Lecture 4:

Decision Structures: *if* / *else* and *switch* statements

Prof. Shervin Shirmohammadi
University of Ottawa

The *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:

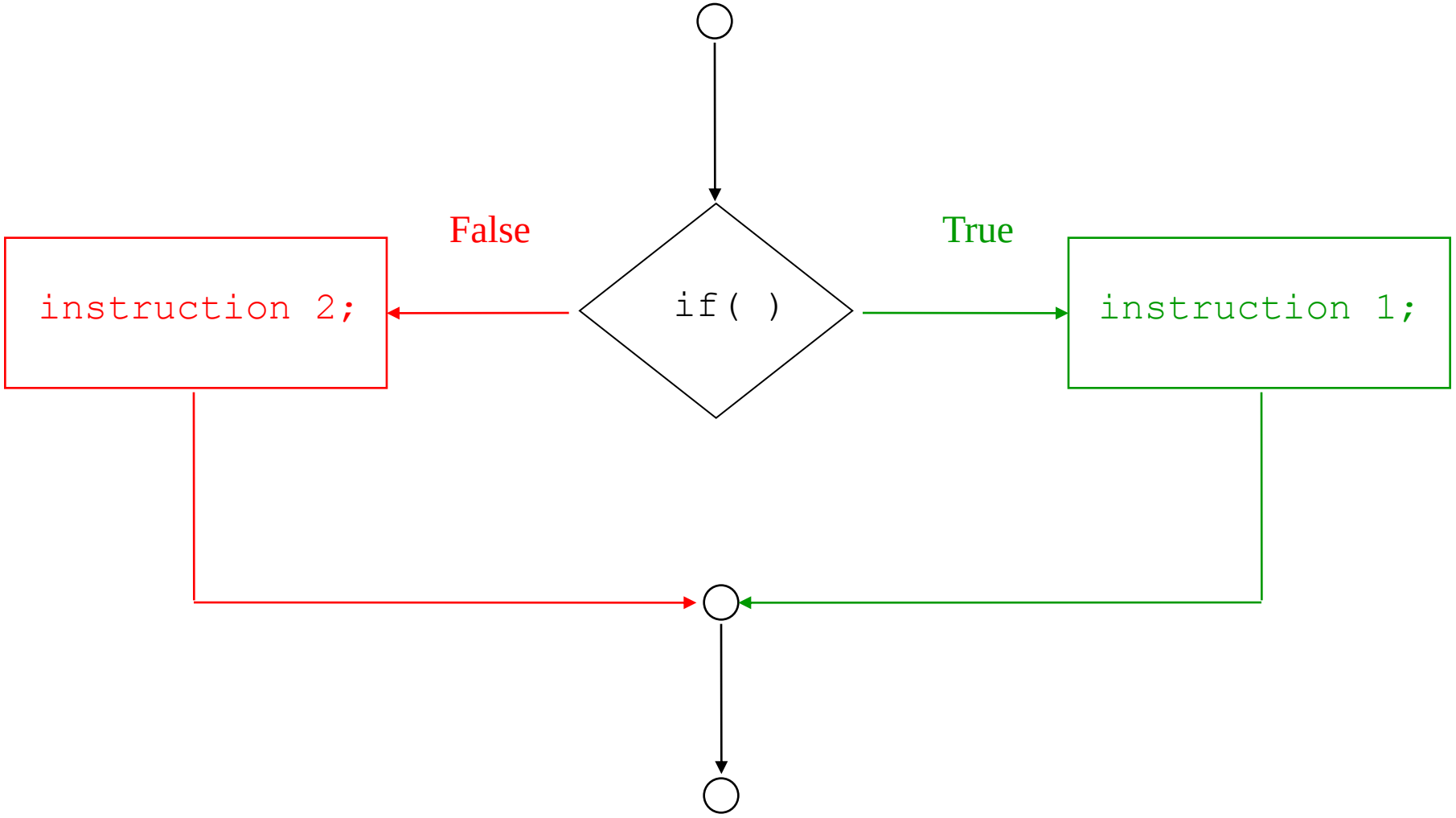
```
if (logical expression)
  instruction 1;
else
  instruction 2;
```

- Syntax to isolate multiple C commands:

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

Careful: there is no `;` here.

- 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 have other `if/else` structures inside 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

- **NOTE:** An `else` is associated with the most recent `if`, regardless of indentation. The only exception is, as always, brackets, which dictate precedence.

- The use of `{}` is highly recommended in a complex logic structure to clarify the intended scope of the `if` and `else` 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;
```

Executed if x>y and y<z.

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;
```

Executed if x>y and y<z.

Executed if x≤y.

Example

- Write a program that prompts the user to input the temperature in degrees centigrade. The program should then print "Freezing" if the temperature is less than zero, "Clear" if the temperature is between 0 and 32, “Too hot" if the temperature is more than 32.

- 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; i.e., 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 a bad programming style.

Example

- Write a program that prompts the user to input the grade of a student and prints out the equivalent letter mark. Use the following grading system:

  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
  Otherwise => print: “This grade is invalid!”

- 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`?

The *switch* Decision Structure

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

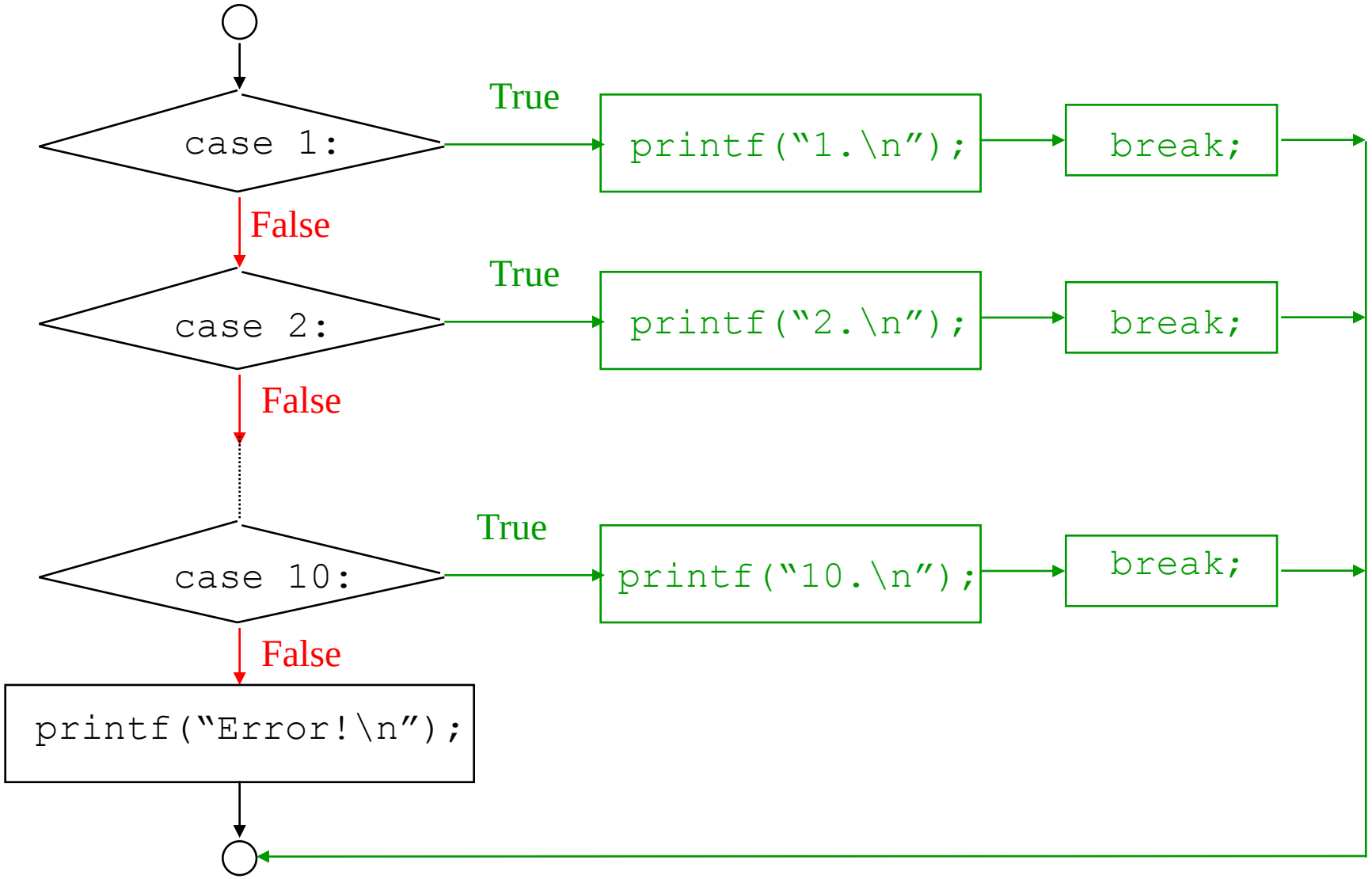
```
switch (choice)
{
  case 1:
    printf("1.\n");
    break;

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

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

- The `switch` structure compares an expression to a 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.
- NOTE: expression can only be integer, char, short, and long. **Float and double are not allowed**!

- Flowchart of the `switch` decision structure:



Example

Write a program that prompts the user to input an integer value between 1 and 5 and prints the corresponding word for the number. (if the user inputs 4, the program prints: ‘Four’)

Using = and = =

- Careful when using the assignment operator = and the equality operator ==; mixing them 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.