# Introduction to Programming

https://www.unive.it/data/course/493929

# 3. Control flow

**Giulio Ermanno Pibiri** — giulioermanno.pibiri@unive.it
Department of Environmental Sciences, Informatics and Statistics

Academic year 2023/2024

# Overview

- Conditional statements: `if-else` and `switch`
- Loop statements: `for`, `while`, `do-while`
- Un-conditional statements: `break` and `continue`
- Nested loops and infinite loops
- Exercises

# Control flow statements

- In a C program, statements are executed **sequentially**, i.e., one after the other.
- Sometimes it is, however, necessary to not execute a statement (or a block of statements) **based on some conditions**.
- For example, we would like to express in code following pattern:

  "*If some condition is satisfied, then execute some statements, otherwise do not (or execute other statements).*"

# Control flow statements

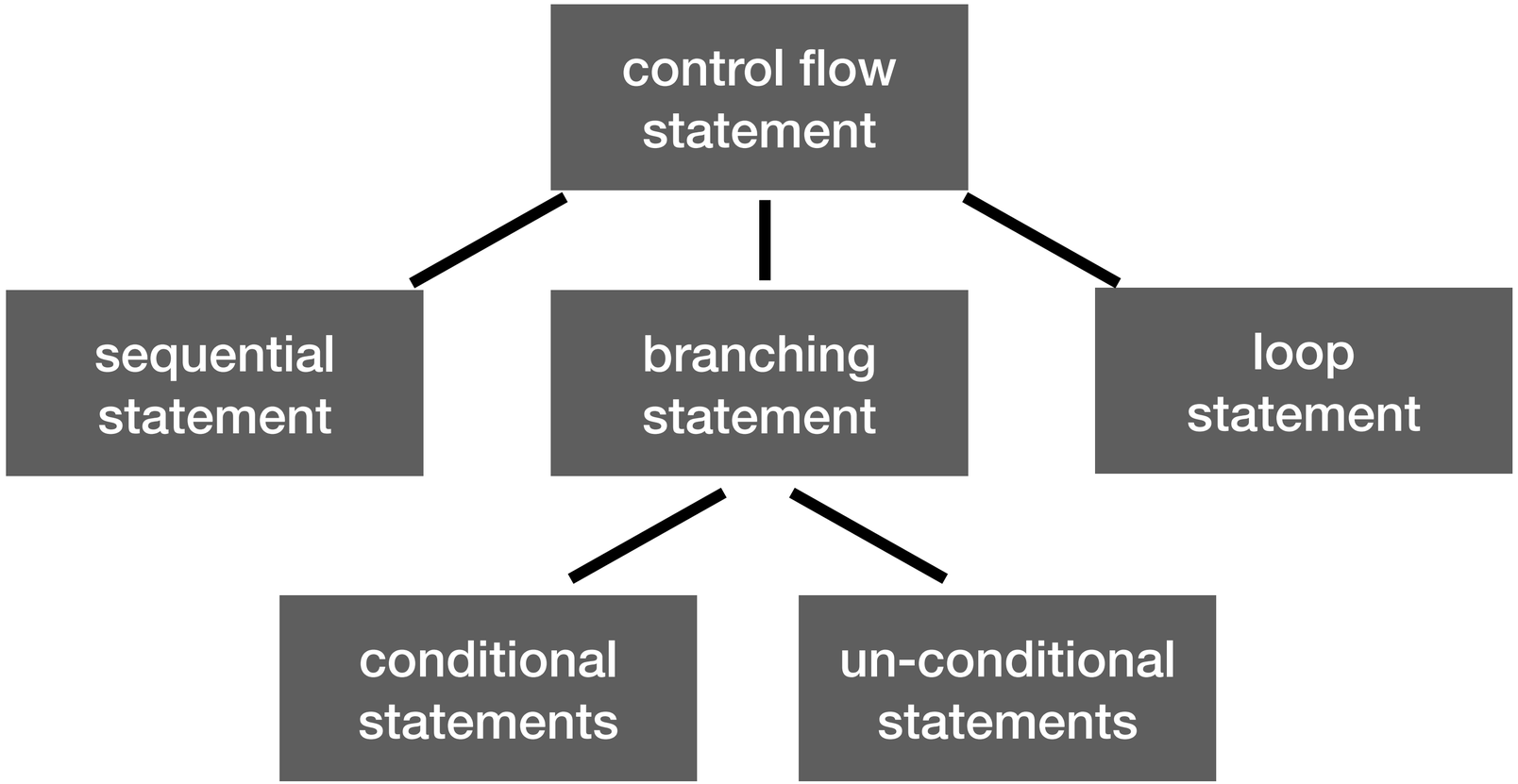
- In a C program, statements are executed **sequentially**, i.e., one after the other.
- Sometimes it is, however, necessary to not execute a statement (or a block of statements) **based on some conditions**.
- For example, we would like to express in code following pattern:

  "*If some condition is satisfied, then execute some statements, otherwise do not (or execute other statements).*"

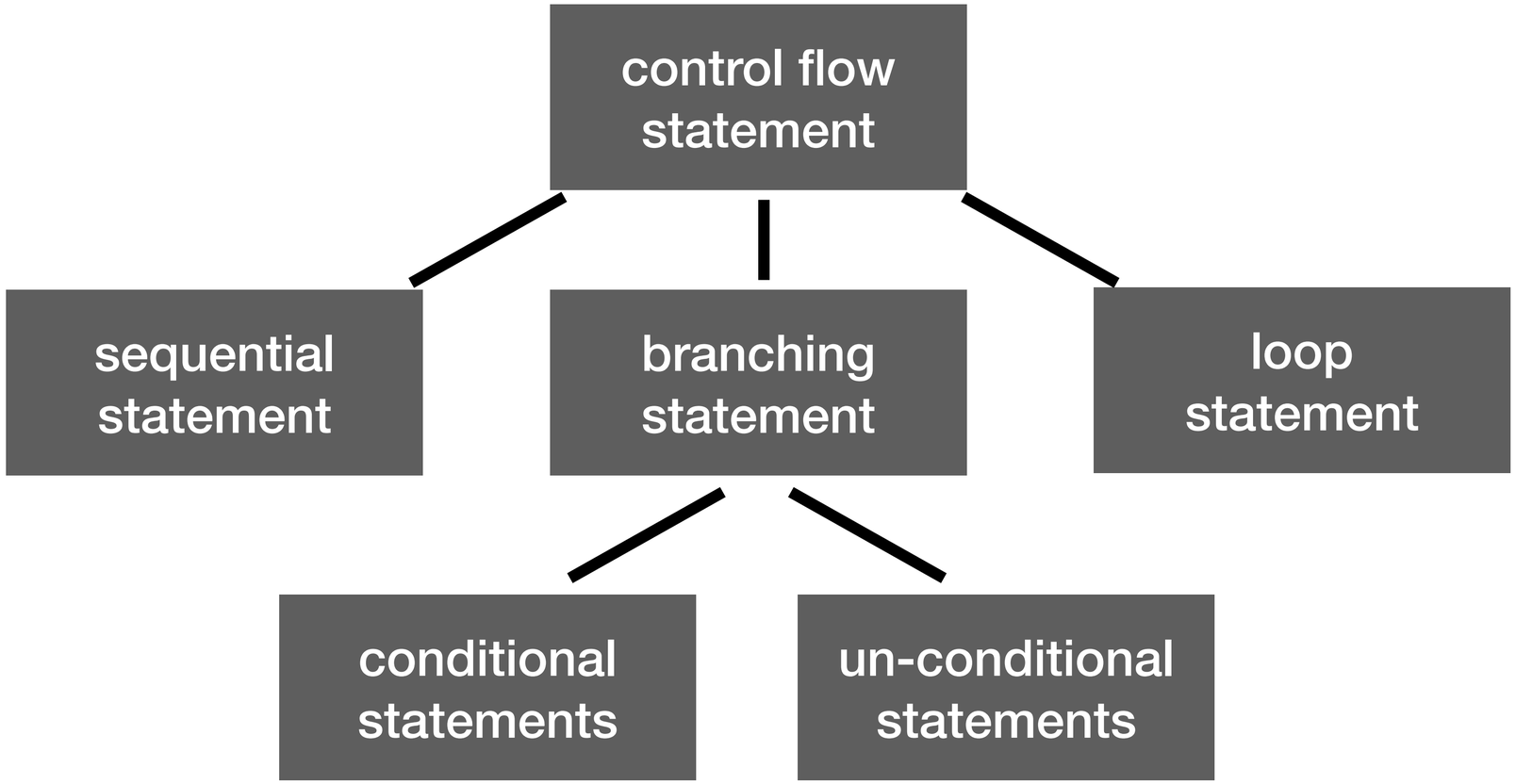
- Also, it is useful to have the ability to **repeat** a block of statements until a given condition is not met.

  Example: "*Until the value of the variable `i` is not 100, increment `i` by 2.*"

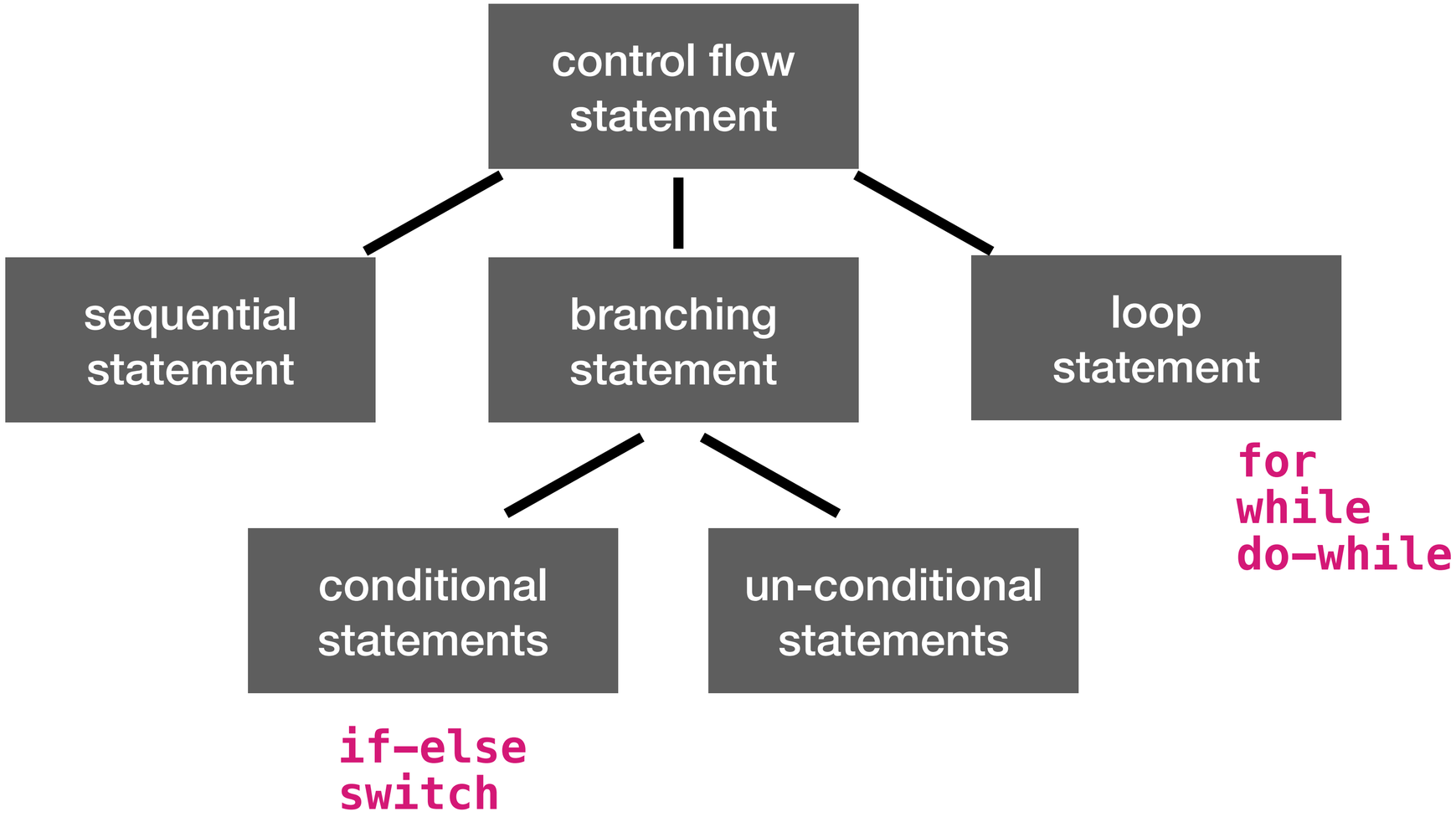
# Control flow statements



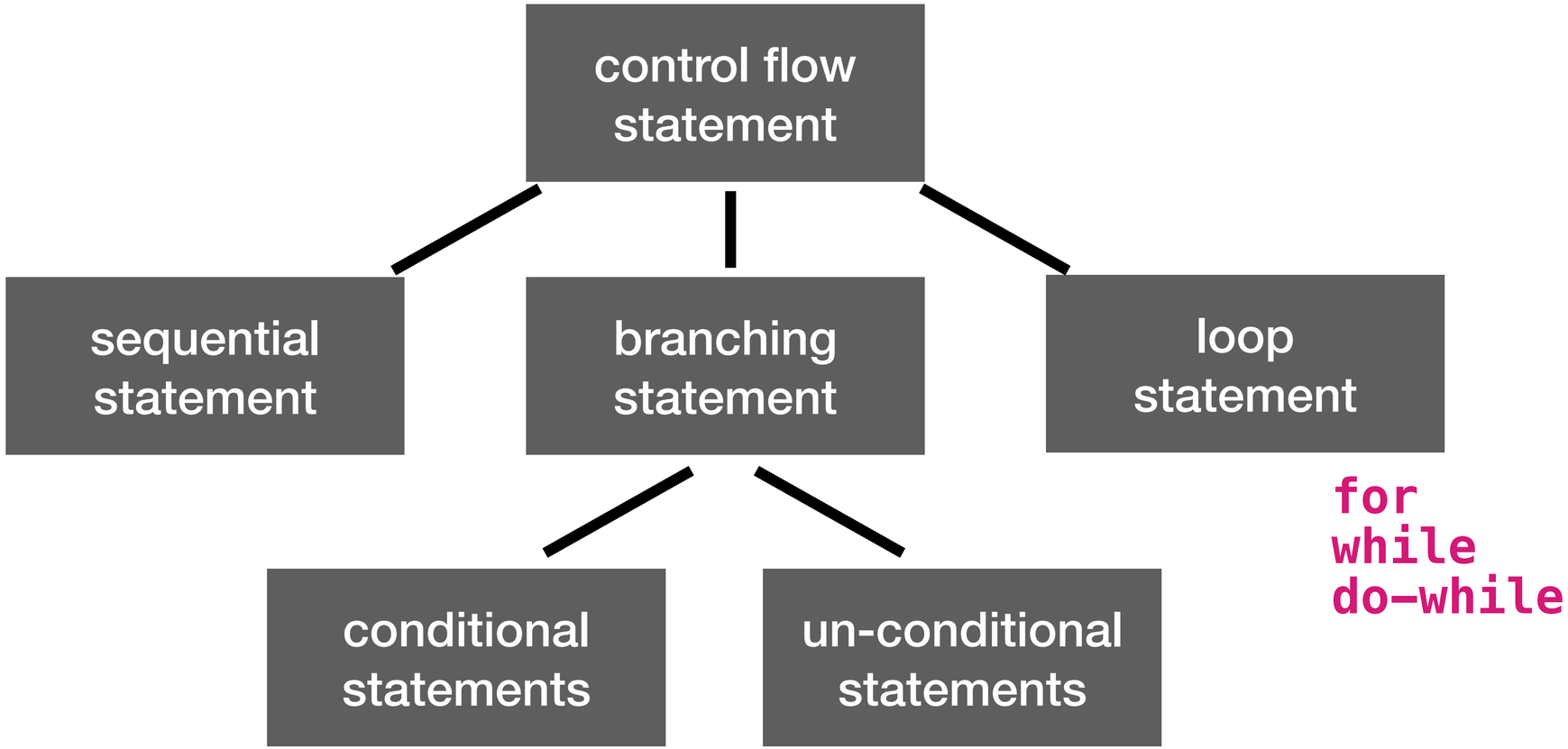
# Control flow statements



# Control flow statements



# Control flow statements



# Conditional statement: `if`

- Syntax:

```
if (boolean-expression) {

    statements

}
```

- Meaning: **only if** `boolean-expression` evaluates to **true** all the statements in the body of the `if` are executed.

- Example:

```
if (exam_mark >= 18) {
    printf("Congratulations!\n");
}
```

# Conditional statement: `if-else`

- Syntax:

```
if (boolean-expression) {
    statements-in-if-body
} else {
    statements-in-else-body
}
```

- Meaning: if `boolean-expression` evaluates to **true**, then all the statements in the body of the `if` are executed; **otherwise**, all the statements in the body of the `else` are executed.

- Example:

```
if (exam_mark >= 18) {
    printf("Congratulations!\n");
} else {
    printf("Damn. Try again.\n");
}
```

# Conditional statement: `if-else` ladder

- Syntax:

```
if (boolean-expression-1) {
    statements-1
} else if (boolean-expression-2) {
    statements-2
} else if (boolean-expression-3) {
    statements-3
}

(...)

else {
    statements-in-else-body
}
```

- **Meaning**: if `boolean-expression-1` evaluates to **true**, then all `statements-1` are executed; otherwise, `boolean-expression-2` is evaluated, etc.
  If no condition is satisfied, then the statements in the body of the `else` branch are executed.

- **Note**: the `else` branch may be missing.

# Nested `if`s

- Multiple if decision statements can be nested: among the statements in the body of an `if` there can be **another `if`** statement.

- Example: determine the largest integer among three integers.

# Nested `if`s

- Multiple if decision statements can be nested: among the statements in the body of an `if` there can be **another `if` statement.**
- Example: determine the largest integer among three integers.

```c
#include <stdio.h>

int main() {
    int num1, num2, num3;
    printf("Enter three numbers: ");
    scanf("%d%d%d", &num1, &num2, &num3);

    if (num1 > num2) {
        if (num1 > num3) {
            /* If num1 > num2 and num1 > num3 */
            printf("Num1=%d is max.\n", num1);
        } else {
            /* If num1 > num2 but num1 <= num3 */
            printf("Num3=%d is max.\n", num3);
        }
    } else {
        if (num2 > num3) {
            /* If num1 <= num2 and num2 > num3 */
            printf("Num2=%d is max.\n", num2);
        } else {
            /* If num1 <= num2 and num2 <= num3 */
            printf("Num3=%d is max.\n", num3);
        }
    }
    return 0;
}
```

# Boolean expressions and `ifs`

- From Part 2, we learnt that a **boolean expression** is an expression that returns either **true** or **false**.
- Example: let `a = 10` and `b = 5`. Then `(a > 5)`, `(a == b)`, and `(b != 3)` are boolean expressions returning, respectively, `true`, `false`, and `true`.
- **Very important:**
  - **In C, the value 0 means false and any value different than 0 means true.**
  - **In C, the value of a boolean expression is converted to the integer 0 if it is false or to the integer 1 if it is true.**
- So

  `if (a) { ... }` means `if (a != 0) { ... }`

# Beware of the bugs...

- **Important note**: do not confuse == (relational equality) with = (assignment) !
- **In C, every expression returns a value. Also assignments.**
- Hence, `a = 5` assigns 5 to a and the whole expression returns 5.
- `a == 5` determines if the value of a is equal to 5, hence it returns either `0` or 1.
- in C, you can write

  `if (a = 5) { ... }` and `if (a == 5) { ... }`

# Conditional statement: `switch`

- Suppose you have an expression that evaluates to a **limited number of possible values**.
- In the simplest form: think of a variable `var` that only takes a limited number of values, say, `val1`,...,`valn`.
- Examples: days in a week (only seven possibilities), number of university degree programs you are enrolled to (either 0 or 1), number of exam calls you have in a year per exam (four calls), etc.

# Conditional statement: `switch`

- Suppose you have an expression that evaluates to a **limited number of possible values**.
- In the simplest form: think of a variable `var` that only takes a limited number of values, say, `val1`,...,`valn`.
- Examples: days in a week (only seven possibilities), number of university degree programs you are enrolled to (either 0 or 1), number of exam calls you have in a year per exam (four calls), etc.
- Of course you can inspect which value `var` is taking with a sequence of `if-else`:

```
if (var == val1) { ... }
else if (var == val2) { ... }
else if (var == val3) { ... }
(...)
```

# Conditional statement: `switch`

- Suppose you have an expression that evaluates to a **limited number of possible values**.
- In the simplest form: think of a variable `var` that only takes a limited number of values, say, `val1`,...,`valn`.
- Examples: days in a week (only seven possibilities), number of university degree programs you are enrolled to (either 0 or 1), number of exam calls you have in a year per exam (four calls), etc.
- Of course you can inspect which value `var` is taking with a sequence of `if-else`:

```
if (var == val1) { ... }
else if (var == val2) { ... }
else if (var == val3) { ... }
(...)
```

- Or you can use a **`switch`** statement.

# Conditional statement: `switch`

```
switch (var) {

    case val1: {
       statements
       break;
    }

    case val2: {
       statements
       break;
    }

    (...)

    case valn: {
       statements
       break;
    }

    default: {
       statements
    }
}
```

- **`break`** is used to jump to the end of the switch statement, without executing all the other statements.

# Conditional statement: `switch`

```
switch (var) {

    case val1: {
       statements
       break;
    }

    case val2: {
       statements
       break;
    }

    (...)

    case valn: {
       statements
       break;
    }

    default: {
       statements
    }
}
```

- **`break`** is used to jump to the end of the switch statement, without executing all the other statements.
- Example: given a letter grade ('A', 'B', 'C', 'D', or 'E'), print the corresponding numeric grade, assuming that
  - 'A' corresponds to a number $\geq 90$;
  - 'B' corresponds to a number $\geq 80$;
  - 'C' corresponds to a number $\geq 70$;
  - 'D' corresponds to a number $\geq 60$;
  - 'E' corresponds to a number $< 60$.
- Let's do it.

# Exercises - `if-and-else`

1. Write a C program to find maximum between two numbers.
2. Write a C program to find maximum between three numbers.
3. Write a C program to check whether a number is negative, positive, or zero.
4. Write a C program to check whether a number is divisible by 5 and 11, or not.
5. Write a C program to check whether a number is even or odd.

# Exercises – if-and-else

6. Write a C program to check whether a character is alphabetical or not.

7. Write a C program to input any alphabetical character and check whether it is vowel or consonant.

8. Write a C program to input any character and check whether it is alphabetical, a digit, or something else.

9. Write a C program to check whether a character is uppercase or lowercase alphabetical.

10. Write a C program to input a number in between 1 and 7 and print the corresponding day of the week. (1 corresponds to Monday, 2 to Tuesday, etc.)

# Exercises - if-and-else

11. Write a C program to input a month number and print the number of days in that month.

12. Write a C program to count the total number of banknotes to form a given sum of money. (Assume banknotes are 500, 100, 50, 10, 5, 1.)

13. Write a C program to input angles of a triangle and check whether the triangle is valid or not.

14. Write a C program to input the lengths of the three sides of a triangle and check whether the triangle is valid or not.

15. Write a C program to check whether a triangle is equilateral, isosceles, or scalene, given its sides.

# Exercises – if-and-else

16. Write a C program to find all roots of a quadratic equation.

17. Write a C program to input basic salary of an employee and calculate its gross salary according to the following rules:
    Basic Salary $\leq$ 10,000 : HRA = 20%, DA = 80%
    Basic Salary $\leq$ 20,000 : HRA = 25%, DA = 90%
    Basic Salary $>$ 20,000 : HRA = 30%, DA = 95%
    Gross Salary is given by the sum between basic salary and HRA and DA.

18. Write a C program to input electricity unit charges and calculate total electricity bill according to the given condition:
    For first 50 units, charge: 0.50/unit
    For next 100 units, charge: 0.75/unit
    For next 100 units, charge: 1.20/unit
    For unit above 250, charge: 1.50/unit
    An additional surcharge of 20% is added to the bill.

# Loop statement: `for`

- A `for` loop is used to repeat a block of statements **until a condition becomes false** (equivalently: as long as the condition is true).
- Syntax:

```
for (variable-declaration-and-initialization;
     condition;
     variable-update)
{
    statements
}
```

- Example:

```
for (int i = 0; i != 10; ++i) printf("i is %d\n", i);
```

# Loop statement: `while`

- Also a **`while`** loop is used to repeat a block of statements **until a condition becomes false** (equivalently: as long as or while the condition is true).

- It has a simpler syntax compared to the for loop:

```
while (condition) {
    statements
}
```

- Same example as before but written with a `while` loop:

```
int i = 0;
while (i != 10) {
   printf("i is %d\n", i);
   ++i;
}
```

# Infinite loops!

- A good programmer takes care of infinite loops and guarantees that **every loop terminates**.
- If a loop does not terminate, also your program does not terminate (and this is usually something you do not want).
- Infinite loops are, for example:

```
while (1) { ... }

for (;;) { ... }
```

# Infinite loops!

- A good programmer takes care of infinite loops and guarantees that **every loop terminates**.
- If a loop does not terminate, also your program does not terminate (and this is usually something you do not want).
- Infinite loops are, for example:

```
while (1) { ... }

for (;;) { ... }
```

- Common mistakes:

```
int i = 0;
while (i != 10) {
   printf("i is %d\n", i);
}
```

# Infinite loops!

- A good programmer takes care of infinite loops and guarantees that **every loop terminates**.
- If a loop does not terminate, also your program does not terminate (and this is usually something you do not want).
- Infinite loops are, for example:

```
while (1) { ... }

for (;;) { ... }
```

- Common mistakes:

```
int i = 0;
while (i != 10) {
   printf("i is %d\n", i);
}
```

```
int i = -1;
while (i) {
   printf("i is %d\n", i);
   --i;
}
```

# Infinite loops!

- A good programmer takes care of infinite loops and guarantees that **every loop terminates**.
- If a loop does not terminate, also your program does not terminate (and this is usually something you do not want).
- Infinite loops are, for example:

```
while (1) { ... }

for (;;) { ... }
```

- Common mistakes:

```
int i = 0;
while (i != 10) {
   printf("i is %d\n", i);
}
```

```
int i = -1;
while (i) {
   printf("i is %d\n", i);
   --i;
}
```

```
for (int i = 0; i != 31; i += 2) {
   printf("i is %d\n", i);
}
```

# Un-conditional statement: `break`

- You can use **`break`** to stop the execution of a `for` or `while` loop.
- Usually, we execute **`break`** if some condition is met.
- Examples:

```
int i = -1;
while (i) {
   printf("i is %d\n", i);
   --i;
   if (i == -10) break;
}
```

# Un-conditional statement: `break`

- You can use **`break`** to stop the execution of a `for` or `while` loop.
- Usually, we execute **`break`** if some condition is met.
- Examples:

```
int i = -1;
while (i) {
   printf("i is %d\n", i);
   --i;
   if (i == -10) break;
}
```

```
for (int i = 0; i != 10; ++i)
    printf("i is %d\n", i);
```

is equivalent to

```
for (int i = 0; ; ++i) {
    if (i == 10) break;
    printf("i is %d\n", i);
}
```

# Un-conditional statement: `continue`

- The jump statement **`continue`**, instead, skips the remaining statements in the body of the loop and the control flow is passed to the next iteration.

- Example:

```
for (int i = 0; i != 10; ++i) {
    if (i % 2 == 0) continue;
    printf("i is %d\n", i);
}
```

**Q.** What does the above code print?

# Un-conditional statement: `continue`

- The jump statement **`continue`**, instead, skips the remaining statements in the body of the loop and the control flow is passed to the next iteration.

- Example:

```
for (int i = 0; i != 10; ++i) {
    if (i % 2 == 0) continue;
    printf("i is %d\n", i);
}
```

**Q.** What does the above code print?

- Equivalent to:

```
for (int i = 0; i != 10; ++i) {
    if (i % 2) printf("i is %d\n", i);
}
```

# Loop statement: `do-while`

- There is a variant of the `while` loop called `do-while` whose syntax is:

```
do {
    statements
} while (condition);
```

- Note that the condition is checked **after** the statements are executed. Hence, it is useful when we want to execute the loop body **at least once**.

# Nested loops

- As with `if-else` statements, also loops can be nested!
- For example, we can have a for loop in the body of another for loop, or a while loop in the body of a for loop, etc.
- **Common pattern**: a "double" for loop (outer and inner loop) where the number of iterations of the inner loop depends on the control variable of the outer loop.
- Example:

```
for (int i = 0; i != n; ++i) {
    for (int j = 0; j <= i; ++j) {
        printf("(i,j)=(%d,%d)\n", i, j);
    }
}
```

# Exercises - Loops

1. Write a C program to print all natural numbers from 1 to n, using a `for` and a `while` loop.
2. Write a C program to print all natural numbers in n to 1, using a `for` and a `while` loop.
3. Write a C program to print all characters from a to z, using a `for` and a `while` loop.
4. Write a C program to print all even numbers between 1 to 100, using a `while` loop.
5. Write a C program to print all odd number between 1 to 100.
6. Write a C program to find sum of all natural numbers between 1 to n.
7. Write a C program to find sum of all even numbers between 1 to n.
8. Write a C program to find sum of all odd numbers between 1 to n.
9. Write a C program to print a multiplication table of any number n. (n = 1,...,9)
10. Write a C program to count number of digits in a number.

# Exercises - Loops

11. Write a C program to find first and last digit of a number.
12. Write a C program to find sum of first and last digit of a number.
13. Write a C program to swap first and last digits of a number.
14. Write a C program to calculate sum of digits of a number.
15. Write a C program to calculate product of digits of a number.
16. Write a C program to enter a number and print its reverse.
17. Write a C program to check whether a number is palindrome or not.
18. Write a C program to enter a number and print it in words. Example: 348 --> three four eight
19. Write a C program to print all ASCII character with their values.

# Exercises - Loops

20. Write a C program to find power of a number, e.g., $x^y$, using a `for` loop.

21. Write a C program to find all factors of a number.

22. Write a C program to calculate the factorial of a number.

23. Write a C program to check whether a number is prime or not.

24. Write a C program to print all prime numbers between 1 to n.

25. Write a C program to find sum of all prime numbers between 1 to n.

26. Write a C program to find all prime factors of a number.

27. Write a C program to check whether a number is an *Armstrong* number or not.
    An Armstrong number if a n-digit number whose value is equal to the sum of the n-power of its digits. Example: $317 = 3^3 + 1^3 + 7^3$.

# Exercises - Loops

28. Write a C program to print all Armstrong numbers between 1 to n.

29. Write a C program to check whether a number is *perfect* number or not.
    A number is perfect if it is the sum of all its divisors (excluding itself, of course).
    Example: 6 = 1 + 2 + 3.

30. Write a C program to print all perfect numbers between 1 to n.

31. Write a C program to check whether a number is *strong* number or not.
    A number is strong if it is the sum of the factorials of all its digits.
    Example: 145 = 1! + 4! + 5!.

32. Write a C program to print all strong numbers between 1 to n.

33. Write a C program to print the *Fibonacci* series up to n terms.