

University Of Science And Technology Houari Boumediene  
Faculty of Computer Science

# Algorithmic and Data Structures

## Control Statement

First year in Computer Science engineering

By: PhD. HAOUARI Ahmed Taha  
Info.ahmed.taha@gmail.com



1

# Conditional statements

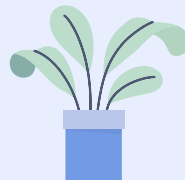


What we find behind AI

# Can you solve these problems ?

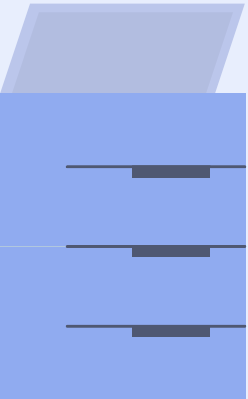
- With all we saw in the previous lectures try to solve these problems :
1. Write an algorithm that displays the sign of a number.
  2. Write a program that solves an equation of 2nd degree (quadratic equation)
  3. Write a system that prints and compute the sum of the first 100 numbers
  4. Write an algorithm that verifies if a number is a prime. (a number divided by only himself and 1)

**We need something to give a computer the power to reason and change the flow of the execution if needed.**



# What's a Control Statement ?

- The control statements cause **the flow of execution** to advance **and branch** based on control action . Those instruction allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable.
- There are two type of control Statement:
  1. Conditional Statement
  2. Loop Statement,



# Conditional statement

- In algorithmic, we have multiple conditional statements that support the **decision-making** in a program, These statements allow us to control the flow of our program's execution based upon conditions known only **during run time**. which are :
  1. **If statement**
  2. **If and else statement**
  3. **Nested if**
  4. **conditional ladder**
  5. **“Case of” statement**



# If statement

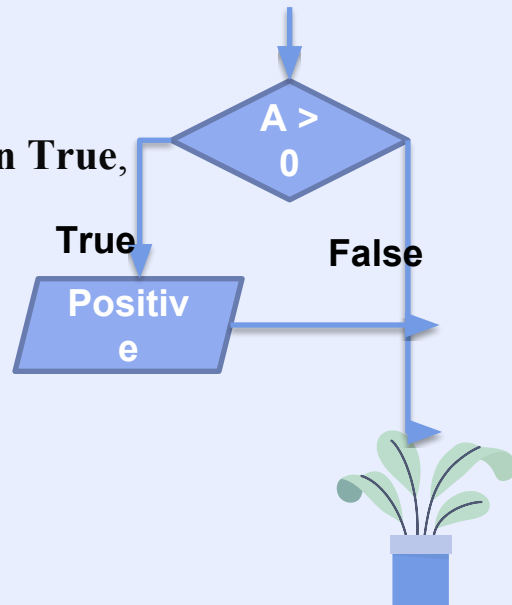
- if statement executes an action **if and only if** the condition is true. If the Boolean expression is evaluates to true, the statements in the **block** following the if is executed.
- The syntax of the If statement :
  - **If** (Boolean operation) **then** instructions; **EIF**;
- Where:
  - Instructions: are set of action to be execute if the **Boolean operation return True**,
  - we Generally called the **block of instruction If**,
  - EIF : represent the end of the block if, **don't forget the “;” after EIF**.

Example:

If (a > 0 ) then

Write(“the value of the variable a is positive”);

EIF;



# If and else statement

While the instruction after the statement if are executed only if the Boolean operation is **true** ; the statement after the **Else** are execute only it the Boolean expression return **false**,

The syntax of if and else :

**If** (Boolean operation) then instructions;  
**else** instructions; **End**;

Example:

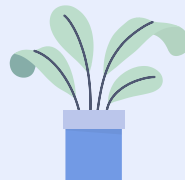
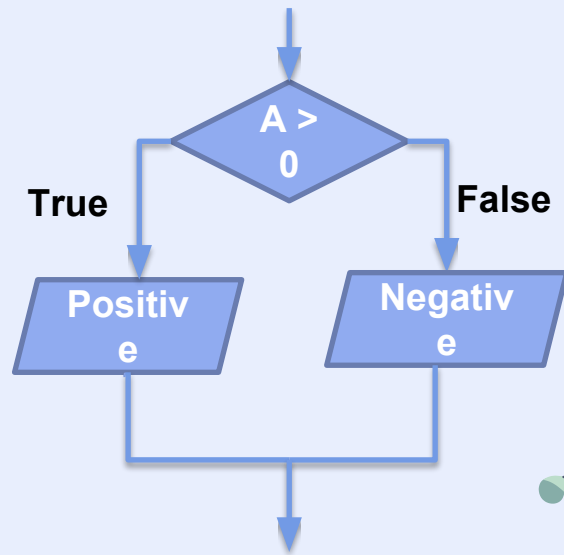
If (a>0) then

Write(“the value of A is positive”);

Else

Write ( “the Value of A is negative” );

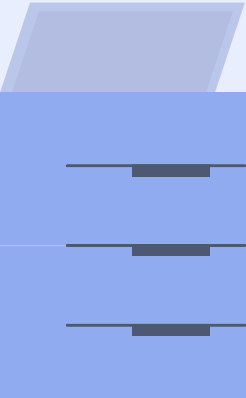
**What we do if the value of A equal 0 ?**




# Nested If statement

A nested if is an if statement which **contains another if or else**. Nested ifs are very common in programming and algorithmic.

Example : Write an algorithm that check the value of the Variable A is Positive, negative or neutral ?



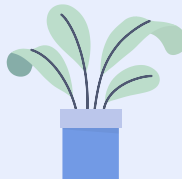
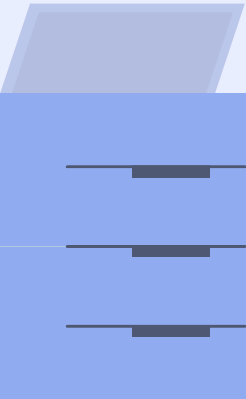
```
Algorithm check;  
Var a: Integer;  
Begin  
  Write("input the value of variable A");  
  Read(a);  
  If(a <> 0) then  
    if (a > 0) then write("A is positive");  
    else write("A is negative");  
  Eif;  
  Else write ("A is neutral"); Eif;  
End.
```





# Ladder of If and else statement

- We can transform the solution seen in the previous slide, by putting the nested if and else statement in a form of a ladder, Where we only access to next statement depend on the result of the Boolean operation of the previous one (**false result**), the last else will play the role of the **default** case (all the Boolean operation return false).
- Example:
  - If( $a > 0$ ) then write(" A is positive");
  - **Else if**( $a < 0$ ) then write("A is negative");
  - **Else** write ("A is neutral");
  - EIF;
  - EIF;
- **Note:** Each else depend or return to the previous IF;
- **Note:** We can't check a Boolean operation with the else statement, Example: ~~else ( $a < 0$ )~~



# Exercise 1

Write an algorithm that verifies if the value of a variable is even or odd ?

Algorithm even\_odd

Var a: Integer;

Begin

Write("input the value of variable A");

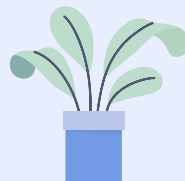
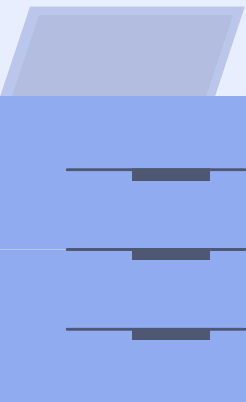
Read(a);

If(a mod 2 = 0) then

    write ("A is even");

    Else write ("A is odd"); Eif;

End.



## Exercise 2

Write an algorithm that display the parity of a variable and it's sign?

Algorithm number;

Var a: Integer;

Begin

Write("input the value of variable A");

Read(a);

If(a mod 2 = 0 and a > 0) then

    write ("A is even and positive");

Else if(a mod 2 = 0 and a < 0) then

    write ("A is even and negative");

Else if (a mod 2 <> 0 and a > 0) then

    write ("A is odd and positive");

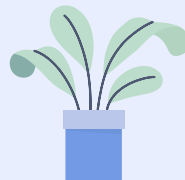
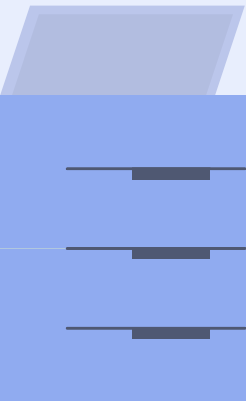
Else write ("A is odd and negative");

Eif;

Eif;

Eif;

End.



## Exercise 2 (the power of the And operation)

Write an algorithm that display the parity of a variable and it's sign?

Algorithm number;

Var a: Integer;

Begin

Write("input the value of variable A");

Read(a);

If(a mod 2 = 0 and a > 0) then

    write ("A is even and positive");

Else if(a mod 2 = 0 ) then // **We assume that one of the expression returned false; the and operation**  
    write ("A is even and negative"); // **return True only if both expression are correct**

Else if (a > 0) then

    write ("A is odd and positive");

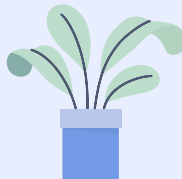
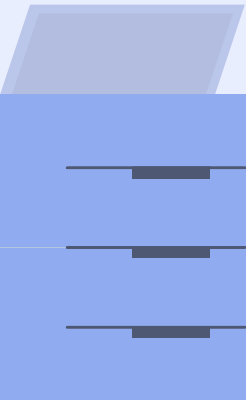
Else write ("A is odd and negative");

Eif;

Eif;

Eif;

End.



## “Case of” statement

. The if statement makes selections based on a single true or false condition. But "Case of" has **multiple choices** for the selection of the statements or we can see it as **a multiway branch statement**. It is like an if-else-if ladder statement. It adds an easy way to dispatch execution to different parts of your algorithm based on the value of an **expression**.

**It syntax:**

**Case** <Expression> **of**

**Val1:** <Statement Block1>;

**Val2:** <Statement Block2>;

**ValN:** <Statement BlockN>;

**Else :** <Other Block>;

**Ecase;**

where :

<Expression>: is an expression of type Integer, or Character only

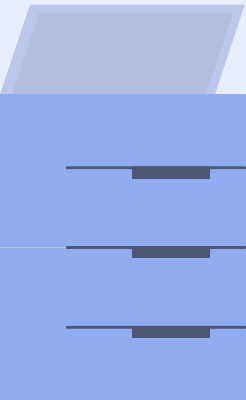
<Statement Block i>: a sequence of actions separated by ;

Val i: the various possible values of <Expression>

**ELSE** here is action is optional. (Or; if something doesn't belong to the list of choices)

**Note: if multiple choices lead to the same sets of instruction, we could group them together, and we Will separate them wih coma “,”**

**Example :** Val1,Val2 : <Statement Block1>;



# “Case of” statement Example

Write an algorithm that display the name of the month inputted by the user ?

**Algorithm** month;

M: integer;

**Begin**

Write(“input the month number”);

Read(M);

Case M of

1: write(“January”);

2: write(“February”);

3: write(“March”);

4: write(“April”);

5: write(“May”);

6: write(“June”);

7: write(“July”);

8: write(“August”);

9: write(“September”);

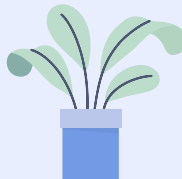
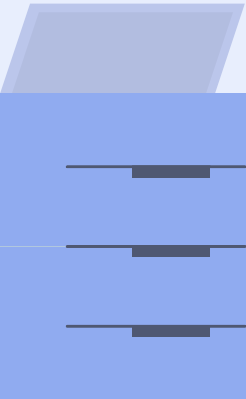
10: write(“October”);

11: write(“November”);

12: write(“December”);

Else write(“this month doesn’t exist”);

Ecase; **End.**



## Exercise 3

Write an algorithm that display the number of days of the month inputted by the user ?

**Algorithm** month;

M: integer;

A: Integer;

**Begin**

Write("input the month number and the year");

Read(M,A);

Case M of

1: write("31 days");

2: if (  $A \bmod 4 = 0$ ) then write("29 days"); else write("28 days"); E

3: write("31 days");

4: write("30 days");

5: write("31 days");

6: write("30 days");

7: write("31 days");

8: write("31 days");

9: write("30 days");

10: write("31 days");

11: write("30 days");

12: write("31 days");

Else write("this month doesn't exist");

Ecase; **End.**

**Begin**

Write("input the month number and the year");

Read(M,A);

Case M of

1,3,5,7,8,10,12: write("31 days");

4,5,9,11: write("30 days");

2: if (  $A \bmod 4 = 0$ ) then write("29 days"); else write("28 days"); Eif;

Else write("this month doesn't exist");

Ecase; **End.**

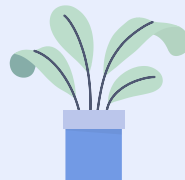
## If statement In C

- In algorithmic, if statement executes an action **if and only if** the condition is true. As C doesn't support Boolean variable, the statement is proceeded when the Boolean operation return **1**, In the case of the Boolean expression return 0, The sets of actions behind **else** will be executed.
- The syntax of the If statement :
  - **If** (Boolean operation) **{//begin**  
instructions;  
**//end**  
**else {**  
instructions; **}**

**Note: We can ignore the brackets when we have only one instruction after the if or else**

Example:

```
If (a > 0 ) {      printf("the value of the variable a is positive");  
}  
else { printf("the value of the variable a is negative");  
}
```





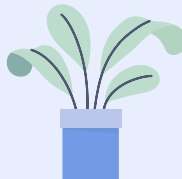
## Switch case “Case Of” in C

- The "Case Of" C is a bit different than the algorithmic pseudocode, but it follows the same set of **rules and reasoning**. We call it the **switch case statement**.

- Its syntax :

```
switch(expression) {  
  case Val1: instructions; break;  
  case Val2: instructions; break;  
  default: instructions;  
}
```

- **switch** assess the expression only once,
- The value of the expression is compared with the values of each **case**. If there is a match, the associated block of code is executed.
- **The break** statement breaks out of the switch block and stops the execution, if not the program will continue to the next block of instructions.
- **The default** statement is optional, and specifies some code to run if there is no case match.



## Exercise 4

Write an algorithm that display the parity of a variable and it's sign using C syntax?

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
    int number;
```

```
    printf("Input the value of the number\n");
```

```
    scanf("%d",&number);
```

```
    if(number%2==0 && number>0) printf("The number %d is Even and Positive",number);
```

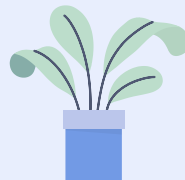
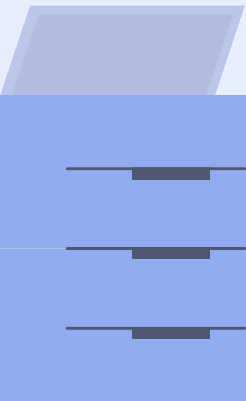
```
    else if(number%2==0) printf("The number %d is Even and negative",number);
```

```
    else if(number>0) printf("The number %d is odd and Positive",number);
```

```
    else printf("The number %d is odd and negative",number);
```

```
    return 0;
```

```
}
```



## Exercise 5

Write an algorithm that display the number of days of the month inputted by the user ?

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
    int a,m;
```

```
    printf("Input the month\n");
```

```
    scanf("%d",&m);
```

```
    switch(m){
```

```
        case 1:case 3:case 5:case 7:case 8:case 10:case 12: printf("31 days\n");break;
```

```
        case 4:case 6:case 9:case 11: printf("30 days\n");break;
```

```
        case 2: printf("Input the year\n");
```

```
        scanf("%d",&a);
```

```
        if (a%4==0) printf("29 days\n");
```

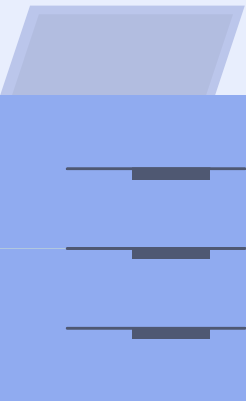
```
        else printf("28 days\n"); break;
```

```
        default : printf("This month doesn't exist");
```

```
    }
```

```
    return 0;
```

```
}
```



## 2

# Loops statements



Work smart not hard

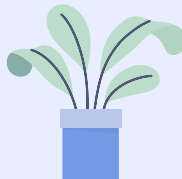
# The concepts of iteration

**If I ask you to create a program that display your age 10 times, what do you do ?**

- the simplest answer is to code 10 Write(age) instructions printing your age.

**Now, I want to be a 1000 times**

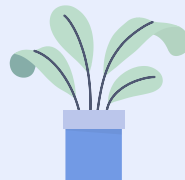
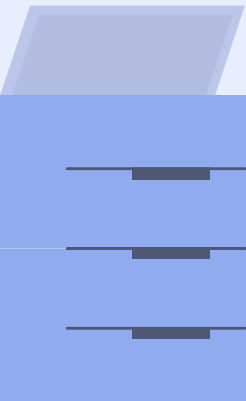
- it became tedious, even though is possible, but computer scientists are too lazy for that, we need something else to represent multiple **iterations** of an instruction or blocks of statements.
- Where iterations is :
- the process of repeatedly executing a set of statements or a block of code. It is a fundamental concept in computer programming and is essential for handling repetitive tasks efficiently. Iteration allows you to perform the same operation multiple times with slight variations, making it an integral part of problem-solving in programming. For that we use **Loops statements**,



# Loops statements

Loops are a fundamental concept in algorithm design and computer programming. They enable the repetition of a set of instructions or a block of code, allowing you to solve problems efficiently, manipulate data, and automate tasks.

- There are three type of loops in algorithmic :
  1. The **For** loop
  2. The **while** loop
  3. **Repeat until** loop
- Besides repetition, loops reduce redundancy in code by encapsulating repetitive actions within a loop structure, which leads to shorter and more maintainable, thus efficient algorithms. They are essential for working with collections of data, like arrays, lists, or databases. They allow you to iterate through these data structures and perform operations on each element. Plus, they facilitate dynamic decision-making. They enable you to respond to changing conditions or input, making your algorithms more adaptable and versatile.



# For loop

- A "for" loop is a common type of loop in algorithmic that allows you to execute a block of code a specified number of times. It's often used when you know the exact number of iterations you need in advance.

```
for var ← init to stop do
// Code to be repeated
done;
```

- Where :
- Var: is the control variable (commonly called **counter**)
- Init: the initialization of the counter
- Stop: the condition to stop the loop. Also, it can be seen as the **number of iterations**.
- Do and done is the start and end of the block for.

Example:

```
For i ← 1 to 1000 do
```

```
Write(age);
```

```
done;
```

**Note:** at each iteration the counter is **incremented** by one, So at the start i variable will receive 1, at the next iteration, the value of i becomes 2 and it will repeat until the i variable is superior to the stop condition.

**Note: In this class, decrementing in the for loop is allowed. just inverse between initialization and stop condition.**



## Exercise 6

- Write an algorithm that computes the sum of the first hundred numbers.

Algorithm sum;

S, I:integer;

Begin

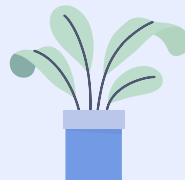
For I  $\leftarrow$  1 to 100 do

S=S+I;

Done;

Write(“the sum = ”, S);

End.





# While loop

The "while" loop is another fundamental type of loop in programming. Unlike the "for" loop, which is used when you know the number of iterations in advance, the "while" loop is used when you want to repeat a block of **code as long as a certain condition is true**. The loop continues executing the code block as long as the condition remains true. Here's the basic structure of a "while" loop in Algorithmic:

```
While (Condition) do
//Block of the loop while
Done;
```

While:

Condition: it's a Boolean operation that has to be true to execute the block of instruction inside the loop while

Example:

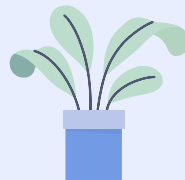
```
i ← 1; //Initialization
```

```
While(i ≤ 100) do
```

```
Write(age);
```

```
i++; // i ← i+1; increment
```

```
Done;
```



# While loop problems

- The loop uses a logical condition to determine if the block of instructions inside it is executed or not.
- Unlike the "for loop" we don't have a way to initialize for example the control variable, or a way to stop the loop. Which lead in both case to a problem.
- Either we cannot get into the block of instructions, or we get stuck in the while block for an infinite number of iterations (Infinite loop)
- for those reasons, We have to make sure that the set of instructions before the while loop has an initialization of the control variables.
- We must also find a way to get out of the infinite loop, by using for example an assignment instruction or reading procedure (read).
- **Let see our previous example:**

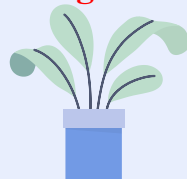
$i \leftarrow 1$ ; //Initialization, **here we made sure that i is lower than 100 which satisfy the while condition**

While( $i \leq 100$ ) do

Write(age);

$i++$ ; //  $i \leftarrow i+1$ ; **Here at each iteration of the loop we increment the counter by 1, till i became 101 which let us get out of the loop cause the Boolean condition will return false.**

Done;



## Exercise 7

- We have a series of number that ends with 999, write a program that computes the product of those number.

Algorithm product;

N, P: integer;

Begin

Write("Input the Number");

Read(N);

$p \leftarrow 1$ ;

While( $N \neq 999$ ) do

$p \leftarrow 1 * N$ ;

Write("Input the Number");

Read(N);

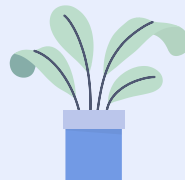
Done;

If ( $P \neq 1$ ) then

Write("the product = ", P);

Else Write("we can't compute the product"); Eif;

End.



# Repeat until loop

The "repeat until" loop is a looping construct in algorithmic that is similar to the "do-while" loop in some other programming languages. It is used when you want to execute a block of code **at least once** and then repeat it as long as a certain condition is **false**. The loop checks the condition at the end of each iteration. Here's the basic structure of a "repeat until" loop in Algorithm:

```
Repeat
//block of instructions
Until(Condition);
```

Where, The condition is checked at the end of each iteration. If the condition is false, the loop continues; if it becomes true, the loop terminates.

Example : (display your age 100 time)

```
i ← 0;
```

```
Repeat
```

```
Write(age);
```

```
i++;
```

```
Until(i=99);
```

**Note:** The loop “repeat until” has the same problem as the while loop seen in the previous slide

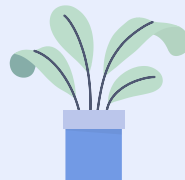


## Exercise 8

- Write a program that prompts the user to enter a password. The program should keep prompting the user until the correct password is entered.

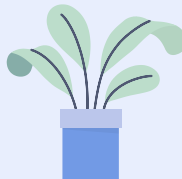
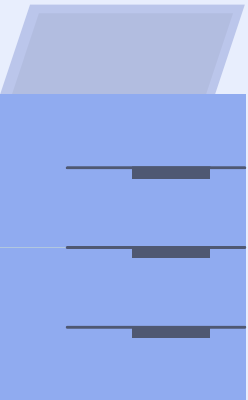
Algorithm PasswordPrompt;

```
const CorrectPassword = 82442;  
var UserPassword: integer;  
begin  
  repeat  
    write("Enter the password: ");  
    read(UserPassword);  
    if (UserPassword <> CorrectPassword) then  
      write("Incorrect password. Try again.");  
    Eif;  
  until (UserPassword = CorrectPassword);  
  write("Correct password entered. Access granted.");  
  
end.
```



# Loop For vs While vs repeat until

- Choosing the right type of loop depends on the specific problem you are solving. Each type of loop offers a different approach to repetition, and the choice depends on the nature of the task and the behavior you want to achieve in your program. It's common to encounter all three types of loops in programming, and the decision of which one to use depends on the requirements of the particular situation.
1. The "for" loop is ideal when you know the number of iterations in advance, such as iterating over a fixed range of values.
  2. The "while" loop is suitable when you want to repeat a block of code as long as a certain condition is true. The condition is checked before each iteration.
  3. The "repeat until" loop is used when you want to ensure that a block of code is executed at least once and then repeat it as long as a certain condition is true. The condition is checked at the end of each iteration.
- Anything that can be solved using the for loop, can be done by while or repeat until loop. However, it's not always the case in reverse.



# For loop In C

The loop for in C, plays the same role as in algorithmic, with a slightly different syntax shown below:

```
for (initialization ; condition ; increment/decrement) {  
    // Code to be repeated  
}
```

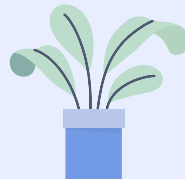
Where:

**Initialization:** This part is typically used to initialize a loop control variable (commonly an integer) to an initial value before the loop begins. It's executed only once at the beginning of the loop.

**Condition:** This is a Boolean expression that is evaluated before each iteration of the loop. If the condition is true, the loop continues; if it's false, the loop terminates.

**Increment/Decrement:** This part is responsible for modifying the loop control variable after each iteration. It's often used to change the variable in a way that eventually makes the condition false.

**Code Block:** This is the block of code enclosed in curly braces {}. It contains the instructions that are executed in each iteration of the loop.



## Exercise 9

- Write an algorithm that computes the sum of the first hundred numbers. (Using C syntax)

```
#include <stdio.h>
```

```
int main() {  
    int i,sum = 0;
```

```
    for (i = 1; i <= 100; i++) {  
        sum += i;  
    }
```

```
    printf("The sum of the first 100 numbers is: %d\n", sum);
```

```
    return 0;  
}
```





# While loop In C

- The while loop in C has almost the same syntax as in algorithmic, Also, she harbors the same problem as in the algorithm syntax (Infinite loop, lack of initialization ). it's syntax is shown below :

```
while (condition) {  
    // Code to be repeated  
}
```

Where:

Condition: The condition is a Boolean expression that is evaluated before each iteration of the loop. If the condition is true, the loop continues; if it's false, the loop terminates,



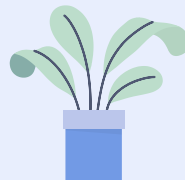
## Exercise 10

- We have a series of number that ends with 999, write a program that computes the product of those number using C language.

#include <stdio.h>

```
int main() {
    int number;
    int product = 1;
    printf("Enter a series of numbers (end with 999):\n");
    scanf("%d", &number); // Read the first number
    while (number != 999) { // Continue the loop until 999 is entered
        product = product * number; // Multiply the product by the entered number
        scanf("%d", &number); // Read the next number
    }
    if (product != 1) // Check if any number was entered before 999
        printf("The product of the entered numbers is: %d\n", product);
    else
        printf("No numbers were entered.\n");

    return 0;
}
```



# The do While loop In C

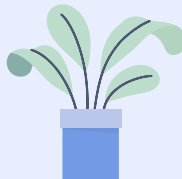
- It's the equivalent of the repeat-until loop, However, the repeat-until loop continues the execution of its block when the condition is not met (return false), and the do-while loop goes through its iterations when the condition returns true. The syntax of the do while loop is as follow:

```
do {  
    // Code to be repeated  
} while (condition); //don't forget the semicolon
```

Where:

**Condition:** The condition is a Boolean expression that is evaluated at the end of each iteration of the loop. If the condition is true, the loop continues; if it's false, the loop terminates.

**Note:** The same problems found in the loop repeat-until and while in the algorithm are applied to the do-while loop in C, don't forget initialization and avoid the infinite loop

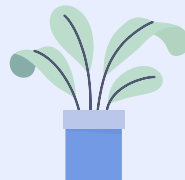


# Exercise 11

- Write a program In C that prompts the user to enter a password. The program should keep prompting the user until the correct password is entered.

```
#include <stdio.h>
```

```
int main() {  
    int userPassword, correctPassword = 82442; // Define the correct password as an integer  
    do {  
        printf("Enter the password : ");  
        scanf("%d", &userPassword);  
        if (userPassword != correctPassword)  
            printf("Incorrect password. Try again.\n");  
    } while (userPassword != correctPassword);  
  
    printf("Correct password entered. Access granted.\n");  
    return 0;  
}
```



**Classroom code :**  
**Whz2ikh**