Chapter 2

Simple sequential algorithm

Chapter 2: Simple Sequential Algorithm

- Parts of an algorithm
- Data: Variables and Constants
- Types of data
- Basic Operations
- Simple algorithm construction
- Representation of an algorithm by a flowchart
- Translation into C language

1. Parts of an algorithm

- An algorithm is a finite description of a computation that associates a result to data.
- It is composed of 3 parts:
  - Header: the name of the algorithm
  - Declarations: the specification of all objects used in the algorithm.
  - Actions or body: the description of the calculation steps in a pseudo-language or algorithm language.

1. Parts of an algorithm

```
Algorithm <Algorithm Name>
        < declaration statement>
Begin
        < action statement >
End.
```

5

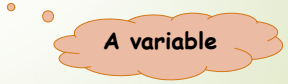
1. Parts of an algorithm

- The body of an algorithm is one or more instructions.
- We can compose these instructions to define new instructions. There are several types of composition, called **control structures**.
- All instructions in an algorithm must be separated by a semicolon "**;**"
- The basic instruction is "**assignment**".

6

2. Data: Variables and Constants

- We often need to store values temporarily.
  - data from the hard disk, given by the user by typing on the keyboard (inputs).
  - intermediate or final results obtained by the program (outputs).
- they can be of different kinds: numbers, text, etc.
- we use what is called.



7

2. Data: Variable and Constant

In other words

- To store initial data or intermediate calculation results, we use "**variables**".
  - From the computer's point of view, a variable is a memory location to which we access the content through an identifier.
  - From an algorithmic point of view, a variable has a fixed name and a value that can change during the execution of the algorithm.
- Therefore, the nature and role of variables in computer science and mathematics are different, although we use the same word.

8

2. Data: Variable and Constant

One more explanation:

- A variable can be likened to a box, which the computer will identify by a label (its **name**). To access the contents (the **value**) of the box, you simply need to specify it by its label.
- The first thing you need to do before you can use a variable is **to create the box and label it**.

9

2. Data: Variable and Constant

Now, the constants:

- Constants are information that have a **fixed value** during the execution of an algorithm. In other words, the algorithm uses them, but **it cannot change** their initial values.
- The constant is designated by **a name** and has **a value** before execution.
- Before being used, **every constant must be defined** by giving it a name (identifier) and a value.

10

Remark

The value of a constant cannot be modified by any action.

11

Identifiers

- Identifiers are used to name all objects manipulated in the program\algorithm.
- Keywords are reserved and cannot be used as identifiers
- An identifier cannot be used unless it has been defined
- An identifier is a series of alphanumeric characters, which must begin with an alphabetic character

12

3. Data types

A type is defined by:

- a domain: the set of values that can be taken by objects of type
- A set of operations that can be applied to objects of the type.

13

3. Data types
Type Categories

There are three main categories of type

- Simple types: integers, reals, etc.
- Structured types: array, file, record, etc.
- The pointer type: list, stack, queue.

14

3. Data types:
The Simple Types

They are the basic types from which all other types are built, and can be classified as follows:

1. Scalar types

a) **Predefined scalar types**

b) **Declared scalar types**

2. The interval type

15

Scalar types
Predefined scalar types

- **integer** :
  - Domain : The set of integers that can be used on a given computer
    - on machines using 16-bit words, the integers can be represented as: -32 768 to 32 767,
  - the arithmetic operators are: +, -, *, div et mod.

16

Scalar types
Predefined scalar types

- **Real** :
  - Domain : the set of real numbers that can be represented on computers.
  - The arithmetic operators are :

    +, -, *, /.

17

Scalar types
Predefined scalar types

- **Logic** (Boolean):
  - A logical variable is a variable indicating an alternative, so it can take one of two values: true or false.
  - Possible operators are: and, or, negation.

18

Scalar types:
Predefined scalar types

- **Character** :
  - It describes the set of alphanumeric character codes available on a given computer,
  - The two most commonly used international codes are: EBCDIC and ASCII.
  - Among the characters, we have :
    - **digits** :' 0 '.. ' 9 ',
    - **letters** :' A'.. ' Z ', ' a '.. 'z ',
    - **Special Characters**: ' + ',- ', ... '? ', ' * ', ' ', '_',...

19

Scalar types:
Predefined scalar types

- Possible operators are:
  - Comparisons : =, <>, <=, <, >=, >
  - successor and predecessor.
  - ord(c) returns the ASCII code of c,
  - chr(n) returns the character with an ASCII code of n
  - ... etc

20

Scalar types:
Predefined scalar types

- Regardless of the code used, we always have:
  - 'A'<'B'<...<'Z'
  - 'a'<'b'<...<' z'
  - '0'<'1'<...<'9'

21

Scalar types: Declared scalar types

- They are user-defined.
- In the definition of such a type, we enumerate the set of values that a variable of this type can take.
- Example : couleur = (bleu, vert, rouge, jaune) ;
- The possible operations are those of the base type.

22

Remark

The relational operators <, >, <=, >=, ≠, and = can be applied to any pair of operands of the same scalar type.

23

The interval type

- It allows the user to define an interval from an associated scalar type (other than the real type).
- This definition specifies a lower bound and an upper bound for the interval.
- Example :
  - Type inter1 = 1..10
    Inter2 = 'a'..'j'

24

Declaration actions *Constant declaration*

CONST ‹Constant Name› = ‹value› ;

Example

```
Const nb_etudiant = 336 ;
      univ = 'université de guelma' ;
      Pi = 3,14;
```

25

Declaration actions
Types declaration

TYPE ‹Type Name› = ‹definition› ;

Example

Type jour=(sam, dim, lun, mar, mer,jeu, ven) ;
index = 1..10;

26

Declaration actions
Variables Declaration

VAR ‹ Variable Name› : ‹Type Name› ;

Example

Var j : jour ;
i : index;
a : char;
x : real;

27

4. Basic Operations
Operator, operand and expression...

- An operator is an operation symbol used to act on variables or perform "calculations".
- An operand is a value or expression that is acted upon by an operator.
- An expression is a combination of operators and operands that is evaluated during the execution of an algorithm. It has a **value** and a **type**.

28

4. Basic Operations
Operator, Operand, and Expression...

- For example, in a+b :
  - a is the left operand
  - + is the operator
  - b is the right operand
  - a+b is called an expression
  - If, for example, a is 2 and b is 3, the expression a+b is 5
  - If, for example, a and b are integers, the expression a+b is an integer

29

4. Basic Operations
Alternative Definition of an Expression...

- An expression is recursively defined by:
  - A constant
  - The name of a variable
  - Applying a basic operation:

    (<expression> <symbol><expression>)

30

4. Basic Operations
Operator...

- An operator can be unary or binary :
  - **Unary** if it admits only one operand, e.g. the non-operand,
  - **Binary** if it admits two operands, e.g. the + operator
- An operator is associated with a data type and can only be used with variables, constants, or expressions of that type.
- For example, the + operator can only be used with arithmetic types (natural, integer and real) or (exclusively) the character string type

31

4. Basic Operations
Operator...

- An integer and a character cannot be added together
- However, *in certain exceptional cases*, it is acceptable to use an operator with two operands of different types, for example with arithmetic types (2+3.5)
- The meaning of an operator can change depending on the type of operands
- For example, the + operator with integers will mean **addition**, but with character strings it will mean **concatenation**.
  - 2+3 equals 5
  - "hello" + "everyone" is "hello everyone"

32

4. Basic Operations
Examples of expressions

Let a, x, y, and som be integer variable names:

- x
- (6 + (5 * 3))
- (true and false)
- ((3 < 8) and ((1 + a) = 7))

Correct Expressions

- (true + a)
- (a or false)

incorrect expressions

33

5. Building a simple algorithm

- A simple algorithm, written in pseudocode or in an algorithmic language, can
  - read input data,
  - process the data by applying the instructions (actions) of the algorithm,
  - and display the results in output.

34

5. Building a simple algorithm

- Input data is stored in variables and constants,
  - each variable has its type.
  - And each constant has its value,
- The results of the calculations are in turn stored in variables using the assignment action (symbol ←).

35

5. Building a simple algorithm

- To write an algorithm, you have to start by answering the following questions
  - What is the problem data (inputs)?
  - What are the required outcomes (outputs)?
  - What is/are the treatment(s) to be carried out?
  - What are the potential errors and special cases that can occur, and what are the proposed solutions?

36

5. Building a simple algorithm
Different Steps in Writing an Algorithm

- Read the problem statement and make sure you understand it :
  - Extract the data manipulated in the problem
  - Think about solving the problem in an abstract way, without considering the constraints imposed by the computer
  - Divide the problem into subproblems, if necessary
- Write the solution (algorithm) on paper, using pseudocode
- Check your solution on an example
- Translate into a programming language
- Test the program on different test sets
  - The general case
  - Check all special cases, and correct any errors

5. Building a simple algorithm

- A simple algorithm can be decomposed into three basic actions, also known as simple actions, which are:
  - Assignment
  - Reading
  - Writing

The Assignment Action: *Effects of an assignment*

The actions carried out during its execution are:

1. Evaluate the expression.
2. If this evaluation returns Error, the assignment is not executed, the general execution ends (it is necessary to avoid this type of situation!). Otherwise, let E be the value of this expression.
3. If the variable has not been declared, the assignment is not executed and the general execution is terminated (this type of situation should be avoided!).
4. If the variable has been declared of a type other than the type of E, the general execution terminates (avoid this type of situation!).
5. Otherwise, the variable's value becomes E.

Simple actions
The Assignment Action

- It is an action that assigns a value to ***a variable***, and ***only one variable***
- It is represented by an arrow ' ←'.
- In general, an assignment is written as follows:

‹variable_name› ← ‹value› or
‹expression› or
‹ variable_name › or
‹const_name›

The Assignment Action:
Example a, b are two variables declared of type Integer.

before		Assignment	after	
Val(a)	Val(b)		Val(a)	Val(b)
×	10	a ← 4		
2	10	a ← (a + 1)		
5	10	a ← (b+2)		
×	10	a ← (a+1)		
×	10	a ← (b+2)		
2	10	a ← ((a+2)*b)		
2	10	a ← ((a et 2)*b)		
2	10	a ← (a< b)		

41

Simple actions
Reading

- This is another instruction used to modify the value of a variable,
- it allows the user to enter a value on the keyboard to be used by the algorithm.
- It is written as follows:

Read (‹variable_name› or ‹A series of variables›);

43

Simple actions
Writing

- This instruction allows the algorithm to communicate values to the user by displaying them on the screen.
- It is written as follows:

write (‹value› or ‹expression› or ‹variable_name› ou ‹constant_name›);

42

Simple actions
Reading: *Effects of a reading action*

1. If the variable has not been declared, the reading action is not executed, and the general execution ends (this type of situation should be avoided!).
2. If the value introduced is of an incompatible type with the type of the variable, the general execution of the program ends. It is important to avoid this type of situation.
3. Otherwise, the value entered on the keyboard is assigned to the variable.

44

Exercise

Write an algorithm to calculate the surface of a square with side C.

45

Solution

Analysis

- Inputs:
  - The square's side
- Outputs
  - The area of the square

The Plan

- Enter the side of the square ***S***
- Calculate of the area ***A*** according to the formula A=S*S
- Display results

46

The Algorithm

```
Algorithm square_Area;
Var S, A : real;
Begin
Write ('please enter the side of the square');
Read (S);
A←S*S;
Write ('The area of this square is: ',A);
end.
```


47

Representing an algorithm by a flowchart

- A flowchart is a schematic representation of an algorithm. It uses a set of symbols to represent the different steps in the algorithm and the flow of control between them.
- Flowcharts are a useful way of visualising and documenting algorithms, and they can be used to communicate algorithms to others.

48

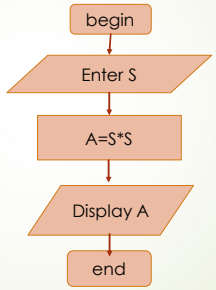
Representing an algorithm by a flowchart

- There is no real standard for flowcharts representing algorithms. However, there are a number of points on which there is consensus:

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectagle represents a process
	Decision	A diamond indicates a decision

49

6. Representing an algorithm by a flowchart



50

7. Translation into C language
general structure of a C program

```
#include <stdio.h>                          /* to be able to read and write*/
int main( )                                          /* Main Program */
{
float S, A;                          /* declaration of two variables S and A */
printf(" please enter the side of the square \n");
scanf("%f", &S);                    /* read the value of S from the keyboard */
A = S*S;                                 /* Calculating the area of the square*/
printf("The area of this square is :%f \n", A);
}
```