Programming 1

Lecture 3
Control Statements
Objectives

- Understand the concept of algorithm
- Understand the need to design algorithms in order to analyse and solve problems
- Understand and manage the different types of existing control statements in a structured programming language
- Understand the syntax and how the control statements work in C language
Topics

1. **Algorithms and Programs**
2. Algorithmic structures
3. Programming structures
4. Sequential statements
5. Selection statements
6. Iteration statements
7. Comments
8. Program trace
9. General structure of a program
10. Information sources
The concept of Algorithm

Algorithm

Ordered sequence of instructions to solve a problem in a finite number of steps

The algorithms are independent both from the *programming language* and the *computer* that executes them.
The concept of Program

Program

Set of ordered instructions (statements or sentences) written in a programming language for a computer to perform a particular task

The programs codify algorithms in a programming language, and are executed in a computer
State of a program

- The state of a program at a given instant is the value of each variable at this instant.

\[ \text{PROGRAM} = \text{State} + \text{Executable statements} \]

- Variables that store the state
- Statements that modify the state
Topics

1. Algorithms and Programs
2. Algorithmic structures
3. Programming structures
4. Sequential statements
5. Selection statements
6. Iteration statements
7. Comments
8. Program trace
9. General structure of a program
10. Information sources
Control flow of an algorithm

• The control flow defines the order followed by the algorithm instructions

• The control flow is determined by the several types of algorithmic structures
Types of algorithmic structures

- **Sequential**
  - Assignment
  - Input
  - Output

- **Selection**
  - Simple
  - Double
  - Multiple

- **Iteration**
  - Loop with initial condition
  - Loop with final condition
  - Loop with counter
Sequential structure

- The actions (instructions) are consecutively performed one after another.

1. Enter the current year by keyboard
2. Enter your birth year by keyboard
3. $age = \text{current\_year} - \text{birth\_year}$
4. Display on the screen the value of $age$
Selection structure

- It allows you to make decisions between different alternative actions depending on the value of a condition.

```
If the condition is true Then
   <action1>
Else
   <action2>
End_selection_statement
```

Diagram:
- If `age < 18` is true, display on the screen "You have a discount" and calculate `price = 5.0 * (18 - age) / 100`.
- If `age < 18` is false, display on the screen "You have no discount" and calculate `price = 5.0`.
- Display on the screen the price of the ticket.
Iterative structure

- It allows you to repeat actions depending on the value of a condition

```
While the condition is true Do
  <action>
End_Iterative_Statement
```

Display on the screen
“The capacity of the theatre is complete”
Topics

1. Algorithms and Programs
2. Algorithmic structures
3. **Programming structures**
4. Sequential statements
5. Selection statements
6. Iteration statements
7. Comments
8. Program trace
9. General structure of a program
10. Information sources
Execution flow of a program

- The execution flow defines the order in which the program statements are executed.
- The execution flow is determined by the different types of **programming structures** (control statements).
Types of control statements

Control statements

- Sequential
  - Assignment
  - Input
  - Output

- Selection
  - Simple
  - Double
  - Multiple
    - if
    - if-else
    - switch or nested if-else

- Iteration
  - Loop with initial condition
    - while
  - Loop with final condition
    - do-while
  - Loop with counter
    - for
Topics

1. Algorithms and Programs
2. Algorithmic structures
3. Programming structures
4. **Sequential statements**
5. Selection statements
6. Iteration statements
7. Comments
8. Program trace
9. General structure of a program
10. Information sources
Simple sequential statements

• Assignment statement

```
variable = value;
```

```
x = 20;
y = 3;
quotient = x / y;
remainder = x % y;
```

• Input statement (read data)

```
cin >> variable;
```

```
cin >> x;
cin >> y;
```

• Output statement (write data)

```
cout << datum;
```

```
cout << “This text is written on the screen”;
cout << quotient;
cout << “\n”;
```
Sequence of statements in C and C++ languages

- A sequence of statements is made up of \( N \) statements, \( N \geq 0 \)
- When \( N > 1 \), the sequence of statements must be written between curly brackets (\{\} )

```c
{ // beginning of the sequence
    cout << "Enter two integer numbers";
    cin >> x >> y;
    quotient = x / y;
    remainder = x % y;
    cout << "The quotient is: " << quotient << endl;
    cout << "The remainder is: " << remainder << endl;
} // end of sequence
```

Remember: all the statements in C and C++ end with a semicolon ;
Topics

1. Algorithms and Programs
2. Algorithmic structures
3. Programming structures
4. Sequential statements
5. Selection statements
6. Iteration statements
7. Comments
8. Program trace
9. General structure of a program
10. Information sources
Select one alternative: if statement

- Decide whether a sequence of statements is executed

```c
if (logic_expression) {
  sequence of statements
}
```

- If the result of evaluating `logic_expression` is **true** then the sequence of statements associated with the if statement is executed

- If the value of `logic_expression` is **false** then the sequence of statements is not executed and the statement following the if is executed

```c
if (speed > 120) {
  cout << "WARNING: you may be fined";
} // end of if statement
cout << "your current speed is: ";
cout << speed << endl;  
```

In C and C++, the **parentheses** enclosing the logical expression are **required**
Select between two alternatives: if-else statement

- Select between two different sequences of statements

```cpp
if (logic_expression) {
    sequence of statements 1
} else {
    sequence of statements 2
}
```

- If the value of `logic_expression` is `true` then the sequence of statements associated with the `if` is executed (sequence of statements 1)

- If the value of `logic_expression` is `false` then the sequence of statements associated with the `else` is executed (sequence of statements 2)

```cpp
if (number % 2 == 0) {
    cout << "the number is even";
} else {
    cout << "the number is odd";
} // end of if-else statement

cout << endl;
cout << "enter another number:";
```
Select from multiple alternatives: nested if-else statement

- Select from several sequences of statements

```cpp
if (logic_expression_1) {
    sequence of statements 1
}
else if (logic_expression_2) {
    sequence of statements 2
}
else if (logic_expression_3) {
    sequence of statements 3
}
```

- Only the sequence of statements associated with the first logic expression that is evaluated as true is executed

- If every logic expression is false
  - The sequence of statements following the whole nested if-else is executed
  - Unless the last alternative is associated with an else. In this case, this branch is executed.

```cpp
if (mark >= 9 && mark <= 10)
    cout << “your mark is SOBRESALIENTE”;
else if (mark >= 7 && mark < 9)
    cout << “your mark is NOTABLE”;
else if (mark >= 5 && mark < 7)
    cout << “your mark is APROBADO”;
else if (mark >= 0 && mark < 5)
    cout << “your mark is SUSPENSO”;
else // last alternative of nested if-else
    cout << “your mark is not correct”;
```

// here is the next sentence
// after the nested if-else
Select from multiple alternatives: switch statement

• Select from several sequences of statements

```c
switch (expression) {
  case value_1 : sequence of statements 1;
      break;
  case value_2 : sequence of statements 2;
      break;
  case value_3 : sequence of statements 3;
      break;
  default : sequence of statements 4;
}
```

• Only the sequence of statements associated with the case is executed, which has a value that corresponds to the result of the expression in switch.

• If the value of the expression in switch is not in any case, the sequence of statements associated with the branch default is executed (this branch is optional)

```c
switch (operator) {
  case '+' : res = x + y;  
      break;
  case '-' : res = x - y;
      break;
  case '*' : res = x * y;
      break;
  case '/' : res = x / y;
      break;
}  // end of switch statement

cout << “Result of the operation : ”;
cout << res;
```
1. Write a program to read two different numbers from the keyboard and display a text message on screen indicating which is the greatest one.

2. Write a program to read a figure in seconds and display this figure as hours, minutes and seconds.

3. Write a program to read the coordinates of three points in a plane and display whether they make up an equilateral triangle.

4. Write a program to display three menu options and let the user select one of them. After that, a message should appear on the screen that shows the selected option or an error message if the option is incorrect:

**Execution example 1**

1. Menu option 1
2. Menu option 2
3. Menu option 3

Enter a menu option: 2
The selected menu option is 2

**Execution example 2**

1. Menu option 1
2. Menu option 2
3. Menu option 3

Enter a menu option: 4
The selected menu option is incorrect
Topics

1. Algorithms and Programs
2. Algorithmic structures
3. Programming structures
4. Sequential statements
5. Selection statements
6. **Iteration statements**
7. Comments
8. Program trace
9. General structure of a program
10. Information sources
Loops

- A loop is a programming structure made up of a sequence of statements, called the **loop body**, that can be repeated several times.

- Each execution of the loop body is an **iteration**.

- The number of times the loop body is executed is controlled by a **condition** (logic expression).

- Therefore, when designing and implementing a loop, we must take into account two aspects:
  
  1. Which one should be the loop body?
  2. How many times must the loop body be iterated?
Loop types

- Depending on where the condition that controls the execution of the loop body is, the following types of loops can be distinguished:
  - Loops with initial condition
    - *While* statement
    - *For* statement (repeat with counter)
  - Loops with final condition
    - *Do-while* statement
Loops with initial condition: while statement

• Repeat zero or more times the execution of a sequence of statements while the condition is true

```c++
while (logic_expression) {
  sequence of statements
}
```

sweets = 0;
cout << “Do you want a sweet?:”;
cin >> answer;
while (answer == ‘Y’ || answer == ‘y’) {
  sweets=sweets+1;
  cout << “Do you want another sweet?:”;
  cin >> answer;
} // end of while statement

cout << “You have ”<< sweets <<” sweets”;
Loops with final condition: do-while statement

- Repeat once or more times the execution of a sequence of statements while the condition is true

\[
\text{do} \{ \\
    \text{sequence of statements} \\
\} \text{ while (logic_expression);}
\]

- First, the \textit{loop body} (sequence of statements) is executed, and then the \textit{logic_expression} is evaluated

- While the result of evaluating the \textit{logic_expression} is \textbf{true}, the sequence of statements (\textit{loop body}) will be executed repeatedly

```cpp
do  {
    cout << “Enter the menu option:”;
    cin >> option;
}  while (option < 1 || option > 4);
```
Loops with initial condition: for statement

- Repeat a given number of times the execution of a sequence of statements
- The number of the loop iterations is controlled by a variable, used as a counter

```c
for (counter initialization ; logic_expression ; counter increment) {
    sequence of statements
}
```

```c
for ( i = 1; i <= 10; i++) {
    cout << “This is the iteration number “ << i;
}
```

In C and C++, the `i++` statement is an assignment statement equivalent to `i = i+1`

The counter increment can be any other number than 1, for instance: `i = i+2; i = i*2; …`

The counter can also be decremented: `i = i-1`
How does the for statement work?

- Step 1: The counter initialization statement is executed (only once)
- Step 2: The logic expression is evaluated:
  - If its value is true, then the loop body is executed
  - If its value is false, then the for statement execution is finished
- Step 3: After executing the loop body, the counter increment statement is executed
- Step 4: Go back to step 2
The for loop vs. the while loop. Equivalence

- Any for loop can be rewritten as a while loop

```cpp
for (expression_1; expression_2; expression_3) {
    sequence of statements;
}
```

Can be rewritten as

```cpp
expression_1;
while (expression_2) {
    sequence of statements;
    expression_3;
}
```

```cpp
for (i = 1; i <= 10; i++) {
    cout << "This is the iteration num." << i;
}
```

Can be rewritten as

```cpp
i = 1;
while (i <= 10) {
    cout << "This is the iteration num." << i;
    i++;
}
```
What type of loop should I use?

Must the loop body be executed at least once?
- No
- Yes

Is the number of iterations controlled by a counter?
- No
- Yes

- No: while loop
- Yes: for loop
- do-while loop
1. Algorithms and Programs
2. Algorithmic structures
3. Programming structures
4. Sequential statements
5. Selection statements
6. Iteration statements
7. Comments
8. Program trace
9. General structure of a program
10. Information sources
Comments in the source code

- A comment is a short explanation to provide information for people who read the code
- They facilitate the program maintenance
- In C++ language, the symbols `//` are used
  - The text between `//` and the end of line is a comment

```c
float partial_mark; // mark of a partial exam (input datum)

// Calculate the average mark and display it on the screen
```

- In C and C++ languages, the symbols `/*` and `*/` are also used
  - The text between `/*` and `*/` is a comment

```c
/* Enter the marks of every partial exam and sum them up
   (only when the entered datum is correct) */
```
Comments

- A comment must clearly and shortly explain what a code section in a program does, but not how it is done (that is the code itself)

- For the following code …

```cpp
for (x=1; x <= 10; x++)
    for (y=1; y<=10; y++)
        cout << x << "*" << y << "=" << x*y << endl;
```

- … what do you think about the following comment?

```c
/* We have two nested for loops that are each repeated 10 times
   In the inner loop a message is printed on the screen to
   indicate the product of the two variables used as counters in the for loops
   A total amount of 100 lines are printed on the screen */
```

Think of a more adequate comment
Comments: where and how many?

- Where must I include a comment?
  - In a module definition (what does the module do?)
  - At the beginning of a code section that makes an important action or which has a meaning that is not obvious by itself
  - At the beginning of a program (a heading with the program name, author, date, program description...)

- How many comments must I include in my program?
  - Too many comments are just as bad as having too few
Topics

1. Algorithms and Programs
2. Algorithmic structures
3. Programming structures
4. Sequential statements
5. Selection statements
6. Iteration statements
7. Comments

8. Program trace

9. General structure of a program
10. Information sources
Program trace concept

- A program trace is a sequence of states through which the program passes, that is, the value taken by the variables as the program is executed.

- The trace is carried out through a sequential manual execution of the program statements.

- The traces are mainly used for debugging the program.

The variables store the program state, and the executable statements modify this state.

Debugging a program is correcting the execution errors that are detected during the execution.
// Let N > 0 be a number, calculate the addition
// all the odd numbers that are less than N

#include <iostream>

using namespace std;

main() {
    int num;  // read number (input datum)
    int sum;  // addition result  (output datum)
    int i;    // loop counter (auxiliary datum)

    cout << “Enter a number, greater than 0:”;
    cin >> num;

    // calculate the addition and display on screen
    sum = 0;
    for (i=1; i < num; i++) {
        if ( (num % 2) != 0)
            sum = sum + i;
    }
    cout << “The result is: ” << sum << endl;
}
Understanding a program using a trace

• A program trace can also be used to understand what a program or a part of it does

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

main() {
    float a, r;
    int b, i;

    cout << "Enter a real number:"; cin >> a;
    cout << "Enter an integer number:"; cin >> b;
    r = 1;
    for (i=0; i < b; i++)
        r = r * a;

    cout << "The result is: " << r << endl;
}
```

What does this program do?
Topics

1. Algorithms and Programs
2. Algorithmic structures
3. Programming structures
4. Sequential statements
5. Selection statements
6. Iteration statements
7. Comments
8. Program trace
9. **General structure of a program**
10. Information sources
What type of program must I be able to do?

```plaintext
#preprocessor directives

Constants declaration

main() {
  Variables declaration:
    of simple data types
  Main body (executable statements)
    Input and Output statements
    assignment statements
    selection statements
    iterative statements
}
```
Program example

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

const int NUM_PARTIALS = 5; // Number of partial exams

main() {
    float partial_mark; // mark of a partial exam (input datum)
    float sum; // total sum of marks (auxiliary datum)
    int i; // counter for 'for' loop (auxiliary datum)
    bool incorrect_mark; // true if the entered mark is incorrect (auxiliary datum)
    float final_mark; // average mark of all the partial exams (output datum)

    sum = 0;
    // Enter the marks of all the partial exams and add them up (only if the datum is correct)
    for (i=1; i <= NUM_PARTIALS; i++) {
        do {
            cout << "Enter the mark of partial number " << i " :";
            cin >> partial_mark;
            incorrect_mark = (partial_mark < 0.0 || partial_mark > 10.0);
            if (incorrect_mark)
                cout << "The entered mark is incorrect" << endl;
        } while (incorrect_mark);
        sum = sum + partial_mark;
    }
    // Calculate the average mark and display it on the screen
    final_mark = sum / NUM_PARTIALS;
    cout << "Your final mark is: " << final_mark << endl;
}
```
Rules for a good style of writing programs

- The variables and constants names must suggest their use
- Use line breaks between parts that are logically separated
- Properly indent the code

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

main() {
    float base, power;
    int exponent, i;

    cout << "Enter a real number:";
    cin >> base;
    cout << "Enter an integer number:";
    cin >> exponent;
    power = 1;
    for (i=0; i < exponent; i++)
        power = power * base;
    cout << "Power: " << power << endl;
}
```

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

main() {  float a, r;  int b, i;  cout << "Enter a real number:";
    cin >> a;
    cout << "Enter an integer number:";
    cin >> b;
    r = 1;
    for (i=0; i < b; i++)
        r = r * a;
    cout << "The result is: " << r << endl; }
```

A program written following a good programming style is easier to read (more readable) and easier to modify (more maintainable)
Exercises

5. After executing each of the following program fragments, which is the final value of variable $x$ in each case?

**Case A**

```c
x = 0;
n = 16;
while (n != 0) {
    x = x + n;
    n = n / 2;
}
```

**Case B**

```c
z = 12;
x = 0;
if ((z % 4) == 0)
    for (j = 0; j < 10; j + 4)
        x = x + j;
else
    for (j = 0; j < 10; j + 2)
        x = x + j;
```

6. Write a program that reads positive numbers and displays their addition and the amount of input numbers.

7. Write a program that reads an integer number, greater than 0, and displays all its divisors.

8. Modify the program from exercise 4 to add a fourth option called “EXIT”. The program must continuously display the menu, let the user select the option, and exit the program only when the option 4 is chosen.
Topics

1. Algorithms and Programs
2. Algorithmic structures
3. Programming structures
4. Sequential statements
5. Selection statements
6. Iteration statements
7. Comments
8. Program trace
9. General structure of a program
10. Information sources
Information sources

<table>
<thead>
<tr>
<th>Book Title</th>
<th>Authors</th>
<th>Publisher and Year</th>
<th>ISBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fundamentos de Programación</td>
<td>Jesús Carretero, Félix García, y otros</td>
<td>Thomson-Paraninfo 2007</td>
<td>978-84-9732-550-9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓ Capítulo 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problemas Resueltos de Programación en Lenguaje C</td>
<td>Félix García, Alejandro Calderón, y otros</td>
<td>Thomson (2002)</td>
<td>84-9732-102-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓ Capítulo 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolución de Problemas con C++</td>
<td>Walter Savitch</td>
<td>Pearson Addison Wesley 2007</td>
<td>978-970-26-0806-6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓ Capítulo 2 (Apartado 2.4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓ Capítulo 7</td>
<td></td>
</tr>
</tbody>
</table>