

Programming 1

Lesson 4. Modular programming

Degree in Computer Engineering

Index

2

1. Concept of module
2. Types of modules
3. Functions in C
4. Procedures in C
5. Declaration and definition of modules in C
6. Parameters of a module
7. Scope of a variable
8. C language libraries
9. Exercises

1. Concept of module

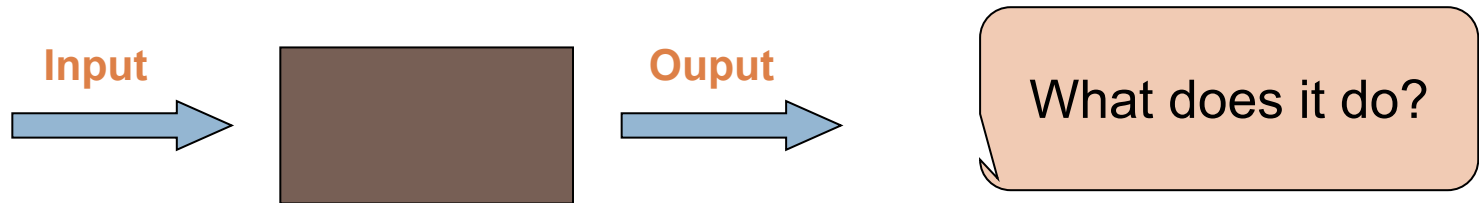
3

- When a program is large and complex, it is not convenient that all the code is inside the main program (`main()` function in C language).
- A module ...
 - is a piece of code that is written independently of the rest of the program.
 - performs a specific task that solves a partial problem of the main problem.
 - can be *invoked* (called) from the main program or from other modules.
 - allows to hide the details of the solution of a partial problem (*black box*).

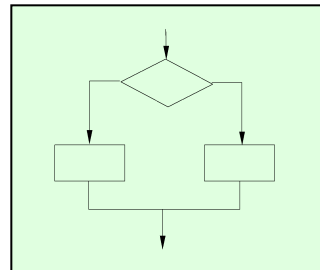
1. Concept of module

4

- Each module is a **black box** for the main program and for the other modules.
- To use a module from the main program or from other modules ...
 - We need to know what it does and its **interface**, i.e. its inputs and outputs.



- We do not need to know the **internal details of operation**



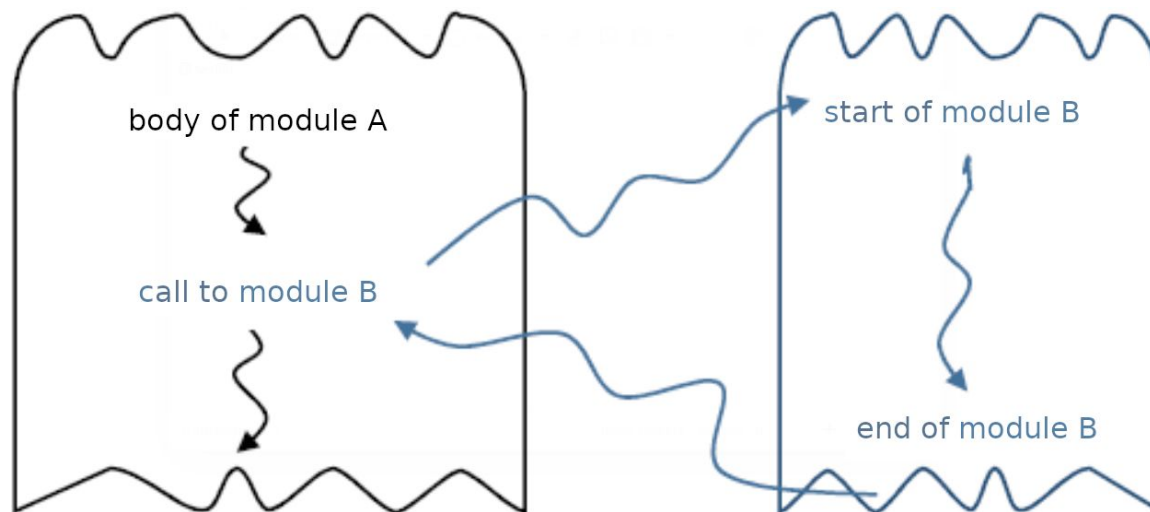
How does it do it?

1. Concept of module

5

Transfer of control flow

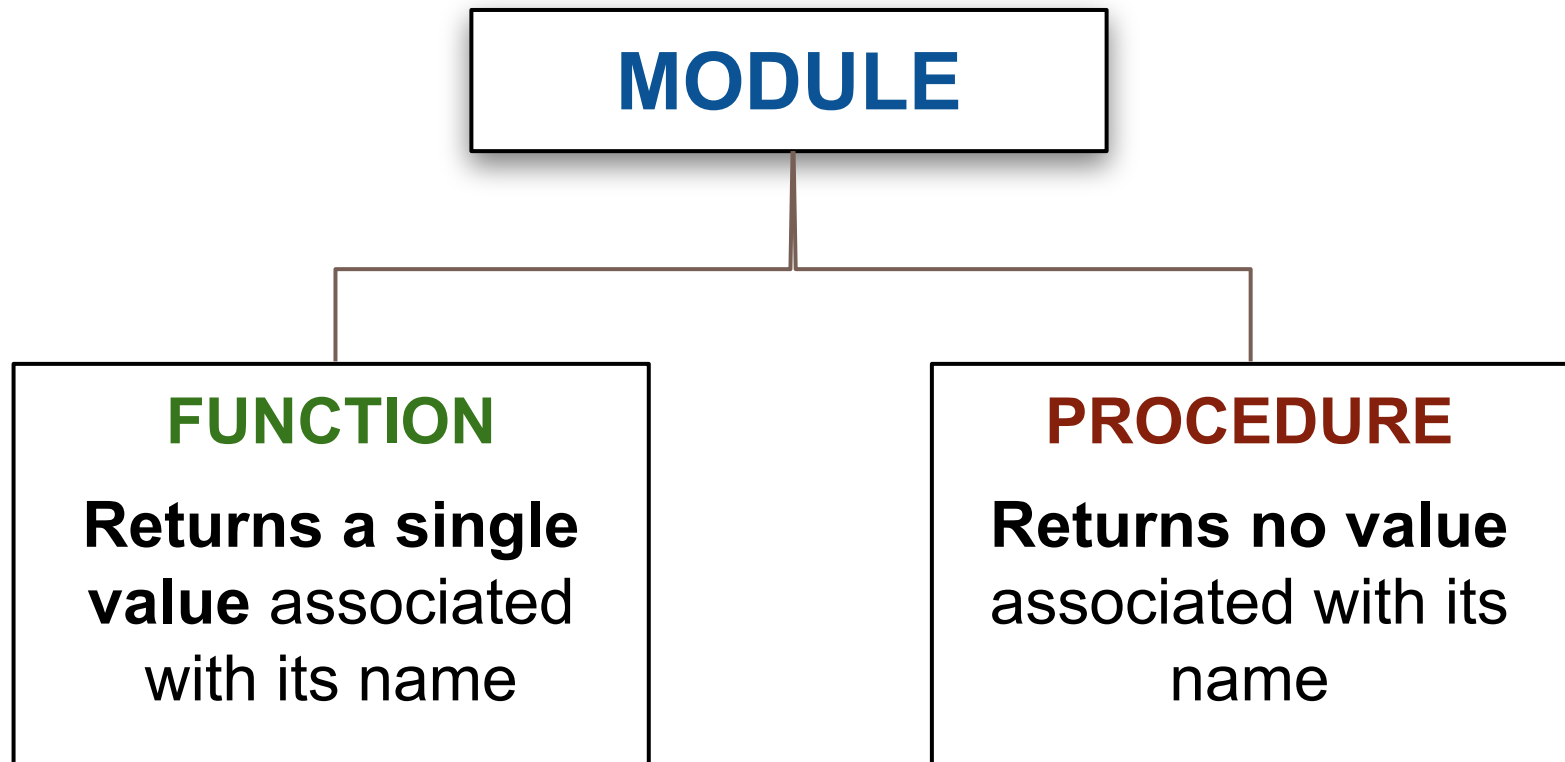
- When a module A calls (invokes) another module B, the control flow (execution flow) passes to module B.
- When module B has finished executing, the flow of control continues in module A, starting from the sentence following the call to module B.



2. Types of modules

6

A module can be of two types **depending on whether it returns a value or not:**



3. Functions in C

7

- A **function** is a module that **returns a single result** associated with its name.
- It may receive zero or more input data (parameters) and, based on that data, generates and returns a result using the **return** statement.
- **Examples**: mathematical functions

| Function | Input | Ouput |
|-----------------|-------|---------|
| Square root (9) | 9 | 3 |
| Power (4, 3) | 4, 3 | 64 |
| Log (4) | 4 | 0.60206 |
| Cos (π) | π | -1 |

3. Functions in C

8

Syntax

```
ret_type name( par_type1 par1, par_type2 par2, ... ) {  
    sequence of sentences  
    return return_value;  
}
```

where:

- **ret_type**: Type of data of the value to be returned by the function.
- **name**: Identifier, name of the function.
- **par_type N** : Type of data of the input parameter (value) in position N .
- **par N** : Name given to the N th parameter received so that it can be used in the function.

3. Functions in C

9

How to use functions

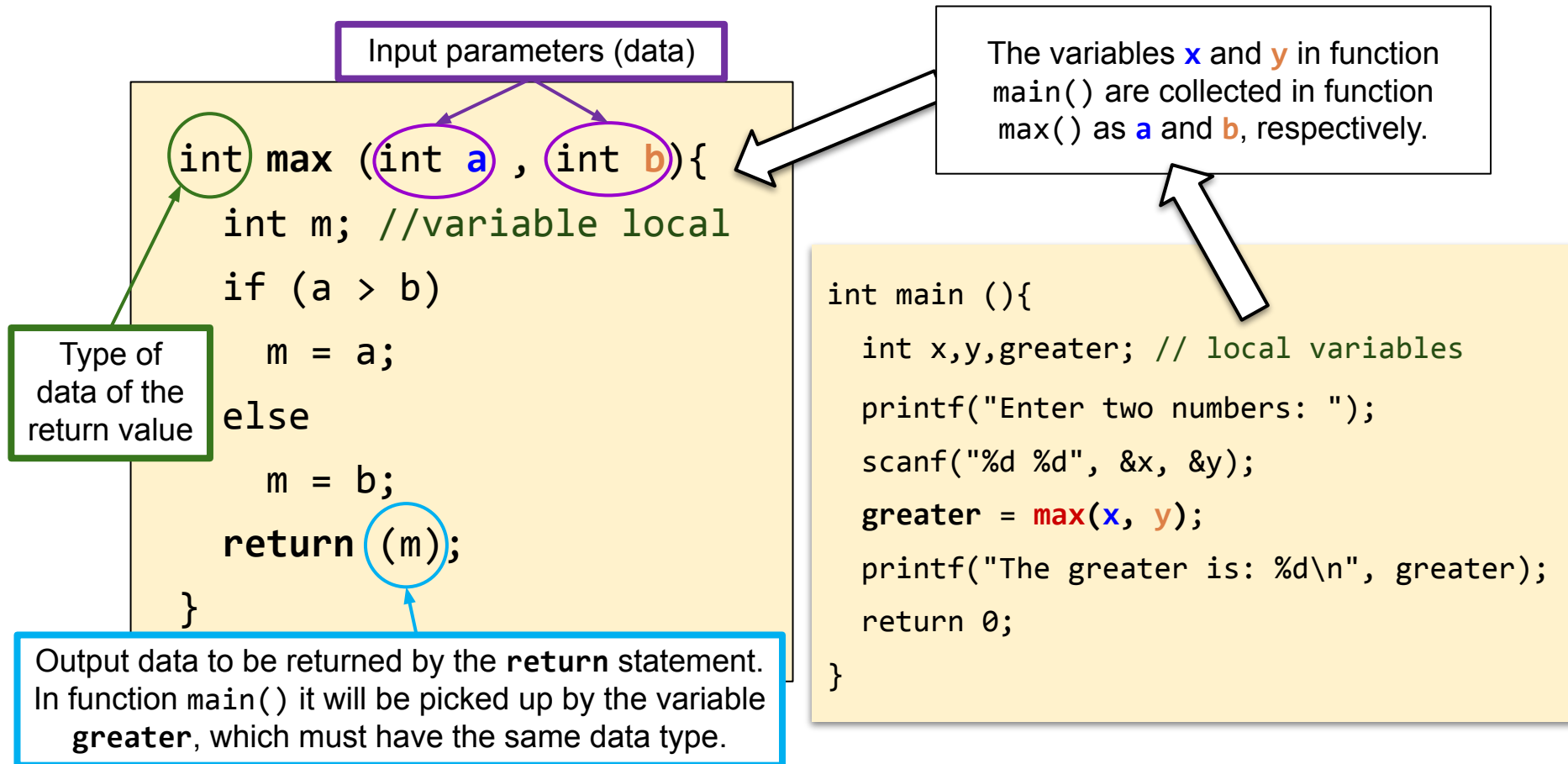
- To execute a function, **a call must be made** from the main module or from another module.
- To call a function, **its name is used** followed by the input parameters in parentheses, sorted and separated by commas.
- If the function has no input parameters, empty parentheses are used.
- When a function is called, a value is obtained as a result. **It is very important that the call is included in an expression that leverages the result.**

3. Functions in C

10

Example:

C function that receives two integer numbers and returns the larger of the two numbers.



3. Functions in C

11

return statement

- **Ends** the execution of the body of the function.
- It is responsible for **returning the return value** of the function, after evaluating its associated expression.
- It is recommended to **use only one return statement** within the body of a function.
- It should be **the last sentence** in the function body.

```
return (expression);
```

4. Procedures in C

12

- A **procedure** is a module that performs a specific task.
- It can receive zero or more input values (parameters) to perform that task.
- It can return zero or more values through the same parameters it receives, **NEVER** through return.
- The difference with functions is precisely that procedures do not use return, so their return data type is **void**.

4. Procedures in C

13

Syntax

```
void name( par_type1 par1, par_type2 par2, ... ) {  
    sequence of sentences  
}
```

where:

- **void**: Empty data type, as it does not return any data associated with its name.
- **name**: Identifier, name of the function.
- **par_type N** : Type of data of the input parameter (value) in position N .
- **par N** : Name given to the N th parameter received so that it can be used in the function.

4. Procedures in C

14

How to use procedures

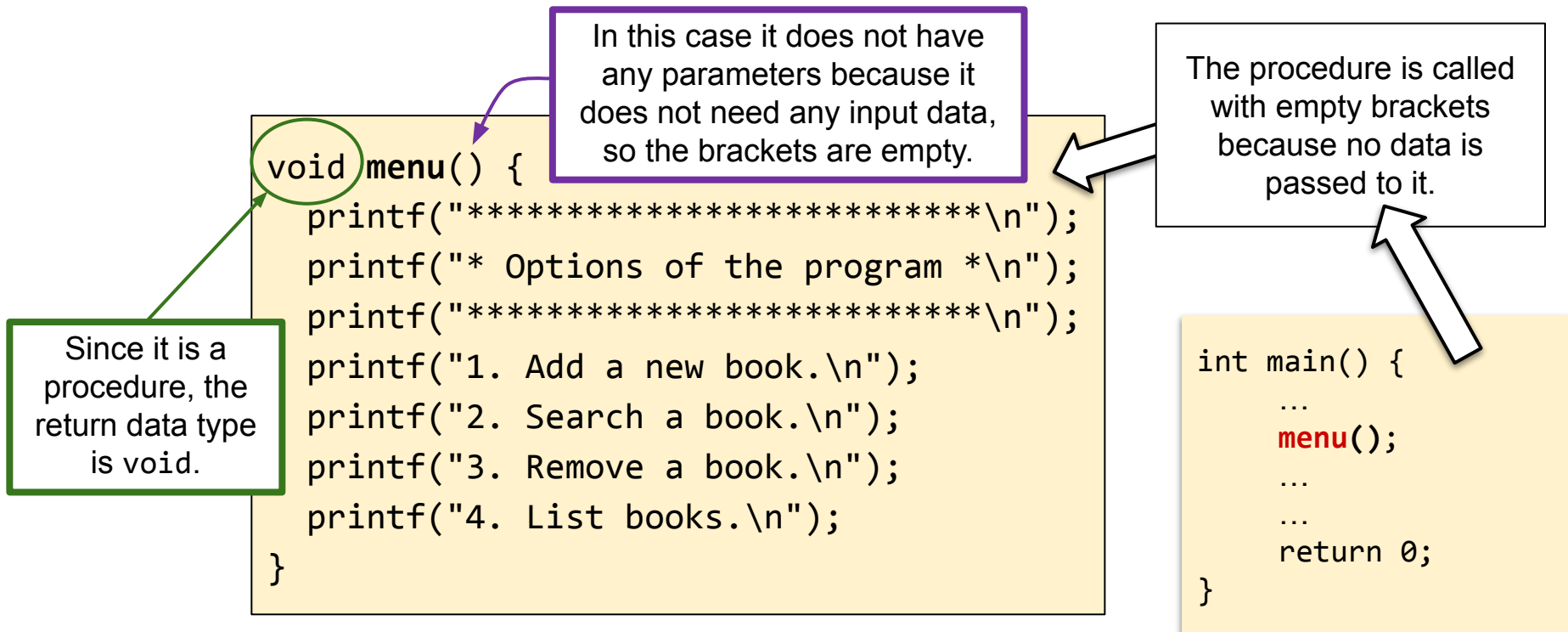
- To execute a procedure, **a call must be made** from the main module or from another module.
- To call a procedure, **its name is used** followed by the input parameters in parentheses, sorted and separated by commas.
- If the procedure has no parameters, empty parentheses are used.
- The call to a procedure is not included in any expression, it will always be a single code sentence.

4. Procedures in C

15

Example:

C procedure that does not receive any parameters and displays all the menu options of an program on the screen.



As a procedure, it does not have any return statement.

5. Declaration and definition of modules in C

16

- **The declaration of a module** consists in associating the module name with the return data type and the data types of the input parameters. This is called the **module prototype**. They are written before the `main()` module.
- **The definition of a module** is the actual implementation of the module. That is, the set of sentences that make up the module. They are written after the `main()` module.
- Although it is not mandatory to declare the modules before defining them, in which case the definition would go before the `main()` module, it is very convenient since it avoids possible errors in the compilation/linking phase of the program.

5. Declaration and definition of modules in C

17

Example of location for the declaration and definition of modules

Prototypes



#preprocessor directives

Declaration of constants

Declaration of procedures and functions

main() {

 Declaration of variables (of simple types)

 Main body

 control sentences

 calls to procedures and functions

}

Definition of procedures and functions

```
#include<stdio.h>
#include<stdbool.h>

// Declaration of modules (Prototypes)
bool isEven(int number);

// Main module
int main(){
    ...

    if( isEven(n) )
        ...

    return 0;
}

// Definition of modules
bool isEven(int number){
    bool retVal;

    if( number % 2 == 0 )
        retVal = true;
    else
        retVal = false;

    return retVal;
}
```

6. Parameters of a module

18

- Parameters are used for the transfer of information between modules. This process is called **parameter or argument passing**.
- A parameter **allows to pass information from one module to another** and vice versa.
- A parameter in a module can be considered as a variable whose value must be provided by another module or returned from the module itself.
- **Three types of parameters** can be identified:
 - Input: their values are provided by the calling module.
 - Output: their values are calculated in the subprogram (module) and returned to the calling module.
 - Input/output: they provide values to the module and these values can also be modified and returned to the calling module.

6. Parameters of a module

19

Formal and current parameters

- **Formal parameters** are those that appear in the module declaration.
- **Actual parameters** are those that appear in the module call.
- The correspondence between formal and actual parameters is as follows:

| Must match | No need to match |
|--|--|
| <ul style="list-style-type: none">• Number of parameters• Data type of parameters• Order of parameters | <ul style="list-style-type: none">• Name of the parameters |

6. Parameters of a module

20

Parameter passing

- Parameter passing **by value**. A copy of the original value of the actual parameter is passed. In this case, if the value of the parameter is changed in the module, the original value of the actual parameter is not changed at the point in the program where the module call was made.
- Parameter passing **by reference**. The original value of the actual parameter is passed. In other words, a reference is passed to the memory address where this value is stored. In this case, if the value of the parameter is modified in the module, the original value of the actual parameter is also modified at the point in the program where the call to the module was made. To do this, & is written in front of the actual parameter in the call and * in front of the formal parameter in the module declaration.

6. Parameters of a module

21

Parameter passing by value

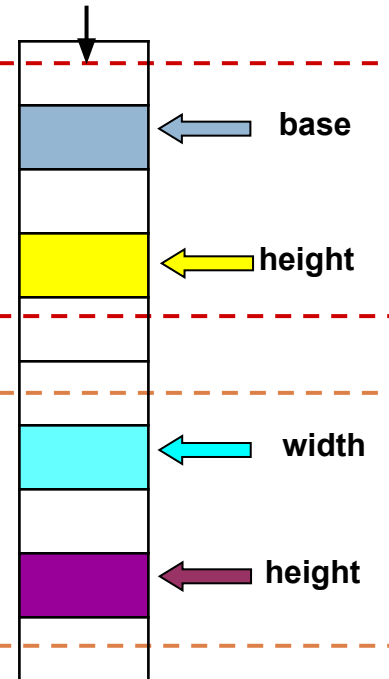
```
int main() {  
    int base, height, area, perimeter;  
  
    printf("Enter the base of the rectangle: ");  
    scanf("%d", &base);  
    printf("Enter its height: ");  
    scanf("%d", &height);  
  
    rectangle(base, height, &area, &perimeter);  
  
    printf("Area: %d\n", area);  
    printf("Perimeter: %d\n", perimeter);  
  
    return 0;  
}
```

Original **base** and **altura** parameters

Passing of **base** and **altura** parameters **by value** in the module call

Copy of the original **base** and **height** parameters received in the module as **width** and **height**

Memory



```
void rectangulo(int width, int height, int *rect_area, int *perim) {  
    rect_area = width * height;  
    perim = 2 * (width + height);  
}
```

Reception **by value** of the **base** and **height** parameters at the point of call

6. Parameters of a module

22

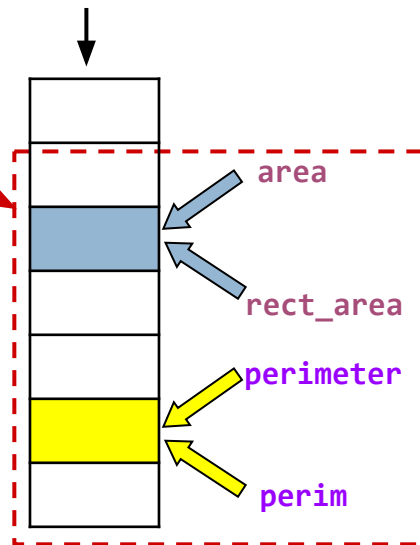
Parameter passing by reference

```
int main() {  
    int base, height, area, perimeter;  
  
    printf("Enter the base of the rectangle: ");  
    scanf("%d", &base);  
    printf("Enter its height: ");  
    scanf("%d", &height);  
  
    rectangle(base, height, &area, &perimeter);  
  
    printf("Area: %d\n", area);  
    printf("Perimeter: %d\n", perimeter);  
  
    return 0;  
}
```

Original **area** and **perimeter** parameters, which are the same as **rect_area** and **perim** in the module

Passing the parameters **area** and **perimeter** by **reference** in the module call. The operator **&** is needed in front of each parameter to be passed by reference

Memory



```
void rectangle( int width, int height, int *rect_area, int *perim ) {  
    *rect_area = width * height;  
    *perim = 2 * (width + height);  
}
```

When receiving parameters **by reference**, the operator ***** is used in front of them whenever they are used in the module

Receiving as parameters **by reference** the variables **area** and **perimeter** at the point of call. The operator ***** is used in front of formal parameters in the declaration

7. Scope of a variable

23

The scope of a variable defines the visibility of the variable, that is to say, **the part of the program where the variable is accessible.**


```
#include<stdio.h>
#include<stdbool.h>

bool is_prime(int);

int main() {
    int n; // number entered by
           // keyboard (input data)

    printf("Enter an integer number: ");
    scanf("%d", &n);
    if ( is_prime(n) )
        printf("That number is prime\n");
    else
        printf("That number is not prime\n");


    return 0;
}
```



scope of **n**

```
// This module checks whether a number
// is prime or not
bool is_prime(int num) {
    int cont; // counter (auxiliary data)
    bool prime; // is prime or not
                // (output data)

    prime = true;
    cont = 2;
    while ( (cont < num) && prime) {
        // check if it is divisible by
        // another number
        prime = ! (num % cont == 0);
        cont = cont + 1;
    }
    return prime;
}
```



scope of
num, **cont**, **prime**

7. Scope of a variable

24

Local variables and global variables

■ **Local** variable

- Its scope is the body of the module in which it is declared.
- It is created when it is declared and destroyed when the execution of the module ends.

■ **Global** variable

- Its scope is the whole program (all its modules).
- It is created when it is declared and destroyed when the execution of the program ends.

7. Scope of a variable

25

Communication between modules

- It must be done through parameters and **NOT by using global variables**.
- The use of global variables is not advisable because:
 - It makes readability of code difficult.
 - It can produce side effects when changing the value of a global variable in a module.
 - It is contrary to one of the most important concepts in programming: **modularity**.

Note: In Programming 1, the use of global variables is not allowed.

7. Scope of a variable

26

Advantages of modular programming

- It makes top-down design and structured programming easier
- Reduces programming time
 - *Reusability*: structuring in specific *libraries* (module library)
 - *Division* of the programming task among a team of programmers
- Decreases the overall size of the program
 - A *module* is only written once and can be used several times from different parts of the program.
- It makes it easier to detect and correct errors
 - By individual checking of the modules
- It makes program maintenance easier
 - Programs are easier to modify
 - Programs are easier to understand (more readable)

8. C language libraries

27

- Most programming languages provide a collection of commonly used procedures and functions (**libraries**).
- In the C language, the compiler directive **#include** is used to make use of the modules included in a library.
- A wide variety of libraries are available:
 - Mathematical functions
 - Character and string handling
 - Data input and output handling
 - Time handling (date, time, ...)
 - etc.

8. C language libraries

28

Examples of C language libraries

| Library | Function | Description |
|------------|--------------------------------|---|
| <math.h> | double cos(double x) | Returns the cosine of x |
| | double sin(double x) | Returns the sine of x |
| | double exp(double x) | Returns e^x |
| | double fabs(double x) | Returns the absolute value of x |
| | double pow(double x, double y) | Returns x^y |
| | double round(double x) | Returns the rounded value of x |
| | double sqrt(double x) | Returns the square root of x |
| <ctype.h> | int isalnum(int c) | Returns true if the parameter is a letter or a digit |
| | int isdigit(int c) | Returns true if parameter is a digit |
| | int toupper(int c) | Returns the character in uppercase |
| <stdlib.h> | void srand(unsigned int s) | Initializes to the value passed as an argument the seed of the random number sequence to be generated |
| | int rand(void) | Returns a random number between 0 y RAND_MAX |

9. Exercises

29

1. Make a function that returns the letter corresponding to a DNI number passed as a parameter using the following algorithm:
 - a) Calculate the remainder of the division of the DNI by 23.
 - b) Depending on the value of the remainder, associate the corresponding letter according to the following table:

| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| T | R | W | A | G | M | Y | F | P | D | X | B | N | J | Z | S | Q | V | H | L | C | K | E |

2. Design a module that receives as parameter a number n and draws on screen a square of size n formed by asterisks.
3. Improve exercise 2 by adding another parameter that allows the square to be drawn with the character sent as a parameter.
4. Design a module that receives two variables as parameters and exchanges their values.
5. Design a module that asks the user for a number greater than 0 and less than 100 and, if the number is correct, it returns the sum and count of the odd numbers between 1 and that value.

9. Exercises (//)

30

6. Module that checks if a number received as an argument is negative, returning `true` if it is or `false` if it is not.
7. Module that receives an integer number and returns how many divisors it has.
8. Module that divides two integers passed as arguments and returns the quotient and the remainder. It must check that the divisor is not 0.
9. Module that receives a gross amount and the percentage of tax added and must return the net amount and the part corresponding to the tax.
10. Write a program that asks the user for an integer number and prints on the screen whether it is prime or not. In addition, the program also prints the next prime number greater than the number entered. Use a module to find out if a number is prime; another one to calculate the next prime number to a given number; and another module (the `main()` module) that makes use of the previous two.