Chapter 4

Decision Structures

Objectives:

- Logical operations and expressions
- Decision structures
- If statement, if/else statement, nested if/else statement
- Switch statement
- Exercises

4. Decision Structures

- Recall that computers execute instructions one at a time
  - In fact most pseudo-code and C instructions are broken down into smaller machine instructions
  - A program can be viewed as a list of instructions to be executed by the CPU
  - Would be nice to have the CPU make decisions on executing certain instructions
- The decision structures
  - Computers can skip instructions, more precisely jump to instructions based on a value of TRUE or FALSE (in reality a value of 1 or 0)
  - From our point of view, decisions are taken once logical expressions have been evaluated.

4.1 Logical Expressions

- Expressions involving operators comparison operators such as “equal”, “greater than”, and logical operators such as AND, OR, and NOT
- Like arithmetic expressions the computer evaluates logical expressions to obtain a value: FALSE (0) or TRUE (1).
- Truth tables for the logical operators:

X	Y	X AND Y	X OR Y	NOT X
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

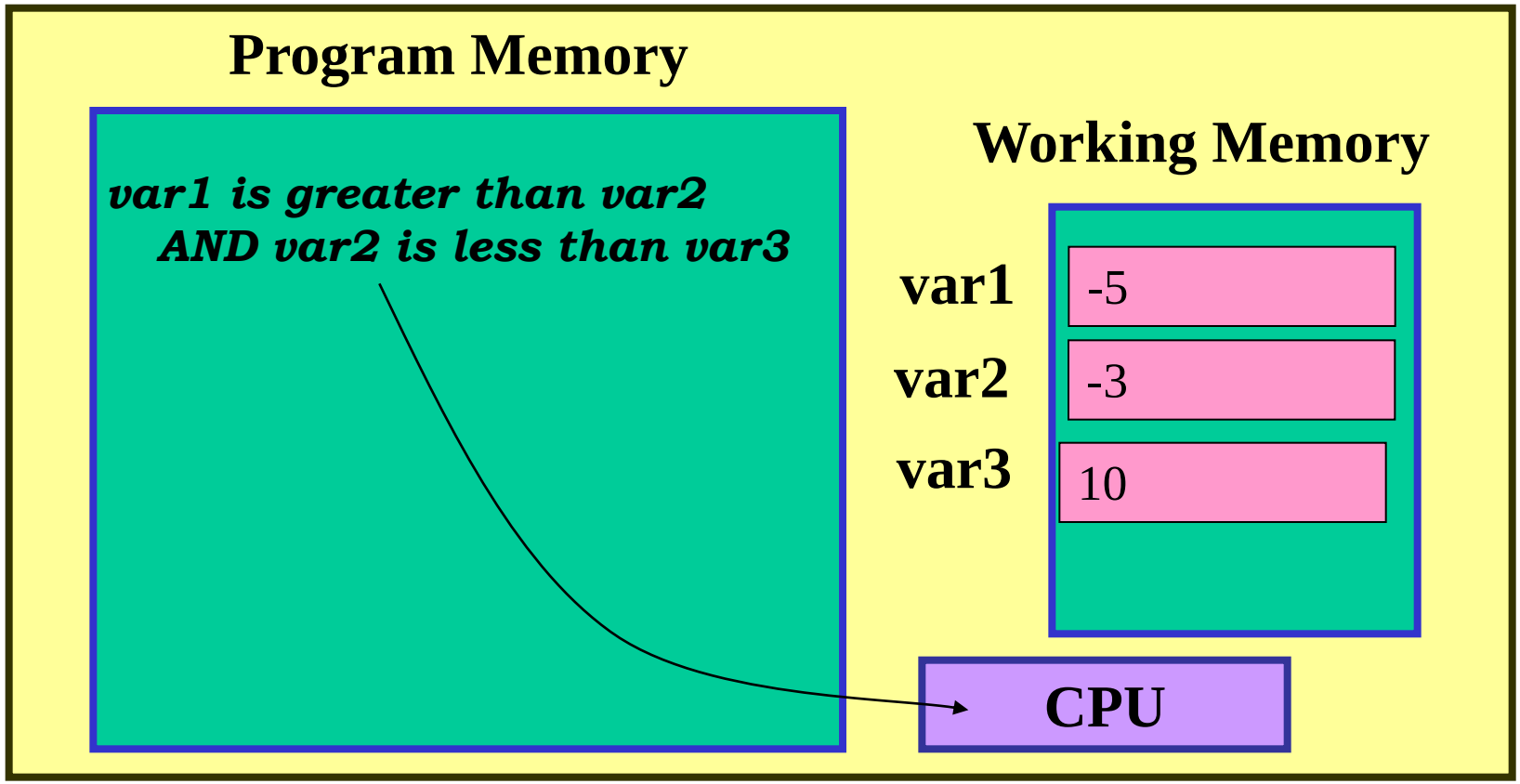
  – In the computer, T is represented with 1 and F represented with 0.

- In pseudo-code, shall write out the comparison and logical operators as in:

  var1 is greater than var2 AND var2 is less than var3

Logical Expressions (continued)

- How does the CPU evaluate the pseudo-code given below
- What is done with the resulting value?
  - As we shall see later, such results can be used in a decision structure



Relational, Equality and Logical Operators in C

- The following relational operators exist in C:

Operation	C Operator	C Example	Pseudo-code
x>y	>	`x > y`	***is greater than***
x<y	<	`x < y`	***is less than***
x≥y	>=	`x >= y`	***is greater or equal to***
x≤y	<=	`x <= y`	***is less than or equal to***

- The following equality operators exist in C:

Operation	C Operator	C Example	Pseudo-code
x=y	==	`x == y`	***is equal to***
x≠y	!=	`x != y`	***is not equal to***

- The following logical operators exist in C:

Operation	C Operator	C Example	Pseudo-code
AND	`&&`	`a && b`	***and***
OR	`||`	`a || b`	***or***
NOT	!	`!a`	***not***

  - The `!` operator is a unary operator while the `&&` and `||` operators are binary operators.

Rules of Logical Operator Precedence in C

- In a logical expression,
  - the logical not `!` operator has highest precedence,
  - the relational operators have the next highest precedence,
  - then come the equality operators,
  - then the logical `&&` operator,
  - and finally the logical `||` operator which has the lowest precedence.

- The following table summarizes in order of precedence the associativity of all the C operators that we have seen to date.

`()`	left to right	
`- ! (type)`	right to left	**unary operators**
`* / %`	left to right	
`+ -`	left to right	
`< <= > >=`	left to right	
`== !=`	left to right	
`&&`	left to right	
`||`	left to right	
`=`	right to left	

- Parentheses can be used in a logical expression. They have the highest precedence and thus can be used to change the precedence of logical operations just like in an arithmetic expression.

- The above rules of precedence and associativity are followed by the C compiler when evaluating expressions that contain an arithmetic part and a logical part.
  - ➔ **You are strongly encouraged to use parentheses in order to clarify such mixed expressions.**

Logical Expressions in C

- The result of a logical expression can only be true or false.

- In C, a logical expression that evaluates to **false** has the integer result `0` and a logical expression that evaluates to **true** has the integer value `1`.

- Here are a few examples of valid logical expressions in C:

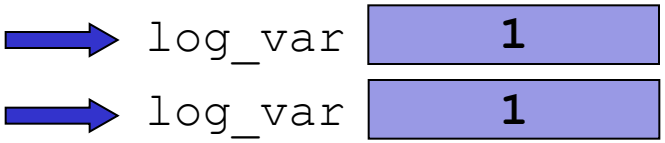
```
int k = -3, l = -4;
float a = 5.5, b = 1.5;
```

	Result
`a < 10.5 + k;`	1
`a + b >= 6.5;`	1
`k != a - b;`	1
`b - k > a;`	0
`!( a == 3 * b);`	1
`-k <= k + 6;`	1
`a < 10 && a > 5;`	1
`k / l >= 0.75;`	0

- In C, logical expressions always evaluate to the integers `1` or `0` (for true or false).
  - ➔ We can use a variable of type `int` to store the result of a logical expression and treat this variable as a logical variable.

E.g.:

```
int log_var, k = -3, l = -4;
float a = 5.5, b = 1.5;
log_var = a < 10.5 + k;
log_var = (a + b) >= 6.5;
```



- In C, a non-zero integer value (`1`, `2`, `-1`...) is interpreted as being true in a logical expression. Suppose that the variables `x` and `y` are of type `int`; shown below are the truth tables for our logical operators in C, operating on the variables `x` and `y`.
  - Truth table for the logical and:

X	Y	X and Y
T	T	T
T	F	F
F	T	F
F	F	F

In a C expression

x	y	x && y
NZ	NZ	1
NZ	0	0
0	NZ	0
0	0	0

  - Truth table for the logical or:

X	Y	X or Y
T	T	T
T	F	T
F	T	T
F	F	F

In a C expression

x	y	x || y
NZ	NZ	1
NZ	0	1
0	NZ	1
0	0	0

  - Truth table for the logical not:

X	not X
T	F
F	T

In a C expression

x	!x
NZ	0
0	1

Example 1

- Translate the following pseudo-code expressions into C expressions.

var2 is larger than or equal to var1

var3 is a positive number

var2 is less than var1 or larger than var3

var2 is between var1 and var3

var3 is not equal to var1

var2 is equal to 5

var3 is even

4.2 The Decision Structure

- Upon evaluation of a logical expression, can specify a set of instructions to execute when the expression is TRUE and another set when the expression is FALSE

  If logical_expression

  Set of instructions to execute if logical_expression is true

  Otherwise

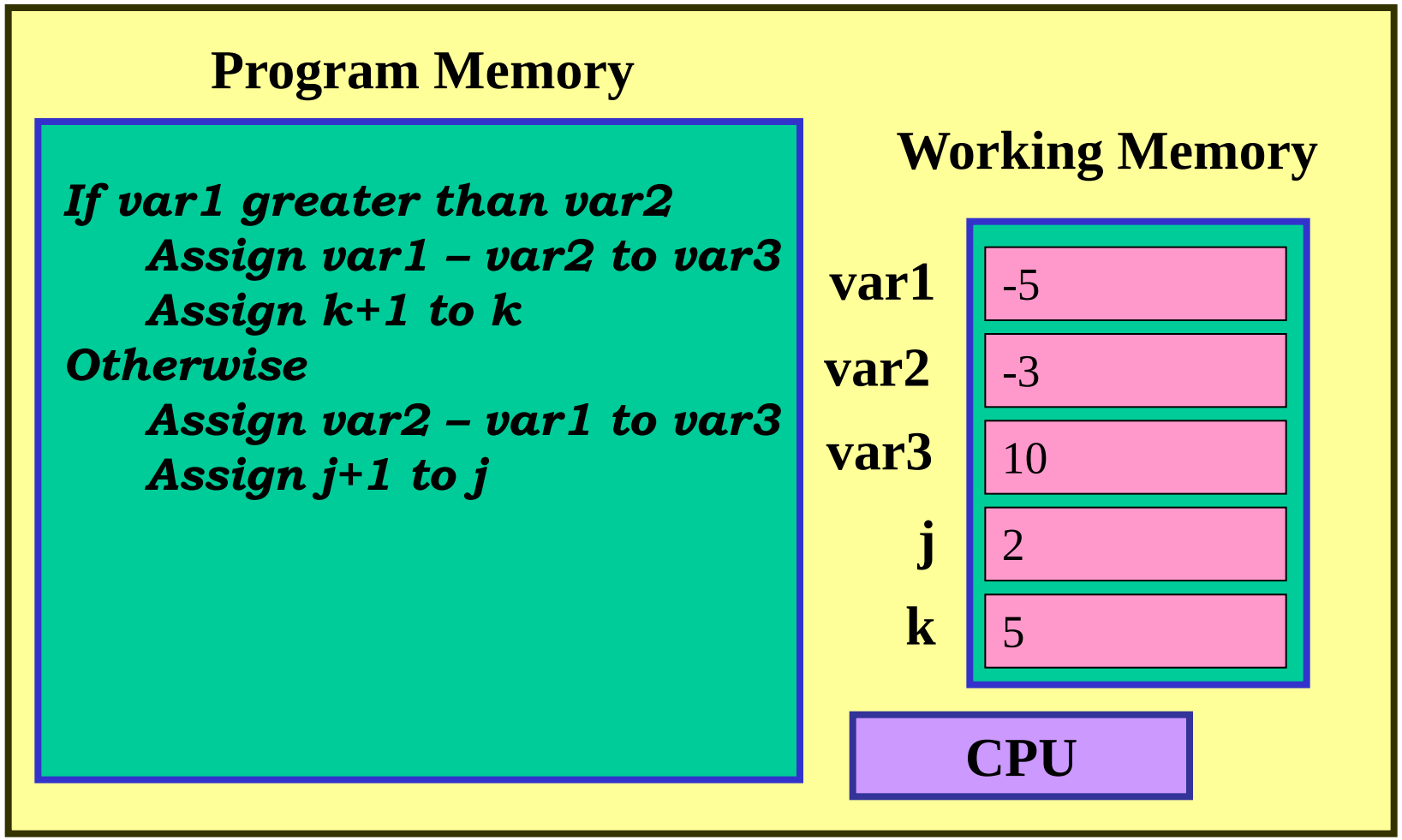
  Set of instructions to execute if logical_expression is false

  – In pseudo-code, all instructions at the same indentation level make up a set of instructions (see example on next slide)

- Nesting decision structures
  – The decision structure is considered a single instruction
  – So it is possible to include another decision structure as part of the set of instructions within a decision structure

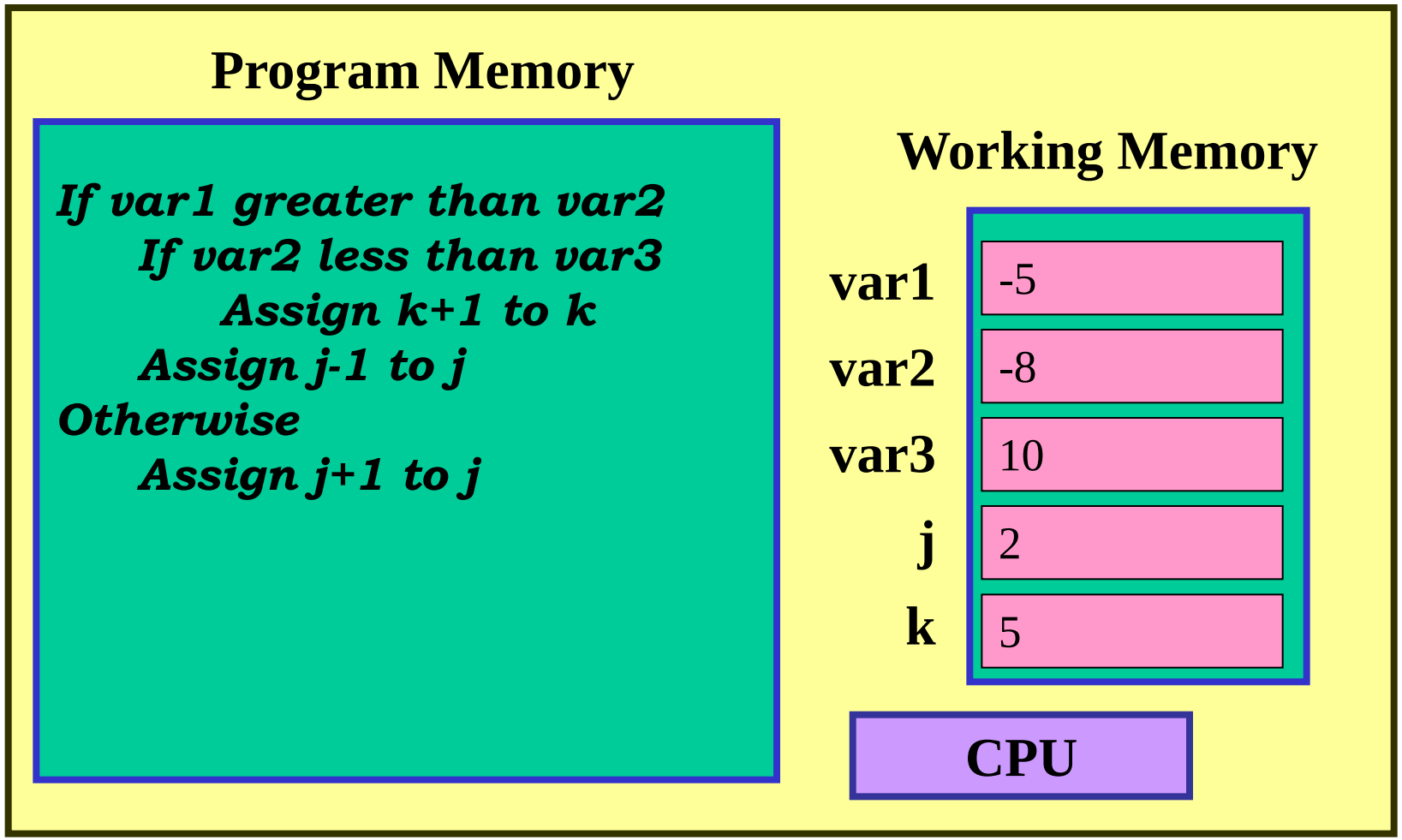
Decision structure: example in pseudo-code

- How does the CPU execute the pseudo-code given below?



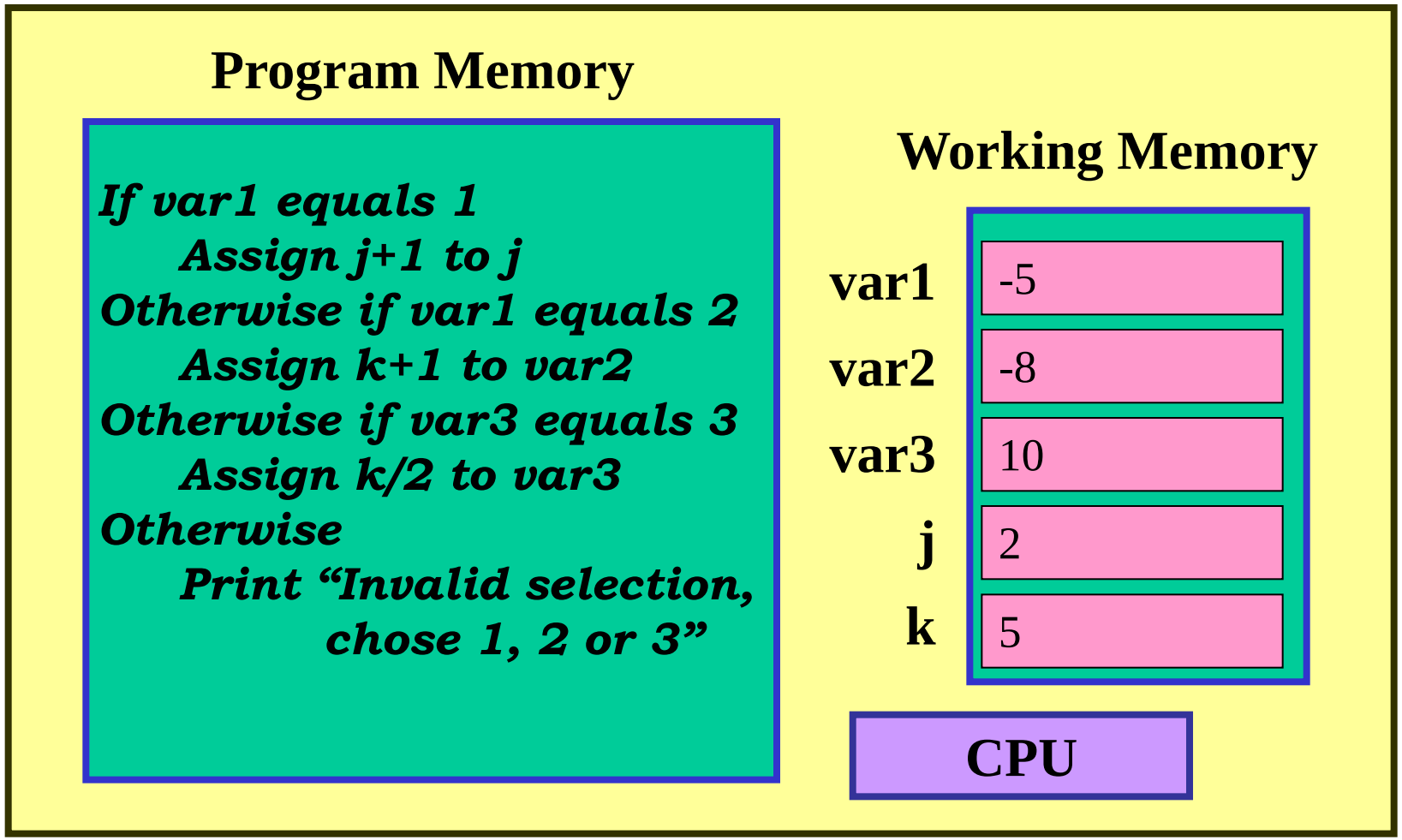
Nested Decision Structure: example in pseudo-code

- How does the CPU execute the pseudo-code given below?
  - Note that the "otherwise" portion of a decision structure is optional



Nested Decision Structure: the otherwise-if structure

- The following form provides a short-hand notation for multiply nested decision structures.



4.3 Decision Structures in C

- Three decision structures exist in C
- The `if/else`
  - The basic decision structure with a set of instructions to be executed if a logical expression is TRUE and an optional set of instructions if the logical expression is FALSE.
- The `Else-if`
  - The short-hand form multiply nested decision structures
- The `switch` (optional use in this course)
  - An alternate form of the `Else-if` structure when testing integer values.

if Decision Structure

- The `if` decision structure allows us to isolate C commands and to have the program execute these instructions only if a logical expression is true.

- Syntax to isolate a single C command:

```
if (logical expression)
    instruction;
```

Careful: there is no `;` here.

The C instruction directly below the `if ()` is executed only if `logical expression` evaluates to true.

- Syntax to isolate many C commands:

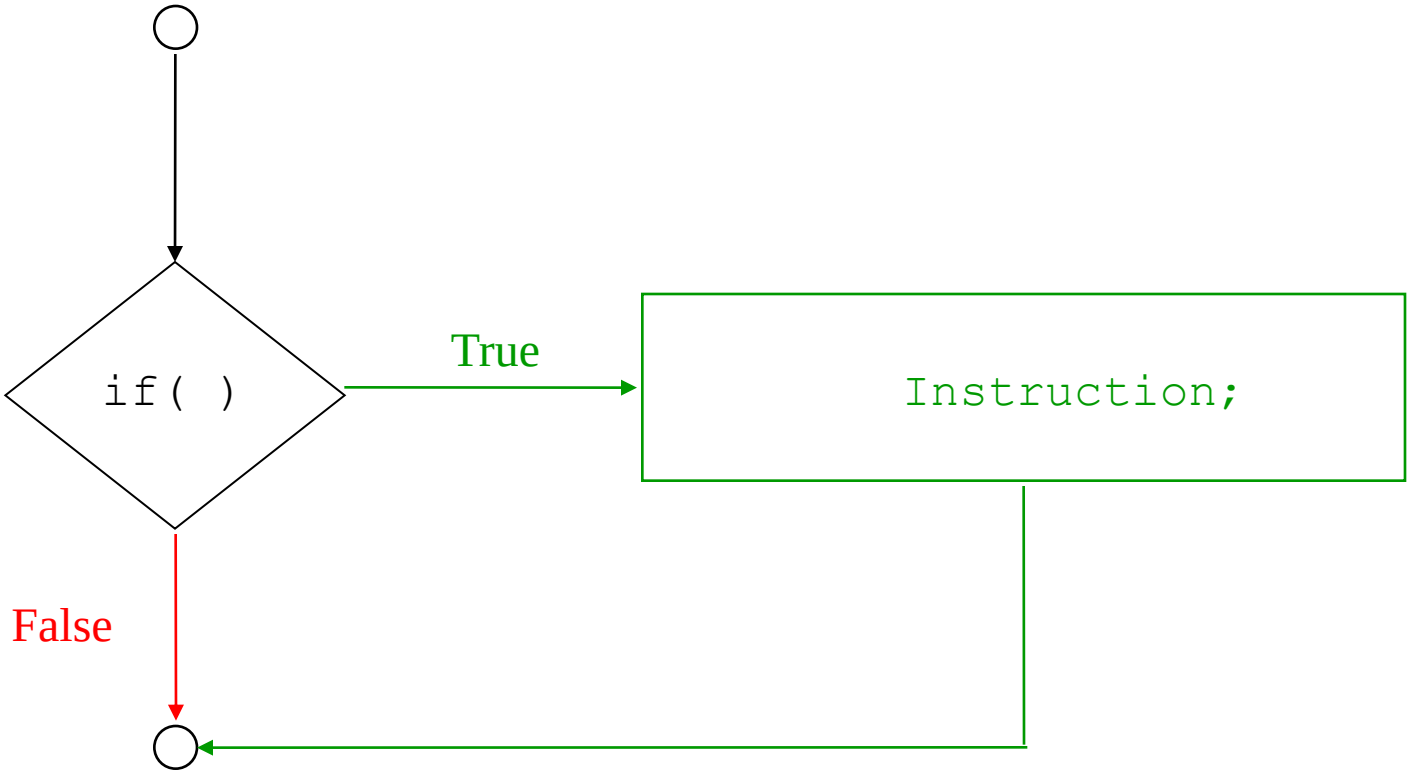
```
if (logical expression)
{
    instruction 1;
    instruction 2;
    :
    instruction n;
}
```

The C instructions delimited by a pair of `{}` directly below the `if ()` are executed only if `logical expression` evaluates to true.

- It is very good practice to indent the instructions under the `if` in order to help visualize the logical structure of the program. Using `{}` to isolate a single instruction is possible and can also help clarify the code.

- A **flowchart** is graphical description of a language structure, algorithm or small program.
  - A flowchart is a convenient way to the visualize logical flow.
  - Lozenges are used to represent decisions and simple instructions are shown as boxes.

- Flowchart of the `if` decision structure:



- Using the `if` structure to isolate one instruction:

```
if (a < 10.0)
    a = a + 1.0;
```

  - or to isolate many instruction:

```
if (a < 10.0)
{
    a = a + 1.0;
    b = b + 1.0;
}
```

If `a` was declared as:
`float a = 5.5;`
then in all of these examples the logical expression evaluates to true and the commands isolated by the `if`'s are executed.

- We can also have nested `if`'s:

```
if (a > 1.0)
    if (a < 10.0)
        a = a + 1.0;
```

  - The above can also be written using logical operators:

```
if (a < 10.0 && a > 1.0)
    a = a + 1.0;
```

- Using the `if` structure to isolate one instruction:

```
if (a > 10.0)
  a = a + 1.0;
```

  - or to isolate many instructions:

```
if (a > 10.0)
{
  a = a + 1.0;
  b = b + 1.0;
}
```

- … nested `if`'s:

```
if (a < 1.0)
    if (a > 7.0)
      a = a + 1.0;
```

  - the above written using logical operators:

```
if (a < 1.0 && a > 7.0)
  a = a + 1.0;
```

If `a` was declared as:
`float a = 5.5;`
then in all of these examples the logical expression evaluates to false and the commands isolated by the `if`'s are not executed.

if/else Decision Structure

- The `if/else` structure allows us to specify a set of instructions to be executed if the logical expression is true and another set of instructions if the logical expression is false.

- Syntax to isolate single C commands:

```
if (logical expression)
  instruction 1;
else
  instruction 2;
```

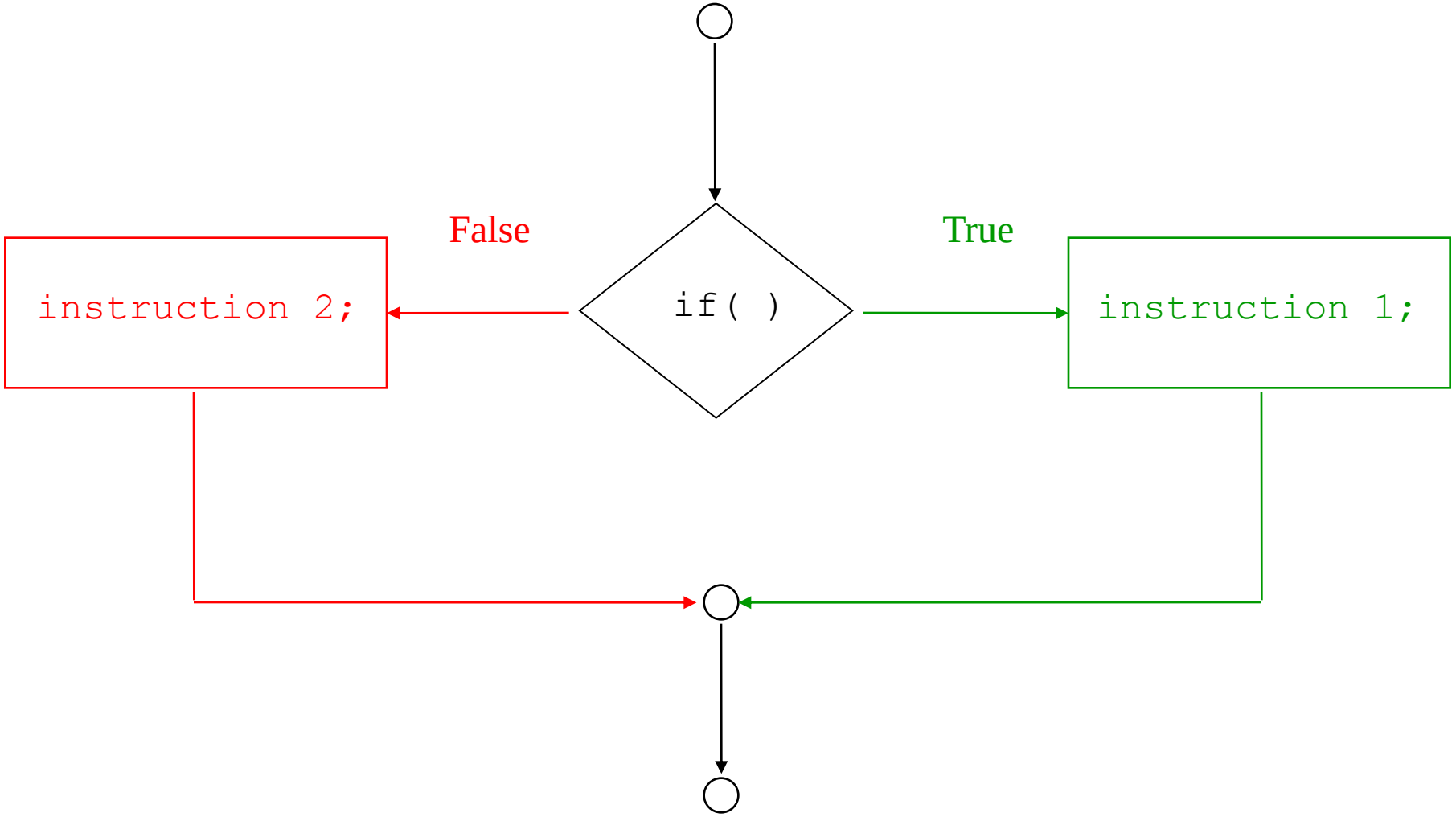
Careful: there is no `;` here.

- Syntax to isolate multiple C commands:

```
if (logical expression) {
    instruction 1;
    instruction n;
}
else {
    instruction 1;
    instruction n;
}
```

Note the difference in the arrangement of the pair of {} in this case; this is also acceptable to the compiler and still clearly delimits the scope of the if and else parts.

- Flowchart of the `if/else` decision structure:



- Example:

```
if (grade >= 90)
    printf("You get an A+.\n");
else
{
    printf("Less than 90.\n");
    printf("Less than A+.\n");
}
```

Executed if grade is greater than or equal to 90.

Executed if grade is less than 90.

- We can also introduce other `if/else` structures into an `if/else` structure:

```
if (x > y)
    if (y < z)
        k = k + 1;
    else
        m = m + 1;
else
    j = j + 1;
```

Executed if x>y and y<z

Executed if x>y and y≥z

Executed if x≤y

- An `else` is associated with the most recent `if`, regardless of indentation.

- The use of `{}` is highly recommended in a complex logic structure to clarify the intended scope of the `if`'s and `else`'s and to help avoid programmer logic errors.
  - Consider the following structure with misleading indentation:

```
if (x > y)
    if (y < z)
        k = k + 1;
else
    j = j + 1;
```

`k = k + 1;` → Executed if x>y and y<z.

`j = j + 1;` → Executed if x>y and y≥z; not if x≤y as was probably intended by the programmer. Recall: an `else` is always associated with the most recent `if`.

  - What the programmer probably wanted is:

```
if (x > y)
{
    if (y < z)
    k = k + 1;
}
else
    j = j + 1;
```

`k = k + 1;` → Executed if x>y and y<z.

`j = j + 1;` → Executed if x≤y.

Exercise 1

- Problem Statement: Software is required to determine if three number are sorted in ascending order.

- Gathering of Information and Input/Output Description
  - Numbers are in ascending order if all numbers are greater than or equal to the previous number in the sequence.
  - Input: Three numbers.
  - Output: Either SORTED or NOT SORTED.

- Test Cases and Algorithm Design
  - Discuss the possible test cases for the software.
  - Design an algorithm using pseudo-code.

- Implementation
  - Translate the pseudo-code into a C-Program

Exercise 2

- Problem Statement: Software is required to evaluate a grade to see if it is pass or fail.

- Gathering of Information and Input/Output Description
  - Grades are assigned from 0 to 100 and a passing grade is 60.
  - Input: A grade value.
  - Output: Either PASS or FAIL.

- Test Cases and Algorithm Design
  - Discuss the possible test cases for the software.
  - Design an algorithm using pseudo-code.

- Implementation
  - Translate the pseudo-code into a C-Program

Exercise 3

- Problem Statement: Software is required to compute the week's pay for an employee given his hourly rate and the number of hours worked for the week.

- Gathering of Information and Input/Output Description
  - The employee is paid at the hourly rate for the first 40 hours worked and double the hourly rate for the time worked after the forty hours.
  - Input: Hourly rate and number of hours worked.
  - Output: Week's pay.

- Test Cases and Algorithm Design
  - Discuss the possible test cases for the software.
  - Design an algorithm using pseudo-code.

- Implementation
  - Translate the pseudo-code into a C-Program

- The following decision structure is commonly encountered and is created from many levels of nested `if/else`'s:

```
if (choice == 1)
  printf("First choice selected.\n");
else if (choice == 2)
  printf("Second choice selected.\n");
else if (choice == 3)
  printf("Third choice selected.\n");
else
  printf("Wrong selection!\n");
```

  - The variable choice is compared with the integer values 1 to 3.
  - If a logical expression is true (say the first one) then the message that follows is printed to the screen (`First choice selected.`) and execution will continue with the command following the structure; ie: after `printf("Wrong selection!\n");`
  - The last `else` (`Wrong selection!`) is executed only if none of the `if`'s are true.
  - The above is such a popular decision structure that programmers rarely add indentation and `{}`. Exceptionally, this is not considered to be in bad programming style.

- Note however, how indentation and { } clarifies the intended logic in the previous example:

```
if (choice == 1)
    printf("First choice selected.\n");
else
{
    if (choice == 2)
        printf("Second choice selected.\n");
    else
    {
        if (choice == 3)
                printf("Third choice selected.\n");
        else
                printf("Wrong selection!\n");
    }
}
```

  - The above executes in exactly the same way.

Exercise 4

- Problem Statement: Software is required to test a temperature value in centigrade to see if it freezing, normal, or too hot.

- Gathering of Information and Input/Output Description
  - The temperature (°C) is freezing when below 0 °C, normal when in the range 10 to 25 °C, and too hot when above 40 °C, and considered out of range when it does not fall in either of the 3 ranges specified.
  - Input: A temperature value in °C.
  - Output: Either FREEZING, NORMAL, TOO HOT, or OUT OF RANGE.

- Test Cases and Algorithm Design
  - Discuss the possible test cases for the software.
  - Design an algorithm using pseudo-code.

- Implementation
  - Translate the pseudo-code into a C-Program

Exercise 5

- Problem Statement: Software is required to translate a student numerical mark to a letter grade.
- Gathering of Information and Input/Output Description
  - The grade must fall in the range of 0 to 100.
  - Grade is translated according to the following marking scheme:

    Grade between 90 and 100 => A

    Grade between 80 and 90 => B

    Grade between 70 and 80 => C

    Grade between 60 and 70 => D

    Grade less than 60 => F
  - Input: Student numerical grade.
  - Output: Student letter grade.
- Test Cases and Algorithm Design
  - Discuss the possible test cases for the software.
  - Design an algorithm using pseudo-code.
- Implementation
  - Translate the pseudo-code into a C-Program

- Be Careful with the logic in nested `if/else` structures.
  - Let's say that we have the following decision structure:

```
if (grade >= 90)
  printf("You have an A+\n");
else if (grade >= 85)
  printf("You have an A\n");
```

  - The above works correctly; the variation shown below however, does not.

```
if (grade >= 85)
  printf("You have an A\n");
else if (grade >= 90)
  printf(You have an A+\n");
```

  - See the problem? What happens if `grade` is equal to `90`?

Using = and = =

- Be Careful when using the assignment operator = and the equality operator ==; mixing then up may not cause a compilation error but will of course cause an execution error that can be more difficult to detect.
  - Consider the following commonly encountered error:

Intended code	Erroneous code
`if (a == 2)`	`if (a = 2)`
`printf("…\n");`	`printf("…\n");`
True only when `a` is equal to `2`.	Always true since the logical result of an assignment is always true.

  - The following is another popular mistake:

Intended code	Erroneous code
`a = 2;`	`a == 2;`
Assigns the integer `2` to the variable `a`.	Logical expression is evaluated during execution but no assignment occurs; `a` remains unchanged.

`switch` Decision Structure

- The `switch` decision structure is commonly used when a variable must be compared to many different values and an appropriate action(s) taken when a `case` is true.
- Example of the syntax (the variable `choice` is declared to be an `int`):

```
switch (choice)
{
  case 1:
    printf("1.\n");
    break;

  case 2:
    printf("2.\n");
    break;

  default:
    printf("Error!\n");
    break;
}
```

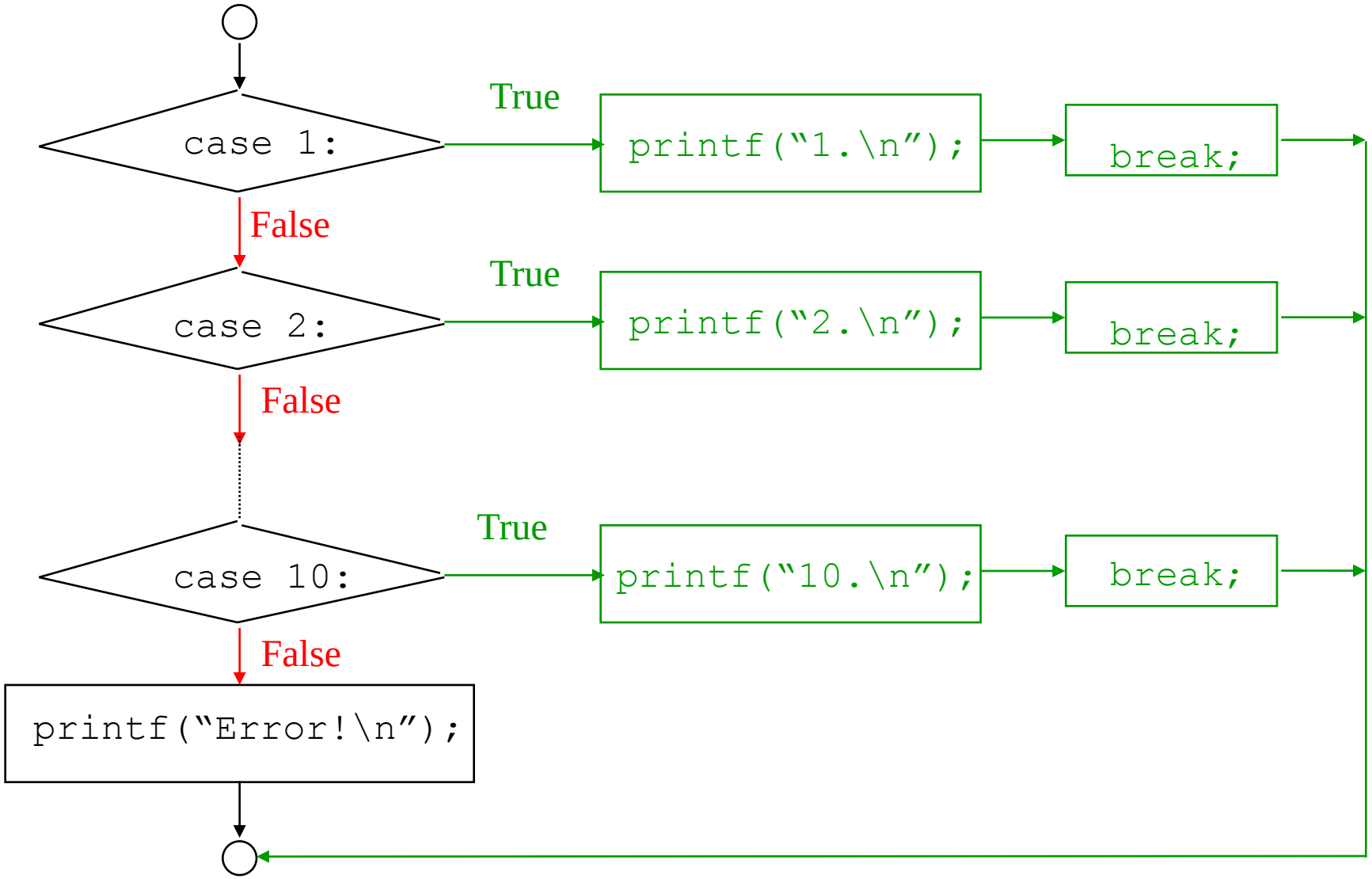
- The `switch` structure compares an expression that yields an integer to an integer value and executes the command(s) after the first `case` that is found to be true.

- The command `break;` forces the execution of the program to resume after the `switch` structure.
  - If `break;` is not included, then all of the commands that occur after the first true `case`, will be executed.

- If none of the `case`s are true then the commands located after `default:` will be executed.

- The `{}` are not required under a `case` since all commands will be executed until a `break;` or the end of the `switch` structure is encountered.

- Flowchart of the `switch` decision structure:



Exercise 6

Develop a program that prompts the user to input a choice to determine one of the followings. (use the switch statement)

1. The area of a circle. A = pi*R2
2. The surface area of a cylinder. A = 2*pi*R*L
3. The volume of a cylinder: Vc = pi*R2*L
4. The volume of a sphere. Vs = (4/3)*pi*R2

- Start with developing the pseudo-code
- Then translate the pseudo-code to a C program


Thank You!
Ευχαριστώ
MULTUMESC
ขอบคุณ
Vielen Dank
Merci
Teşekkürler
DMnvwd
شكراً
متشكرم
Gracias
Grazie
Bedankt
Dankie
THANK YOU
Köszönettel
Hvala
Obrigado!
Asante
شکریا
Díky
謝謝
GADDA GUEY
WAD MAHAD
SAN TAHAY
감사합니다
Urakoze