Programming 1

Lecture 2
Simple Data Types
Objectives

- Understand the use of data in a program
- Know the simple data types in a programming language
- Learn to manage, read and display the simple data types in the C language (C++)
Topics

1. **Data types in a program**
2. Variable and constant data
3. Managing variables and constants in a program
4. Assignment statement
5. Arithmetic and logic expressions
6. Data input and output statements
7. General structure of a program
8. Information sources
Data in a program

- **Datum**: fact or value from which a conclusion can be inferred (information)

- **Data in a program**: data which the computer operates with
  - **Input data** are a starting point to obtain knowledge (**output data**)
  - The program may also need **auxiliary data** (internal) to obtain the result
Example: data in a program

- **Program**: calculate the arithmetic mean of \( n \) numbers
- **Input data**: a set of \( n \) numbers
- **Output data**: arithmetic mean of the \( n \) numbers
- **Auxiliary data**:
  - Addition of the numbers
  - Amount of numbers

\[
\begin{array}{c}
\text{INPUT DATA} \\
3 \quad 7 \quad 15 \quad 4 \quad 23
\end{array}
\]

\[
\begin{array}{c}
\text{PROGRAM to} \\
\text{calculate the} \\
\text{arithmetic mean of} \\
\text{of} \ n \ \text{numbers}
\end{array}
\]

\[
\begin{array}{c}
\text{AUXILIARY DATA} \\
52 \quad 5
\end{array}
\]

\[
\begin{array}{c}
\text{OUTPUT DATA} \\
10,4
\end{array}
\]
Data Types in a Program

- **Data Type** = Values + Operations
  - Set of *values* that a datum can take in the program
    - If a value not included in the set is tried to be given, the program must produce an error message
  - Set of *operations* that can be defined with the data
Example: Data Type

• **Boolean Data Type**
  • Values = \{true, false\}
  • Operations = \{and, or, not\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>not a</th>
<th>a and b</th>
<th>a or b</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td></td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Simple Data Type

- They are elementary types that are not derived from other types
- Each particular value of a simple data type is specified by a literal
  - For instance, the integer literals can be expressed as a:
    - Decimal (base 10): 255
    - Octal (base 8): 0377 (3*8^2+7*8^1+7=255)
    - Hexadecimal (base 16): 0xff (15*16^1+15=255)
# Predefined Simple Data Types in C

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>character</td>
<td>1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>unsigned character</td>
<td>1</td>
</tr>
<tr>
<td><strong>Integer</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>2-4</td>
</tr>
<tr>
<td>short</td>
<td>short integer</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>long integer</td>
<td>4</td>
</tr>
<tr>
<td>long long</td>
<td>long integer</td>
<td>8</td>
</tr>
<tr>
<td>unsigned</td>
<td>unsigned integer</td>
<td>2-4</td>
</tr>
<tr>
<td>unsigned short</td>
<td>unsigned short integer</td>
<td>2</td>
</tr>
<tr>
<td>unsigned long</td>
<td>unsigned long integer</td>
<td>4</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>unsigned long integer</td>
<td>8</td>
</tr>
<tr>
<td><strong>Real</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>floating point (real numbers)</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>double precision floating point</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>extended double precision floating point</td>
<td>16</td>
</tr>
<tr>
<td><strong>Boolean</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

The bool type doesn't exist in standard ANSI C (It is emulated using int type: 0 means false; ≠ 0 means true. We will use bool type (belonging to C++ and standard C99)
## Values for Simple Data Types in C

<table>
<thead>
<tr>
<th>Type</th>
<th>Bytes</th>
<th>Meaning</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>Alphabetic: 'a', 'b', …'z'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>'A', 'B', …'Z'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Digits: '0', '1', '2', …'9'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Special: '+' , '-' , '/' , '=' , '(' ...</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>-2.147.483.647 .. 2.147.483.647</td>
<td></td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-32.767 .. 32.767</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>Approx. 10^{-38} .. 10^{38}</td>
<td>7 digits</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>Approx. 10^{-308} .. 10^{308}</td>
<td>15 digits</td>
</tr>
<tr>
<td>bool</td>
<td>1</td>
<td>true, false</td>
<td></td>
</tr>
</tbody>
</table>
User-defined Data Types

• Generally, programming languages have:
  • Predefined Data Types
  • User-defined Data Types

• In C language:
  • The user can define enumerated data types made up of a set of identifiers representing an integer value.
  • There is no impression format for these types. The first element has an associated value of 0, the second element of 1 and so on.

```c
enum T_WeekDay {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};
enum T_Primary_Colour {red, green, blue};
```
Topics

1. Data types in a program
2. **Variable and constant data**
3. Managing variables and constants in a program
4. Assignment statement
5. Arithmetic and logic expressions
6. Data input and output statements
7. General structure of a program
8. Information sources
Variables and Constants in a Program

• Common features:
  • They both represent data in a program
  • They use a memory space, reserved to store a value of a data type
  • They are identified by a name

• Differences:
  • The value of a variable can change during the program execution
  • The value of a constant never changes during the program execution
Representation of variables in the memory

- The memory is a list of numerated positions (bytes)

- A variable represents a portion of the memory made up of a consecutive amount of bytes

- A variable in the memory is determined by:
  - The **address**: place of the first byte dedicated to this variable
  - The **type**: it determines how many bytes are required to store this variable
A football club X, owner of a stadium Y, needs to calculate the income of each match played in its stadium. There are three types of tickets, depending on the kind of seat: back seats (stands behind the goals), general seats (uncovered side stands) and preferential seats (covered side stands). Throughout the season, the price of a back seat ticket is half of a general ticket, and the preferential ticket price is double of a general one. For each match, the football club sets a price for the general tickets, and it also establishes discounts for children (80%) and pensioners (50%) for the whole season.

- What would you use to store …
- … the price of a general ticket? Real Variable
- … the discount for children? Constant
- … the amount of sold preferential tickets? Integer Variable
- … the type of a ticket?
- … whether a discount can be applied or not?
- … the price of a preferential ticket?
Identifiers

• An **identifier** is a name used by the programmer to make reference to the data and other program elements

• General rules to construct an identifier:
  
  1. It must be significant
  2. It cannot match reserved words belonging to the programming language
  3. Its length should not be too long
  4. It should begin with an alphabetic character or the *underscore* symbol. It can contain alphabetic characters, digits and the *underscore* symbol
  5. It cannot be accentuated
  6. Depending on the programming language, it may be used interchangeably or not, in uppercase or lowercase

**C and C++ are case sensitive**
Variable and Constant Identifiers

• Widespread notations among most programmers:
  1. Variables in lowercase, constants in uppercase
  2. Identifiers made up of several words:
     − Lowercase, separating words with underscores
       `nombre_alumno`
     − Uppercase, separating words with underscores
       `NOMBRE_ALUMNO`
     − Initials in uppercase, the remaining lowercase
       `NombreAlumno`
     − Same length words
       `nom_alu`

**Important**: don't change notation arbitrarily. Follow only one notation to keep your programs consistent, readable and understandable
Identifiers: Example

Right

- distance
- euclidean_distance
- _date
- dateOfBirth
- NUMBER_PI
- number1
- number_2

Wrong

- euclidean-distance
- 3books
- Number$1
- More_questions?
- número

The following identifiers are different in C:

Car_colour  car_colour  CAR_COLOUR  car_Colour  Car_Colour
Topics

1. Data types in a program
2. Variable and constant data
3. Managing variables and constants in a program
4. Assignment statement
5. Arithmetic and logic expressions
6. Data input and output statements
7. General structure of a program
8. Information sources
Managing variables and constants

Step 1: Declaration
- Give them a name and determine the data type for the compiler to reserve space in the memory to store a value of this data type

Step 2: Initialization
- Assign the first value before using them

Step 3: Use
- Use them in the allowed places (sentences), following the syntactic rules of the programming language

Step 4: Destruction
- The compiler frees the space in memory that was previously reserved
- Usually, this is done by the compiler, but the programmer must take it in account, so that they are not used once destroyed
### Variable declaration in C

- A data type must be associated to the variable, so that it can store any value of this data type.

```
char  letter_dni;  // variable to store the letter of the dni of any person
int    pages;  // variable to store the amount of pages of any book
float  salary;  // variable to store the salary of any person
bool   passed;  // variable to store whether a student has passed an exam or not
```
Constant declaration and initialization in C/C++

In C language

It indicates that it is a constant declaration

#define identifier value

#define HOURS_DAY 24 // constant to store the number of hours in a day

In C++ language

Reserved word for constants in C++

const data type constant name = value;

const int HOURS_DAY = 24; // constant to store the number of hours in a day
Variable initialization in C

• Using the **assignment statement**

  ![Diagram of assignment statement]

  **Variable name** → **Assignment operator** → **Value assigned to the variable** → **‘;’ end of assignment statement**

**In C language**

```c
letter_dni = ‘A’;  // store the ‘A’ character in variable letter_dni
pages = 365;      // store the number 365 in variable pages
salary = 1000.20;  // store the real number 1000.20 in variable salary
passed = true;     // store the value true in variable passed
```

**Warning:** In C and C++, the variables can be initialized in the declaration. For example: `int pages = 365;`
Using variables and constants in C language

• A **variable** is used …
  • On the left side of an assignment statement
  • In an arithmetic of logic expression
  • In the input and output statements

• A **constant** is used …
  • In an arithmetic of logic expression
  • In an output statement

Why not a constant on the left side of an assignment?

Why not a constant in an input data statement?
Topics

1. Data types in a program
2. Variable and constant data
3. Managing variables and constants in a program
4. Assignment statement
5. Arithmetic and logic expressions
6. Data input and output statements
7. General structure of a program
8. Information sources
Assignment statement: syntax

In C language

```c
pages_bookA = 430;  // store the number 430 in variable pages_bookA, previously declared as type int
salary = 35616.44;  // store the real number 35616.44 in variable salary, previously declared as type float
```
How does the assignment statement work?

- Step 1: Evaluate the right side of the assignment operator
- Step 2: Assign the value of the right side to the left side of the assignment operator

In C language

```c
// suppose variables price_carA and price_carB declare as type float
price_carA = 10500.00; // store the number 10500.00 in variable price_carA
price_carB = 40200.00; // store the number 40200.00 in variable price_carB
final_price = price_carA + price_carB;
```

What is the value stored in variable final_price?
Topics

1. Data types in a program
2. Variable and constant data
3. Managing variables and constants in a program
4. Assignment statement
5. Arithmetic and logic expressions
6. Data input and output statements
7. General structure of a program
8. Information sources
Arithmetic and logic expressions

- An **expression** in a program is a combination of variables, constants, operators, brackets and function identifiers, whose evaluation produces a value
  - Expressions can be written in any place in the program in which its evaluation value can be used
- An **arithmetic expression** ...
  - Is built using arithmetic operators
  - Returns a numerical value

\[
\frac{x_{\text{rad}} \times 360}{2 \times \pi}
\]

*It calculates the degrees corresponding to a value in radians stored in variable \(x_{\text{rad}}\), using the constant \(\pi\)*

- A **logic expression** ...
  - Is built using relational and logical operators
  - Can also contain arithmetic operators
  - Returns a boolean value

\[(\text{year modulus 4} == 0) \text{ AND} (\text{NOT (year modulus 100} == 0) \text{ OR (year modulus 400} == 0) )\]
# Operators in C language

<table>
<thead>
<tr>
<th>Arithmetic operators</th>
<th>Meaning</th>
<th>Operand types</th>
<th>Result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>+       -   *   /</td>
<td>Addition, subtraction, multiplication, division</td>
<td>Integer or real</td>
<td>Integer or real</td>
</tr>
<tr>
<td>%</td>
<td>Division remainder</td>
<td>Integer</td>
<td>Integer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relational operators</th>
<th>Meaning</th>
<th>Operand types</th>
<th>Result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;   &gt;   &lt;=   &gt;=</td>
<td>Less than, greater than, less than or equal to, greater than or equal to</td>
<td>Simple types</td>
<td>Boolean</td>
</tr>
<tr>
<td>==   !=</td>
<td>Equal to, not equal to</td>
<td>Simple types</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logic operators</th>
<th>Meaning</th>
<th>Operand types</th>
<th>Result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>Logic AND</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Logic OR</td>
</tr>
<tr>
<td>!</td>
<td>Logic NOT</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

**Important:** The assignment operator '==' is different from the equality operator '=='. Very often some people wrongly use the operator '==' instead of '=='. It generates errors which are difficult to detect.

**Important:** Notice that the division operator '/' , when the operands are of integer type, returns the integer part of the division quotient. To obtain a result with a fractional part, some of the operands must be of real type.
Operator Precedence and Associativity

- The **precedence** or **priority** of an operator indicates the order in which the operations in an expression with several operands are executed.
- The **associativity** of an operator indicates the order in which the operations in an expression with several operands with the same priority are executed.

### Operator precedence in C language

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operators</th>
<th>Meaning</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>- !</td>
<td>Negative number, logic NOT</td>
<td>From right to left</td>
</tr>
<tr>
<td>2</td>
<td>* / %</td>
<td>Multiplication, division, division remainder</td>
<td>From left to right</td>
</tr>
<tr>
<td>3</td>
<td>+ -</td>
<td>Addition, subtraction</td>
<td>From left to right</td>
</tr>
<tr>
<td>4</td>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>Relational operators</td>
<td>From left to right</td>
</tr>
<tr>
<td>5</td>
<td>== !=</td>
<td>Equality operators</td>
<td>From left to right</td>
</tr>
<tr>
<td>6</td>
<td>&amp;&amp;</td>
<td>Logic AND</td>
<td>From left to right</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Always use brackets '( )' …
- when you have any doubt in the evaluation order
- to make the expression more readable
- to modify the evaluation order
Topics

1. Data types in a program
2. Variable and constant data
3. Managing variables and constants in a program
4. Assignment statement
5. Arithmetic and logic expressions
6. Data input and output statements
7. General structure of a program
8. Information sources
Data input and output statements

• The **input statements** allow the variables to store data that the user enters by keyboard

• The **output statements** allow the display of data on the screen

• Variables can be used in input statements

• Variables, constants and expressions in general can be used in output statements

The input and the output can be associated to several **sources** and **devices**, such as files, printers, touchscreens, a mouse and so on.

We will mainly use the **keyboard** and the **screen** as input and output of our programs. They are the default devices for input and output.
Output statement in C++

- It writes on the screen any combination of values of variables, constants, expressions and text strings

**Examples**

```cpp
cout << “The price of the laptop is “ << price << “ euros” << endl;
cout << “The total price is : “ << (price1 + price2);
cout << price;
cout << “this is a text string with no new line”;
cout << “this is a text string with new line\n”;
cout << endl;
cout << “\n”;
```

In C language, the data output is done by using the library function `printf()`. In our case, we prefer to use the statement `cout` of C++, because of its easier use.
Input statement in C++

- It stores in variables the values entered by keyboard

```
Examples

cout << "Enter your age:";
icin >> age; // age is a variable, declared as type int
cout << "Enter the marks for the two practice exams:";
icin >> mark1 >> mark2; // mark1 are mark2 are variables declared as type float or double
cout << "Would you like to enter more data? (Y/N) : ";
icin >> answer; // answer is a variable, declared as type char
```

In C language, the data input is done by using the library function `scanf()`. In our case, we prefer to use the statement `cin` of C++, because of its easier use.

`cin` ignores whitespace and the newline character.
Topics

1. Data types in a program
2. Variable and constant data
3. Managing variables and constants in a program
4. Assignment statement
5. Arithmetic and logic expressions
6. Data input and output statements
7. **General structure of a program**
8. Information sources
General structure of a program in C

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

const float PI = 3.1415926;

main() {
    float area, radio;
    cout << "Enter the circle radius:";
    cin >> radius;
    area = PI * radius * radius;
    cout << "The circle area is:" << area;
    cout << endl;
}
```

All the statements in C and C++ end with a **semicolon ';;'**
Topics

1. Data types in a program
2. Variable and constant data
3. Managing variables and constants in a program
4. Assignment statement
5. Arithmetic and logic expressions
6. Data input and output statements
7. General structure of a program
8. Information sources
Information sources

- Fundamentos de Programación
  Jesús Carretero, Félix García, y otros
  - Capítulo 2 (Apartados 2.4)
  - Capítulo 4 (Apartados 4.1; 4.2; 4.3; 4.4; 4.10)

- Problemas Resueltos de Programación en Lenguaje C
  Félix García, Alejandro Calderón, y otros
  - Capítulo 2 (Apartados 2.1; 2.2; 2.3)

- Resolución de Problemas con C++
  Walter Savitch
  - Capítulo 2