PROGRAMMING LOGIC AND TECHNIQUES

Algorithms: Conditionals

Exchanging the values of two variables

- The following algorithm serves to exchange the content of one variable with that of another. In other words, this algorithm serves to swap the two variables' values.

- This algorithm is often used in programming, especially for sorting data.

Exchanging the values of two variables

- The program below asks the user to enter two names, and then displays these two names in the order they were entered; that is, **name1** followed by **name2**.

- The program then exchanges the values of the two variables and displays the new contents of the variables, still in the order of **name1** followed by **name2**.

Exchanging the values of two variables

Analysis

Input:

The two names

Output:

The two names in the order they were entered, followed by the two names in reverse order

Constants:

–

Procedure:

Read the two names into two variables. Display the values of these variables in the order they were entered. Then swap the content of the one variable with that of the other, and then display the contents of the two variables again.

Exchanging the values of two variables

- Pseudocode

```
VARIABLES
  String : name1, name2        //The two names entered by the user
  String : temp                //Temporary variable for exchanging
                               //the values of variables name1 and name2

START
  WRITE "Enter the first name: "
  READ name1
  WRITE "Enter the second name: "
  READ name2
  WRITE name1, " and ", name2
  temp ← name1
  name1 ← name2
  name2 ← temp
  WRITE name1, " and ", name2
END
```

Conditional statements

- Programs are not limited in their abilities to just reading, writing, and performing calculations, all in a linear sequence.
- One of the tasks that program must also perform is establishing choices for executing procedures.
- These choices are comprised of one or several precisely-specified **conditions**.
- These conditions are in fact **value comparisons**. Depending on the outcomes of these comparisons, this or that procedure may or may not be executed.

Conditional statements

- Before getting to conditional statements, note the syntactic composition of a condition, given below:

<value1> <relational operator> <value2>

Conditional statements

- Here are the different **relational operators:**

Operator	Meaning
==	Equal to (same value)
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to (different value)

Relational Operators

- Note that =< and => do not perform any meaningful operation.
- In a (compiled) programming language, writing either one of these "operators" would produce a compile-time error

- Also, both sides of a relational operator must be comparable
- Trying to compare a Boolean with an Integer or a String with a Real will not work

Conditional statements

- Here is an example of a **condition:**

  (a == 2)

- Here is an example of a **conditional statement:**

  IF (a == 2) THEN

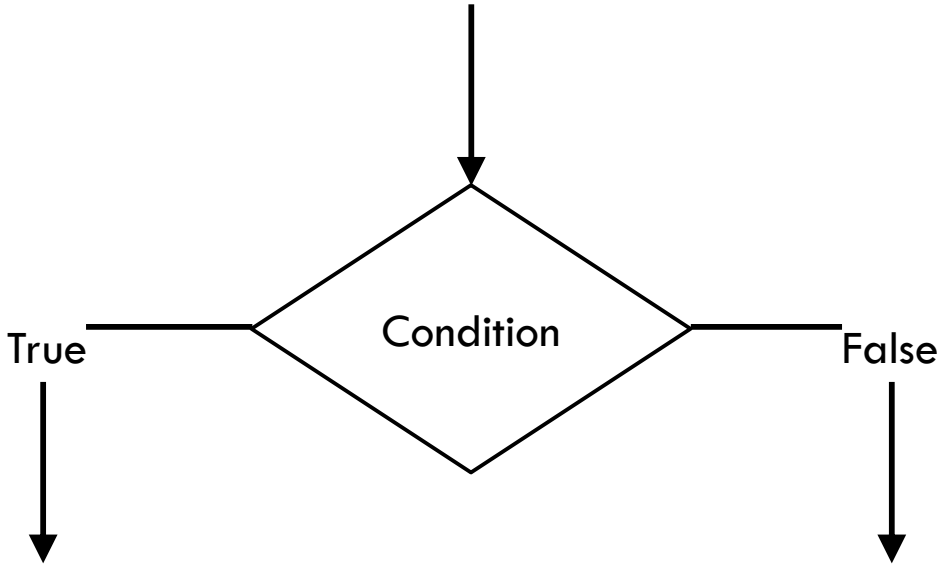
  …

  ENDIF

- This conditional statement is understood as follows:
  if the variable **a**'s value **is equal to** the constant value **2**, then…

Conditional statements

- Here is a representation of a **conditional statement** in a flowchart:



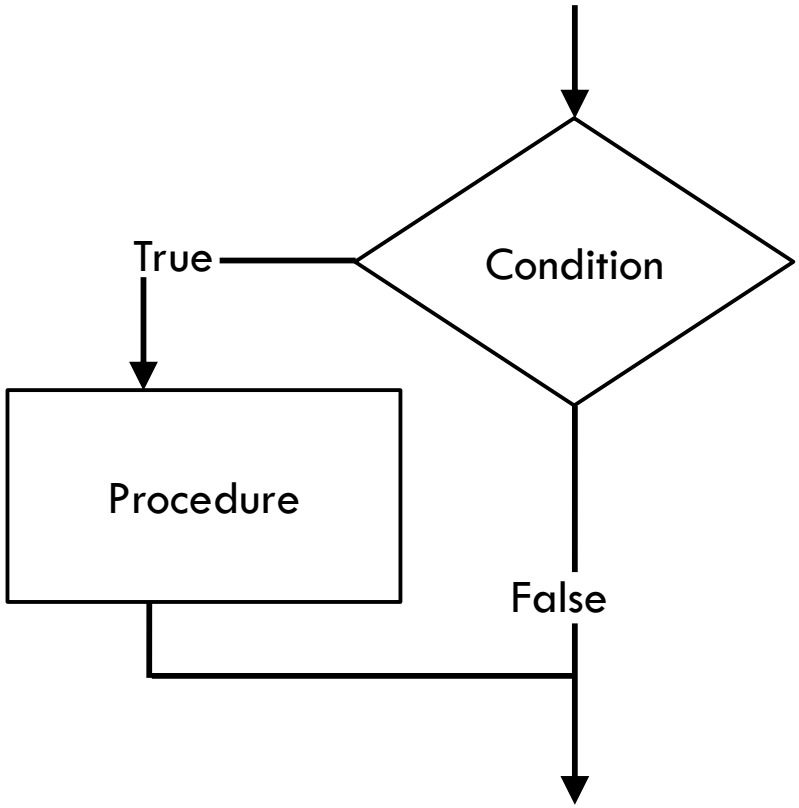
Simple conditional statements: IF

- Syntax: **IF** (<condition>) **THEN**

  <procedure>

  ENDIF

Flowchart diagram:



Simple conditional statements: IF

Here is an example of a **simple conditional statement:**

```
IF (a < 6) THEN
   WRITE "a is less than 6"
ENDIF
```

In this example, the procedure will be executed if and only if the value of variable **a** is less than 6.

Exercise

- Given the hourly wage and the number of hours worked by an employee, make a program that calculates the gross salary of the employee. In addition, for all hours worked overtime (over 40), increase the hourly wage by 50%.

Exercise

Analysis

Input:

The hourly wage of the employee

The number of hours worked

Output:

The gross salary of the employee

Constants:

–

Procedure:

Read the hourly wage and number of hours worked.

If the number of hours worked is greater than 40, then the number of hours worked = the number of hours worked + (the number of hours worked – 40) × 0.5

The gross salary of the employee = the hourly wage × the number of hours worked. Output the result.

Exercise

- Pseudocode

```
VARIABLES
  Real : hourlyWage      // (Input) The hourly wage of the employee
  Real : hoursWorked     // (Input) The number of hours worked
  Real : grossSalary  // (Output) The gross salary of the employee
START
  WRITE "Enter the hourly wage of the employee: "
  READ hourlyWage
  WRITE "Enter the number of hours worked by the employee: "
  READ hoursWorked
  IF (hoursWorked > 40) THEN
     hoursWorked ← hoursWorked + (hoursWorked – 40) * 0.5
  ENDIF
  grossSalary ← hourlyWage * hoursWorked
  WRITE "The gross salary is: ", grossSalary
END
```

Conditional statements: IF-ELSE

- Syntax:

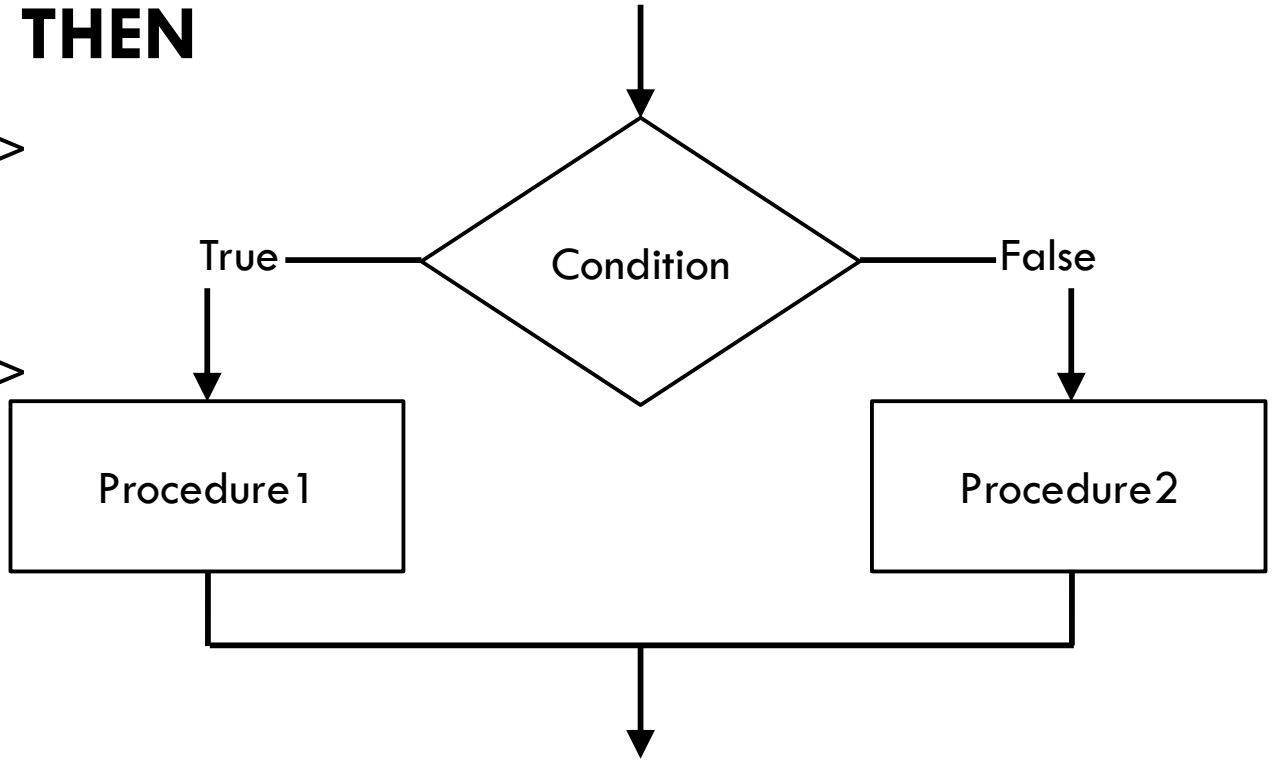
IF (<condition>) **THEN**

<procedure1>

ELSE

<procedure2>

ENDIF



Flowchart diagram

Conditional statements: IF-ELSE

- An **IF-ELSE** conditional statement is similar to a simple conditional statement, but in the case of an **IF-ELSE** statement, if the condition is **not** met, then another procedure is executed. Here is an example of an **IF-ELSE** statement:

```
IF (a < 6) THEN
        WRITE "a is less than 6"
ELSE
        WRITE "a is greater than or equal to 6"
ENDIF
```

Conditional statements: IF-ELSE

```
IF (a < 6) THEN
        WRITE “a is less than 6”
ELSE
        WRITE “a is greater than or equal to 6”
ENDIF
```

If the value of variable **a** is less than 6, then the first procedure will be executed. Otherwise, the second procedure will be executed.

Conditional statements: IF-ELSE

- It is important to note that if there is no procedure to be executed upon a condition **not** being met, then no **ELSE** instruction should be given.

- Example of what not to do:

```
IF (a < 6) THEN
    <procedure1>
ELSE
ENDIF
```

Conditional statements: IF-ELSEIF

It is also possible to carry out a series of conditional executions using an **IF-ELSEIF** or **IF-ELSEIF-ELSE** statement.

Syntax:

IF (<condition1>) **THEN**
    <procedure1>
ELSEIF (<condition2>) **THEN**
    <procedure2>
ELSEIF (<condition3>) **THEN**
    <procedure3>
ELSE
    <procedure4>
ENDIF

Exercise

- Given the hourly wage and the number of hours worked by an employee, make a program that calculates and displays the net salary of the employee. In addition, for all hours worked overtime (over 40), increase the hourly wage by 50%.
- Tax rates vary as follows:
  - 20% for less than $500
  - 30% for $500 and over

Nested conditional statements

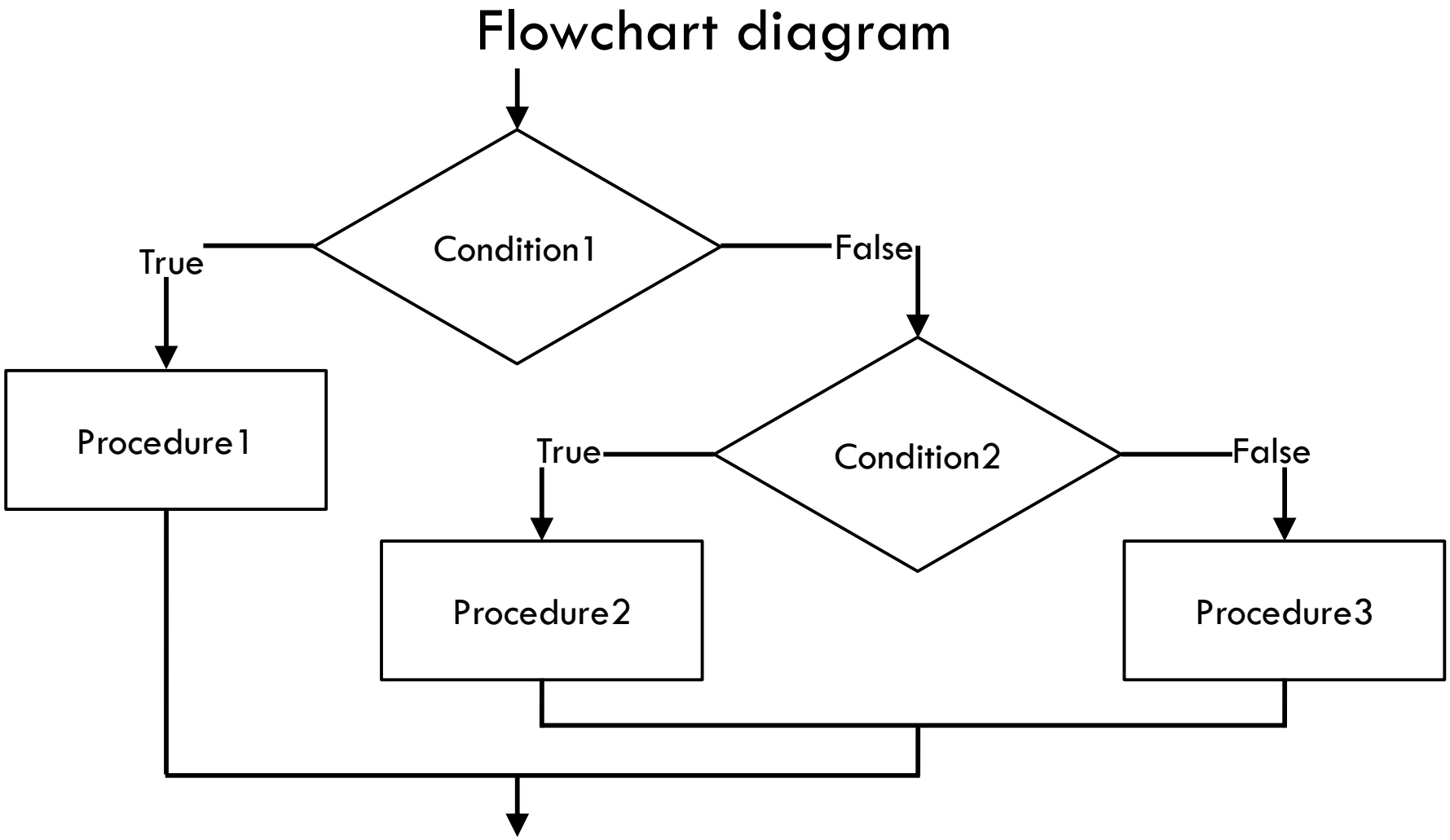
- Sometimes, in order to be executed, certain procedures must be part of a hierarchical sequence of conditional statements.
- To be able to support such a sequence of statements, it is necessary to use **nesting,** which is to say that certain conditions will be evaluated only provided that other conditions have been satisfied.

Nested conditional statements

Syntax:

```
IF (<condition1>) THEN
    <procedure1>
ELSE
    IF (<condition2>) THEN
        <procedure 2>
    ELSE
        <procedure 3>
    ENDIF
ENDIF
```

Nested conditional statements



Nested conditional statements

Here is an example to help explain how nesting works:

```
IF (a == 2) THEN
    <procedure1>
ELSE
    IF (a == 4) THEN
        <procedure2>
    ELSE
        <procedure3>
    ENDIF
 ENDIF
```

Nested conditional statements

- If the value of variable **a** is 2, then **procedure1** will be executed.
- Otherwise, if the value of variable **a** is 4, then **procedure2** will be executed.
- But if neither of those conditions are met, then **procedure3** will be executed.

```
IF (a == 2) THEN
    <procedure1>
ELSE
    IF (a == 4) THEN
        <procedure2>
    ELSE
        <procedure3>
    ENDIF
ENDIF
```

Complex conditional statements

- It is possible to produce a conditional statement with a **complex condition** that is constructed out of two or more simple conditions.

- To construct a complex condition, you must join some simple conditions together with **logical operators**.

Complex conditional statements

Operator	Meaning
AND	The **conjunction** operator. The two conditions that it links must **both** be true for the result to be true.
OR	The **disjunction** operator. Either **one or both** of the conditions that it links must be true for the result to be true.
NOT	The **negation** operator. Gives the **opposite** value of the condition it is applied to.

Complex conditional statements

AND operator

For a complex condition that uses an **AND** operator to link two conditions to be true, **both** of the component conditions (operands) must be true for the complex condition to be true.

The following is a truth table that represents the logic of the **AND** operator.

Complex conditional statements

Truth table: **AND** operator

Condition1	Condition2	Condition1 **AND** Condition2
F	F	**F**
F	T	**F**
T	F	**F**
T	T	**T**

Complex conditional statements

Example of a condition using the **AND** operator:

```
VARIABLE Integer : c
…
IF ((c > 8) AND (c < 12)) THEN
      <procedure>
ENDIF
```

In this example, the procedure will be executed on the condition that the value of variable **c** is greater than 8 **and** less than 12.

So, since **c** is an integer variable, the values of **c** that would cause the procedure to execute are 9, 10, and 11.

Complex conditional statements

OR operator

A complex condition that uses an **OR** operator to link two conditions is true, if and only if: **either one or both** of the component conditions (operands) are true.

Truth Table: **OR** operator

Condition1	Condition2	Condition1 **OR** Condition2
F	F	**F**
F	T	**T**
T	F	**T**
T	T	**T**

Complex conditional statements

Example of a condition using the **OR** operator:

```
VARIABLE Integer : b
…
IF ((b == 2) OR (b == 4)) THEN
    <procedure>
ENDIF
```

Here, the procedure will be executed if and only if the value of **b** is either 2 or 4.

Complex conditional statements

NOT operator

The **NOT** operator **inverts** the value of the condition it is applied to. Thus, if a condition is true, then the negation of that condition is false, and if a condition is false, then its negation is true.

Truth table: **NOT** operator

Condition	**NOT** Condition
F	**T**
T	**F**

Complex conditional statements

Example of a condition using the **NOT** operator:

```
IF (NOT (b == 2)) THEN
    <procedure>
ENDIF
```

The most important thing to understand about this condition is that if there wasn't a **NOT** operator, the condition would be true if the value of **b** was 2, and the procedure would thus be executed. However, with the **NOT** operator, the condition becomes true if and only if the variable **b** is not 2.

Complex conditional statements

In this case, the condition could also have been written as follows:

```
IF (b <> 2) THEN
    <procedure>
ENDIF
```

Complex conditional statements

Complex conditions can be built up out of more than two simple conditions, and each of these can use multiple variables.

Complex conditional statements

Thus, it would be possible to have the following statement:

```
IF (c == 8 OR (a > 12 AND a < 29))
    <procedure>
ENDIF
```

Order of Priority

1. Parentheses ()
2. NOT
3. Arithmetic
    a) * / %
    b) + -
4. Comparison:
    a) Relational
    b) Equality
5. Boolean
    a) AND
    b) OR

Algorithm tracing

- One way to figure out what a given algorithm does (or what a program does, regardless of the language it's written in) is to perform a **trace**.
- Tracing an algorithm consists of putting yourself in the shoes of the computer and executing each line of the algorithm.
- When tracing an algorithm, a table is used to record the values of each of the variables as they change throughout the operation. The last column of the table is used to record what the program displays.

Algorithm tracing: trace tables

Here is an example of a **trace table:**

#	Variable1	Variable2	Variable3	WRITE
	<initial value>			
	<modified value>			
	…			

Tracing becomes very useful when the values of variables change often. This technique is widely used to check that programs, or parts of programs, are doing what they are supposed to, and to locate problems when programs do not behave as expected.

Algorithm tracing: verification test sets

- Verification testing using **test sets** consists of checking whether a program is behaving properly by assigning different values to the variables and performing a trace for each set of values.

- This is very useful for ensuring that a program functions properly in all cases. Verification testing with test sets can help detect some errors that occur only for certain variable values.

Exercise

- A cinema offers two discounts to its customers.
- The first discount is for 10% off if you're a member, and the second discount is for 20% off if you're a student.
- For the discounts to apply, you must present either a membership card to get 10% off, a student card to get 20% off, or both cards to get 30% off.
- The base price of a ticket is $8.50.
- Given this information, make a program that calculates the final price of a ticket.