M'hamed Bougara University - Boumerdès
Faculty of Sciences
Computer Science Department

Algorithms and Data Structures 1

L1 Computer Science

Dr RAHMOUNE Nabila

2023/2024

Chapter 1
Basic Concepts

Outline

- Introduction
- Definition of the algorithm
- Variables
- Diferrents types of variables
  Applications

Introduction

- Writing a program is a complex operation that requires many steps.
- The most important thing is to understand the end goal and stick to it.
- For this, it is often preferable to break down the desired treatment into a succession of smaller and simpler operations.

Structure of an algorithm

Algorithms are intended to make us think, but not to run on a computer: for that, it will be necessary to translate the algorithm into a programming language.

Structure of an algorithm

Definitions

Algorithm

An algorithm is a series of elementary operations making it possible to obtain the final result determined for a problem.

Property of an algorithm:

An algorithm, under similar execution conditions (with identical data) always provides the same result.

Instruction block:

A block of instructions is a processing part of an algorithm, made up of elementary operations located between Start and End or between braces.

Structure of an algorithm

The structure of an algorithm is as follows:

```
Algorithm algorithm_name;     //    header part
Begin                         //   processing part
Instruction block
End.
```

- Each line has a single instruction
- The execution of the algorithm corresponds to the realization of all the instructions, line after line, from the first to the last, in this order.

Structure of an algorithm

Comment:

Comments are textual explanations in the algorithm by the programmer following the two characters // .

They are not executed: they are invisible to the execution of the algorithm.

These comments will be useful to programmers who want to understand or modify the algorithm .

Data

Declaration and use of variables

The values, in order to be able to be manipulated, are stored in variables.

Variable:

A variable is a memory location that stores a value.

A variable is defined by:

a single name

a unique type of definition

a value assigned and modified during the course of the algorithm: ***Variable1*** ← value;

Data

The syntax

The structure of an algorithm (declaring a variable named **index and of integer** type) is then the following:

```
Algorithm algorithm-name    // header part
Var index : integer          // variable declaration part
   Begin                     // processing part
Instruction block;
 END
```

Data

The name – the type – the value

Errors to evade

- A variable is declared only once in an algorithm
- A variable is declared at the beginning of the algorithm and not in the part reserved for processing instructions.
- Before you can use a variable, you must have declared it in the variable block.
- Before you can use the value of a variable, a value must be assigned to it.

Data

The name – the type – the value

Errors to evade

Variable-error
Var: **number, result:** real;
Begin result ← number x 2; value ← 1;
END

error : *number* has no value

error : *value* has not been set

Types

- Types can already be **Predefined** :

  They are called the “ **Predefined Types**” of the language

  Example: type Integer, Real,…

- ..Types **defined** (by you) :

  Do not exist in your language, you must create them.

  Example: Human type; Ben;…

Types

Operations permitted on integers

- The standard operations{+,-,*}.
- Div : i ← 9 div 2; (i will be equal to 4)
- Mod : i ← 9 mod 2 (i will be equal to 1)
- abs(x) absolute value of x.
- pred (x):x-1.
- succ (x): x+1.
- odd (x) true if x is odd, false otherwise.
- sqr (x) the square of x.

Types

- **The equality operator:**

This is the operator found in all simple types that allows you to know if the two operands are equal

It is represented by the character =

The result of an expression containing this operator is a boolean

- **We also have the inequality operator** : ≠

- And for types with an order the **comparison operators**

<, ≤ , >, ≥

Types

- **The real type is larger than the integers but also has representation limits.**

```
Multiplication program
Var Radius, :integer;
Var Perimeter: real;
const  Pi:Real ;
Begin
Radius ← 5;
Pi ← 3.147;
Perimeter ←2*P i*Radius;
Write('Your perimeter is', Perimeter, 'cm');
END.
```

- Arithmetic operator:
  - +,-,/,*
- Relationship Operator:
  - =, ≠ <, ≤ , >, ≥

Types and DATA

Types of variables

The real type and the integer type

The real ty
– the i

typ

the numeric type variables used in the algorithm have as usual domains those provided by mathematics: real or integer.

Real-type

Var **number1, number2, result:** real;

var1 : integer;

Begin

number1 ← 1.2;

number2 ← 15;

var1 ← 2;

result ← number1/number2 x var1

END.

Types and DATA

Types of variables

The real type and the integer type

Conversion

- Converting an integer to a real is natural: this operation does not cause any loss of information; For example, the integer 15 will become 15.0.
- Converting a real to an integer results in a loss of information: the decimal digits are lost. For example, the real 15.75 will become 15.

Numerical-conversion algorithm

Var: **number1:** integer;

number2: real;

Begin

number1 ← 15;

number2 ← number1; //number2 is 15.0

number1 ← number2 + 0.5 ; //impossible error

END

Types and DATA

Types of variables

The Character type

The Character type	*This is the domain consisting of alphabetic, numeric, and punctuation characters.*

DO NOT CONFUSE BETWEEN

AND

The character _	the whole
' 3 '	3

Types and DATA

Types of variables

The Character type

- The only elementary operations for character type elements are the comparison operations:

  $> < \neq = \geq \leq$

- In fact, each character is associated with a unique whole numeric value (the ASCII code establishes this correspondence: for example, the letter 'A' corresponds to the value 65).
- To write an algorithm, we don't have to know the values of the ASCII table by heart. But we will use three principles:
  - The integers corresponding to the characters 'A','B'…'Z' follow each other in this order.
  - The integers corresponding to the characters ' a','b '…'z' follow each other in this order.
  - The integers corresponding to the numeric characters '0' to '9' follow each other in this order.

Types and DATA

Types of variables

Conversion

- So to convert a lowercase character to uppercase, just add the difference between them: 'c'+ (' A'-'a ') equals 'C'
- Character to integer type conversion: to convert the character '3' into an integer value 3, simply calculate the difference between the two characters: '3'-'0', which is equal to 3.
- Conversion from integer to character type: to convert the integer 3 into a value of the character '3', simply calculate the sum between the two characters: 3+'0', which equals '3'.

Types and DATA

Types of variables

Conversion

```
Integer-character-conversion
Var: number1: integer;
  char: character ;
 Begin
char ← '3';
  number ← '3' – '0'; //number is 3
  number ← number + 2; // number is 5
  char ← '0' + number; // because is '5'
 END
```

Types and DATA

Types of variables

The Boolean type

The Boolean type	*The domain of booleans is the set formed of the only two values (true, false).*

- The admissible operations on the elements of this domain are carried out using all the logical connectors, denoted:
  - AND: for the "logical and"
  - OR: for the "logical or"
  - NO: for the "logical no"

AND operation	Fake	TRUE
Fake	Fake	Fake
TRUE	Fake	TRUE

OR operation	Fake	TRUE
Fake	Fake	TRUE
TRUE	TRUE	TRUE

Types and DATA

- Reminders on Boolean logic...

Possible values: True or False

- Associativity of operators **and** and **Or**

a and (b and c) = (a and b) and c

- Commutativity of operators and and or

a and b = b and a

a or b = b or a

- Distributivity of 'and' and 'or' operators

a or (b and c) = (a or b) and (a or c)

a and (b or c) = (a and b) or (a and c)

- Involution

no no a = a

Types and DATA

Types of variables

The Boolean type *- Example*

```
Boolean-type algorithm
Var: boolean1, boolean2: boolean;
 Begin
  boolean1 ← 5<6; //boolean1 evaluates
  boolean2 ← NOT boolean1; // boolean2 evaluates to False
  boolean2 ← (5<7) OR (3>8); //boolean2 evaluates to True
  boolean1 ← true; // boolean1 evaluates
 END
```

Types and DATA

Mistakes to avoid

- During an assignment, the value of the right part must be of the type of the variable whose value is modified.

error-type
Var: **char:** character ;
Begin
a← 1.56;
b← 5<8;
END

//error: a is not a real

// error: b is not a boolean

Priority between Arithmetic operator

A ← 2;	A =2;
B←A+2;	B=4;
B←BA*3;	B=-2;
C←(BA)*3;	C=-12;
D←A-8 div B mod C;	D=-2;
D←D+A mod 9*3;	D=12;
D ←D*(5 div 8)-2*3;	D =-6;

Priority between Logical operator

A ← True;

B←False;

B←B and A or not (A);

C←(5=4) or (2>3);

D←(4<>4) or not(C);

E←3;F←5;

G←(F+E div 2=2) or not(C) and D

A = True;

B=False;

B=False;

C=False;

D=True;

E=3;F=5;

G=True;

Input – Output Functions

- The programs frequently use instructions allowing the display on the screen and the entry of values on the keyboard by the user.
- We are going to provide ourselves with two analogous operations allowing us to simulate:
  - Displaying a sentence with the write() or print();
  - Entering a value by the user with the read() instruction.

Input – Output Functions

The read function

- The on-screen display instruction (output device) of an expression is:

  write(expression) or print(expression)

- This statement simply performs the display of the expression passed between parentheses.

Input – Output Functions

- **Example**

READ AND WRITTEN

```
Alg orithme example-read-write
Var: nb: real;
Begin
read( nb) //the user enters the number on the keyboard
write(' 'the value of nb''); //a sentence is displayed on the screen
write (nb); // a value is displayed on the screen
write (''the value of nb is: '', nb) ; //a sentence followed by the value //are displayed on the screen
END
```

Input – Output Functions

Exercise

- Write an algorithm that asks the user to enter three real numbers on the keyboard and displays the sum of these three numbers on the screen ;

```
Sum_3_real
Var: nb1, nb2, nb3, sum: real;
Begin
read(nb1);
read(nb2);
read(nb3);
sum ← nb1+nb2+nb3;
write(sum);
END
```