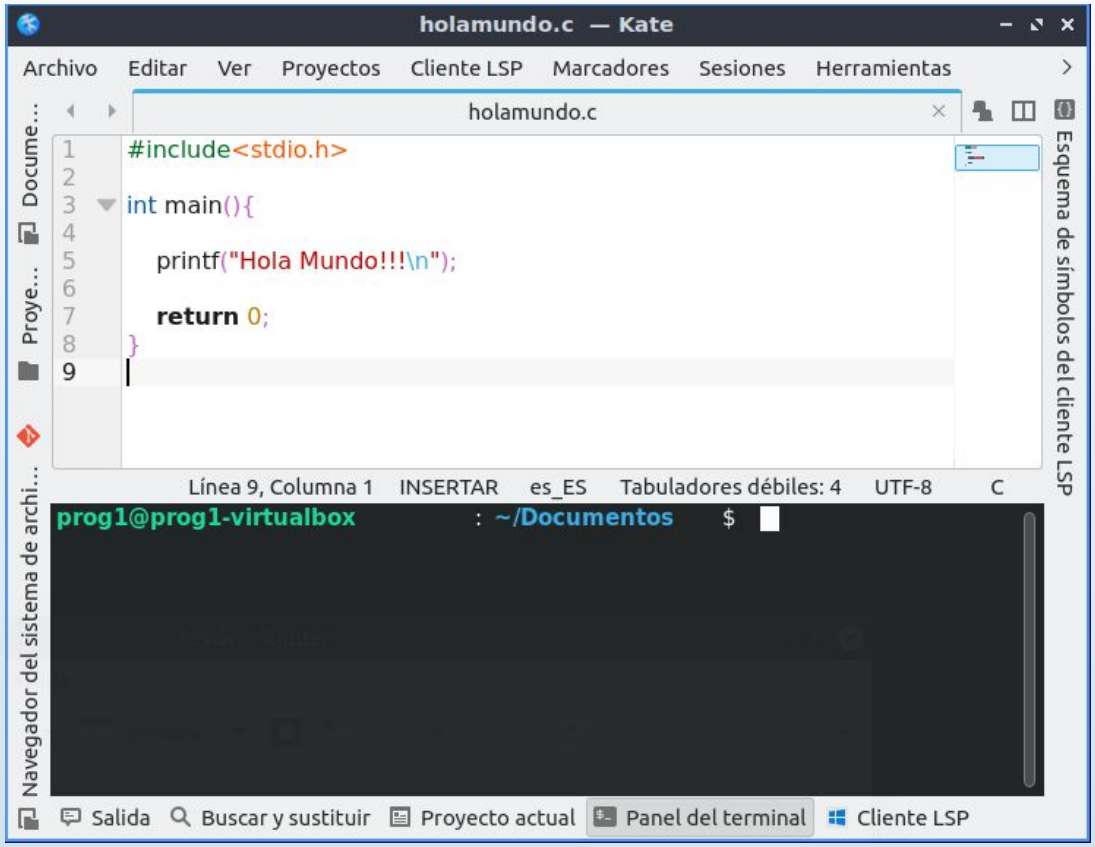



# Programming 1

## Lesson 3. Control structures

**Degree in Computer Engineering**

# Syllabus



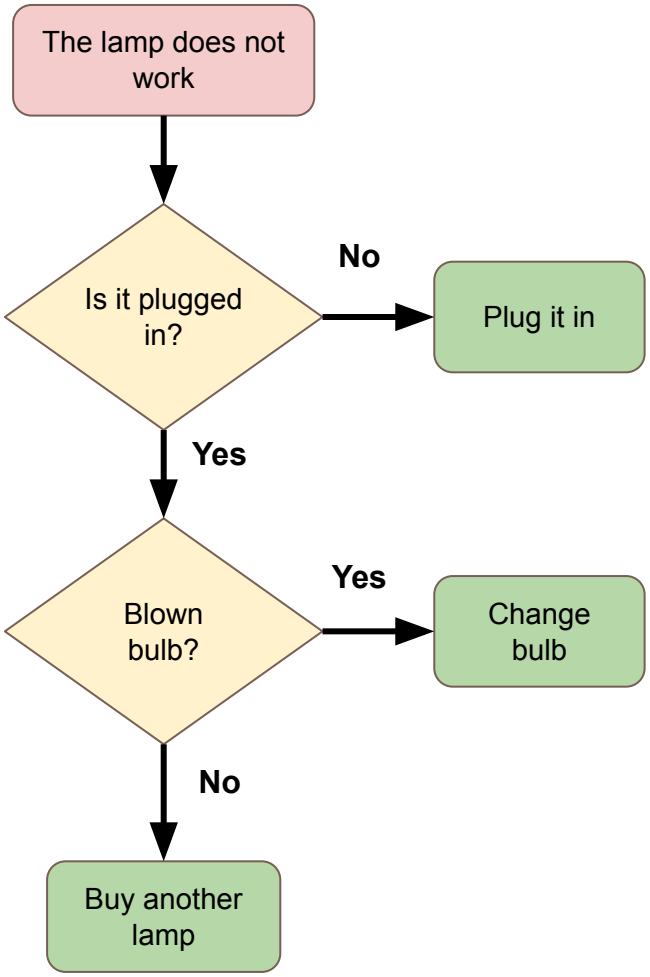
1. Algorithms and Programs
2. Algorithmic structures
3. Control structures in C
4. Comments in the C source code
5. Trace of a program
6. General structure of a program
7. Good practices for readable code

# 1. Algorithms and Programs



## Concept of algorithm

- An algorithm is an ordered sequence of instructions to solve a problem in a finite number of steps.
- In computer science, algorithms are **independent of both the programming language** used and **the computer** on which they will be executed.



# 1. Algorithms and Programs



## Concept of program

- Set of ordered instructions (statements) written in a programming language for a computer to carry out a given task.
- A computer program is nothing else but a set of ordered algorithms encoded in a programming language.

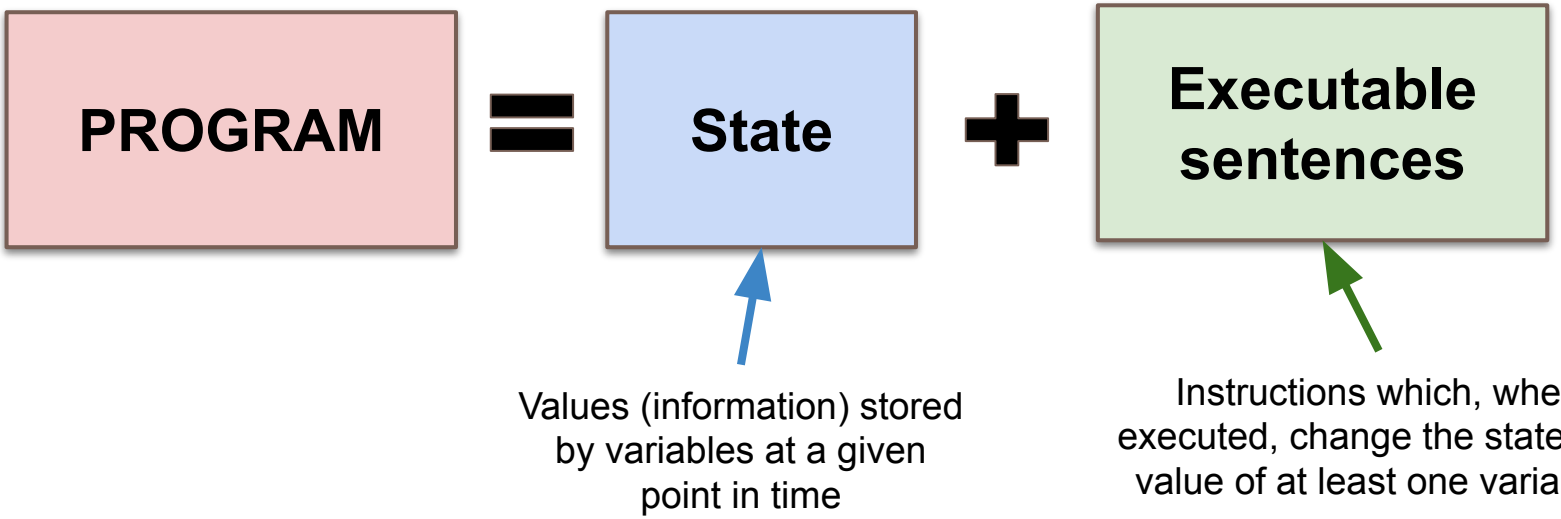
```
3
4  #include <stdio.h>
5
6  void main(){
7
8      int x, y, z;
9      int i = 1;
10
11     printf("\nInput upper limit: ");
12     scanf("%d", &z);
13
14     printf("\nInput 1st divisor: ");
15     scanf("%d", &x);
16     printf("Input 2nd divisor: ");
17     scanf("%d", &y);
18
19     if (x==y||x<=0||y<=0||z<=x||z<=y){
20         printf("\nInvalid input!\n");
21         main();
22     } else {
23         do{
24             if(i<x||i<y){
25                 if(i>=x){
26                     if (i%x==0){
27                         printf(" %d ", i);
28                     }
29                 } else if (i>=y){
30                     if (i%y==0){
31                         printf(" %d ", i);
32                     }
33                 }
34             } else if(i%x==0||i%y==0){
35                 if(i%y!=0){
36                     printf(" %d ", i);
37                 } else if(i%x!=0){
```

# 1. Algorithms and Programs



## State of a program

The state of a program at a given instant is a unique configuration of the information it handles. In other words, it is the value of each of its variables at that instant.



# 1. Algorithms and Programs



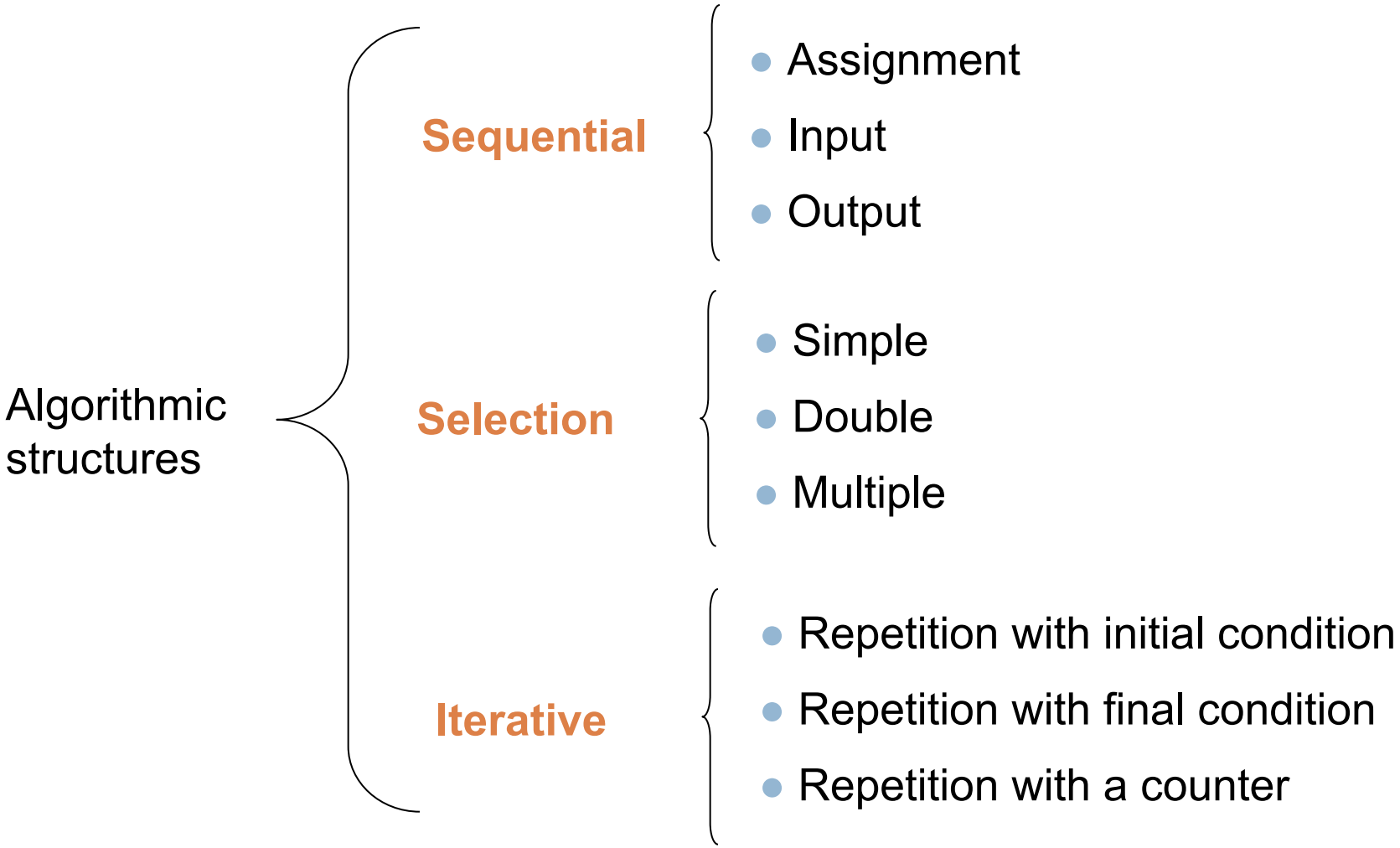
## Control flow of an algorithm

- The order in which the instructions of an algorithm are executed.
- By default, the order in which instructions are executed is sequential, from top to bottom and left to right.
- **Algorithmic structures** can be used to alter the control flow of an algorithm.

# 2. Algorithmic structures



## Types of algorithmic structures

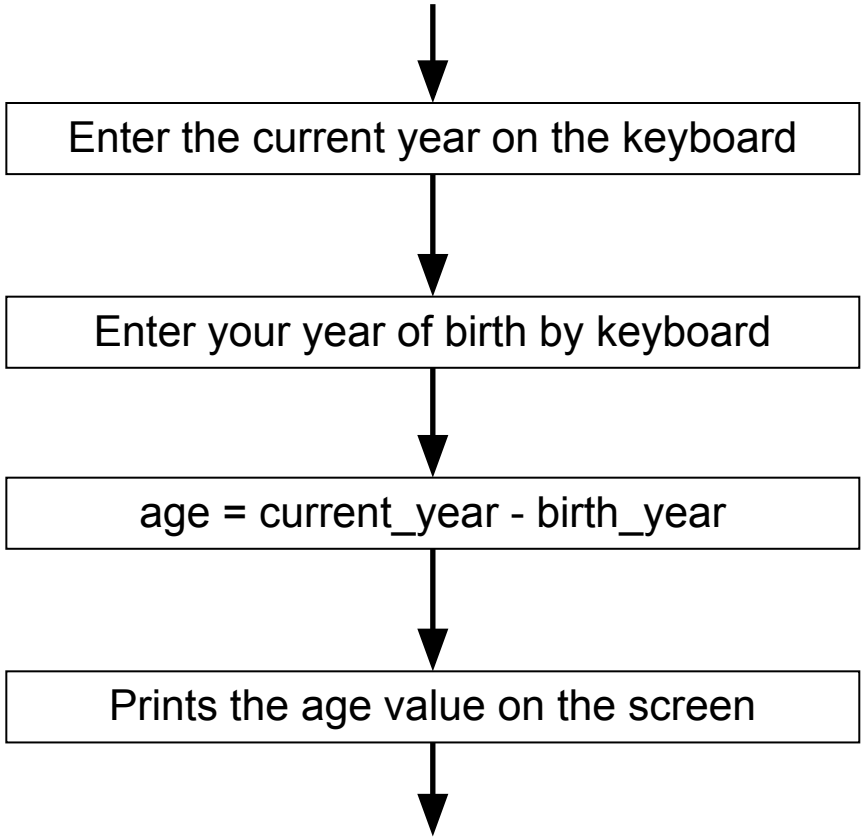


# 2. Algorithmic structures



## Sequential structure

Actions (instructions) are carried out one after the other, consecutively



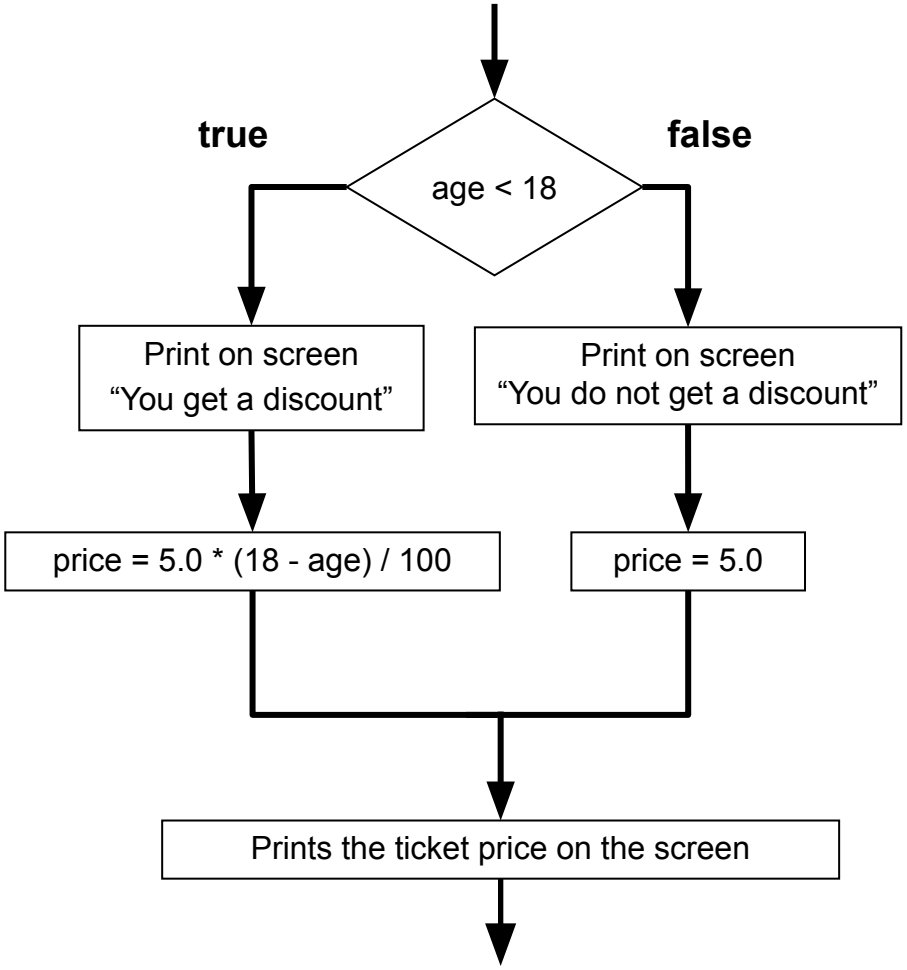
# 2. Algorithmic structures



## Selection structure

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

```
If condition_true Then
   <actions1>
Else
   <actions2>
End_Selection_Statement
```



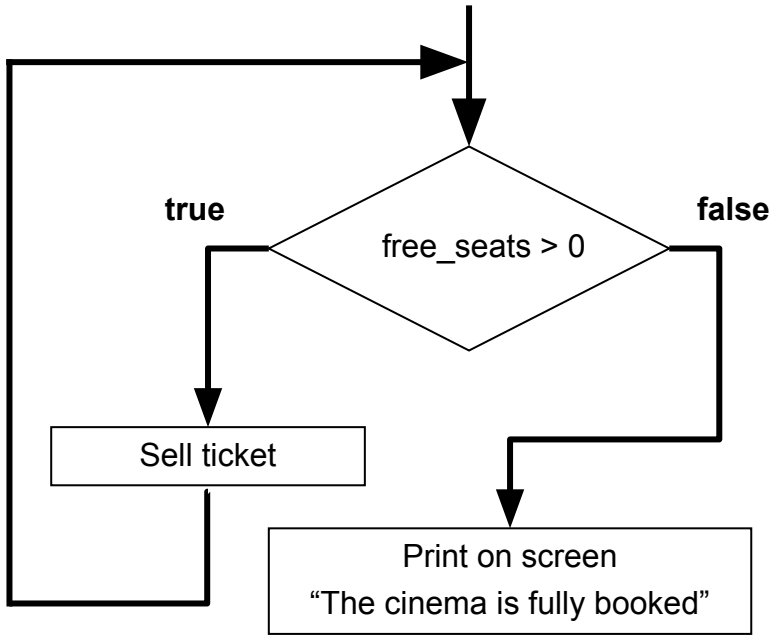
# 2. Algorithmic structures



## Iterative structure

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

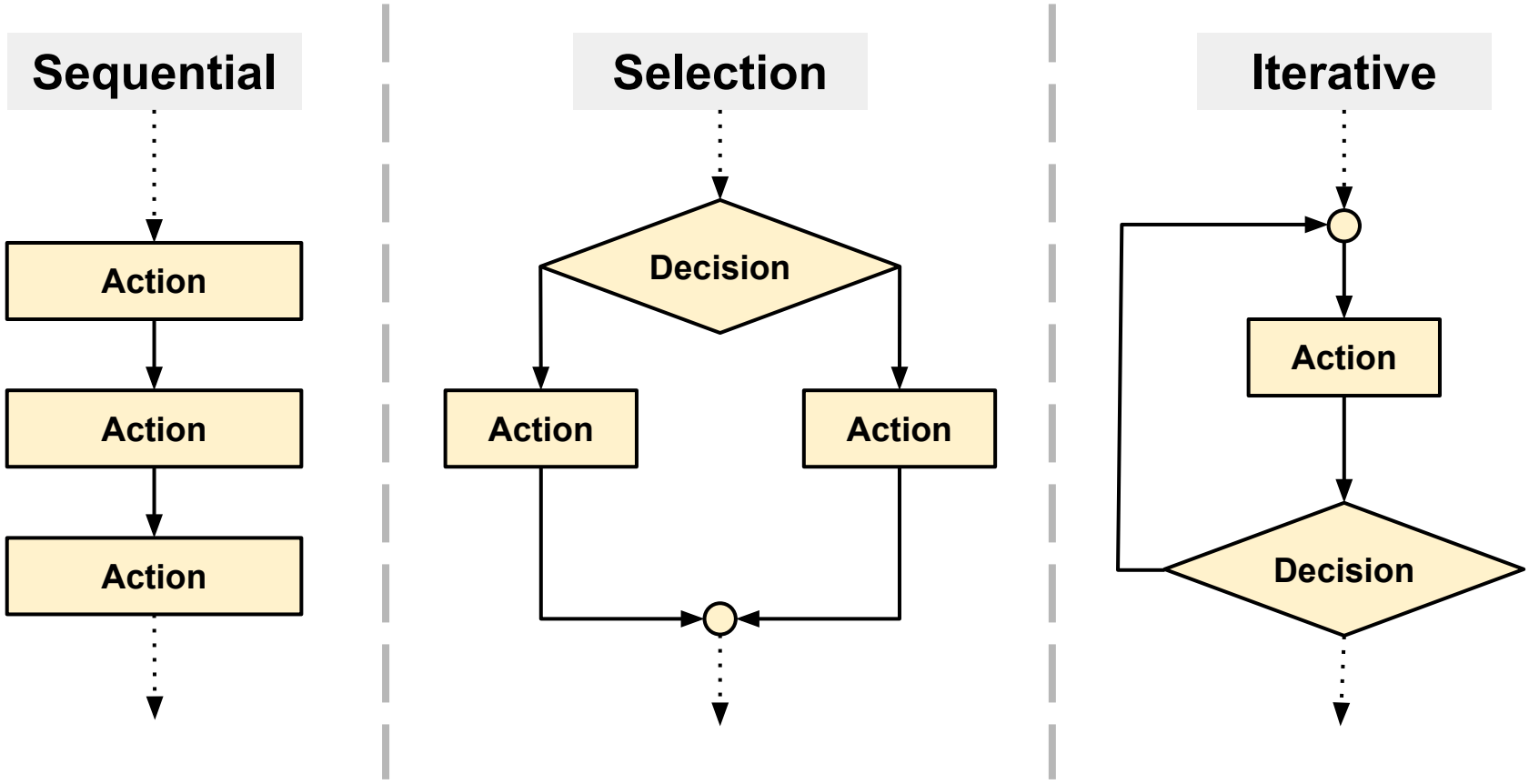
```
While condition_true Do
   <actions>
End_Iterative_Statement
```



# 2. Algorithmic structures



## Summary of types of algorithmic structures



# 3. Control structures in C



- **Control or programming structures** are the algorithmic structures carried over into a programming language.
- All programming languages implement the same control structures, although the way they are written may vary.

# 3. Control structures in C



Control structures

- **Sequential**
  - Assignment: Operator `=`
  - Input: `scanf`
  - Output: `printf`
- **Selection**
  - Simple: `if`
  - Double: `if-else`
  - Multiple: `switch` or ***nested*** `if-else`
- **Iterative**
  - Repetition with initial condition: `while`
  - Repetition with final condition: `do-while`
  - Repetition with a counter: `for`

# 3. Control structures in C



## Sequential structures

- **Assignment sentence**

```
variable = value;
```

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

- **Input sentence (data reading)**

```
scanf(format, arguments);
```

```
scanf("%d", &x);
scanf("%f", &y);
```

- **Output sentence (data writing)**

```
printf(format, arguments);
```

```
printf("Text to screen");
printf("Value: %d\n", x);
```

# 3. Control structures in C



## Sequence of sentences

- A sequence of sentences may consist of N sentences (N ≥ 0).
- When N > 1, the sequence of sentences **must** be enclosed in **curly brackets**.

```
{
   sequence of sentences
}
```

```
{  // beginning of sequence of sentences

  printf("Type two integer numbers: ");
  scanf("%x %y", &x, &y);
  quotient = x / y;
  remainder = x % y;
  printf("The quotient is: %f\n", quotient);
  printf("The remainder is: %d\n", remainder);

}  // end of sequence of sentences
```

**Important**: All C statements end with a semicolon.

# 3. Control structures in C



## Selection structures: `if` statement

- It allows to decide whether a sequence of statements is to be executed next.

```
if (logical_expression) {
   sequence of sentences
}
```

```
if (speed > 120) {
   printf("WARNING: You can be fined");
}  // end of if sentence

printf("Your current speed is: %f\n", speed);
```

- If the result of evaluating *logical_expression* is **true**, then the sequence of statements associated with the **`if`** statement is executed.
- If the value of *logical_expression* is **false**, then the sequence of statements is not executed and the next statement following the **`if`** statement is executed.

**Important**: The **parentheses** enclosing the logical expression are **mandatory**.

## Selection structures: `if-else` statement

- It allows to select between two different sequences of sentences.

```
if (logical_expression) {
  sequence of sentences 1
}
else {
  sequence of sentences 2
}
```

```
if (number % 2 == 0) {
  printf("The number is even\n");
}
else {
  printf("The number is odd\n");
} // end of if-else statement

printf("Type another number: ");
```

- If the value of *logical_expression* is **true** then the sequence of statements following the `if` statement (*sequence of sentences 1*) is executed.
- If the value of *logical_expression* is **false** then the sequence of statements following the `else` statement (*sequence of sentences 2*) is executed.

# 3. Control structures in C



## Selection structures: nested `if-else` statement

- It allows to select among multiple sequences of different sentences.

```
if (logical_expression_1 ) {
  sequence of sentences 1
}
else if (logical_expression_2) {
  sequence of sentences 2
}
else if (logical_expression_3) {
  sequence of sentences 3
}
```

```
if (mark >= 9  &&  mark <= 10)
  printf("Your grade is OUTSTANDING");
else if (mark >= 7  &&  mark < 9)
  printf("Your grade is REMARKABLE");
else if (mark >= 5  &&  mark < 7)
  printf("Your grade is PASS");
else if (mark >=0  && mark < 5)
  printf("You grade is FAIL");
else // last alternative of nested if-else
  printf("You mark is not correct");

// here comes the next sentence after the
// nested if-else structure
```

- Only the sequence of statements associated with the logical expression that first evaluates to **true** is executed.
- If all logical expressions evaluate to **false**, the sequence of sentences of `else` will be executed, if the alternative `else` exists.

## Selection structures: `switch` statement

- It allows to select between multiple sequences of sentences. It is equivalent to the nested `if-else` structure.

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

```
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 structure
printf("Result: %f\n", res);
```

- Only the sequence of sentences associated with the case whose value is equal to the result of the `switch` *expression* evaluation is executed.
- If the result of the `switch` expression does not correspond to any value in a `case`, the sequence of sentences associated with the `default` part (which is optional) is executed.

# Exercises



1. Write a program that asks the user for an integer number and writes a message telling whether it is odd or even.
2. Write a program that, for a month (1-12) entered by the user, prints the number of days it has (consider a non-leap year).
3. Write a program that reads the coordinates (x,y) of three points on a plane and prints whether these points form an equilateral triangle.
4. Write a program that displays three options of a menu and allows the user to select one of them. Then, a message should appear on the screen showing the selected option or an error message if the option is incorrect:

*Execution example 1*

```
1. Option 1
2. Option 2
3. Option 3

Choose an option (1-3):  2
The selected option is 2
```

*Execution example 2*

```
1. Option 1
2. Option 2
3. Option 3

Choose an option (1-3):  4
The selected option is not correct
```

## Iterative structures: loops

- A **loop** is a programming structure consisting of a sequence of statements, called the loop **body**, which 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** (logical expression).
- Therefore, when designing and implementing a loop, two aspects must be considered:

    1. **What is the body of the loop?**
    2. **How many times should the loop body be iterated (executed)?**

# Iterative structures: loops

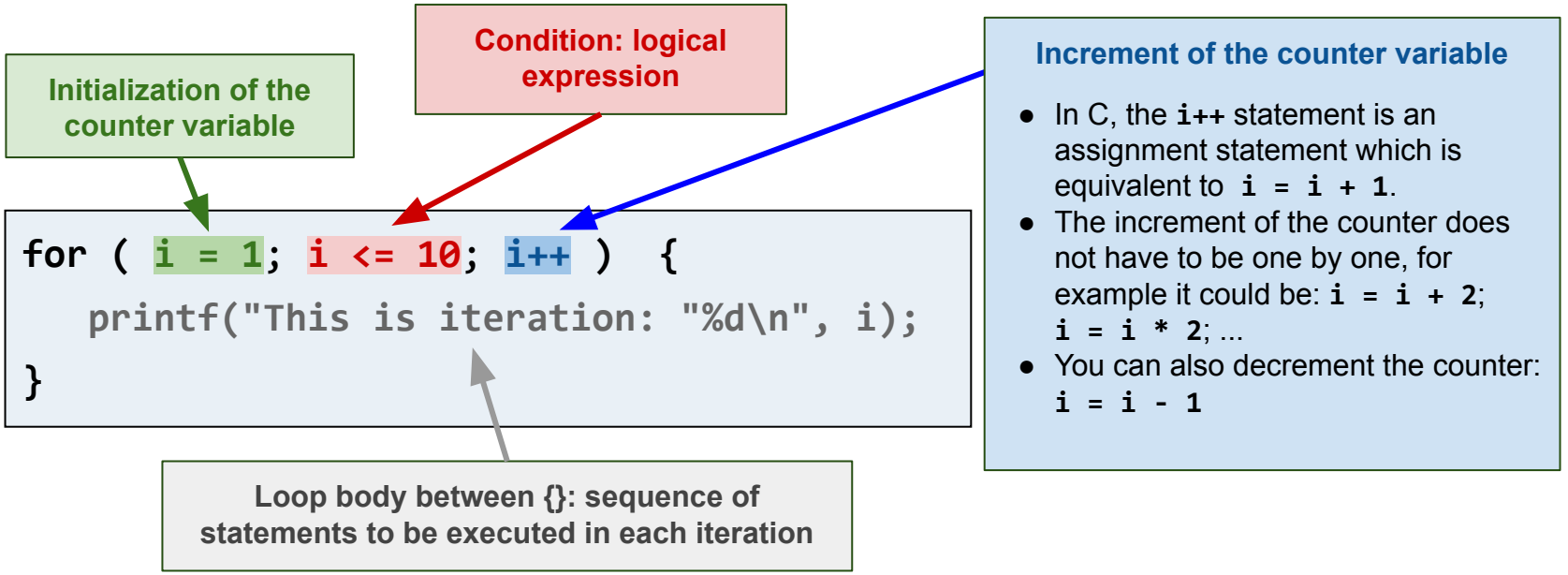
## Types of loops

- Depending on whether or not the number of times the loop body is to be repeated (iterations) is known a priori, loops can be divided into:
  - Determined loops. The number of iterations is known before the loop is executed.
    - `for` sentence (repetition is controlled with a counter)
  - Undetermined loops. Before executing the loop, the number of iterations is not known and depends on the fulfilment of a condition.
    - `while` sentence
    - `do ... while` sentence

# Iterative structures: determined loops

## for loop:

- It allows the execution of a sequence of statements to be repeated a certain number of times (known a priori).
- The number of iterations of the loop is controlled by a variable used as a **counter**.



# Iterative structures: determined loops

## `for` loop. How does it work?

**Step 1**. The counter initialization statement is executed (once only).

**Step 2**. The logical expression is evaluated so that:

- If its value is **true** then the body of the loop is executed.
- If its value is **false** then the `for` statement **ENDS** its execution.

**Step 3**. After the loop body is executed, the counter increment statement is executed.

**Step 4**. Volver al **Step 2**.

```
for ( i = 1;  i <= 10;  i++ )  {
   printf("This is iteration %d\n", i);
}
```

# Iterative structures: undetermined loops

## `while` loop

- it allows the execution of a sequence of statements (the body of the loop) to be repeated zero or more times as long as the condition (logical expression) is **true**.

```
while (logical_expression) {
   sequence of sentences
}
```

```
candies = 0;

printf("Do you want a candy?: ");
scanf(" %c", &res);

while (res == 'S' || res == 's') {
  candies = candies + 1;
  printf("Do you want another candy?: ");
  scanf(" %c", &res);
}  // end of while structure

printf("I gave you %d candies\n", candies);
```

# Iterative structures: undetermined loops

## do...while loop

- It allows to repeat one or more times the execution of a sequence of sentences as long as the condition (logical expression) is **true**.

```
do {
   sequence of sentences
} while (logical_expression);
```

```
do {
  printf("Choose option (1-4): ");
  scanf(" %d", &option);
} while (option < 1  ||  option > 4);
```

- The loop body (sequence of sentences) is executed first and then the logical expression is evaluated.
- As long as the result of evaluating *logical_expression* is **true**, the loop body is executed repeatedly.

# Iterative structures: undetermined loops

## Equivalence between the `for` and the `while` loop

- Any **`for`** loop can be rewritten as a **`while`** loop.

```
for (init ; log_expr ; incr) {
   sequence of sentences;
}
```

```
for ( i = 1;  i <= 10; i++)  {
  printf("This is iteration %d\n", i);
}
```

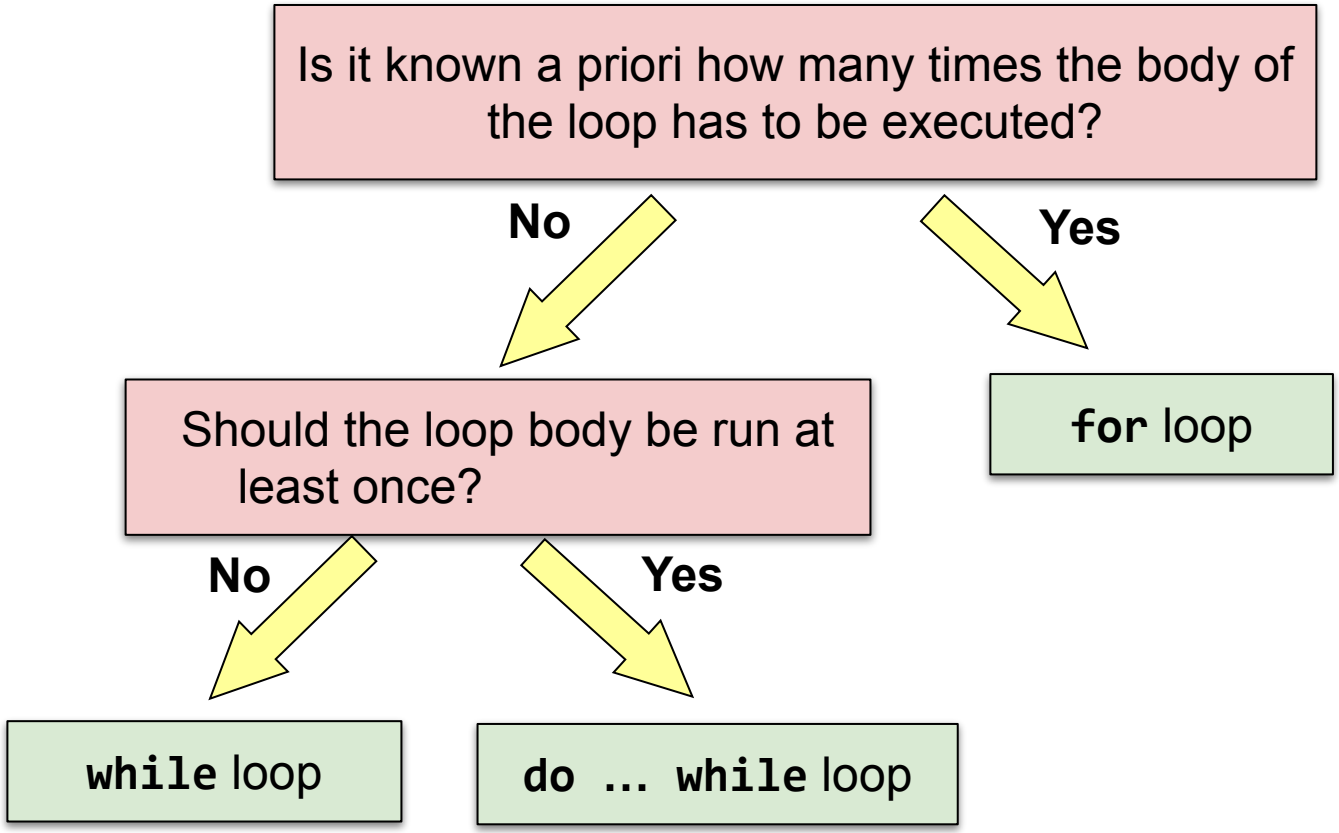
It can be rewritten as

```
init ;
while  (log_expr)  {
   sequence of sentences;
   incr;
}
```

```
i = 1;
while ( i <= 10 )  {
  printf("This is iteration %d\n", i);
  i++;
}
```

## Iterative structures: loops

How to know what type of loop to use



# 4. Comments in the C source code



- They are explanatory notes that we can include in the source code of a program.
- They make program maintenance easier.
- There are two types of comments:
  - Line comment. The symbol **//** is used and anything that goes to the right of that symbol in the same line is considered a comment by the compiler.

```
float midterm_mark;  // mark of the midterm exam (input data)
// Calculate the average mark and print it on the screen
```

  - Block comment. The symbols **/*** and ***/** are used. Anything that goes between the symbols is considered a comment by the compiler.

```
/* Enter the marks of all midterms and add them up (only
   when the data entered is correct) */
```

# 4. Comments in the C source code



## How should a comment be?

- A comment should explain clearly and concisely **what** a section of code in a program does, **not how** it does it (that's what the code itself is for).

  Given the following code:

```
for (x = 1; x <= 10; x++)
  for (y = 1; y <= 10; y++)
    printf("%d * %d = %d\n", x, y, x*y);
```

  What do you think about the following comment for the code above?

```
/* Tenemos 2 bucles for anidados que se repiten 10 veces cada uno. En el
bucle interno se imprime un mensaje de texto en la pantalla que indica el
producto de las dos variables usadas como contador en los bucles for. En
total se imprimen 100 líneas en la pantalla */
```

  What would be the most appropriate comment?

# 4. Comments in the C source code



## Where should I write comments?

- In the definition of a module (what the module does).
- At the beginning of a section of code that performs an important action and it is not obvious what it does.
- At the beginning of the program (a header with the program name, author, date, program description, etc.).

## How many comments should be written?

- Too many comments are as bad as too few.

# 5. Trace of a program



- This is the **sequence of states** through which a program passes, that is, the value that the variables take as the program's sentences are executed.
- The trace is carried out by **manually executing**, in a sequential way, the sentences that make up the program.
- Traces are mainly used to **debug** a program, i.e., to **correct errors** detected during its execution.

# 5. Trace of a program



## Example of how to make a trace

```
// Given a number N > 0, calculate the sum of
// all odd numbers less than N
#include<stdio.h>

int main() {
  int num;  // read number (input data)
  int sum;  // result of the sum (output data)
  int i;    // loop counter (auxiliary data)

  printf("Type a number > 0: ");
  scanf("%d", &num);

  /* calculate the sum and print it
     on the screen */
  sum = 0;
  for (i=1; i < num; i++) {
     if ( (num % 2) != 0 )
        sum = sum + i;
  }
  printf("The result is: %d\n", sum);

  return 0;
}
```

| | num | sum | i |
|---|---|---|---|
| scanf | 5 | | |
| Initialize sum | 5 | 0 | |
| Initialize counter | 5 | 0 | 1 |
| 1st loop iteration | 5 | 1 | 1 |
| Increase counter | 5 | 1 | 2 |
| 2nd loop iteration | 5 | 3 | 2 |
| Increase counter | 5 | 3 | 3 |
| 3rd loop iteration | 5 | 6 | 3 |
| Increase counter | 5 | 6 | 4 |
| 4th loop iteration | 5 | 10 | 4 |
| Increase counter | 5 | 10 | 5 |

**Why is it not working?**

# 5. Trace of a program



## Knowing what a program does through a trace

A trace can also be used to find out what a program or part of a program's code does.

```
#include<stdio.h>

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

  printf("Enter a real number: ");
  scanf("%f", &a);
  printf("Enter an integer number: ");
  scanf("%d", &b);
  r = 1;
  for (i = 0; i < b; i++)
    r = r * a;

  printf("The result is: %.2f\n", r);

  return 0;
}
```

**What does this program do?**

# 6. General structure of a program



```
#preprocessor directives
Declaration of constants


main() {
  Declaration of variables:
    of simple types
  Main body (executable sentences)
    input and output sentences
    assignment sentences
    selection sentences
    iterative sentences
}
```

# 6. General structure of a program



```c
#include<stdio.h> // to use input/output sentences such as printf and scanf
#include<stdbool.h> // to use boolean variables

const int NUM_MIDTERMS = 5;  // number of midterm exams

int main() {
  float midterm_mark;  // mark of a midterm (input data)
  float sum; // total sum of marks (auxiliary data)
  int i;  // for loop counter (auxiliary data)
  bool wrong_mark; // true if the mark entered is not correct (auxiliary data)
  float final_mark; // average mark of all midterms (output data)

  sum = 0;

  // ask for all midterm marks and add them up when the input is correct
  for (i = 1; i <= NUM_MIDTERMS; i++) {
    do {
      printf("Enter your mark of midterm %d: ", i);
      scanf("%f", &midterm_mark);
      wrong_mark = (midterm_mark < 0.0 || midterm_mark > 10.0);
      if (wrong_mark)
        printf("The mark entered is wrong\n");
    } while (wrong_mark);
    sum = sum + midterm_mark;
  }

  // Calculate the final average mark and print it on screen
  final_mark = sum / NUM_MIDTERMS;
  printf("Your final mark is: %.2f\n", final_mark);

  return 0;
}
```

**Example of a C program**

# 7. Good practices for readable code



- Variable and constant names must be meaningful.
- Correct use of tabs and line breaks between parts which, because of their logic, must be considered separately.
- Proper use of comments in the code.
- Avoid deep nested `if-else` structures.

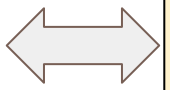
```c
#include<stdio.h>

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

  printf("Enter a real number: ");
  scanf("%f", &base);
  printf("Enter an integer number: ");
  scanf("%d", &exponent);
  power = 1;
  for (i = 0; i < exponent; i++)
    power = power * base;

  printf("The result is: %.4f\n", power);

  return 0;
}
```



```c
#include<stdio.h>

int main() {   float a, r;   int
 b, i;   printf("Enter a real number:
";  scanf("%f", &a);
printf(
"Enter an integer number: ");
  scanf("%d", &b);   r
 = 1;   for
(i=0
; i < b;
i++)       r = r * a;
  printf( "The result is: %f\n", r);
     return 0;}
```

Both programs do the same thing, but a program written in 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, what will be the final value of the variable x in each case?

**Case A**

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

**Case B**

```
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 entered by the user and displays the value of their sum and the amount of numbers read when the program ends. The program ends when a negative number is entered.

7. Write a program that reads an integer greater than zero from keyboard. The program builds another number made up of the same digits but in the opposite direction and print it on screen.

8. Modify the program in exercise 4 to add a fourth option which is EXIT. The program will display the menu continuously, after the user chooses an option, until option 4 is chosen.