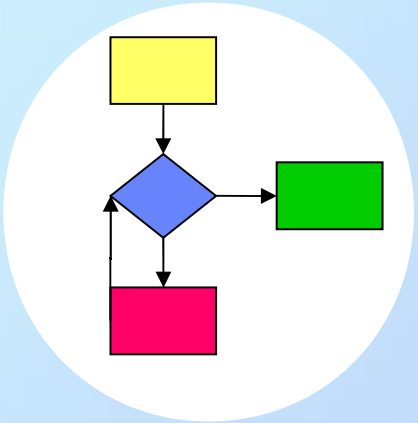
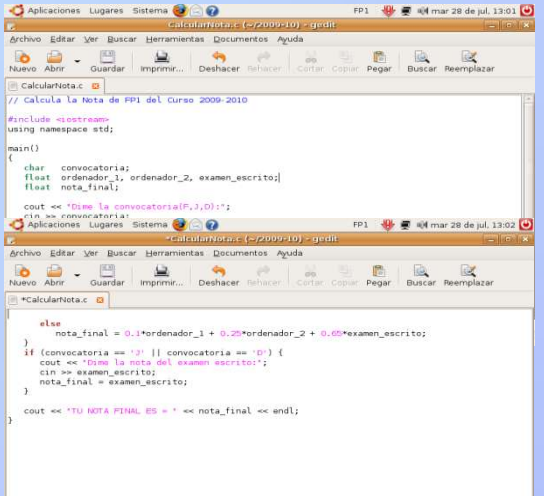


Programming 1

Lecture 4
Modular Programming



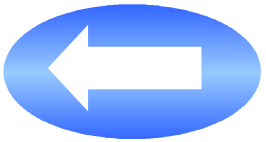
Dpnt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial

Objectives

- Use a top-down design to solve relatively complex problems
- Understand the differences between procedures and functions
- Modularize programs in C language

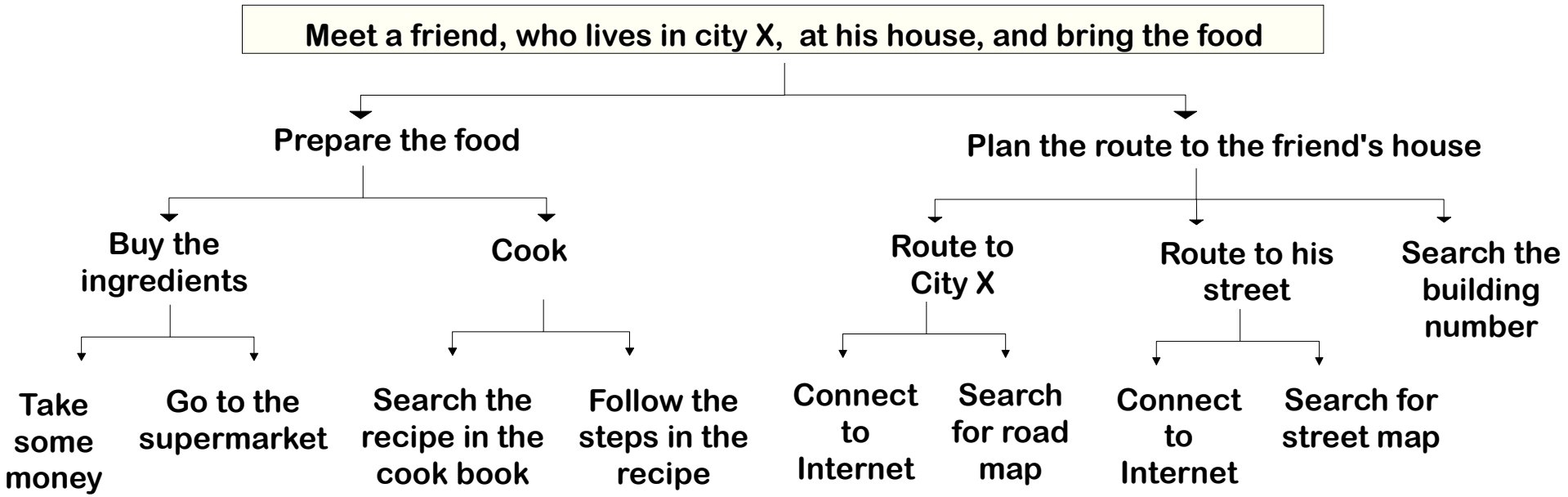
Topics

1. **Modular decomposition**
2. Communication between modules
3. Procedures and Functions
4. Scope of a variable
5. General structure of a program
6. Predefined functions in C language
7. Information sources



Top-down design

- **Decomposition** of a problem into some other smaller problems (subproblems)
  - The problem decomposition is done in a set of levels or consecutive **refinement** steps, so that a **hierarchical structure** is obtained
  - Each level in the hierarchy includes a **higher level of detail**



Concept of Module

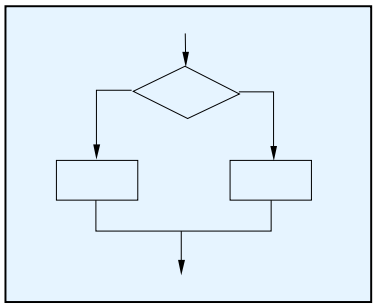
- In a large and complex program, all the code should not be included in the main program (function main() in C language)
- A module or **subprogram**...
  - is a block of code that is written separate to the main program
  - is responsible for performing a specific task that solves a partial problem of a major problem
  - can be invoked (called) from the main program or from other modules
  - hides the details of the solution of a partial problem (**Black Box**)

Black Box

- Each module is a black box for the main program or for other modules
- To use a module from the main program or from other modules …
  - **we need** its **interface**, that is, its inputs and outputs

What does it do?

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



How does it do it?

Modules: example

```
main() {
   int    n1, n2;    // numbers entered by keyboard  (input data)
   int    greater;   // the greater of the entered numbers (output data)
   int    lesser;    // the lesser of the entered number (output data)

   cout << "Enter two integer numbers:  ";
   cin >> n1 >> n2;
   greater = maximum(n1, n2);
   lesser = minimum(n1, n2);
   cout << "The greater number is:" << greater;
   cout << "The lesser number is:" << lesser;
   cout << endl;
}
```

What does the module "maximum()" do?

How does it do it?

```
// This module returns the lesser of two numbers
int minimum(int a, int b)
{
   int  m;  // the lesser of two numbers (output datum)

   m = a;
   if (b < m)
      m = b;
   return(m);
}
```

```
// This module returns the greater of two numbers
int maximum(int a, int b)
{
   int  m;   // the grater of two numbers (output datum)

   if (a > b)
      m = a;
   else
      m = b;
   return(m);
}
```

Modules: declaration, definition and call

Module declaration

Module_name (parameter_declaration)

```
int maximum(int a, int b);
```

Module definition

Module_name (parameter_declaration)

Local_variable_declaration

Body_of_module: executable statements

Fin_del_módulo

```
int maximum(int a, int b)
{
   int  m;

   if (a > b)
      m = a;
   else
      m = b;
   return(m);
}
```

Module call

Module_name (parameter_list)

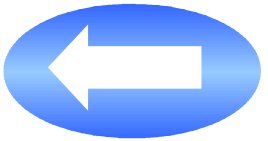
```
greater =  maximum (n1, n2);
```

Advantages of modular programming

- Easier top-down design and structured programming
- Reduction of programming time
  - Reusability: structure in specific libraries (modules library)
  - Division of the programming task among a team of programmers
- Smaller size of the whole program
  - A module is written only once and it can be used several times from different parts of the program
- Easier error detection and correction
  - By testing the individual modules
- Easier program maintenance
  - The programs are easier to modify
  - The programs are easier to understand (more readable)

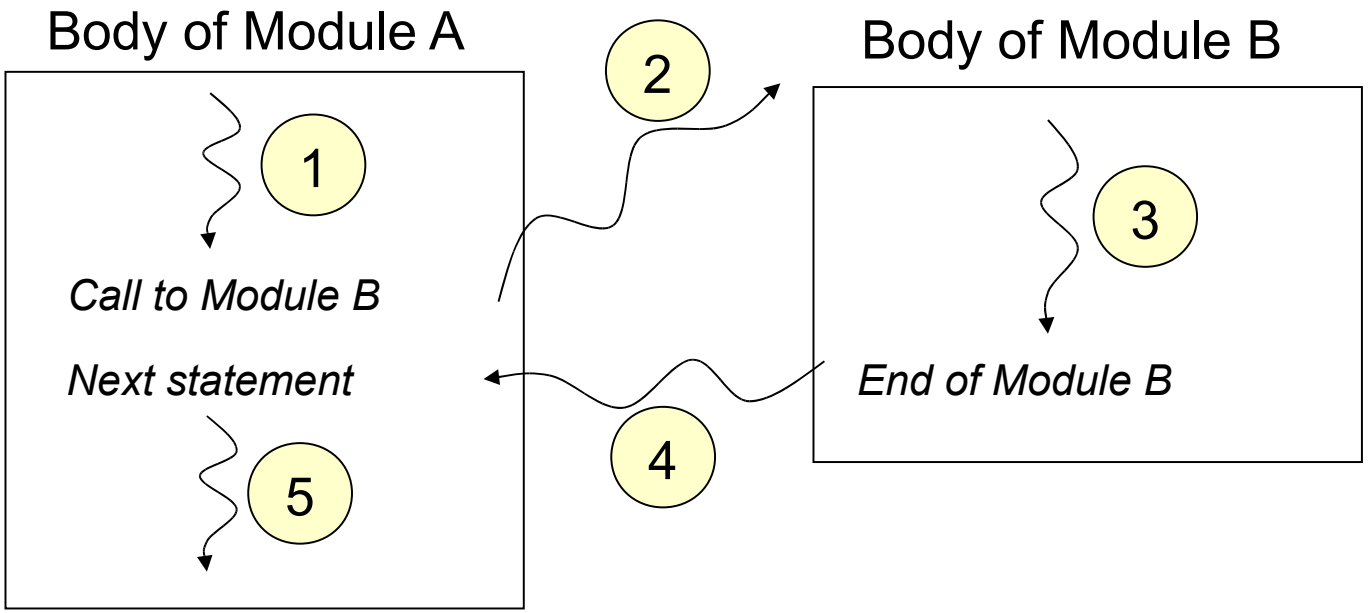
Topics

1. Modular decomposition
2. **Communication between modules**
3. Procedures and Functions
4. Scope of a variable
5. General structure of a program
6. Predefined functions in C language
7. Information sources



Transfer of the control flow

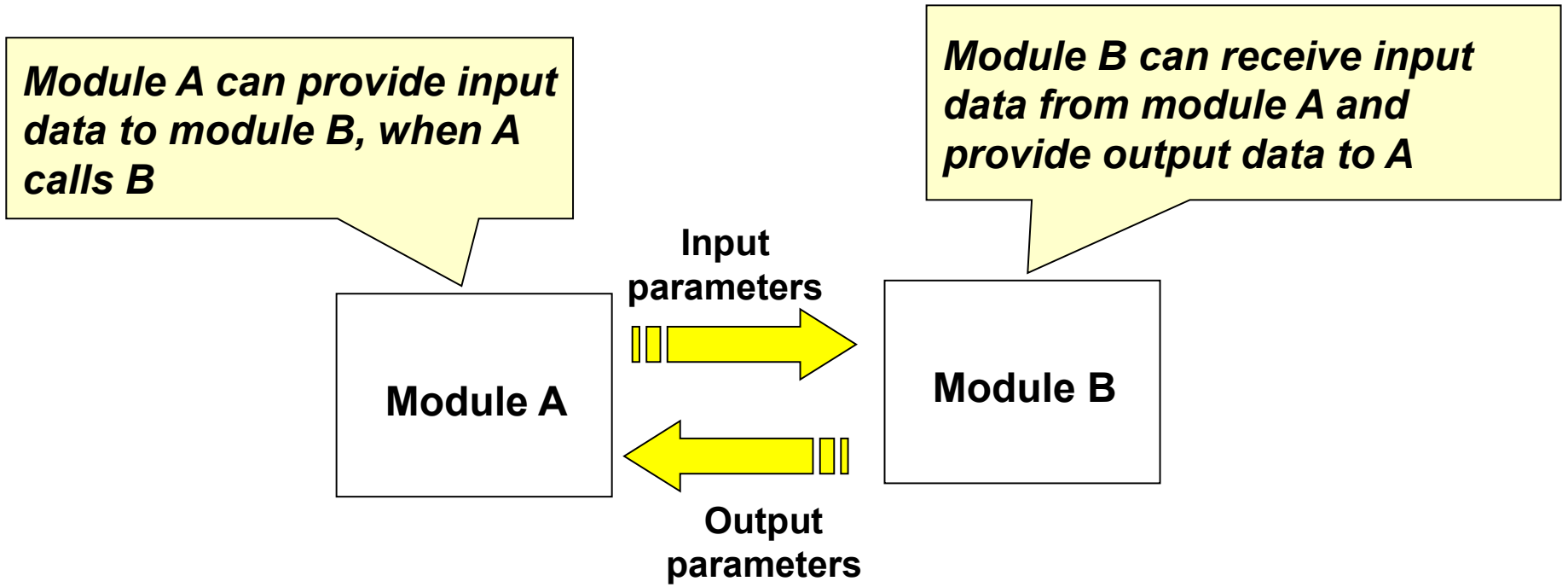
- When a module A calls (invokes) another module B, the control flow (execution flow) passes to module B
- When the execution of module B is finished, the control flow continues in module A, from the statement following the call to the module B



The main program can be considered as a module that can call other modules, but that cannot be called by any other module

Transfer of information

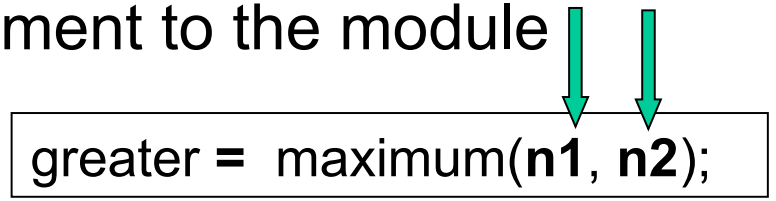
- Transferring information between modules is carried out through the use of **parameters** (**arguments**)
- A module can have input and/or output parameters



Actual and formal parameters

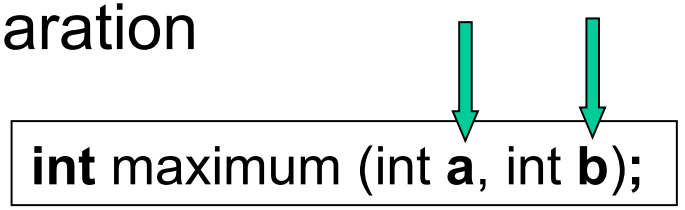
- **Actual** parameters (or **arguments**)
  - The ones appearing in the call statement to the module

Module_name (pr1, pr2, …, prN)



- **Formal** or fictitious parameters (or **parameters**)
  - The ones appearing in the module declaration

Module_name (type1 pf1, type2 pf2, …, typeN pfN)

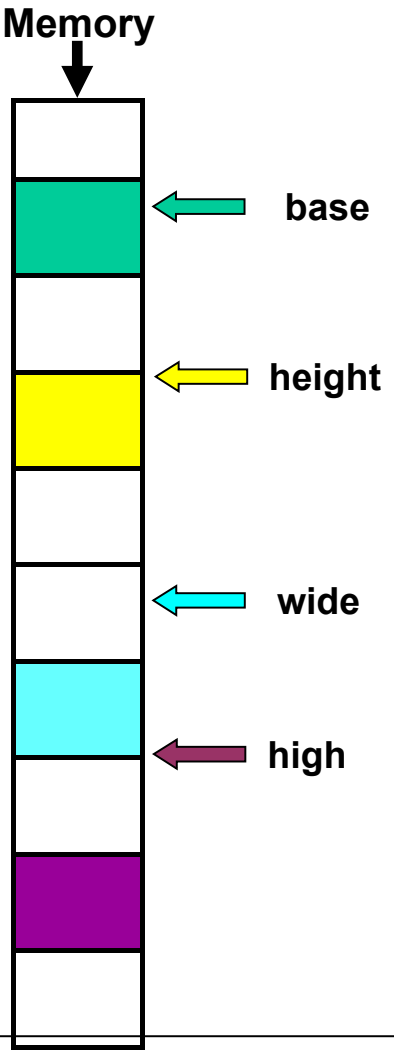


- **Relation** between actual and formal parameters

✓ number of parameters
✓ type of parameters
✓ order of parameters
✘ ~~name of parameters~~

Pass-by-value parameters

- The module receives a copy of the data value (actual parameter), passed by the calling module
- The actual parameter can be any expression that can be evaluated at the moment of the call
- If the corresponding formal parameter is modified inside the module, the actual parameter is not changed in the calling module



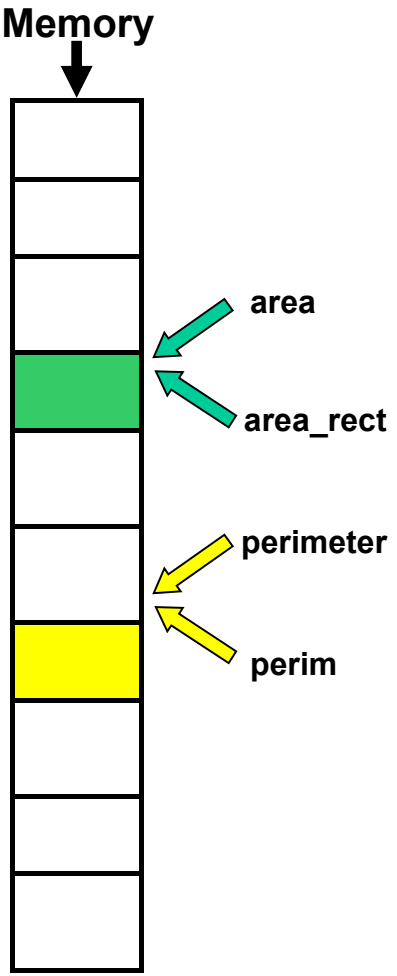

```
main() {
   int  base, height, area, perimeter;

   cout << "Enter the base of  rectangle:";
   cin >> base;
   cout << "Enter its height:";
   cin >> height;

   rectangle(base, height, area, perimeter);

   cout << "Area: " << area << endl;
   cout << "Perimeter: " << perimeter;
   cout  << endl;
}
```

Pass-by-value parameters

```
void rectangle( int wide, int high, int &area_rect, int &perim )

{
   area_rect = wide * high;
   perim = 2 * (wide + high);
}
```

Pass-by-reference parameters

- The module receives the reference to the memory position where the value is (memory address of the variable)
- The actual parameter must compulsorily be a variable (which may or may not contain a value)
- If the corresponding formal parameter is modified inside the module, the actual parameter is changed as the memory content is changed



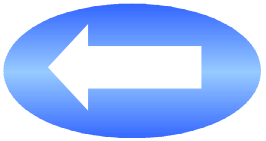

```
main() {
   int  base, height, area, perimeter;

   cout << "Enter the base of  rectangle:";
   cin >> base;
   cout << "Enter its height:";
   cin >> height;

   rectangle(base, height, area, perimeter);

   cout << "Area: " << area << endl;
   cout << "Perimeter: " << perimeter;
   cout  << endl;
}
```

Pass-by-reference parameters

```
void rectangle( int wide, int high, int &area_rect, int &perim )

{
   area_rect = wide * high;
   perim = 2 * (wide + high);
}
```

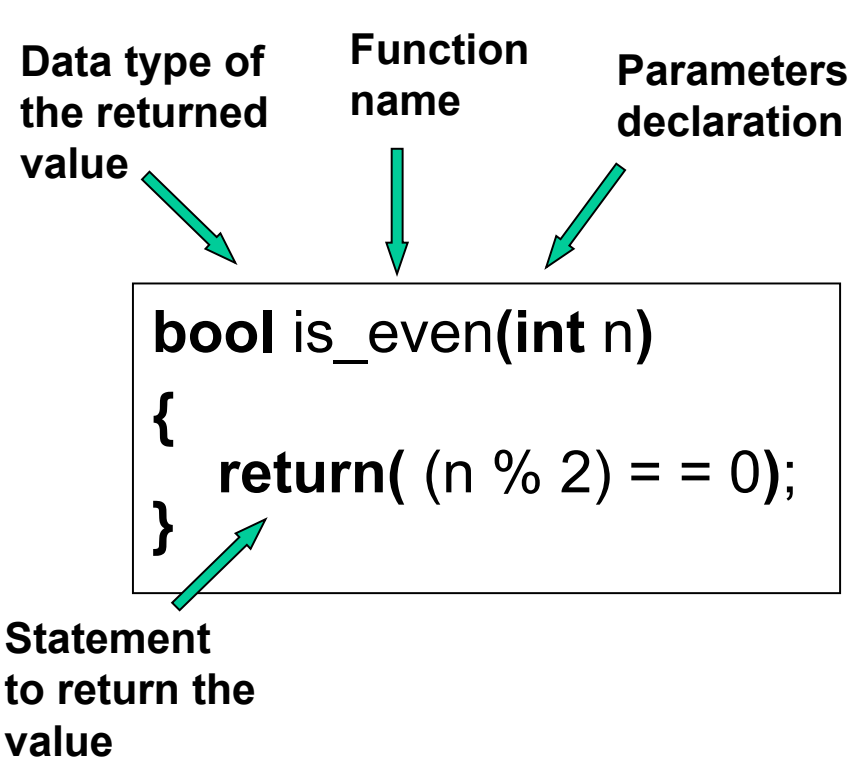
Topics

1. Modular decomposition
2. Communication between modules
3. **Procedures and Functions**
4. Scope of a variable
5. General structure of a program
6. Predefined functions in C language
7. Information sources



Concept of Function

- A function **returns a value** *associated with the function name*
- It is usually defined with N parameters (N≥ 1)
- Only **pass-by-value parameters** should be used




```
int  x;
bool fin;
```

Correct function calls

```
if (is_even(x) == true)
if (is_even(3))
if (is_even(x) && (x > 10) )
fin = is_even(x);
fin = is_even(4) || (x < 0);
```

Incorrect function call

```
is_even(x);
```

Concept of Procedure

- It can be defined with N parameters (N ≥ 0)
- It can use **pass-by-value** and/or **pass-by-reference** parameters
- It is called by using a statement made up of its name and the list of actual parameters (the call is a statement itself)

Indicates that the values can only be returned as parameters

Procedure name

Parameter declaration

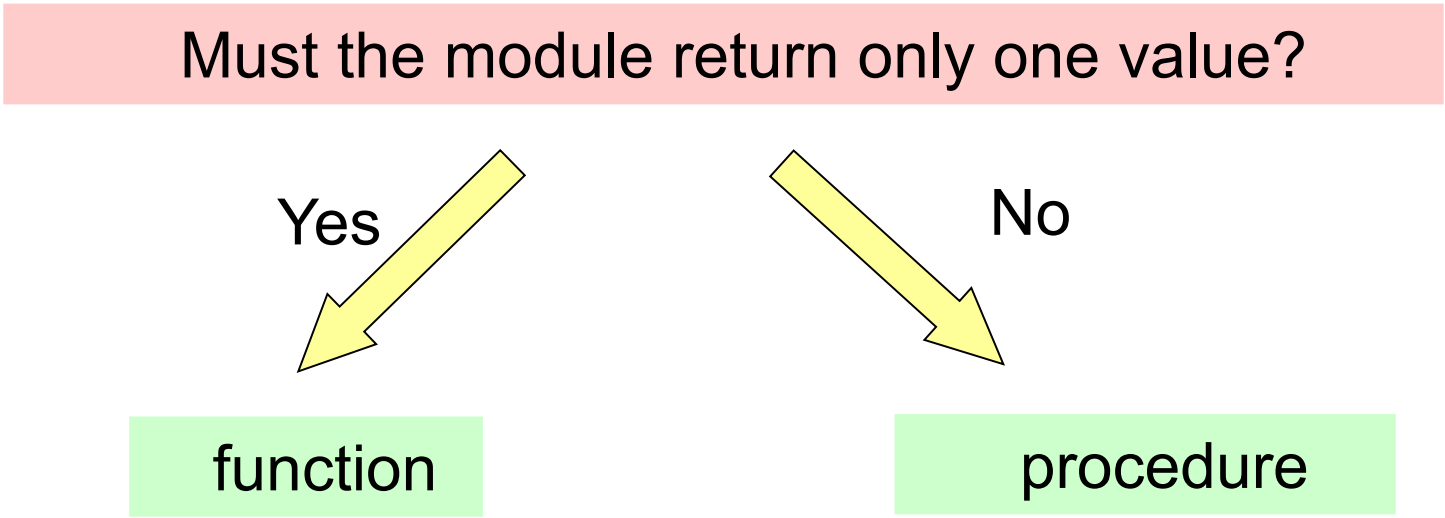
```
void Write_character(char c, int n)
{
   int i;
   for (i=1; i <= n; i++)
      cout << c;
   cout << endl;
}
```

```
int  num;
char c;
```

Correct procedure calls

```
Write_character('*',20);
Write_character(c, num);
Write_character('c', 5);
```

Must I use a procedure or a function?



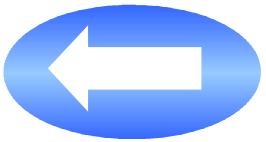
About return statement

return (*expression*);

- It ends the execution of the function body
- It returns the return value of the function, after evaluating the associated expression
- Recommendation: use a single return statement within a function body
- It should be the last statement in the function body

Topics

1. Modular decomposition
2. Communication between modules
3. Procedures and Functions
4. **Scope of a variable**
5. General structure of a program
6. Predefined functions in C language
7. Information sources



Concept of scope of a variable

- The scope of a variable defines its visibility, that is, from where the variable can be accessed

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

   cout << "Enter an integer number:  ";
   cin >> n;
   if (is_prime(n))
      cout << "The number is prime";
   else
      cout << "The number is not prime";
   cout << endl;
}
```

Scope of ***n***

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

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

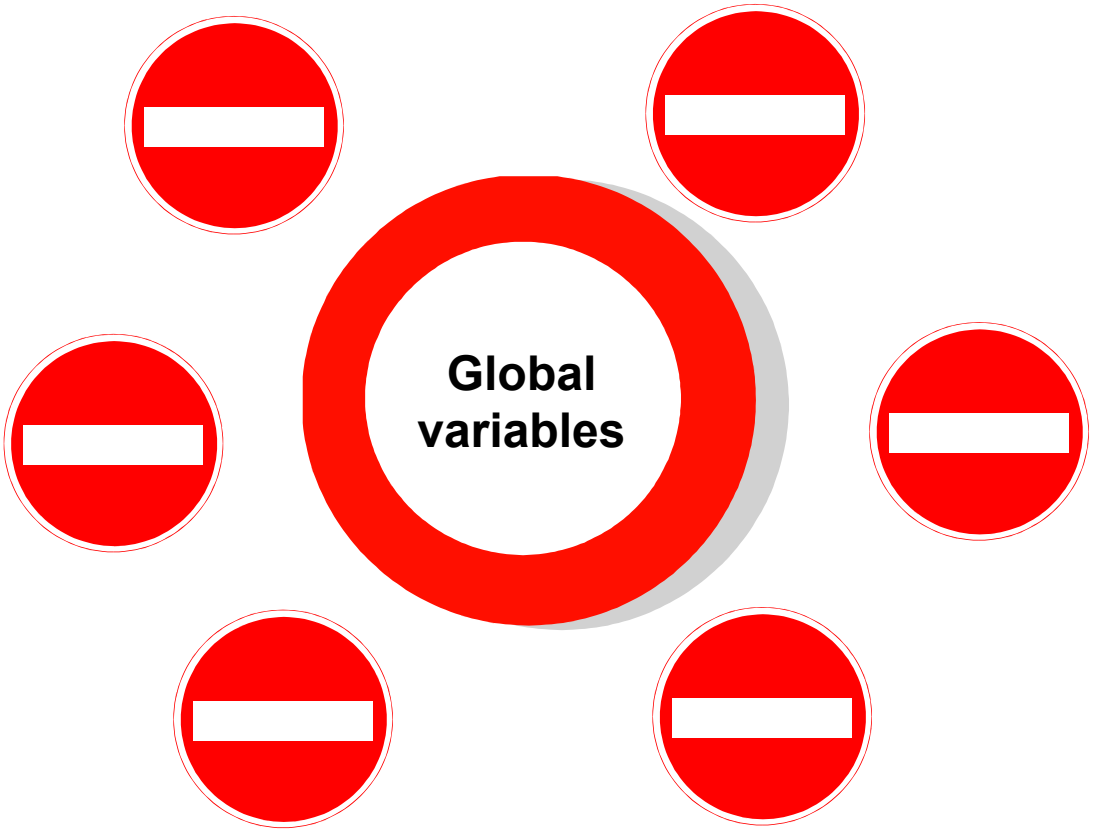
Scope of
num, count, prime

Local and global variables

- Local variable
  - Its scope is the body of the module where it is declared
  - It is created when it is declared, and it is destroyed when the module completes its execution
- Global variable
  - Its scope is the entire program (all modules and the main program)
  - It is created when it is declared, and it is destroyed when the program completes its execution

Don`t use global variables

- The **communication** between the modules must be done through **parameters**, *never through global variables*

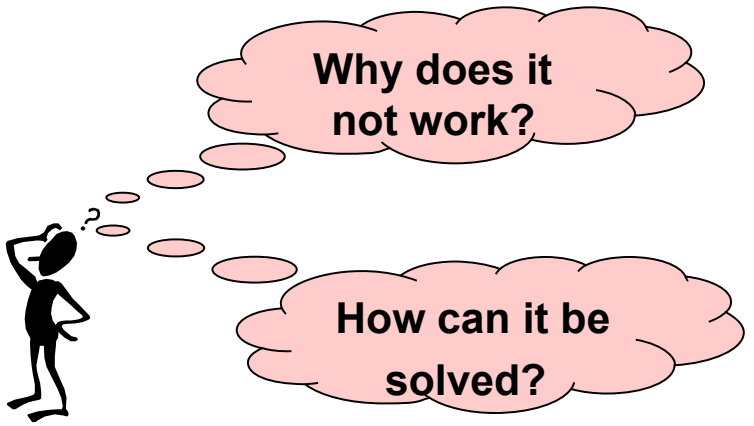


Side effect

- Any data communication between modules outside the parameters and the returning of results is called **a side effect**

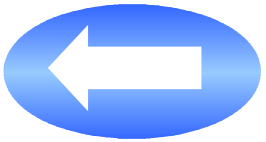
```
#include <iostream>
using namespace std;

int result;  // declaration of a global variable

int Greater_Num(int n1, int n2);

main() {
   int  n1, n2;   // number entered by keyboard (input data)
   int  greater;  // the greater of two numbers (output data)

   cout << "Enter two integer numbers :";
   cin >> n1 >> n2;
   result = n1 + n2;
   greater = Greater_Num (n1, n2);
   cout << "The addition of the two numbers is: " << result;
   cout << "  and the greater number is:"  << greater;
   cout <<  endl;
}
```

```
// function that returns the greater of two numbers
int Greater_Num(int n1, int n2)
{
   if (n1 > n2)
      result = n1;
   else
      result = n2;

   return(result);
}
```



Topics

1. Modular decomposition
2. Communication between modules
3. Procedures and Functions
4. Scope of a variable
5. **General structure of a program**
6. Predefined functions in C language
7. Information sources



What type of program must I be able to do?

#preprocesor directives

Constants declaration

Procedures and functions declaration

main() {

Variables declaration:

of simple data types

Main body (executable statements)

control statements

calls to procedures and functions

}

Procedures and functions definition

Program example

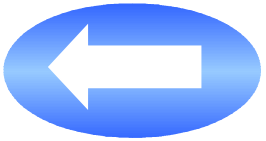
```
#include <iostream>
using namespace std;

// Currency exchange to euros
const float US_DOLAR_EURO =  1,4696;
const float UK_POUND_EURO =  1,4696;

// Procedures and functions declarations
void Read_Amount(float &amount, char &currency);
float Change_In_Euros(float amount, char currency);

main() {
   float amount;  // money amount  (input datum)
   char  currency; // currency type  (input datum)
   char  answer; // answer to continue (input datum)
   float  euros;  // equivalent amount in euros (output datum)

   do {
      Read_Amount(amount, currency);
      euros = Change_In_Euros(amount, currency);
      cout << "The change in euros is:" << euros << endl;
      cout << "Another amount? (Y/N) :";
      cin >> answer;
   } while ( (answer == 'y') || (answer == 'Y'));
}
```

```
// Read amount and currency type from keyboard, validate
// the entered data until they are correct
void Read_Amount(float &amount, char &currency)
{
   bool correct_data;
   do {
      cout << "Enter an amount of money and currency (D/P):";
      cin << amount << currency;
      correct_data = (amount > 0.0) &&
                              ( currency == 'D'  ||  currency == 'P');
   } while ( ! correct_data);
}

// Return the equivalent change in euros, given an amount and
// a currency
float Change_In_Euros(float amount, char currency)
{
   switch (currency) {
      case 'D' : euros = amount * US_DOLAR_EURO;
                 break;
      case 'P' : euros = amount * UK_POUND_EURO;
   }
   return (euros);
}
```

Remember to write a comment for every defined module, explaining what it does

Topics

1. Modular decomposition
2. Communication between modules
3. Procedures and Functions
4. Scope of a variable
5. General structure of a program
6. **Predefined functions in C language**
7. Information sources



Libraries in C/C++ language

- Most programming languages provide a collection of commonly used procedures and functions (**libraries**)
- In C / C + +, to make use of the modules included in a library, the compiler directive *#include* is used
- There is a large amount of libraries available:
  - Mathematical functions
  - Characters and character strings management
  - Input and output data management
  - Time management (date, time, ...)
  - And many others

Some predefined functions in C/C++ language

Library C++	Library C	Function	Description
<math.h>	<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
<iostream>	<ctype.h>	int isalnum(int c)	Returns true if the parameter is a letter or a digit
		int isdigit(int c)	Returns true if the paremeter is a digit
		int toupper(int c)	Returns the character in uppercase
	<stdlib.h>	int rand(void)	Returns a random number between 0 and RAND_MAX

Library C++	Library	Constant	Description
<iostream>	<stdint.h>	INT_MIN	Lowest representable integer number
		INT_MAX	Greatest representable integer number

Exercises

1. In cold weather, meteorologists report a so called *Wind Chill Factor* that takes into account wind speed and temperature. This factor can be approximated by the following formula:

   $$W = 13.12 + 0.6215 * t - 11.37 * s^{0.16} + 0.3965 * t * s^{0.016}$$

   where s = wind speed in m/s
   t = temperature in degrees Celsius: $t \leq 10$
   W = wind chill factor (in degrees Celsius)

   Design a module to request the value of the wind speed and temperature, and calculate W, taking into account the restriction imposed for the temperature.

2. Design a module that receives a number ***n*** as a parameter, and displays a square of asterisks of size ***n*****x*****n.***

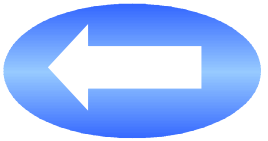
3. Improve exercise 2, adding another parameter representing the character to make up the square.

4. How can exercise 3 be modified to indicate that the square is empty or solid?

5. Design a module to read and validate an input number so that its value is greater that 0 and lower than 100, and return the sum and the amount of figures between 1 and the input value.

Topics

1. Modular decomposition
2. Communication between modules
3. Procedures and Functions
4. Scope of a variable
5. General structure of a program
6. Predefined functions in C language
7. **Information sources**



Information sources

Fundamentos de Programación
Jesús Carretero, Félix García, y otros
Thomson-Paraninfo 2007. ISBN: 978-84-9732-550-9

✓ Capítulo 7

Problemas Resueltos de Programación en Lenguaje C
Félix García, Alejandro Calderón, y otros
Thomson (2002) ISBN: 84-9732-102-2

✓ Capítulo 5

Resolución de Problemas con C++
Walter Savitch
Pearson Addison Wesley 2007. ISBN: 978-970-26-0806-6

✓ Capítulo 4