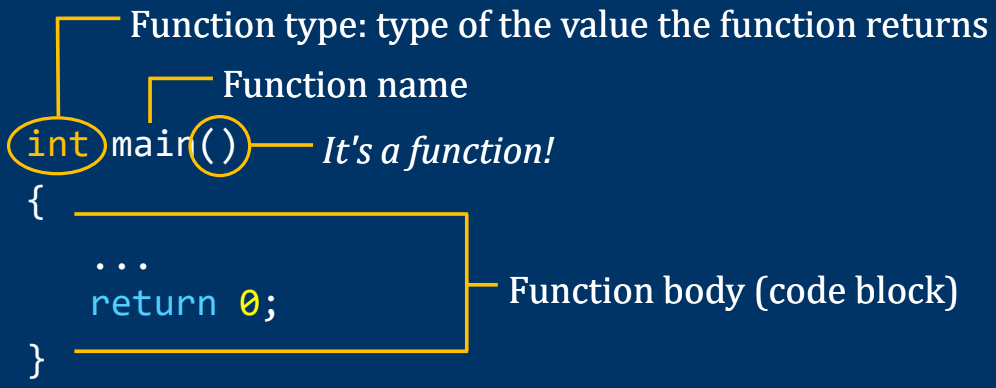
```
#include <iostream>
using namespace std;

int main()
{
   cout << "Hello world!" << endl;
   return 0;
}
```

Contains instructions to be executed

First C++ Program

Main program

`main()` function:



`return 0;` Returns the result (`0`) of the function

First C++ Program

Documenting the code...

Comments (ignored when compiling):

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

int main() // main() is where execution starts
{
   cout << "Hello world!" << endl;
   ...
```

Until the end of the line: `// One line comment`

Several lines:

```
/* Comment in several
consecutive lines */
```

Luis Hernández Yáñez

First C++ Program

The infrastructure

Code to reuse:

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

int main()
{
   cout << "Hello world!" << endl;
   return 0;
}
```

← A directive: starts with #

Function libraries for C++ programmers

Luis Hernández Yáñez

First C++ Program

Libraries

Are included with `#include` *directive*

```
#include <iostream>
```

(Utilities for console I/O)

Programs must include `iostream` library for console I/O

Namespaces

`iostream` has several namespaces: Which one do we need?

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

← Instruction: ends with `;`

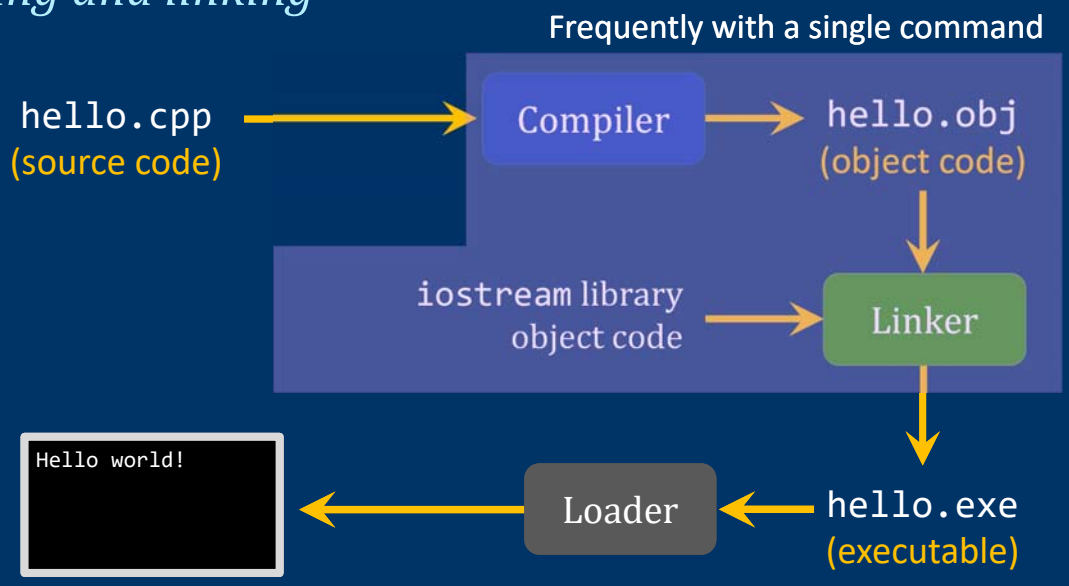
We will always use `std` namespace (standard)

Many libraries do not have namespaces

Luis Hernández Yáñez

First C++ Program

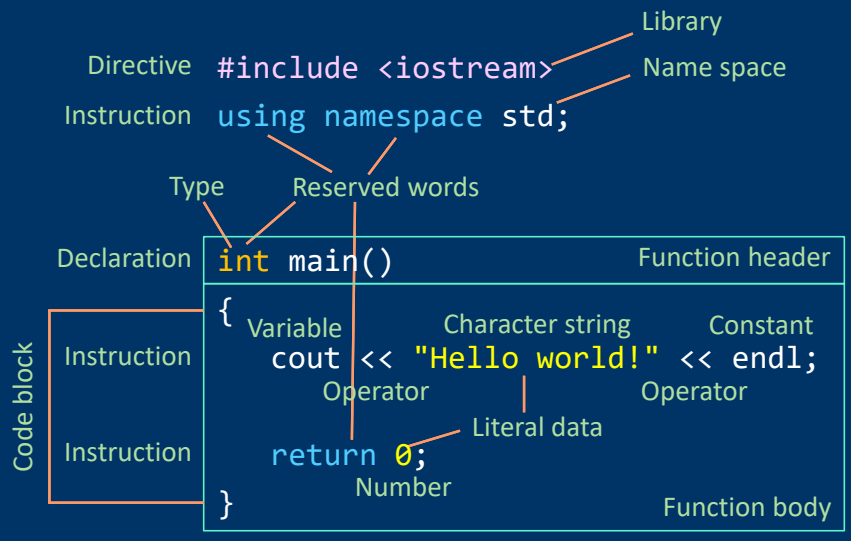
Compiling and linking



Luis Hernández Yáñez

The First C++ Program

Program elements (w/o comments)



Syntactic Coloring

Instructions end with ;

First C++ Program

Using blank space

Spaces, Tabs and New line

Separation of elements

Ignored when compiling

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

int main()
{
   cout << "Hello world!" << endl;
   return 0;
}
```

```
#include<iostream> using namespace std; int
main(){cout<<"Hello world!"<<endl; return 0;}
```

Which one is easier to read?

Program Code Lines

Luis Hernández Yáñez

Minimal Program

Program with console I/O

A template to begin with:

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

int main()
{

   return 0;
}
```

Your code here!

Luis Hernández Yáñez

Don Quixote...

... recited in the console

Texts are displayed with `cout << ...`

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

int main()
{
   cout << "En un lugar de la Mancha," << endl;
   cout << "de cuyo nombre no quiero acordarme," << endl;
   cout << "no ha mucho tiempo que vivía un hidalgo de los de lanza en
astillero, ..." << endl;
   return 0;
}
```


Code Lines

Inputting the program code

Finish each code line with a carriage return (↲):

```
#include <iostream>↲
using namespace std;↲
↲
int main()↲
{↲
   cout << "En un lugar de la Mancha," << endl;↲
   cout << "de cuyo nombre no quiero acordarme," << endl;↲
   cout << "no ha mucho tiempo que vivía un hidalgo de los de lanza en
astillero, ..." << endl;↲
   return 0;↲
}↲
```


Code Lines

Inputting the program code

Never break a literal string between two lines:

```
cout << "no ha mucho tiempo que vivía un hidalgo de ↲
los de lanza en astillero, ..." << endl;↲
```

Errors...

The string doesn't end (1st line)!

`los` *is not understood (2nd line)!*

Check by yourself how the compiler notifies you those errors!

Program thinking in Future Changes

Maintainability and reusability

- ✓ Use blank space to separate program elements:

```
cout << "En un lugar de la Mancha," << endl;
```

is better than

```
cout<<"En un lugar de la Mancha,"<<endl;
```

- ✓ Use indentation inside blocks of code:

```
{
    cout << "En un lugar de la Mancha," << endl;
    ...
    return 0;
}
```

Tab or 3 spaces

Style matters!

Calculations in Programs

Luis Hernández Yáñez

Arithmetic operators

- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division

Binary operators

`left_operand operator right_operand`

Operation	Result
`3 + 4`	`7`
`2.56 - 3`	`-0.44`
`143 * 2`	`286`
`45.45 / 3`	`15.15`

Luis Hernández Yáñez

Calculations in Programs

Literal numbers

- ✓ Integer numbers: no decimals

  Minus sign (optional) + digit sequence

  `3` `143` `-12` `67321` `-1234`

  Don't use thousands comma!

- ✓ Real numbers: with decimals

  Minus sign (optional) + digit sequence
  + decimal point + digit sequence

  `3.1416` `357.0` `-1.333` `2345.6789` `-404.1`

Luis Hernández Yáñez

Calculations in Programs

`arith.cpp`

Example

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

int main()
{
   cout << "133 + 1234 = " << 133 + 1234 << endl;
   cout << "1234 - 111.5 = " << 1234 - 111.5 << endl;
   cout << "34 * 59 = " << 34 * 59 << endl;
   cout << "3.4 * 5.93 = " << 3.4 * 5.93 << endl;
   cout << "500 / 3 = " << 500 / 3 << endl; // Integer div.
   cout << "500.0 / 3 = " << 500.0 / 3 << endl; // Real div.

   return 0;
}
```

Text

Number

Luis Hernández Yáñez

Calculations in Programs

Integer division

Real division

```
D:\FP\Less02>arith
133 + 1234 = 1367
1234 - 111.5 = 1122.5
34 * 59 = 2006
3.4 * 5.93 = 20.162
500 / 3 = 166
500.0 / 3 = 166.667

D:\FP\Less02>_
```

Luis Hernández Yáñez

Calculations in Programs

Integer division or real division?

Both operands are integer numbers → Integer division

Any operand is a real number → Real division

Division	Result
`500 / 3`	166
`500.0 / 3`	166.667
`500 / 3.0`	166.667
`500.0 / 3.0`	166.667

Always double-check if the kind of division is the one you intended!

Luis Hernández Yáñez

Variables

Luis Hernández Yáñez

Variables

Data maintained in memory

Variable: data accessed by means of its name

Literal data: a concrete value (*literally written*)

Variable: can change value (*variant*)

```
age = 19; // variable age and literal 19
```

Variables must be declared first

What type of data we want to maintain?

- ✓ Numerical value without decimals (integer number): `int` type
- ✓ Numerical value with decimals (real number): `double` type

Declaration: *`type name;`*

Luis Hernández Yáñez

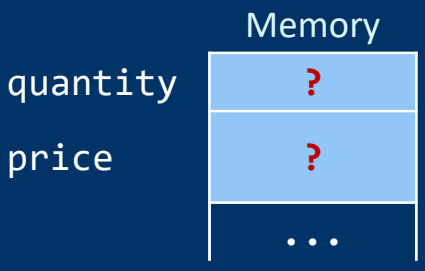
Variables

Variable declaration

type name;

```
int quantity;
double price;
```

Enough space is reserved in memory



VARIABLES ARE NOT INITIALIZED

Don't use them before any value has been assigned

Where do we put declarations?

Always before the first use

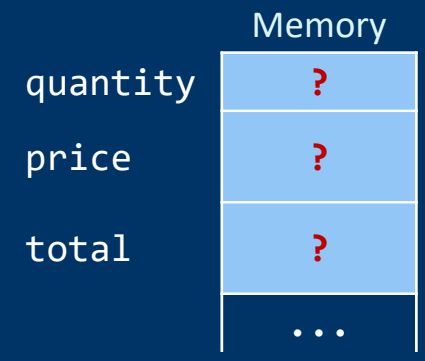
Usually at the beginning of the block of code

Luis Hernández Yáñez

Variables

Variable declaration

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

int main()
{
   int quantity;
   double price, total;


   return 0;
}
```



Several variables of the same type can be declared by separating their names with commas

Luis Hernández Yáñez

Variables

Variable capacity

`int`

-2,147,483,648 ... 2,147,483,647

`-2147483648 .. 2147483647`

`double`

+/- 2.23 x 10^{-308} ... 1.79 x 10^{+308} ← Scientific Notation

`[+|-] 2.23e-308 .. 1.79e+308`

Precision problems...

Luis Hernández Yáñez

Variables

Assigning values to variables (= operator)

variable = *expression*; ← Instruction: ends with ;

```
quantity = 12; // int
price = 39.95; // double
total = quantity * price; // Assigns 479.4
```

quantity ← 12

Type concordance: ~~`quantity = 12.5;`~~ `// No decimals!`

Always a variable to the left of = !!!

Luis Hernández Yáñez

Expressions

Luis Hernández Yáñez

Expressions

Operand/operator sequences

```
operand operator operand operator operand ...
total = quantity * price * 1.21;
```

Expression

Operators are usually evaluated from left to right

Parenthesis to force certain operations:

```
total = quantity1 + quantity2 * price;
total = (quantity1 + quantity2) * price;
```

≠

Some operators are evaluated before others

Luis Hernández Yáñez

BY NC SA

Expressions

Operator precedence

```
quantity1 = 10;
quantity2 = 2;
price = 40.0;
```

* and / are evaluated before + and -

```
total = quantity1 + quantity2 * price;
```

* before + → 10 + 2 * 40,0 → 10 + 80,0 → 90,0

```
total = (quantity1 + quantity2) * price;
```

(+) before * → (10 + 2) * 40,0 → 12 * 40,0 → 480,0

Variables and expressions

variables.cpp

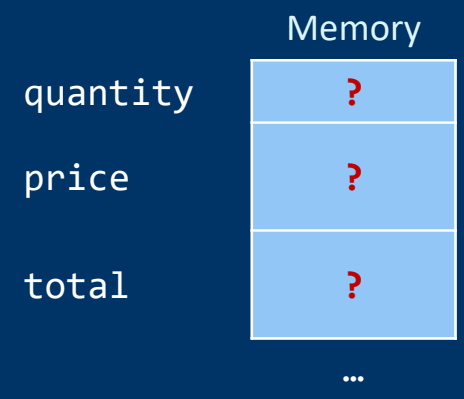
Example

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

int main()
{
   int quantity;
   double price, total;
   quantity = 12;
   price = 39.95;
   total = quantity * price;
   cout << quantity << " x " << price << " = "
        << total << endl;

   return 0;
}
```

Variables and expressions

Example

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

int main()
{
   int quantity;
   double price, total;
```

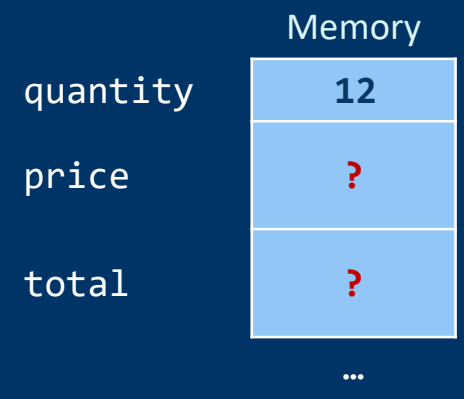


Luis Hernández Yáñez

Variables and expressions

Example

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

int main()
{
   int quantity;
   double price, total;
   quantity = 12;
```



Luis Hernández Yáñez

Variables and expressions

Example

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

int main()
{
   int quantity;
   double price, total;
   quantity = 12;
   price = 39.95;
```

Memory

quantity	12
price	39.95
total	?
	…

Variables and expressions

Example

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

int main()
{
   int quantity;
   double price, total;
   quantity = 12;
   price = 39.95;
   total = quantity * price;
```

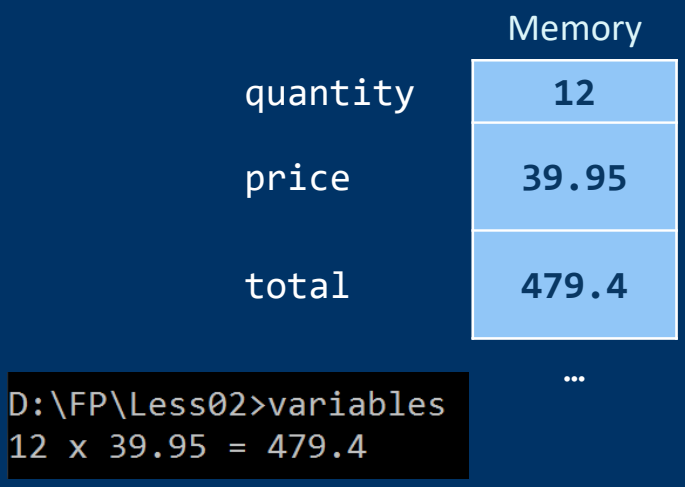
Memory

quantity	12
price	39.95
total	479.4
	…

Variables and expressions

Example

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

int main()
{
   int quantity;
   double price, total;
   quantity = 12;
   price = 39.95;
   total = quantity * price;
   cout << quantity << " x " << price << " = "
        << total << endl;
```



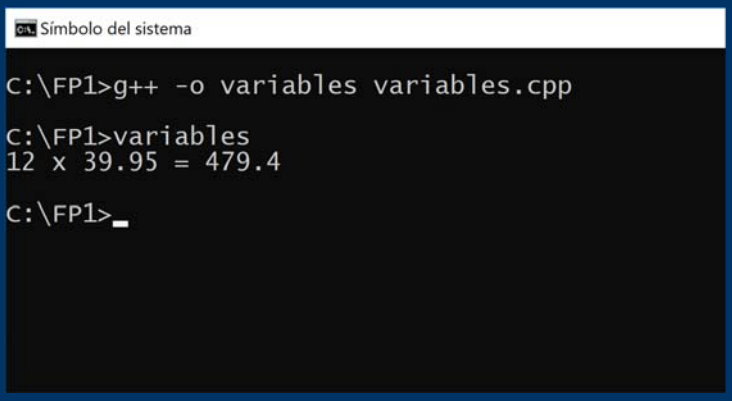

```
D:\FP\Less02>variables
12 x 39.95 = 479.4
```

Luis Hernández Yáñez

Variables and expressions

Example

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

int main()
{
   int quantity;
   double price, total;
   quantity = 12;
   price = 39.95;
   total = quantity * price;
   cout << quantity << " x " << price << " = "
        << total << endl;

   return 0;
}
```



Luis Hernández Yáñez

Reading Data from the Keyboard

Luis Hernández Yáñez

Values Provided by the User

`cin` (`iostream`) — *character input stream*

Input of variable's values: operator `>>` (*extractor*)

```
cin >> quantity;
```

Luis Hernández Yáñez

BY NC SA

Values Provided by the User

Extractor (>>)

```
cin >> variable;
```

Transforms input characters into data

Blinking cursor: where the next character will be input

Input ends with Enter (↲) (cursor to the next line)

To the right of the operator THERE MUST BE a variable!

Initial blank spaces are ignored

Luis Hernández Yáñez

Values Provided by the User

Reading integer values (`int`)

Digits are read until a non-digit character is encountered

`12abc↲` `12 abc↲` `12 abc↲` `12↲`

Value 12 is assigned to the variable; the rest of the input is pending

Recommendation: Read each variable in one line `12↲`

Reading real values (`double`)

Digits, a decimal point, and other digits are read

`39.95.5abc↲` `39.95 abc↲` `39.95↲`

Value 39.95 is assigned to the variable; the rest is pending

Recommendation: Read each variable in one line `39.95↲`

Luis Hernández Yáñez

Values Provided by the User

What happens if the user makes a mistake?

Data will not be correct

Professional application: checking code and help

We will assume that users never make mistakes

On some occasions we will add simple checking code

To avoid mistakes, read each piece of data in a separate line!

Values Provided by the User

`variableinput.cpp`

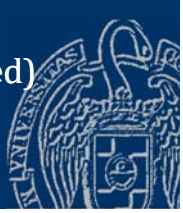
What happens if the user makes a mistake?

```
int quantity;
double price, total;
cout << "Please input quantity: ";
cin >> quantity;
cout << "Please input price: ";
cin >> price;
cout << "Quantity: " << quantity << endl;
cout << "Price: " << price << endl;
```

User friendly!
What are we asking for?

```
Please input quantity: abc
Please input price: Quantity: 0
Price: 1.79174e-307
```

An integer couldn't be read
→ 0 for `quantity` and Error
`price` input fails (no value is assigned)



Values Provided by the User

What happens if the user makes a mistake?

```
Please input quantity: 12abc
Please input price: Quantity: 12
Price: 0
```

12 for `quantity`
A real # couldn't be read
→ 0 for `price` and Error

```
Please input quantity: 12.5abc
Please input price: Quantity: 12
Price: 0.5
```

12 for `quantity`
`.5` → 0,5 for `price`
The rest is pending

```
Please input quantity: 12
Please input price: 39.95
Quantity: 12
Price: 39.95
```

Correct input!!!

A Program with Data Reading

Number division

Ask the user for two real numbers and show the result from dividing the first number by the second number

Algorithm.- Data / Calculations

1. Ask for the `numerator` — Variable `numerator` (`double`)
2. Ask for the `denominator` — Variable `denominator` (`double`)
3. Perform the division, keeping the `result` — Variable `result` (`double`)

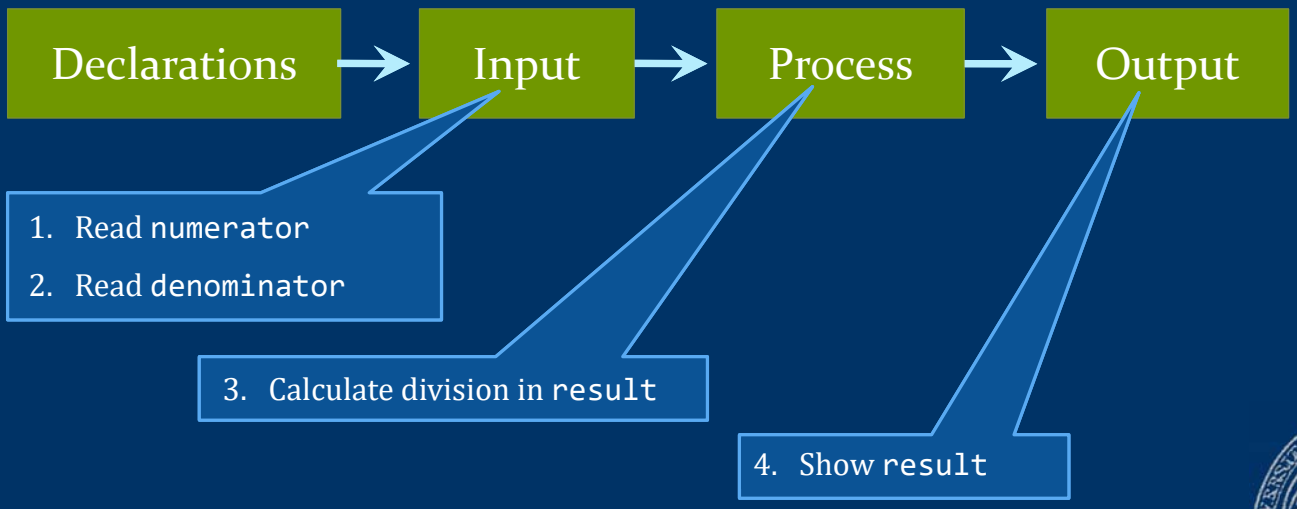
```
result = numerator / denominator
```

4. Show the `result`

A General Scheme

Input-Process-Output

Many programs follow a simple scheme:



Luis Hernández Yáñez

A Program with Data Reading

Number division

Ask the user for two real numbers and show the result from dividing the first number by the second number

1. Read `numerator` `cin >> numerator;`
2. Read `denominator` `cin >> denominator;`
3. Calculate division in `result` `result = numerator / denominator;`
4. Show `result` `cout << result;`

Luis Hernández Yáñez

A Program with Data Reading

Implementation

Number division

division.cpp

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

int main()
{
    double numerator, denominator, result;
    cout << "Numerator: ";
    cin >> numerator;
    cout << "Denominator: ";
    cin >> denominator;
    result = numerator / denominator;
    cout << "Result: " << result << endl;
    return 0;
}
```

Declarations	`double numerator, denominator, result;`

Input	`cout << "Numerator: ";`
	`cin >> numerator;`
	`cout << "Denominator: ";`
	`cin >> denominator;`

Process	`result = numerator / denominator;`
Output	`cout << "Result: " << result << endl;`

```
Numerator: 129
Denominator: 2
Result: 64.5
_
```

Luis Hernández Yáñez

Fundamentals of Programming I

Analysis and Design

Luis Hernández Yáñez

Analysis and Design

Problem statement

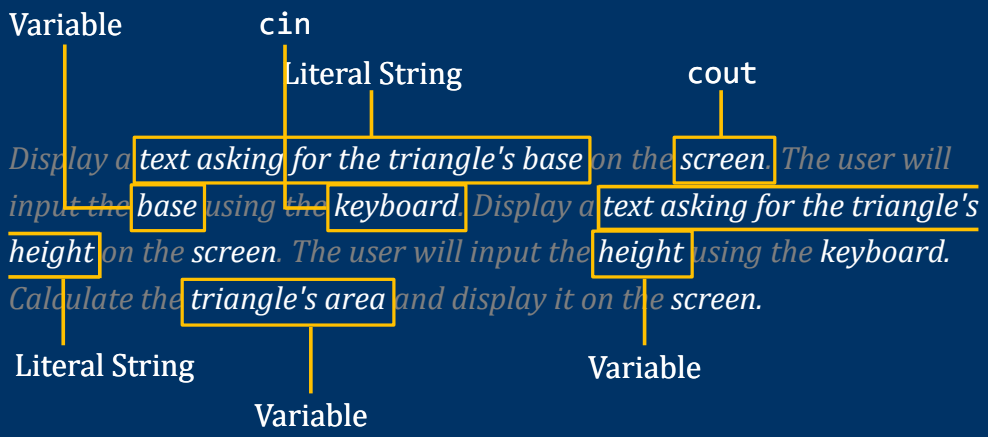
With the base and height of a triangle, show its area

Refinement

Display a text asking for the triangle's base on the screen. The user will input the base using the keyboard. Display a text asking for the triangle's height on the screen. The user will input the height using the keyboard. Calculate the triangle's area and display it on the screen.

Analysis and Design

Objects: Data the program manages



Analysis and Design

Program data types

Object		*Type*	*Name*
Screen	I/O System		`cout`
"Please input triangle's base: "	Literal		-
Triangle's Base		`double`	`base`
Keyboard	I/O System		`cin`
"Please input triangle's height: "	Literal		-
Triangle's Height		`double`	`height`
Triangle's Area		`double`	`area`

Luis Hernández Yáñez

Analysis and Design

Operations (Actions)

```
cout << ...                cin >> ...
```

Display a text asking for the triangle's base on the screen. The user will input the base using the keyboard. Display on the screen a text asking for the triangle's height. The user will input the height using the keyboard. Calculate the triangle's area and display it on the screen.

```
area = base * height / 2
```

Luis Hernández Yáñez

The Algorithm

Sequence of actions (steps) to take for solving the problem

1. Display a text asking for triangle's base on the screen
2. Read the value for triangle's base from the keyboard
3. Display a text asking for triangle's height on the screen
4. Read the value for triangle's height from the keyboard
5. Calculate triangle's area
6. Display triangle's area

Luis Hernández Yáñez

The Program

```
#include <iostream>
using namespace std;
int main()
{
    Declarations

    Algorithm
    translated
    to
    C++ code

    return 0;
}
```

1. Display a text asking for triangle's base on the screen
2. Read the value for triangle's base from the keyboard
3. Display a text asking for triangle's height on the screen
4. Read the value for triangle's height from the keyboard
5. Calculate triangle's area
6. Display triangle's area

Luis Hernández Yáñez

The Program

triangle.cpp

Implementation

```
D:\FP\Less02>triangle
Please input triangle's base: 34.7
Please input triangle's height: 12
Area of a triangle with base 34.7 and height 12: 208.2
```

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

int main()
{
   double base, height, area;                  // Declarations
   cout << "Please input triangle's base: ";          // 1
   cin >> base;                                       // 2
   cout << "Please input triangle's height: ";        // 3
   cin >> height;                                     // 4
   area = base * height / 2;                          // 5
   cout << "Area of a triangle with base " << base    // 6
        << " and height " << height << ": " << area
        << endl;

   return 0;
}
```

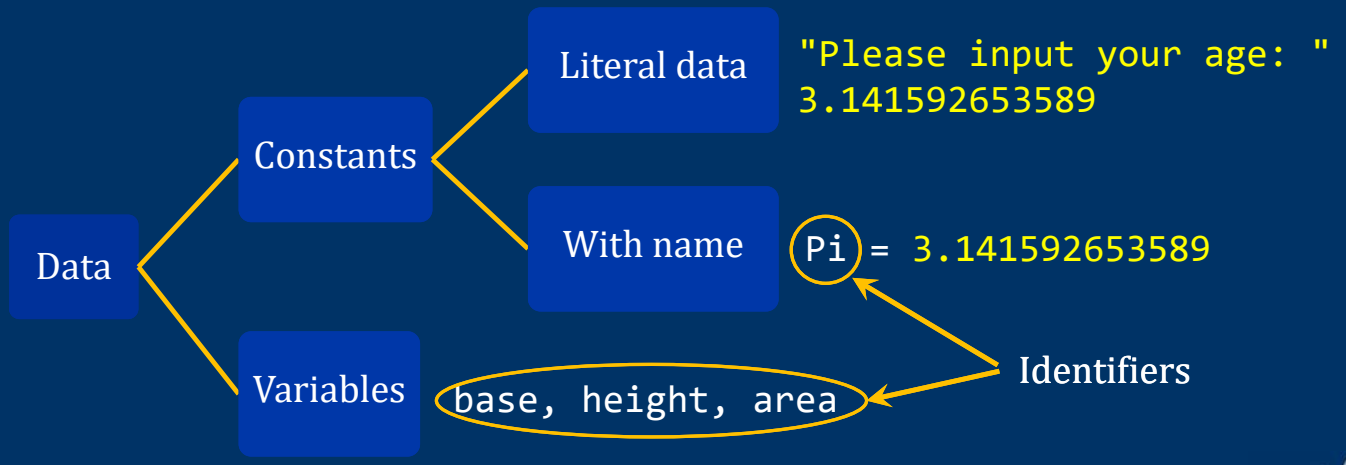
👀 Remember: instructions end with ;

Fundamentals of Programming I

Program Data

Program Data

Data variability



Luis Hernández Yáñez

Fundamentals of Programming I

Identifiers

Luis Hernández Yáñez

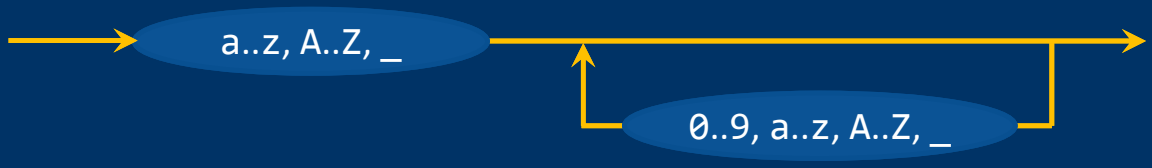
Identifiers

≠ reserved keywords

Variables and named constants

- *Name* of data (to access it, modify it, ...)
- *Descriptive!*

Syntax




```
quantity  price  total  base  height  area  numerator
```

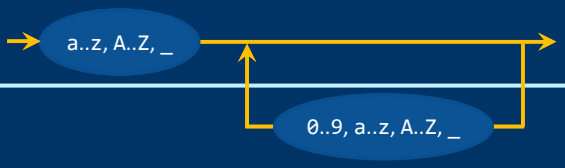
At least 32 significant characters

Identifiers

C++ reserved keywords

```
asm  auto  bool  break  case  catch  char  class  const
const_cast  continue  default  delete  do  double
dynamic_cast  else  enum  explicit  extern  false  float
for  friend  goto  if  inline  int  long  mutable  namespace
new  operator  private  protected  public  register
reinterpret_cast  return  short  signed  sizeof  static
static_cast  struct  switch  template  this  throw  true
try  typedef  typeid  typename  union  unsigned  using
virtual  void  volatile  while
```

Identifiers



Which identifiers are valid?

`balance` ✓	`anualRate` ✓
`_tax_base` ✓	`años` ✗
`AGE12` ✓	`salary_1_month` ✓
`__age` ✓	`cálculoNómina` ✗
`value%100` ✗	`AnyValue` ✓
`100characters` ✗	`value?` ✗
`_12_months` ✓	`____value` ✓

Fundamentals of Programming I

Data Types

Data Types

Types

Each piece of data... of a certain type

Each type states:

- The valid value set (range)
- The set of operations that can be applied to the values

`125` `true` `'a'` `3.14159` `"Hello"`

Expressions with data of different (compatible) types:
Automatic type promotion

Technical details: Lesson Supplement

Basic Data Types

`int`	Integer numbers (no decimals)	`1363`, `-12`, `49` ✓
`float`	Real numbers	`12.45`, `-3.1932`, `1.16E+02`
`double`	Real numbers (larger range, better precision)	✓
`char`	Characters	`'a'`, `'{'`, `'\t'`
`bool`	Logic values (true/false)	`true`, `false`
`string`	Character strings (`string` library)	`"Hello world!"`
`void`	*Nothing!* No type, no data (*functions*)	

char

Characters

1 byte

Value range: char set (ASCII)

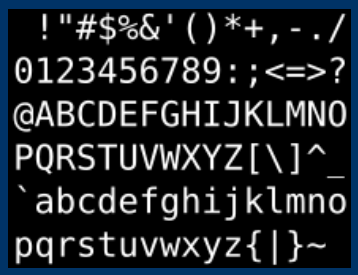
Literals:

```
'a'    '%'    '\t'
```

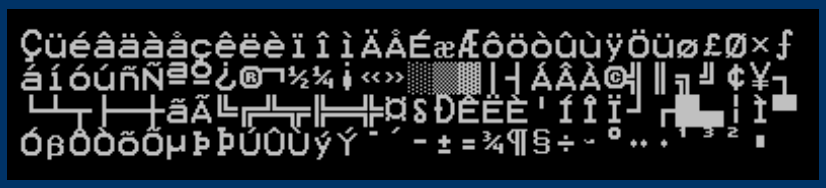
Backslash constants (or *escape sequences*):

Control characters

`'\t'` = tab `'\n'` = new line ...



ASCII (codes 32..127)



ISO-8859-1
(Extended ASCII: codes 128..255)

bool

Logic Values

Only two possible values:

- *True*
- *False*

Literals:

```
true    false
```

Number `0` is equivalent to `false`

Any other number is equivalent to `true`

Upper-case / Lower-case

C++ distinguishes between upper- and lower-case

C++ is case-sensitive...

`int`: C++ reserved keyword for declaring integer values

`Int`, `INT` or `inT` are not C++ reserved keywords

`true`: C++ reserved keyword for *true* value

`True` or `TRUE` are not C++ reserved keywords

`string` *Character String*

```
"Hello"   "Input numerator: "   "X142FG5TX?%A"
```

Character sequences

Programs with `string` variables must include the `string` library:

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

Typographic quotes (open/close) “...” are NOT ok
Be sure to use straight quotes: "..."

Basic Data Types: Example

types.cpp

```
#include <iostream>
#include <string>
using namespace std; // Just one using... for both libraries

int main() {
   int integer = 3; // Can assign (initialize) at declaration
   double real = 2.153;
   char ch = 'a';
   bool ok = true;
   string str = "Hello";
   cout << "Integer: " << integer << endl;
   cout << "Real: " << real << endl;
   cout << "Character: " << ch << endl;
   cout << "Boolean: " << ok << endl;
   cout << "String: " << str << endl;

   return 0;
}
```

```
D:\FP\Less02>types
Integer: 3
Real: 2.153
Character: a
Boolean: 1
String: Hello
```

How many numbers are there in the program?
Characters? Strings? Booleans?

Luis Hernández Yáñez

Type Modifiers

- `signed`/`unsigned`: with sign (default) / without sign
- `short`/`long` : shorter / larger value range

Type	Range
`int`	`-2147483648 .. 2147483647`
`unsigned int`	`0 .. 4294967295`
`short int`	`-32768 .. 32768`
`unsigned short int`	`0 .. 65535`
`long int`	`-2147483648 .. 2147483647`
`unsigned long int`	`0 .. 4294967295`
`double`	`+|- 2.23e-308 .. 1.79e+308`
`long double`	`+|- 3.37E-4932 .. 1.18E+4932`

Luis Hernández Yáñez

Variable Declaration and Use

Luis Hernández Yáñez

Variable Declaration

```
[modifiers] type variable_list;
```

Optional

variable_list


```
int i, j, k;
short int units;
unsigned short int coins;
double balance, profit, loss;
```

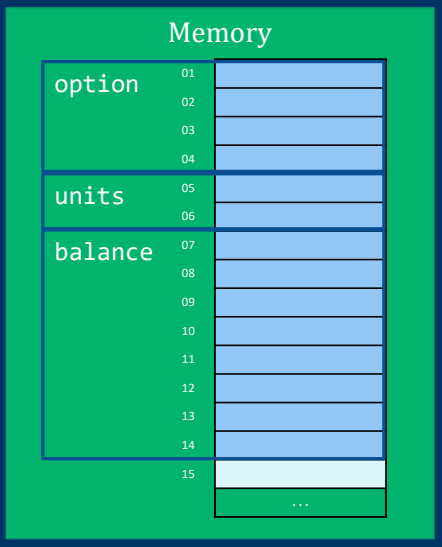
Programming with good style:
Descriptive identifiers
One space after each comma
Variable names in lower-case
Several words: capitalize each initial (`gainByMonth`)

Luis Hernández Yáñez

Data and Memory

Enough memory is reserved for each data type

```
int option;
short int units;
double balance;
```



Luis Hernández Yáñez

Variable Initialization

In C++ variables are not automatically initialized!

A variable must be initialized before being read!

How does a variable get a value?

- Reading from keyboard (`cin >>`)
- Assigning a value (assignment instruction)
- Assigning a value at declaration

Initialization when declaring the variable:

Expression: Compatible value

```
int i = 0, j, l = 26;
short int units = 100;
```

A literal data is also an expression

Luis Hernández Yáñez

Variable Use

Getting variable values

- ✓ A variable name in a expression is replaced with its value:

```
cout << balance; // Outputs balance's value
cout << rateByMonth * months / 100;
```

Changing variable values

- ✓ Variable name to the left of = operator

```
balance = 1214.5;
percentage = value / 30;
```

Variables must have been declared previously!

Luis Hernández Yáñez

Fundamentals of Programming I

Assignment Instructions

Luis Hernández Yáñez

Assignment Instructions

= operator

To the left, ALWAYS a variable!

The *expression* is evaluated, and the result is assigned...

```
int a, b = 2;
a = 23 + b * 5; // a gets the value 33
```

Luis Hernández Yáñez

Assignment Instructions

Errors...

👀 var = exp means var ← exp

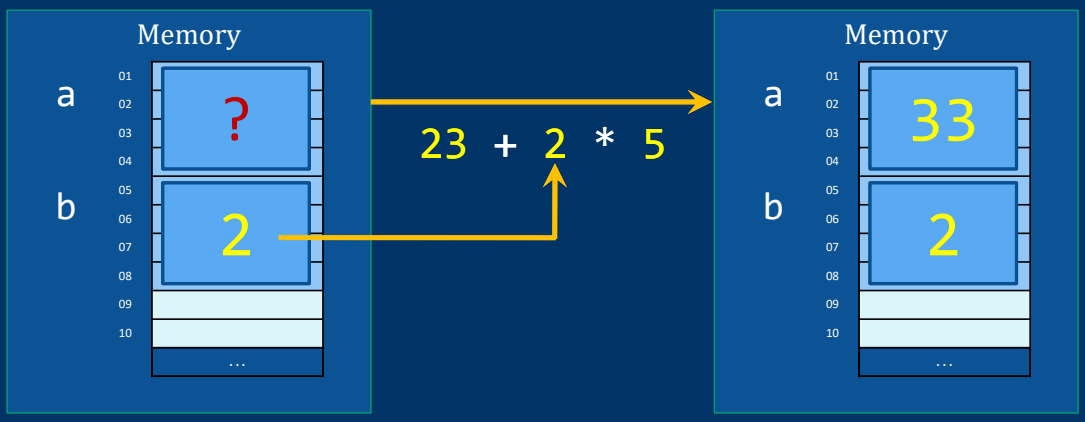
```
int a, b, c;
5 = a;
   // ERROR: A literal data can't receive a value
a + 23 = 5;
   // ERROR: There can't be an expression to the left
b = "abc";
   // ERROR: An integer can't receive a string
c = 23  5;
   // ERROR: Invalid expression (operator missing)
```

Luis Hernández Yáñez

Variables, Assignment, and Memory

```
int a, b = 2;
```

```
a = 23 + b * 5;
```

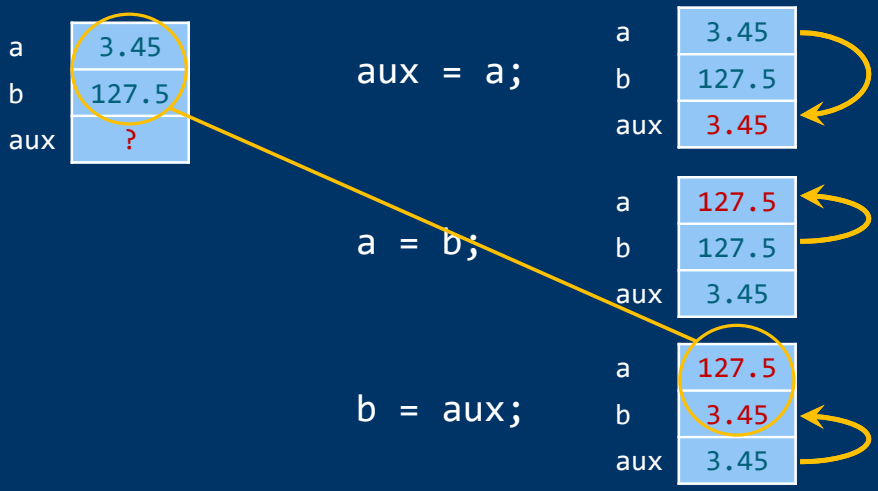


Luis Hernández Yáñez

Example: Value Exchange

We need an auxiliary variable...

```
double a = 3.45, b = 127.5, aux;
```



Luis Hernández Yáñez

Operators

Luis Hernández Yáñez

Operations with values

Each type defines possible operations with its values...

All data types:

— Assignment (=)

Numerical data types (`int`, `float`, `double`):

— Arithmetic operators (+, -, *, /, ...)

— Relational operators (less than, greater than, equal, ...)

Logic data type (`bool`):

— Logic operators (AND, OR, NOT)

Character data types (`char`, `string`):

— Relational operators (less than, greater than, equal, ...)

Luis Hernández Yáñez

Arithmetic Operators

Operators for numerical data types

Operator	Operands	Position	`int`	`float` / `double`
-	1 (unary)	Prefix	Sign Change	
+	2 (binary)	Infix	Addition	
-	2 (binary)	Infix	Subtraction	
*	2 (binary)	Infix	Multiplication	
/	2 (binary)	Infix	Int. Division	Real Division
%	2 (binary)	Infix	Modulus	Not applicable
++	1 (unary)	Prefix / postfix	Increment	
--	1 (unary)	Prefix / postfix	Decrement	

Luis Hernández Yáñez

Arithmetic Operators

Unary operators and binary operators

Unary Operators

- Sign Change (`-`):
  Preceding a variable, constant or expression in parenthesis

```
-balance      -RATIO      -(3 * a - b)
```

- Increment/decrement (only variables) (prefix/postfix):

```
++ratio      --months      j++      // 1 more or 1 less
```

Binary Operators

- Left Operand Operator Right Operand
  Operands: literal data, constants, variables or expressions

```
2 + 3      a * RATIO      -a + b      (a % b) * (c / d)
```

Luis Hernández Yáñez

Arithmetic Operators

Integer division or real division?

/

Both operands are integer numbers: integer division

```
int i = 23, j = 2;
cout << i / j; // Outputs 11
```

Any operand is a real number: real division

```
int i = 23;
double j = 2;
cout << i / j; // Outputs 11.5
```

Luis Hernández Yáñez

Arithmetic Operators

Modulus (integer division remainder)

%

Both operands must be integer numbers

```
int i = 123, j = 5;
cout << i % j; // Outputs 3
```

Integer division:

No decimals are kept → There may be a remainder

Luis Hernández Yáñez

Arithmetic Operators

Increment and decrement operators

++ --

Increases or decreases the numerical variable in one unit

Prefix: Before its value is accessed

```
i=i+1;
j=i;
```

```
int i = 10, j;
j = ++i; // Increments before assignment
cout << i << " - " << j; // Outputs 11 - 11
```

Postfix: After its value is accessed

```
j=i;
i=i+1;
```

```
int i = 10, j;
j = i++; // Assigns and then increments
cout << i << " - " << j; // Outputs 11 - 10
```

Don't mix ++ and -- with other operators

Luis Hernández Yáñez

Arithmetic Operators: Example

operators.cpp

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

int main() {
   int int1 = 15, int2 = 4;
   double real1 = 15.0, real2 = 4.0;
   cout << "Operations with numbers 15 y 4:" << endl;
   cout << "Integer division (/): " << int1 / int2 << endl;
   cout << "Modulus (%): " << int1 % int2 << endl;
   cout << "Real division (/): " << real1 / real2 << endl;
   cout << "Num = " << real1 << endl;
   real1 = -real1;
   cout << "Sign change (-): " << real1 << endl;
   real1 = -real1;
   cout << "Another sign change (-): " << real1 << endl;
   cout << "Incremented first (++ prefix): " << ++real1 << endl;
   cout << "Shown before being incremented (postfix ++): "
        << real1++ << endl;
   cout << "Already incremented: " << real1 << endl;
   return 0;
}
```

Luis Hernández Yáñez

More about Expressions

Luis Hernández Yáñez

Evaluation Order

In which order are the operators evaluated?

Left to right?

Right to left?

Some before others?

```
3 + 5 * 2 / 2 - 1
```

Operator Precedence (*priority*):

Those with higher precedence are evaluated first

And those with the same precedence?

Generally they are evaluated left to right

Parenthesis: force to evaluate its content first

Luis Hernández Yáñez

BY NC SA

Operator Precedence

Precedence

Higher Priority

Lower Priority

Operators
++ -- (postfix)
++ -- (prefix) - (sign change)
* / %
+ -

```
3 + 5 * 2 / 2 - 1  → 3 + 10 / 2 - 1  → 3 + 5 - 1  → 8 - 1 → 7
```

Same precedence: Left first

Higher Precedence

Same precedence: Left first

Luis Hernández Yáñez

Expression Evaluation

```
((3 + 5) * 4 + 12) / 4 - (3 * 2 - 1)
```

Parenthesis first...

```
(8 * 4 + 12) / 4 -        (6 - 1)
```

* before -

```
(32 + 12) / 4 -           5
```

* before +

```
44 / 4 - 5
```

/ before -

```
11 - 5
```

```
6
```

Leave a space before and after any binary operator

Luis Hernández Yáñez

A Formula

formula.cpp

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

int main() {
  double x, f;
  cout << "Introduce X value: ";
  cin >> x;
  f = 3 * x * x / 5 + 6 * x / 7 - 3;
  cout << "f(x) = " << f << endl;

  return 0;
}
```

$$f(x) = \frac{3x^2}{5} + \frac{6x}{7} - 3$$

Use parentheses to improve legibility:

```
f = (3 * x * x / 5) + (6 * x / 7) - 3;
```


Arithmetic Shortcuts

variable = variable operator right_operand;

Same ≡

variable operator= right_operand;

Assignment	*Shortcut*
`a = a + 12;`	`a += 12;`
`a = a * 3;`	`a *= 3;`
`a = a - 5;`	`a -= 5;`
`a = a / 37;`	`a /= 37;`
`a = a % b;`	`a %= b;`

Same precedence as assignment

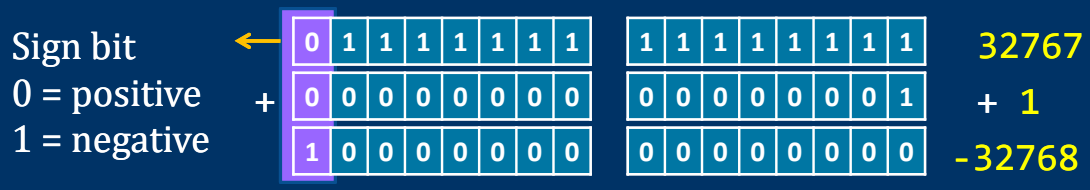
If not sure how to use them it's better to avoid them!

Overflow

Next value greater than the maximum?

Or next value less than the minimum

```
short int i = 32767; // Maximum value for short int
i++; // 32768 doesn't fit in a short int
cout << i; // Outputs -32768
```



Fundamentals of Programming I

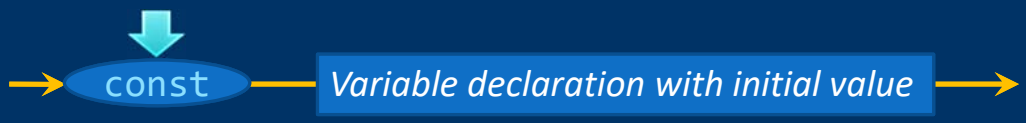
Constants

Constants

Constant declaration

`const` Access Modifier

Initialized variables we don't let them change




```
const short int Months = 12;
const double Pi = 3.141592, RATIO = 2.179 * Pi;
```

A constant can not appear to the left of an = operator after it is declared

Programming with good style: Capitalize constants' first letter or the whole name

Why Use Constants?

Better code legibility...

```
populationVariation = (0.1758 - 0.1257) * population;
```

VS.

```
populationVariation = (BirthRate - DeadRate) * population;
```

Easier code modification...

```
double net1 = sale1 * 21 / 100;
double net2 = sale2 * 21 / 100;
double total = net1 + net2;
cout << total << " (VAT: " << 21 << "%)" << endl;
```

3 changes

```
const int VAT = 21;
double net1 = sale1 * VAT / 100;
double net2 = sale2 * VAT / 100;
double total = net1 + net2;
cout << total << " (VAT: " << VAT << "%)" << endl;
```

VAT change to 22%?

1 change

Constants: Example

constants.cpp

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

int main() {
   const double Pi = 3.141592;
   double radius = 12.2, circumference;
   circumference = 2 * Pi * radius;
   cout << "Circumference of a circle of radius "
        << radius << ": " << circumference << endl;
   const double Euler = 2.718281828459; // e number
   cout << "Squared e number: " << Euler * Euler << endl;
   const int VAT = 21;
   int quantity = 12;
   double price = 39.95, net, qVAT, total;
   net = quantity * price;
   qVAT = net * VAT / 100;
   total = net + qVAT;
   cout << "Total shopping: " << total << endl;
   return 0;
}
```


Fundamentals of Programming I

Functions: cmath Library

Mathematical Functions

`#include <cmath>`

Some ...	`abs(x)`	Absolute value of x
	`pow(x, y)`	x raised to y
	`sqrt(x)`	Squared root of x
	`ceil(x)`	Lesser integer number greater than or equal to x
	`floor(x)`	Higher integer number less than or equal to x
	`exp(x)`	e^x
	`log(x)`	Ln x (natural logarithm of x)
	`log10(x)`	10 base logarithm of x
	`sin(x)`	Sine of x
	`cos(x)`	Cosine of x
	`tan(x)`	Tangent of x
	`trunc(x)`	Decimal part lose(integer part)

cmath Library

`math.cpp`

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

int main() {
  double x, y, f;
  cout << "Value for X: ";
  cin >> x;
  cout << "Value for Y: ";
  cin >> y;
  f = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))
        / abs(x * y) - cos(y);
  cout << "f(x, y) = " << f << endl;
  return 0;
}
```

$$f(x,y) = 2x^5 + \frac{\sqrt{\frac{x^3}{y^2}}}{|x \times y|} - \cos(y)$$

`pow()` with integer arguments:
Use `double()` cast:
`pow(double(i), 5)`

Leave a space after every comma in argument lists

Operations with Characters

Luis Hernández Yáñez

Operations with Characters

`#include <cctype>`

Assignment, `++`/`--` and relational operators

Functions for characters (`cctype` library)

`isalnum(c)`	`true` if `c` is a letter or a digit
`isalpha(c)`	`true` if `c` is a letter
`isdigit(c)`	`true` if `c` is a digit
`islower(c)`	`true` if `c` is a lowercase letter
`isupper(c)`	`true` if `c` is an uppercase letter

`false` in any other case

`toupper(c)`	returns `c` in uppercase
`tolower(c)`	returns `c` in lowercase

…

Luis Hernández Yáñez

Operations with Characters

characters.cpp

```
int main() {
   char char1 = 'A', char2 = '1', char3 = '&';
   cout << "Character 1 (" << char1 << ").-" << endl;
   cout << "Alphanumeric? " << isalnum(char1) << endl;
   cout << "Letter? " << isalpha(char1) << endl;
   cout << "Digit? " << isdigit(char1) << endl;
   cout << "Uppercase? " << isupper(char1) << endl;
   char1 = tolower(char1);
   cout << "In lowercase: " << char1 << endl;
   cout << "Character 2 (" << char2 << ").-" << endl;
   cout << "Letter? " << isalpha(char2) << endl;
   cout << "Digit? " << isdigit(char2) << endl;
   cout << "Character 3 (" << char3 << ").-" << endl;
   cout << "Alphanumeric? " << isalnum(char3) << endl;
   cout << "Letter? " << isalpha(char3) << endl;
   cout << "Digit? " << isdigit(char3) << endl;
   return 0;
}
```

1 ≡ true / 0 ≡ false

Luis Hernández Yáñez

Fundamentals of Programming I

Relational Operators
(simple conditions)

Luis Hernández Yáñez

Logical Expressions (*Boolean*)

Relational operators

Comparisons (conditions)

Simple Condition ::= *Expression Relational_Operator Expression*

Expressions with compatible types

Result: `bool` (`true` or `false`)

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Operators (priority)
...
* / %
+ -
< <= > >=
== !=
= += -= *= /= %=

Relational Operators

Lower priority than + - * / %

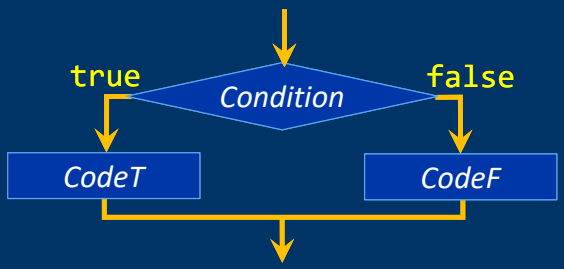
```
bool result;
int a = 2, b = 3, c = 4;
result = a < 5;                 // 2 < 5 → true
result = a * b + c >= 12;       // 10 >= 12 → false
result = a * (b + c) >= 12;     // 14 >= 12 → true
result = a != b;                // 2 != 3 → true
result = a * b > c + 5;         // 6 > 9 → false
result = a + b == c + 1;        // 5 == 5 → true
```

👀 Don't confuse the equality operator (==) with the assignment operator (=)

Decision Making (`if`)

Luis Hernández Yáñez

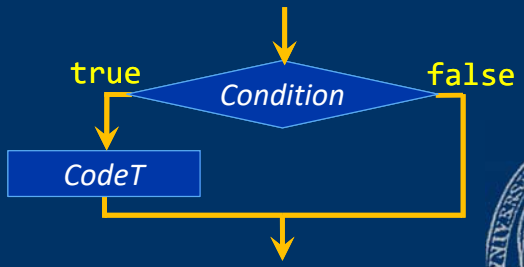
Selection: Conditional branching




```
if (condition) {
    codeT
}
else {
    codeF
}
```

Optional:
There may be no else clause

```
int num;
cout << "Number: ";
cin >> num;
if (num % 2 == 0) {
    cout << num << " is an even number";
}
else {
    cout << num << " is an odd number";
}
```



Luis Hernández Yáñez

if Instruction

selection.cpp

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

int main() {
   int op1 = 13, op2 = 4;
   int option;
   cout << "1 - Sum" << endl;
   cout << "2 - Product" << endl;
   cout << "Option: ";
   cin >> option;
   if (option == 1) {
      cout << op1 + op2 << endl;
   }
   else {
      cout << op1 * op2 << endl;
   }
   return 0;
}
```

```
D:\FP\Less02>selection
1 - Sum
2 - Product
Option: 1
17

D:\FP\Less02>selection
1 - Sum
2 - Product
Option: 2
52
```

Luis Hernández Yáñez

Fundamentals of Programming I

Code Blocks

Luis Hernández Yáñez

Code Blocks

Grouping instructions...

Sequence of instructions to be executed in an `if` branch


```
{
    instruction1
    instruction2
    ...
    instructionN
}
```

Tab or 3 spaces

```
int num, total = 0;
cin >> num;
if (num > 0)
{
   cout << "Positive";
   total = total + num;
}
cout << endl;
```

Local Scope (local declarations)

Code Blocks

Position of braces: a matter of style

```
if (num > 0)
{
   cout << "Positive";
   total = total + num;
}
cout << endl;
```

```
if (num > 0) {
   cout << "Positive";
   total = total + num;
}
cout << endl;
```

We don't need braces for a single instruction

```
if (num > 0) {
   cout << "Positive";
}
```

≡

```
if (num > 0)
   cout << "Positive";
```

Avoid writing the `if` and the target instruction in the same line:

```
if (num > 0) cout << "Positive";
```

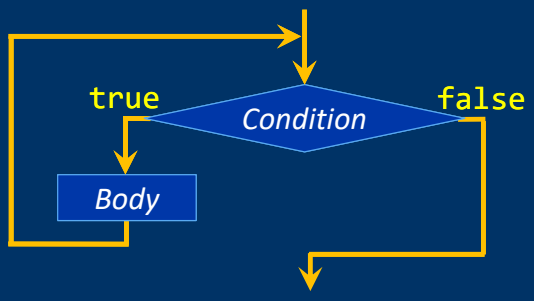
Loops (`while`)

Luis Hernández Yáñez

While the Condition is True Repeat...

Conditional repetition (iteration)

Code block




```
while (condition) {
⟼ body
}
```

If the condition is `false` at the beginning, the body is never executed

Luis Hernández Yáñez

BY NC SA

while Instruction

series.cpp

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

int main() {
   int i = 1, n = 0, sum = 0;
   while (n <= 0) { // Only a positive number
      cout << "How many numbers do you want to add? ";
      cin >> n;
   }
   while (i <= n) {
      sum = sum + i;
      i++;
   }
   cout << "Sum of i (1 to " << n << ") = " << sum << endl;
   return 0;
}
```

$$\sum_{i=1}^{n} i$$

```
How many numbers do you want to add? -3
How many numbers do you want to add? 0
How many numbers do you want to add? 5
Sum of i (1 to 5) = 15
```

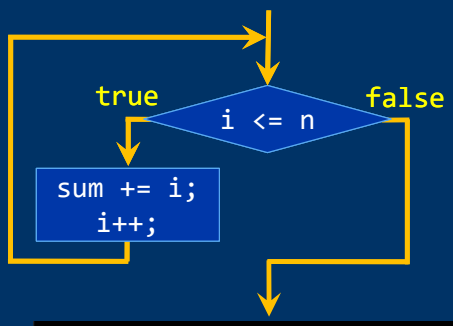
Luis Hernández Yáñez

while Instruction

Conditional iteration

```
while (i <= n) {
   sum = sum + i;
   i++;
}
```

$$\sum_{i=1}^{n} i$$



n	5
i	6
sum	15

```
Sum of i (1 to 5) = 15
```

Luis Hernández Yáñez

Console Input/Output

Luis Hernández Yáñez

Console Input/Output (keyboard/screen)

Text Streams

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

- Connects program execution with I/O devices
- Character sequences
- Input from keyboard: input stream `cin` (`istream` type)
- Output to screen: output stream `cout` (`ostream` type)

`iostream` library with namespace `std`

Luis Hernández Yáñez

BY NC SA

Input from Keyboard

First *blank spaces* are skipped: spaces, tabs, new lines

- `char` One single character is read
- `int`
  Digits are read and converted to the integer value to be assigned
- `float`/`double`
  Digits, and maybe a decimal point and more digits, are read and converted to the real value to be assigned
- `bool` If `0` is read, `false` is assigned; any other values assign `true`

Read each variable in a separate instruction / line!

Be user friendly! Tell the user what are you asking for!

```
cout << "Please enter your age: ";
cin >> age;
```

Luis Hernández Yáñez

Reading strings

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

`cin >> str` ends input when a blank space is found

`cin.sync()` discards any pending input

```
string name, surname;
cout << "Name: ";
cin >> name;
cout << "Surname: ";
cin >> surname;
cout << "Full name: "
     << name << " "
     << surname << endl;
```

```
Name: John Fitzgerald
Surname: Full name: John Fitzgerald
```

surname gets `"Fitzgerald"`

```
string name, surname;
cout << "Name: ";
cin >> name;
cin.sync();
cout << "Surname: ";
cin >> surname;
cout << ...
```

```
Name: John Fitzgerald
Surname: Kennedy
Full name: John Kennedy
```

How to read several words?

Next page...

Luis Hernández Yáñez

Input from Keyboard

Reading without skipping initial blank spaces

Calling functions with the dot (`.`) operator:

The dot operator allows a function to be called on a specific variable

`variable.function(arguments)`

Reading a character without skipping blank spaces:

```
cin.get(c); // Reads the very next character
```

Reading a string without skipping blank spaces:

```
getline(cin, str);
```

Reads everything until the end of the line (Enter) is found

Remember: *Blank spaces* include spaces, tabs and Enter

Output to Screen

Text representation of data

```
int month = 7;
cout << "Total: " << 123.45 << endl << " Month: " << month;
```

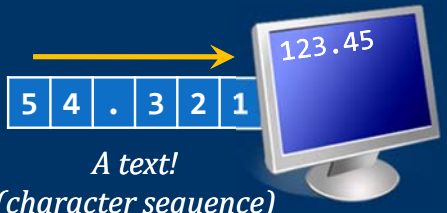
The `double` value `123.45` is stored in memory binary-coded

Its text representation is: `'1' '2' '3' '.' '4' '5'`

```
double d = 123.45;
```

A number!

```
cout << d;
```



A text!
(character sequence)

`iostream` library defines `endl` constant as a line feed

Output to Screen


```
int month = 7;
cout << "Total: " << 123.45 << endl << " Month: " << month;
```

```
cout        << 123.45 << endl << " Month: " << month;
cout        << endl << " Month: " << month;
cout        << " Month: " << month;
cout           << month;
```

```
Total: 123.45
Month: 7
```


Formatting Output

`#include <iomanip>`

Constants and functions for `cout` to adjust output format

Library	Constant/function	Purpose
`iostream`	`showpoint` / `noshowpoint`	Show (or not) the decimal point for numbers without decimals (`34.0`)
	`fixed`	Fixed point notation (real numbers) (`123.5`)
	`scientific`	Scientific notation (`1.235E+2`)
	`boolalpha`	Show `bool` values as `true` / `false`
	`left` / `right`	Align to the left / to the right (default)
`iomanip`	`setw(`*`width`*`)`*	Number of screen positions to be filled
	`setprecision(`*`p`*`)`	Precision: total number of digits With `fixed` or `scientific`, nr. of decimals

*`setw()` only affects the following output piece of data;
the others affect all the following data

Formatting Output

```
bool end = false;
cout << end << "->" << boolalpha << end << endl;
double d = 123.45;
char c = 'x';
int i = 62;
cout << d << c << i << endl;
cout << "|" << setw(8) << d << "|" << endl;
cout << "|" << left << setw(8) << d << "|" << endl;
cout << "|" << setw(4) << c << "|" << endl;
cout << "|" << right << setw(5) << i << "|" << endl;
double e = 96;
cout << e << " - " << showpoint << e << endl;
cout << scientific << d << endl;
cout << fixed << setprecision(8) << d << endl;
```

```
0->false


123.45x62
|  123.45|
|123.45  |
|x   |
|   62|

96 - 96.0000
1.234500e+002
123.45000000
```


Fundamentals of Programming I

User-Defined Functions

Functions in C++

Programs can include other functions besides `main()`

General form of a function in C++:

```
type name(parameters) // Heading
{
    // Body
}
```

- ✓ *Type* of data the function returns as a result
- ✓ *Parameters* for providing data to the function
  Variable declarations separated by commas
- ✓ *Body*: sequence of declarations and instructions A code block!

Function Data

- ✓ Local data: declared in function body
  Auxiliary data the function uses (there can be none)
- ✓ Parameters: declared in function heading
  Input data for the function

Both of exclusive use for the function, and not known outside

```
double formula(double x, double y) {
   // Declaration of local data:
   double result;

   // Instructions:
   result = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))
            / abs(x * y) - cos(y);

   return result; // Returns result and ends...
}
```

$$f(x,y) = 2x^5 + \frac{\sqrt{\frac{x^3}{y^2}}}{|x \times y|} - \cos(y)$$

Arguments

Calling a Function with Parameters

Name(*Arguments*)

```
double formula(double x, double y)
```

Function call:

- As many arguments between parenthesis as parameters
- Same order as declared parameters
- Each argument: Same type as corresponding parameter
- Each argument: A valid expression (result is passed)

  Expression values are copied to corresponding parameters

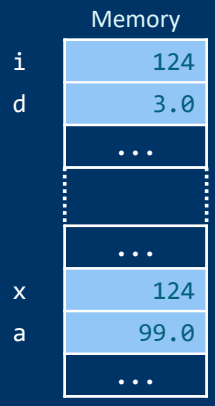
Function calls: in expressions ...

```
double value = formula(2, 3);
```


Argument Passing

Arguments are copied to parameters

```
int function(int x, double a) {
   ...
}

int main() {
   int i = 124;
   double d = 3;
   funcion(i, 33 * d);
   ...

   return 0; // main() returns 0 to O.S.
}
```



Arguments are not modified

Function Result

Functions must return a result

Function ends its execution returning a result

`return` instruction *(only one in each function)*

- Returns the value that follows (same type as the function)
- Ends function execution

Returned data substitutes function calling:

```
int sqr(int x) {
   return x * x;
   x = x * x;
}
```

This instruction will never execute

```
int main() {
   cout << 2 * sqr(16);
   return 0;
}
```

256

Luis Hernández Yáñez

Function Prototypes

Which functions are in the program?

We will write the functions after `main()`

Are program function callings correct?

- Does the function exist?
- Is there correspondence between arguments and parameters?

→ Prototypes after library inclusion

Function prototype: Function heading terminated with `;`

```
double formula(int x, int y);
int function(int x, double a);
int sqr(int x);
...
```

`main()` is the only function that doesn't need a prototype

Luis Hernández Yáñez

A Program with Functions

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

// Function Prototypes (except main())
bool even(int num);
bool letter(char ch);
int summation(int num);
double formula(double x, double y);

int main() {
   int number, sum, x, y;
   char character;
   double f;

   cout << "Integer: ";
   cin >> number;
```

.../...

A Program with Functions

```
   if (even(number))
      cout << "Even number";
   else
      cout << "Odd number";
   cout << endl;
   if (number > 1)
      cout << "Summation from 1 to " << number << ": "
           << summation(number) << endl;
   cout << "Character: ";
   cin >> character;
   if (!letter(character))
      cout << "not ";
   cout << "a letter" << endl;
   cout << "f(x, y) = " << formula(x, y) << endl;
   // Arguments can have, or not, same names as parameters

   return 0;
}
```

.../...

A Program with Functions

```
// Implementation of defined functions

bool even(int num) {
   bool isEven;

   if (num % 2 == 0)
      isEven = true;
   else
      isEven = false;

   return isEven;
}
```

Luis Hernández Yáñez

.../...

A Program with Functions

```
bool letter(char ch) {
   bool isLetter;

   if ((ch >= 'a') && (ch <= 'z') || (ch >= 'A') && (ch <= 'Z'))
      isLetter = true;
   else
      isLetter = false;

   return isLetter;
}
```

Luis Hernández Yáñez

.../...

A Program with Functions

functions.cpp

```
int summation(int num) {
   int sum = 0, i = 1;

   while (i < num) {
      sum = sum + i;
      i++;
   }

   return sum;
}

double formula(double x, double y) {
   double f;

   f = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))
         / abs(x * y) - cos(y);

   return f;
}
```


Promote Open Culture!

Creative Commons License

Attribution
You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Non commercial
You may not use this work for commercial purposes.

Share alike
If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

Click on the upper right image to learn more...