U.P.V. E.H.U.

# C programming basics

U.P.V. E.H.U.

# Outline

1. Introduction
2. Basic concepts
3. Functions
4. Data types
5. Control structures
6. Arrays and pointers
7. File management

# 3.1:

# Introduction

U.P.V. E.H.U.

# Review of programming

- **Program: steps to follow (recipe)**
- **Commands: orders to be carried out**

  Variables: containers of information.

  Assignment: to give a value to a variable.

  Expressions: calculations that give a result.

  Functions: reusable subprograms.

  Control structures: conditional actions of branching control.

# Features of C

- Very powerful standard libraries.
- Low level programming: bits, registers, memory.
- Disadvantages of C:
  - Very flexible: tolerates many unsuitable practices.
  - Memory management is complicated.

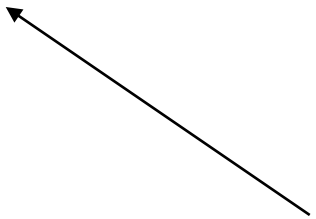
# 3.2:

# Basic concepts

U.P.V. E.H.U.

# Basic program: Hello world!

```
#include <stdio.h>
/* My first program*/
void main(void)
{
   printf(“Hello, world!\n”); //Message
}
```



Standard function

U.P.V. E.H.U.

# Variable, assignment, expression

```
int main(void)

    /* int: Returns an integer*/
{
  int total;              // Declaration of variable
  total=2+2;              // assignment
  return(total);          // returns 4
}
```

U.P.V. E.H.U.

# Operations

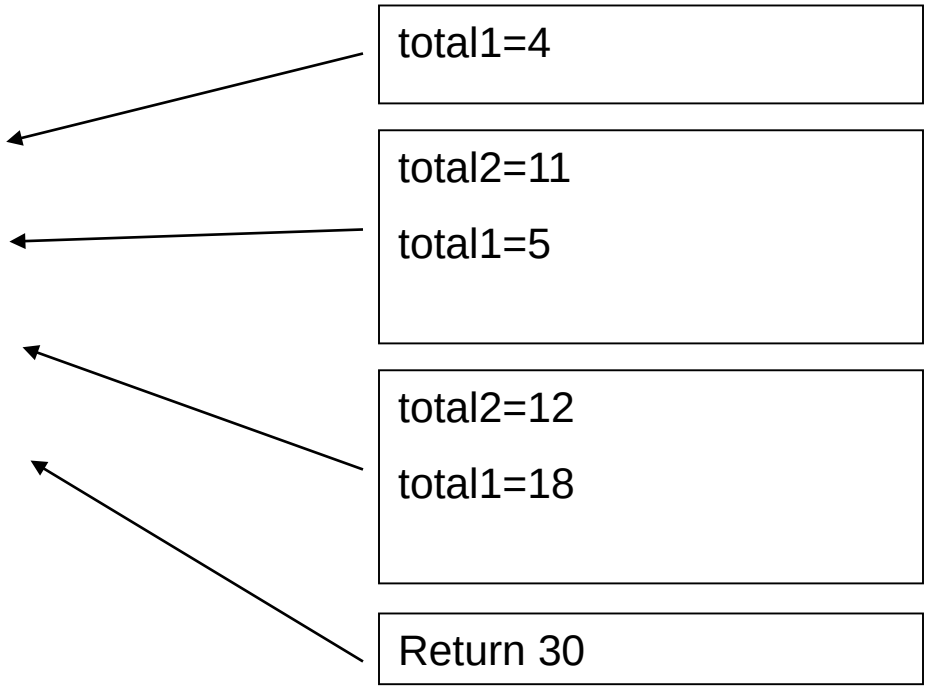
OPERATION / REPRESENTATION / EXAMPLE

| | | |
|---|---|---|
| Assignment | = | a=1; |
| a = a+1 | ++ | a++; ++a; |
| a = a-1 | -- | a--; --a; |
| a = a+b | += | a += b; |
| a = a-b | -= | a -= b; |

U.P.V. E.H.U.

# Example of operations

```
int main(void)
{
   int total1, total2;
   total1=2+2;
   total2=7+(total1++);
   total1=6+(++total2);
   return(total1+total2);
}
```



# 3.3:

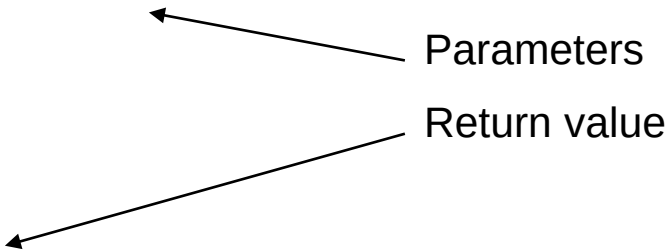
# Functions

# Functions

- **Prototype:** declaration of function.
- Int addit (int a, int b);

- **Implementation:** code.

```
Int addit (int a, int b)
{
  int result;
  result = a + b;
  return (result);
}
```




- **Call:** use.

```
printf("Result: %d\n", addit(4,6));
```

U.P.V. E.H.U.

# Printf function

- Syntax:
  - Printf ("String ...", variables);
  - % letter: insert the content of a variable:
    - % d: integer
    - % f: real number
    - % c: character
    - % s: string

U.P.V. E.H.U.

# Example 1: printf and scanf

```
#include <stdio.h>
#define PI   3.14159
void main(void)
{
float r, area, perimeter;

printf ("Enter value of radius: \ n");
scanf ("% f", & r);
perimeter = 2 * PI * r;
area = PI * r * r;
printf ("Area:% f \ n", area);
printf ("Perimeter:% f \ n", perimeter);
}
```

Constant

By reference

U.P.V. E.H.U.

# Example 2: printf and scanf

```
#include <stdio.h>

void main(void)
{
   char name [20]; // string: array of 20 characters

    printf ("Enter your name: \ n");
    scanf ("% s", name); // name: pointer to 1st character
 }
```

# 3.4:

# Data types

# Data types

- int: integers (short, long)
- float: real numbers (long double)
- char: character
- void: null (special type)
- array: size is initialized
- pointers: variable size

# Conversion between types

- Implicit (automatic, promotion)
  - printf ("Real:% f", integer);
- Explicit (made by the programmer, casting)
  - printf ("Real:% f", (float));

## 3.5:

# Control structures

U.P.V. E.H.U.

# Control structures

- **Conditional:** If a condition is true, the next commands are executed.

```
IF (A> B)
  {C = A;}
IF NOT
  {C = B;}
```

- **Iterative:** while a condition is true, the next commands are executed.

```
WHILE (A> 0)
  {A = A-1;}
```

U.P.V. E.H.U.

# Control structures

- Typical conditional structure

  If (expression) {BlockYES}

  Else {BlockNO}

- Example:
  - If (a> b) {c = a;}
  - Else {c = b;}

U.P.V. E.H.U.

# Conditional structure *switch*

```
switch(c)
{       case '+':   printf("%d\n", a+b);
                    break;
        case '-':   printf("%d\n", a-b);
                    break;
        case '*':   printf("%d\n", a*b);
                    break;
        case '/':   printf("%d\n", a/b);
                    break;
        default:    printf(“Error\n");
                    break;
}
```

# Conditional structure *switch* (2)

```
switch(c)
{       case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U':       printf (“It is a vowel\n");
                        break;
        default:        printf (“It is not a vowel\n");
                        break;
}
```

U.P.V. E.H.U.

# Iterative structure *while*

- Typical iterative structure

  while (expression) {BlockIteration}

- Example:

```
int n = 4;
while (n> 0) {
printf ("Number:% d \ n", n--);
}
```

U.P.V. E.H.U.

# Iterative structure *for*

- It is typically used when the number of iterations is previously known.

```
for (start; condition_to_continue; update)
    {BlockIteration}
```

- Example:

```
for (i = 0; i <100; i ++)
    {printf ("Number:% d \ n", i);}
```

# 3.6:

# Arrays and pointers

U.P.V. E.H.U.

# Arrays

- A group of variables of the same type. Maximum size initialized.

  int arrayintegers [40];

- Example:

  arrayintegers [4] = 14;

  printf ("The fifth element is:% d", arrayintegers [4]);

U.P.V. E.H.U.

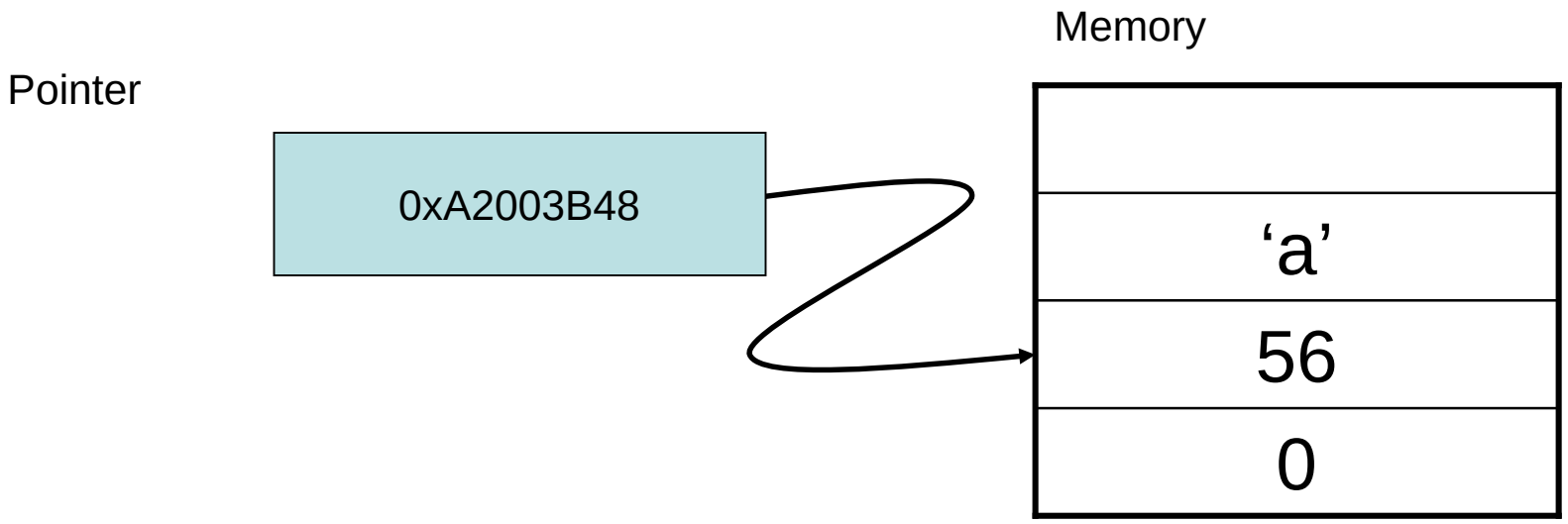
# Examples with arrays

```
int days_each_month [12] = {31,29, ... 31};

int number;
char error [] = ”Error";

scanf ("%f", & number); // address
scanf ("%s", error); // address

printf (“%f”, number); // value
printf (“%s”, *error); // value
```

U.P.V. E.H.U.

# Pointers

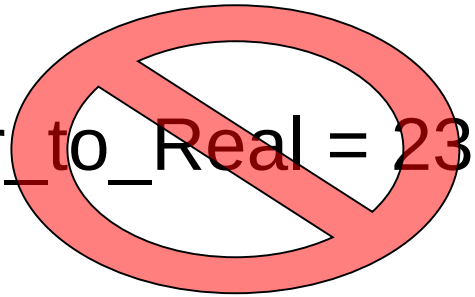
- Variables that store memory addresses.
- Very useful when working with data of variable size.



U.P.V. E.H.U.

# Examples with pointers

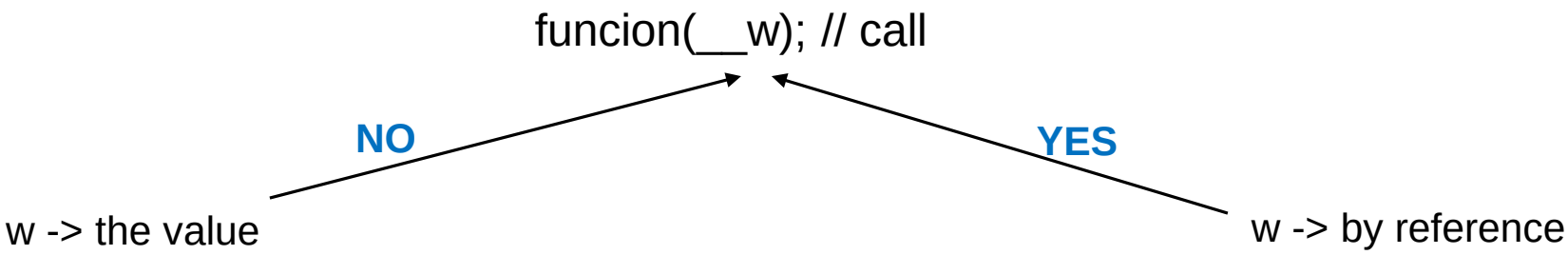
```
float *pointer_to_Real;
float Real_Normal_variable;

Real_Normal_variable = 23.4;

pointer_to_Real = 23.4;\\Would change the direction it points to

*pointer_to_Real =23.4; \\Change the value of the address it points to
```



U.P.V. E.H.U.

# Pointers with functions

- Passing the value or by reference:
  Do I want to change the content?



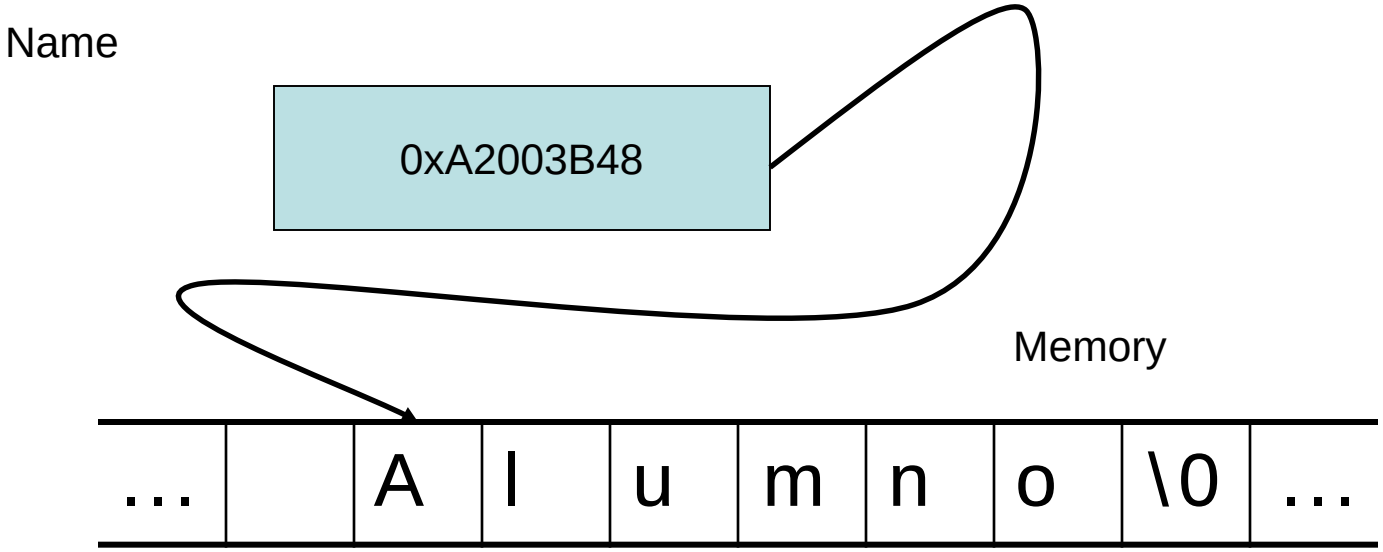

```
void funcion(float r)
{ ...
r=45.0; // No effect outside
...}
```

```
void funcion(float *r)
{ ...
*r=45.0; //w changed outside
...}
```

U.P.V. E.H.U.

# Typical pointer: a string

- String: string of characters.
- It ends always with the character '\0'.



U.P.V. E.H.U.

# Examples with string

/ * Declaration of a string * /

```
char name [5];
...
scanf ("%s", name); // pointer
name [0] = 'A'; // first element of the array
printf ("%s \ n", *name); // value
```

U.P.V. E.H.U.

# Functions for strings

- *strcpy*: copies a string.

  strcpy (string1, "Hello");
- *strcmp*: compares two strings.

  if (strcmp (string1, string2)) ...
- *strcat*: concatenates two strings.

  strcat (string1, string2);
- *strlen*: Returns the length of the string.

  sf (strlen> 10) ...

U.P.V. E.H.U.

# Memory management

- Overally, the problem of strings and pointers is: How much memory?
- Static Memory allocation:

```
char help [8];

char *string = "value";
```

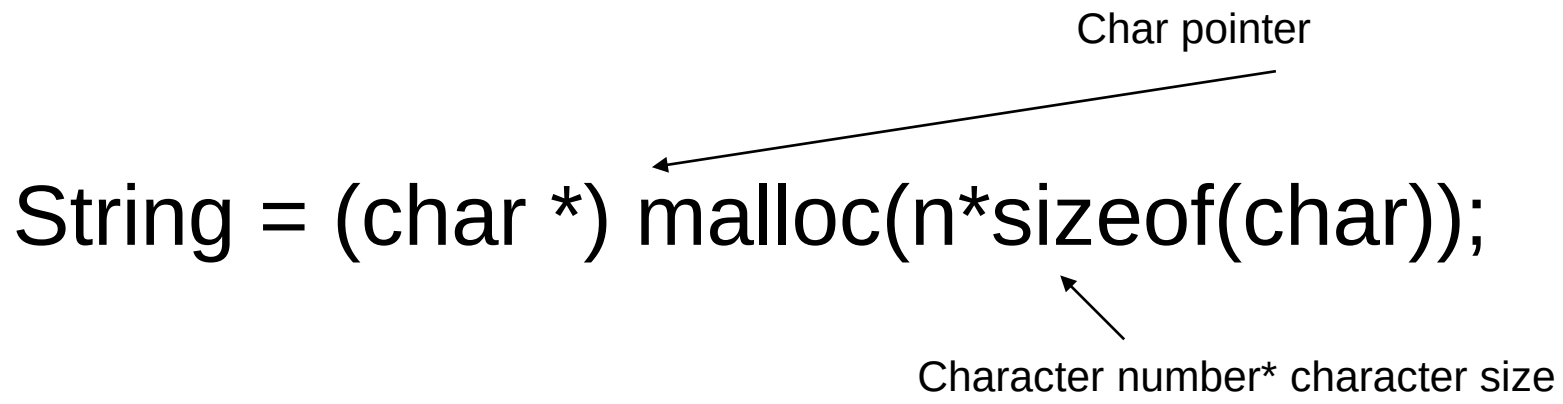
- Not practical. We want real variable size.

U.P.V. E.H.U.

# Memory management

- Dynamic memory allocation:

  malloc: Allocates a size in bytes to a variable.




realloc: reassign the memory.

free: free/deallocate memory. Important.

U.P.V. E.H.U.

# *main* funtion with arguments

- String array to pass parameters to programs: * *argv []* and its counter *argc*.

```
int main (int argc, char * argv [])
{...
    int count;
    for (count = 0; count <argc; count ++)
    printf ("The argument %d is: %s \ n", count, argv [count]);
    ...
    return 0;
}
```

## 3.7:

# File management

U.P.V. E.H.U.

# File management in C

- Functions to work with files:
  - *fopen / fclose*: functions to open / close a file.
  - *fgetc / fputc*: functions to read / write a character.
  - *fgets / fputs*: functions to read / write a string.
  - *fscanf / fprintf*: functions to read / write a block.

U.P.V. E.H.U.

# File management in C

- Standard input and output:
  - *stdin*: standard input, usually related to the keyboard.
  - *stdout*: standard output, usually related to the screen.
  - *stderr*: standard error output, usually related to the screen.

U.P.V. E.H.U.

# Writing examples

```
FILE * file;
char c;
file = fopen ("path \filename", "r");

if (file == NULL)
     fprintf (stderr, "Error opening file \n");
else {while ((c = fgetc (file)) = EOF);
     fputc (c, stdout);}

fclose (file);
```

U.P.V. E.H.U.

# Reading examples

```
FILE * file;
char string [80];
file = fopen ("path \filename", "w");

if (file == NULL)
    fprintf (stderr, "Error opening file \n");
else {scanf ("%s", string);
    fprintf (file, "%s \n", * string); }

fclose (file);
```