PROGRAMMING LOGIC AND TECHNIQUES

Algorithms: Pseudocode

Variables

Definition:

Name used to identify a given location in memory.

Use:

The purpose of using variables is to manipulate values. A variable is therefore a **reserved area in memory**, which has a **name**, and in which it is possible to store a **value**.

Each variable has a type that determines what kind of values it can store. This value can be a number, text, image, sound or other type of data, but it is decided once.

Variables

- In programming, a **variable** can be thought of abstractly as a container that, at any one moment in time, contains a single **value**.

- In a variable/container, there is enough space to fit one value at a time, and no more.
  - (This is true for the basic variables we will be looking at presently.)

Variables

- What is the purpose of variables?
  What do we use them for?
  What do they allow us to do?

- We use a variable as a space or location in which to **store** a piece of information (a value). This allows us to:
  - access and use the information in the variable
  - change and manipulate the information in the variable

Variables

- We can **access** or look at a variable's value at a given point in time while the program is executing.
- We can then use its value in our calculations (i.e., computations that apply operations to values in order to produce other values).
- We can also use its value in computing the path that a given execution of the program will take through the logical flow of the program. This will determine the exact sequence of actions our program will take when it is executing on a particular occasion.

Variables

- We can also change a variable's value (change which value the variable contains) by **assigning** the variable a new value – overwriting the old value by giving the variable a new one.

Variables

- In programming, a variable is a location in memory inside of which we can store a value.

- We use the variable's **name** to identify this reserved area in memory. Using the name in code allows us to both access the variable's value, and to modify the variable's value.

Variables

- When you create your own variable, you get to choose the variable's name.
- As much as possible, variables should be given **meaningful names** that represent their functionality. The more variables a program contains, the more difficult it is to remember the role of each one.
- In addition, you should take into consideration the possibility of other programmers being required to modify your program. The amount of time it can take to properly navigate variables can be considerable if they do not have meaningful names.

Variables

In general, variable names should be:

- Concise (1-30 characters)
- Descriptive of its content and unique
- Case-sensitive (most languages are)
- Use alphanumerical characters and underscores only
- Start with a letter
- Not use reserved words

Data types

- Each variable has a **data type.**
- A variable's type determines all of the possible values that it can have, and how many of these possible values there are.

Data types

- There are many data types.
- Not all programming languages use the same set of data types: some programming languages contain certain data types that other languages do not have.
- Here are the most common types which all programming languages use:
  - **Character:** Any one symbol
    - Ex. A, a, 2, %, , _, >
  - **String:** Data composed of one or more characters (text)
  - **Integer:** Integer
  - **Real:** Real number
  - **Boolean:** Value that is either true or false

Variables

Variable declarations:

Syntax:

VARIABLES

<DataType> : <variableName>

The following two instructions are examples of variable declarations in an algorithm:

VARIABLES

Integer : total

String : c1, letter, z

Variables

VARIABLES

Integer : total

String : c1, letter, z

The first statement declares a variable named **total** that is of the integer type.

The second instruction declares three variables: **c1**, **letter**, and **z**, which are all of the string type.

Variables

- It is important to note that an algorithm's instructions do not take into account the specifications of the language used for program development.

- It is therefore important, when programming, to adjust declarations according to the programming language used.

Constants

Constant declarations:

Syntax:

CONSTANTS

<Type> : <CONSTANT_NAME> ← <value>

The following two instructions are examples of constant declarations in an algorithm:

CONSTANTS

Real : GST_PERCENTAGE ← 0.07

Real : QST_PERCENTAGE ← 0.075

Constants

- The declaration of constants is similar to that of variables, but, as shown above, they must also be **immediately assigned a value** when they are declared.
- This value is permanent

Assignment statements

- Assignment of a variable is an instruction that sets the value of a variable (or, in other words, that puts a value into a variable).

  Syntax:

  <variableName> ← <value>

  Or

  <variableName> = <value>

  Or

  Assign to <variableName> **the value** <value>

Assignment statements

A ← 6 OR

A = 6 OR

Assign to A the value 6

This assignment statement means: put 6 in variable **A.**

It is also possible to assign the value of one variable to another variable. Here is an example:

A ← B OR

A = B OR

Assign to A the value of B

Assignment statements

It is important to note that the following two assignment statements are not identical:

$A \leftarrow B$ (Assign to A the value of B)

$B \leftarrow A$ (Assign to B the value of A)

Assignment statements

There is another kind of assignment method available: the assignment to a variable of values linked by an operator (or multiple operators). Here is an example:

B ← A + 3 OR

B = A + 3 OR

Assign to B the value of A + 3

It is important to note that the value of variable **A** is not changed here, only the value of the variable **B.**

Assignment expressions

- Expressions are in fact assignment statements that allow you to do, among other things, calculations and concatenations. They are grouped into two types: numerical expressions and string expressions.

Numerical expressions

- Numerical expressions are calculations. They can contain the following:

Numerical expressions

- Constants: Constants are numbers. In expressions, these numbers remain fixed, unchangeable, and constant.

  Ex: Price ← BaseAmount * 0.2 (0.2 is a constant)

- Variables
- Operator symbols:

+	(for addition)
–	(for subtraction)
*	(for multiplication)
/	(for division)
%	(modulo: for whole division remainder)
^	(for raising to a power)
SQRT	(for taking the square root)

- Parentheses

String expressions

- String expressions consist of constants and variables. It is important to remember one thing. On one hand, the assignment of numerical literals (constant values typed directly into the code) to variables is done in the following way:

$$A \leftarrow 6$$

String expressions

On the other hand, the same is not true for the assignment of string literals. It is necessary to put string values in quotation marks when we want to put a string directly into a variable. It is therefore necessary to assign values as follows:

A ← “e” and not A ← e

This last assignment would assign to variable **A** the content of variable **e**.

String expressions

- In addition, it is possible to concatenate strings. **Concatenation** is the linking of one string to the end of another. Here is an expression that concatenates two strings:

  A ← “H”, “ello” (A contains “Hello”)

String expressions

It is the same for the concatenation of the values of two variables of type string:

VARIABLE String : E, F, G

E ← “Hello”

F ← “ you”

G ← E, F

G will end up having the value: “Hello you”

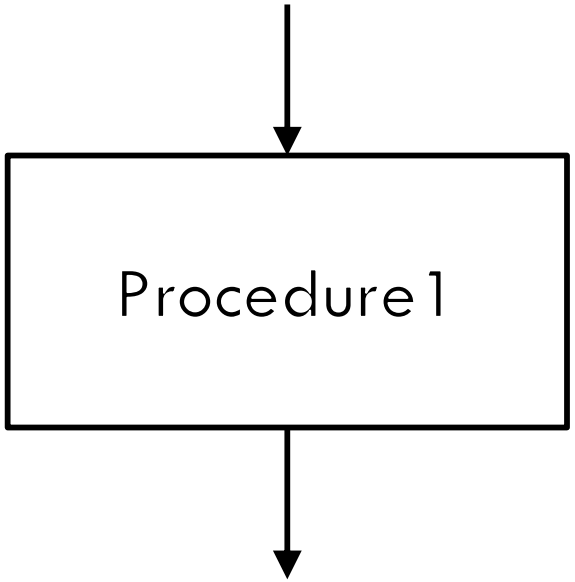
Flowchart diagrams

Algorithms can also be represented as flowcharts.

A flowchart is a diagram composed of boxes that are connected together with arrows. Each box in a flowchart corresponds to a specific function.

Flowchart diagrams

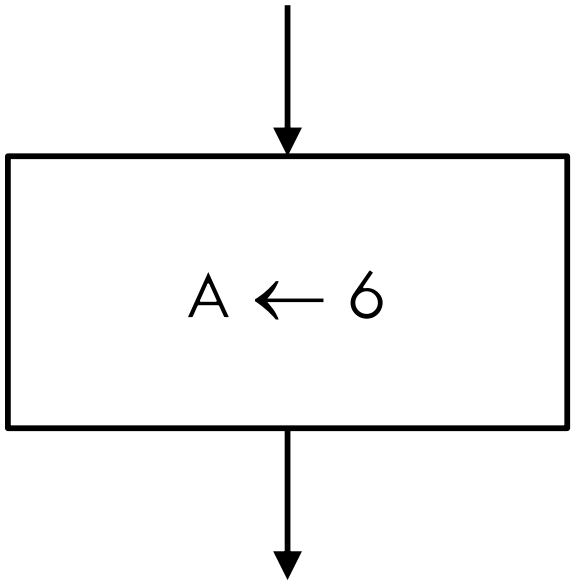
- To represent procedures such as variable declarations and assigning values to variables, we can use flowchart diagrams such as the following:



Flowchart diagrams

- The following flowchart diagram represents the expression

$A \leftarrow 6$



- A flowchart can also be used to represent a collection of instructions.

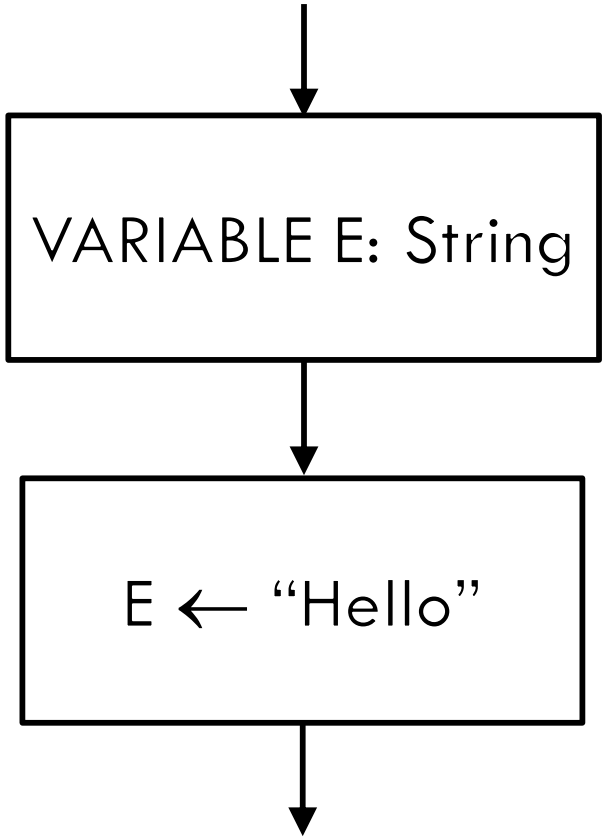
Flowchart diagrams

Here is an example of an algorithm:

VARIABLE String : E

Assign to E the value “Hello”

Algorithm expressed as a flowchart diagram:



Reading and writing

Writing: Displaying or printing a result or the value of a variable on a peripheral device (screen, printer, disk) in the form of one or multiple characters.

Reading: Obtaining data from a peripheral device (keyboard, disk).

Writing

Syntax: **WRITE** <value>

This is an **output statement.**

When a **WRITE** statement is executed, the specified value will be written on (printed to) the device.

Writing

It is thus possible to execute the following statements (assuming the device being written to is a screen):

```
X ← 2
Y ← 3
WRITE X, Y
```

This **WRITE** statement means that the content of variables **X** and **Y** will be written in the order given. So, the result of executing these statements is to display:

23

Writing

Writing data on two lines is done as follows:

```
X ← 2
Y ← 3
Z ← 4
WRITE X,Y
WRITE Z
```

The result of executing these statements is:

23

4

Reading

Syntax: **READ** <variable>

This is an **input statement.**

READ statements are used for obtaining values from some form of input, in order to store them in a variable.

In this course, we can think of this as **user input:** the user gives the program some data by typing it in and pressing Enter.

Reading

Here is an example of using a **READ** statement:

```
READ A
```

This instruction signifies that a value will be obtained from a device and assigned to the variable **A.**

Reading

It is also possible to read multiple values at once. Here is the necessary syntax for reading two variables:

```
READ X, Y
```

This instruction signifies that two values will be obtained and then put in variables **X** and **Y** respectively.

Sequential problems example

- Given the hourly wage and the number of hours worked by an employee, make a program that calculates and displays the employee's gross salary.
- Analysis

  Input:
  The employee's hourly wage
  The number of hours worked

  Output:
  The employee's gross salary

  Constants:
  –

  Procedure:
  The employee's gross salary = the hourly wage multiplied by the number of hours worked.

Sequential problems example

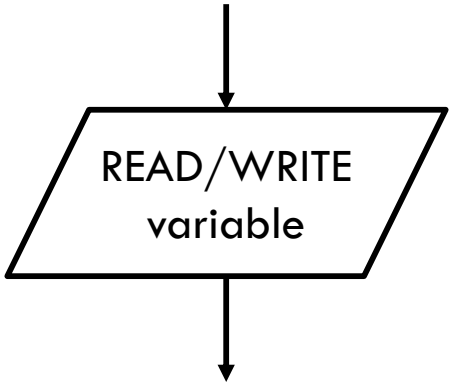
- Pseudocode

```
VARIABLES
  Real : hourlyWage (Input)  // The employee's hourly wage
  Real : numberOfHours (Input) // The number of hours worked
  Real : grossSalary (Output) // The employee's gross salary

START
  READ hourlyWage, numberOfHours
  grossSalary ← hourlyWage * numberOfHours
  WRITE grossSalary
END
```

Flowcharts

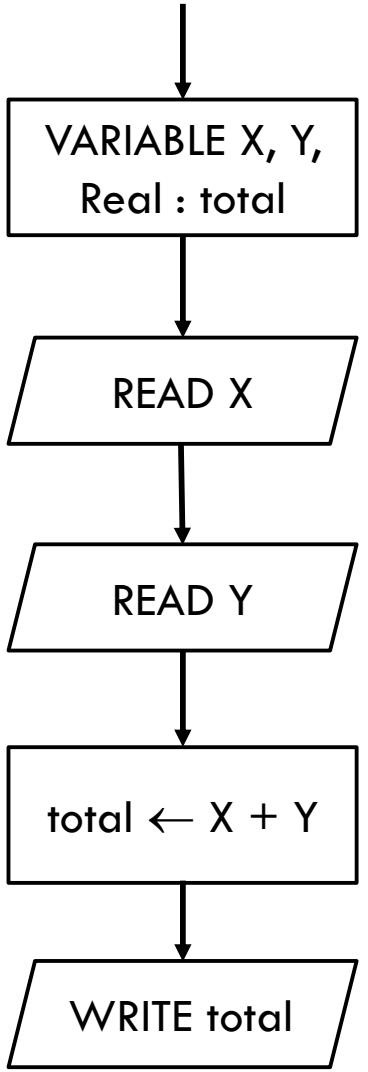
- **READ** and **WRITE** statements are represented in flowcharts by the following box shape:



- This symbol is used in general for input and output (I/O).

Flowcharts

- Here is an example of an algorithm:
  - VARIABLE Real : X, Y, total
    READ X
    READ Y
    total ← X + Y
    WRITE Total
- Algorithm expressed as a flowchart diagram:



Reading and writing in conversational mode

- The reading of data from the keyboard and the writing of resulting values on the screen are often accompanied by the display of additional information. The purpose of such displayed information is to enhance the reading and writing processes for the user, and to better inform the user about the data they are being asked to input and the resulting values that are written to the screen.

Writing a result in conversational mode

- Syntax: **WRITE** "<message>", <value>
  - "<message>": the message to be displayed to the user
  - <value>: value to display

Here's an example of writing a value in conversational mode:

- total ← 5 + 5
  WRITE "Here is the result: ", total

Reading a value in conversational mode

- Syntax: **WRITE** "<message>"
  READ <variable>

Here's an example of reading a value in conversational mode:

```
VARIABLE Real : number
WRITE "Enter a number"
READ number
```

Sequential problems example

- Given the hourly wage and the number of hours worked by an employee, make a program that calculates and displays the employee's gross salary.
- Analysis

  Input:
  The employee's hourly wage
  The number of hours worked

  Output:
  The employee's gross salary

  Constants:
  –

  Procedure:
  The employee's gross salary = the hourly wage * the number of hours worked

Sequential problems example

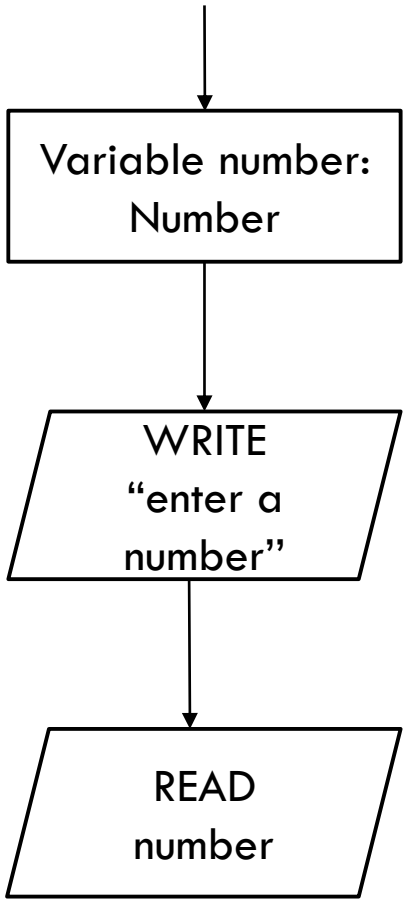
- Pseudocode

```
VARIABLES
  Real : hourlyWage (Input)  // The employee's hourly wage
  Real : numberOfHours (Input) // The number of hours worked
  Real : grossSalary (Output) // The employee's gross salary
START
  WRITE "Enter the employee's hourly wage"
  READ hourlyWage
  WRITE "Enter the number of hours worked by the employee"
  READ numberOfHours
  grossSalary ← hourlyWage * numberOfHours
  WRITE "The gross salary is: ", grossSalary
END
```

Flowchart diagrams

- Conversational mode is represented in flowchart diagrams by the addition of a processing box before the **READ**/**WRITE** box that represents the message displayed to the user.

  The following flowchart represents the algorithm on slide 38:



Exercise

- Write an algorithm that takes an integer value and returns the cube of that value.
- **Input:**

  A value

  Output:

  The cube of the value

  Constants:

  –

  Procedure:

  Raise the value to the power of 3

Answer

```
VARIABLES
   Integer : value

START
   WRITE “Enter an integer value: ”
   READ value
   WRITE “The cube of the entered value is  ”, value ^ 3
END
```

Exercise

Write an algorithm that takes two numbers as input and displays the sum, the difference, the product, and the quotient of the two numbers (without validation)

Input:

The first number

The second number

Output:

The sum, the difference, the product, and the quotient of the two numbers.

Constants:

–

Procedure:

Display the sum of the two numbers

Display the difference of the two numbers

Display the product of the two numbers

Display the quotient of the two numbers

Answer

```
VARIABLE
    Real : number1
    Real : number2

START
    WRITE “Enter two numbers: ”
    READ number1, number2
    WRITE “The sum: ”, number1 + number2
    WRITE “The difference: ”, number1 – number2
    WRITE “The product: ”, number1 * number2
    WRITE “The quotient: ” , number1 / number2
END
```

Exercises

- Make a program that converts US dollars and Euros into Canadian dollars according to the following exchange rates:

US dollars ⟶ Canadian dollars	Rate: $1.48
Euros ⟶ Canadian dollars	Rate: $1.55

This program will need to ask the user to select which type of currency they would like to convert. The user will have to type 1 or 2 to indicate their choice. The program will then require the user to enter the amount of foreign currency they would like to convert. The program will then display the equivalent amount in Canadian dollars. (No input validation required)

Analysis

Input:

The type of currency to convert

The amount of currency to convert

Output:

The equivalent amount in Canadian dollars

Constants:

US dollar exchange rate = $1.48
Euro exchange rate = $1.55

Procedure:

Ask for type of foreign currency (1 for US or 2 for Euro)
Ask for the amount of money to convert
Display the converted amount of money