BTE2313

Chapter 6: CONTROL STRUCTURES (SELECTION)

by
Sulastri Abdul Manap
Faculty of Engineering Technology
sulastri@ump.edu.my

Objectives

- In this chapter, you will learn about:

  1) Flow of control in single and nested if statements given the flow chart

  2) Conditional expressions

  3) Decision/Selection program structure

  4) Repetition/Looping program structure

Introduction

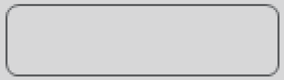
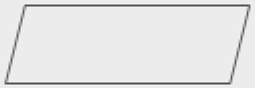
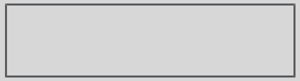
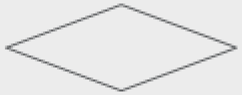
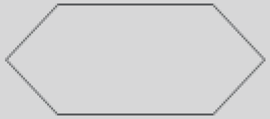
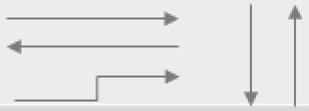
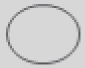
- A computer's pattern of executions are:

1) Sequence (execute from first to last instruction, from top to bottom)
2) Decision/Selection: making a choice which block of instructions to be executed
3) Repetition/Looping: repeat a set of instructions
4) Branch to a separate group of instructions and return (or function calling)

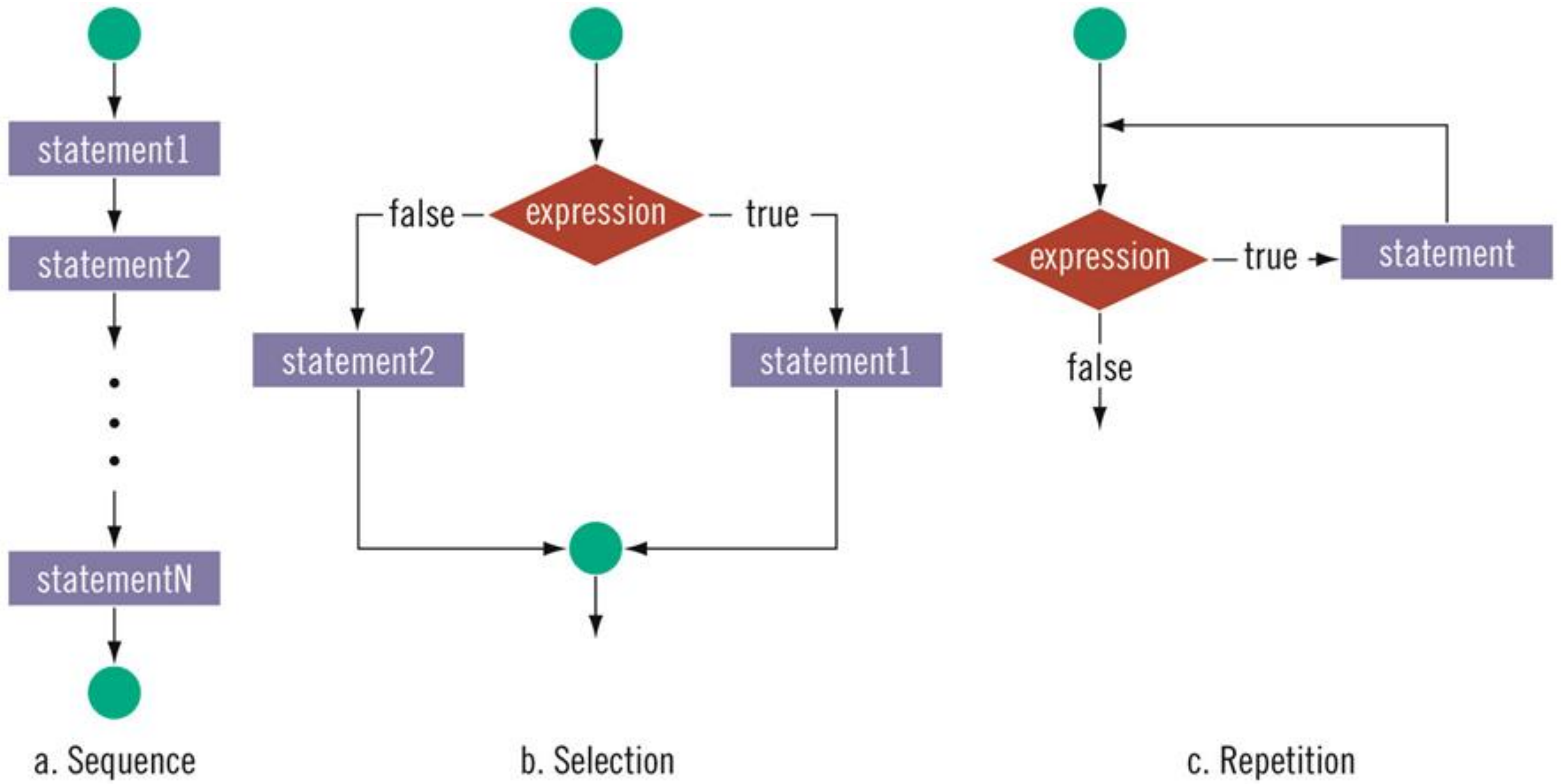
- Any program - no matter how complex - can be written with just the first three patterns: sequence, repetition, decision.
- Branch and return has real advantages in complex programs

Flow Chart

- Any algorithm or process in computer programming can be represented graphically by using Flow chart.
- A flow chart may consists of rectangles, diamonds, ovals, small circles and their flow is represented by arrows.

Shape	Meaning	Description
	Start/End	It marks where the program starts and ends.
	Input/output	Use this symbol to visualize the input and output for the program.
	Process	For any kind of process, usually calculation.
	Selection	When there is a condition involved, if it is true, a different action is taken compared to when it is not true.
	Repetition Selection	When there is a repetition involved, if it is true in the range then a different action will be taken.
	Flow	To visualize the step by step of the actions in the flowchart.
	Connector	Used when there is not enough space to draw the whole flowchart in one page.

Flow Chart (cont.)



Conditional Expressions

- Both repetition/looping and decision/selection statements depend on a conditional expressions (logical expressions either *true* or *false*)
- Usually uses relational operators:

`==`	equality
`!=`	not equal
`>`	greater than
`<`	less than
`>=`	greater than or equal
`<=`	less than or equal
`!>`	not greater than
`!<`	not less than

```
Examples:
(sum == 0)
(i < 10)
(salary != 0.0)
```

Decisions/Selections

- Allows the program to execute one or another set of instructions

→ `if` : a single-selection structure will select or sometimes ignore a single process/action.

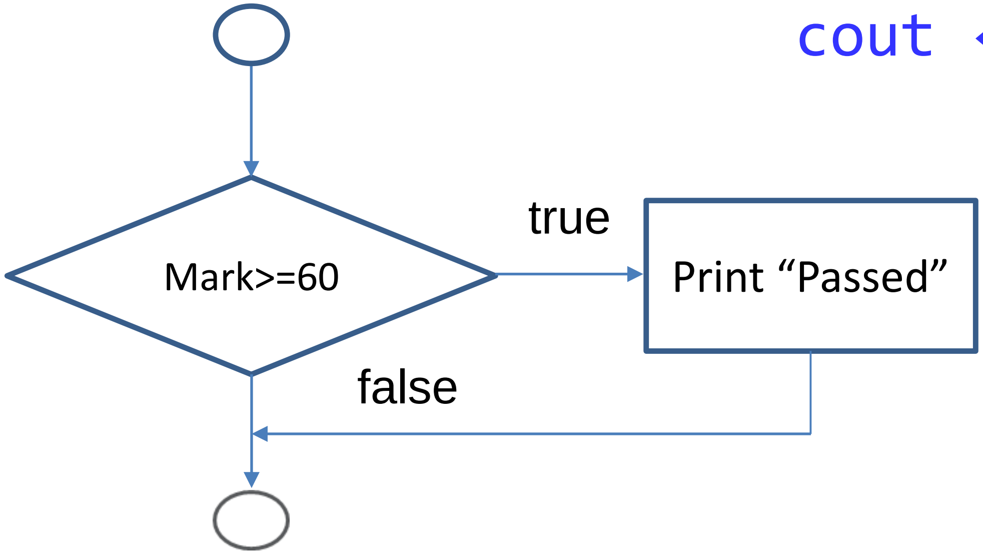
→ `if-else`: a double-selection structure that selects among two different processes/actions.

→ `switch-case`: a multiple-selection structure which selects among several different actions.

`if` Selection Structure

- Used to select or ignore a single action.
- Example: a passing mark on a mid-term test is 60.

```
if (mark >= 60)
   cout << “Passed” << endl;
```



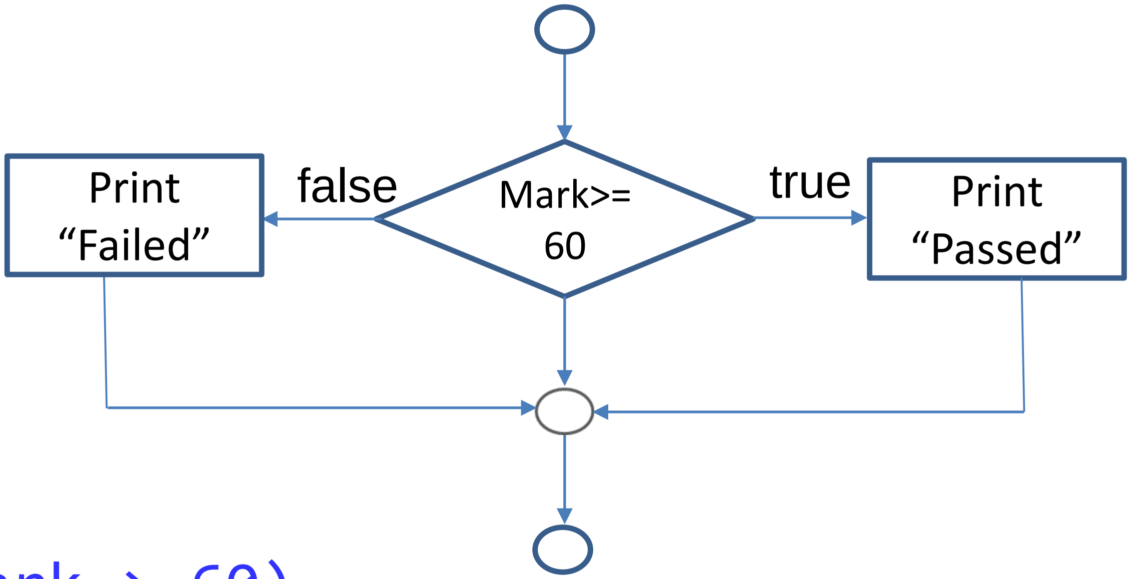
if-else Selection Structure

- if...else statements allow the programmer to select between two different actions
- The syntax for a simple if-else statement is:

```
if (condition)
   statement for condition is true;
else
   statement for condition is false;
```

if-else Selection Structure (cont.)

- Example: if student’s mark is bigger than or equal to 60, display “Passed”. Otherwise, display “Failed”



```
if (mark >=60)
   cout << “Passed” << endl;
else
   cout << “Failed” << endl;
```

`if-else` Selection Structure (cont.)

- If there are more than one statement in the body of `if-else`, put the statements in between curly brackets/braces { and }
- A sequence of statements between curly brackets/braces is called a compound statement.
- Example:

```
if (mark >=60)
   cout << “Passed.” << endl;
else
{
   cout << “Failed.” << endl;
   cout << “See you again next semester.” << endl;
}
```

Nested `if-else` Structure

- Test for multiple cases by placing `if-else` structures inside `if-else` structures.
- Example:

```
If student's mark is bigger than or equal to 80
   Display "A"
else
   If student's mark is bigger than or equal to 70
      Display "B"
   else
      If student's mark is bigger than or equal to 60
     Display "C"
      else
     If student's mark is bigger than or equal to 50
        Display "D"
     else
        Display "F"
```

How its flow chart looks like??

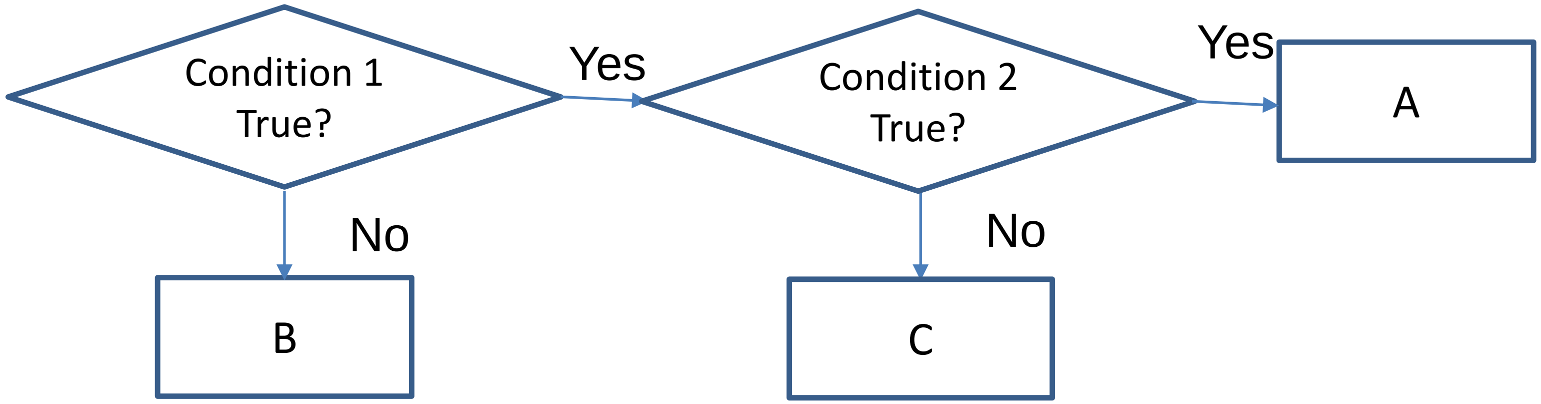
Nested `if-else` Structure (cont.)

```
if (mark >= 80)
    cout <<“A” << endl;
else if (mark >= 70)
    cout <<“B” << endl;
else if (mark >= 70)
    cout <<“C” << endl;
else if (mark >= 50)
    cout <<“D” << endl;
else
    cout <<“F” << endl;
```

If the **`mark`** is bigger than or equal to 80, all 4 conditions from the top are true, but only ONE **`cout`** statement will be executed, which is the first statement. Once that statement is being executed, the rest of statements will be skipped.

Nested `if` Structure

- Involves more than one condition.
- Once the first condition is true, the second condition is checked.

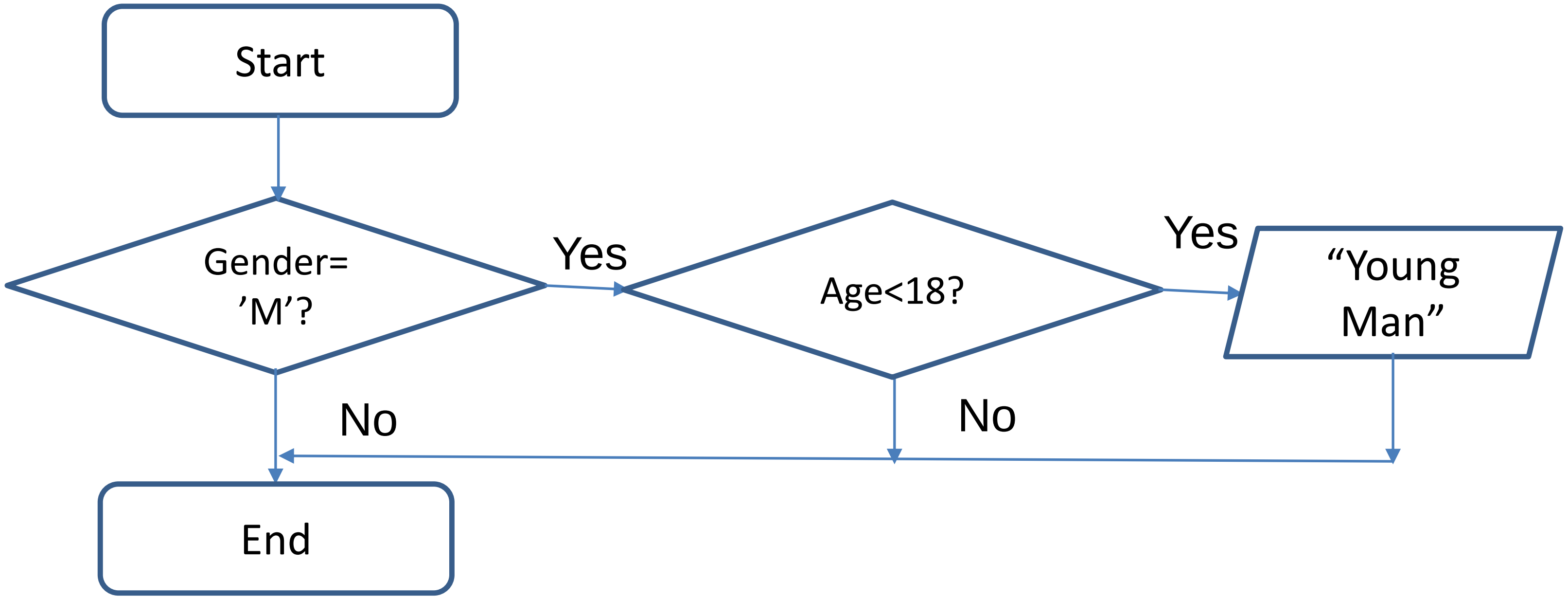


Nested `if` Structure (cont.)

- Example:

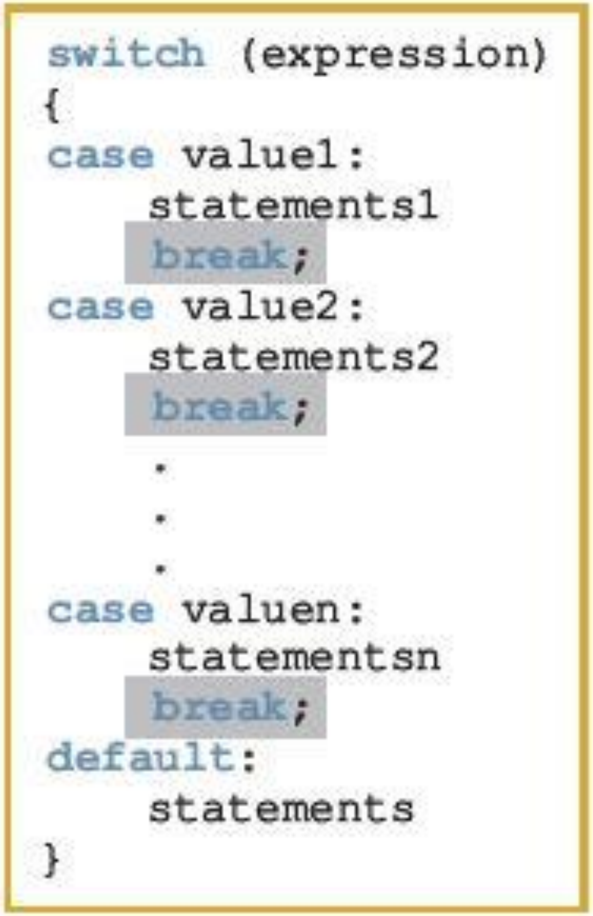
```
if (Gender ==‘M’)
if (Age < 18)
    cout << “Young man”;
```

What if you have too many conditions to check??



`switch-case` Structure

- Provides a convenient alternative to nested `if-else`
- Switch (integral) expression is evaluated first
- Result from the expression determines which action will be taken



```
switch (expression)
{
case value1:
        statements1
        break;
case value2:
        statements2
        break;
          .
          .
          .
case valuen:
        statementsn
        break;
default:
        statements
}
```

switch-case Structure (cont.)

- The expression inside the () after the word *switch* must have an integral value.
- switch statement body must be placed between curly brackets/braces { }
- After each case word, it must be followed by an integral constant value and a colon (:)
- If there is a case that is matched, the statements after that will be executed, until it came upon a break statement.

switch-case Structure (cont.)

- the following if-else statement can be easily converted to a comparable switch statement:

```
if (x == 1) {
    //  Do 1 stuff here . .
}
else if (x == 2) {
    //  Do 2 stuff here . .
}
else if (x == 3) {
    //  Do 3 stuff here . .
}
```

```
switch (x) {
  case 1:
    //  Do 1 stuff here . . .
    break;
  case 2:
    //  Do 2 stuff here . . .
    break;
  case 3:
    //  Do 3 stuff here . . .
    break;
}
```

switch-case Structure (cont.)

- Now consider the following if-else statement:

```
if (x == y) {
    //  Do "y" stuff here . . .
}
else if (x > 2) {
    //  Do "> 2" stuff here . . .
}
else if (z == 3) {
    //  Do 3 stuff here . . .
}
```

- The above code is impossible to be translated into a switch..case statement.
- A variable (in this case y) cannot be taken as a case label.
- The second case checks for an inequality, rather than an exact match, hence it is impossible to translate to a case.