Tutoría 8
Física Computacional I
Grado en Física

Javier Carrasco Serrano, javcarrasco@madrid.uned.es

Física Computacional I, Las Tablas

Prueba evaluable de programación con Maxima

Prueba Maxima - Enunciados

Adobe Acrobat Document

Prueba Maxima – Dudas

Prueba Maxima – Errata clase anterior

La última diferencia del caso de orden 2 respecto a la EDO de orden 1 es la transformación que hay que aplicar a la segunda condición inicial (la que incluye el valor inicial de la función derivada). En este caso, no habrá que sustituir directamente, sino que habrá que sustituir primero la función derivada en x, por la función derivada en $x(y)$ con la expresión que ya hemos utilizado antes: $f'(x) = y'(x) \cdot f'(y)$

Por lo que si la segunda condición inicial es $f'(x_0) - v_0 = 0$, la nueva condición sería, utilizando que $x_0 = x(y_0)$:

$$f'(x) = y'(x) \cdot f'(y) \rightarrow f'(x_0) = y'(x(y_0)) \cdot \boldsymbol{f'(y_0)} \rightarrow v_0 = y'(x(y_0)) \cdot \boldsymbol{f'(y_0)} \rightarrow \boldsymbol{f'(y_0)} - \frac{v_0}{y'(x(y_0))} = 0$$

Tema 11:
El lenguaje C mediante ejemplos

10

10.2. Compilación, enlazado y ejecución de programas

Proyecto en Code::Blocks. Se utiliza un proyecto para dividir un programa grande en pequeños programas (archivos).

- File → New → Project → Console application → C → “HolaMundo” → “HolaMundo.cpb” (una carpeta por proyecto).
- Al elegir *console application* crea una estructura por defecto.
- File → Save file as… → “helloworld0.c” → sobre el panel de la izquierda (*workspace*), sobre “main.c” → remove file from Project → add files → “helloworld0.c”
- Añadimos *system(“PAUSE”)* al programa para que no se cierre al ejecutarse desde consola (o al abrir el .exe, que es lo mismo). Este comando deja abierta la consola hasta que se pulse alguna tecla. *stdlib.h* hace que se cargue esta función.
- Dos posiblidades:
  - Build → Compile current file → crea un objeto (O) del módulo, y ejecuta sólo esta parte del programa; rápido para detectar errores.
  - Build → Build → crea un objeto (O) por cada módulo, ejecuta todo el programa y crea el ejecutable.
  - Build and run, Run, o doble click sobre el .exe que se ha generado para correr el programa.

Introducción

Estructura código C:

- Directivas: *#include*
- Función principal del programa (se debe entender como llamada al sistema): *main()*
- Instrucciones: *;*
- Bloques instrucciones: *{ }*
- Comentarios: */* */*

Ejemplo: Listado 11.1

Adobe Acrobat Document

Adobe Acrobat Document

11.1. Estructura básica de un programa: la función “main”

int main(**int** argc, char** argv) **int** main(**int** argc, char□*argv[])

main → llamada al sistema para ejecutar el programa.

argc → indica cuántos argumentos se pasan a la función main.

argv → donde se guardan los argumentos del programa (matriz o array), “argumentos de línea de comandos”.

En argv[0] se guarda el nombre del programa, y en el resto, los argumentos del programa (si los tiene): argv[1], … , argv[n]

Ejemplo: Listado 11.2

printf() → saca por pantalla; \n → salto de línea; %s → pasamos un string; %d → pasamos un entero, %g → pasamos un número en coma flotante.

11.2. Declaración de variables, asignación, operaciones e impersión

Declaración de variables es obligatoria en C → se debe indicar nombre y tipo.

= → asignación de valores.

atof(), atoi() → sirve para asignar a variables valores (float, integer) de entrada de la función, es decir, los valores que hay en argv[…].

Ejemplo: Listado 11.3

Bucles y condicionales

- Pseudocódigo: https://es.wikipedia.org/wiki/Pseudoc%C3%B3digo
- Bucle: repetir una o varias sentencias un número determinado (o no) de veces.
  1. Para i=valor_inicial hasta i=valor_final ejecutar {sentencias}
  2. Para cada i en los elementos de una lista ejecutar {sentencias}
  3. Para i=valor_inicial hasta i=valor_final ejecutar {sentencias} mientras se cumpla condición
  4. Para i=valor_inicial hasta i=valor_final ejecutar {sentencias} hasta que se cumpla condición

- Sentencia condicional: ejecutar una o varias sentencias dependiendo si se cumple o no una condición.
  1. Si {condición lógica} entonces {sentencias}
  2. Si {condición lógica} entonces {sentencias}, si no {sentencias}

- Combinaciones de sentencias condicionales y bucles.

11.3. Control de flujo: “if. . . else”

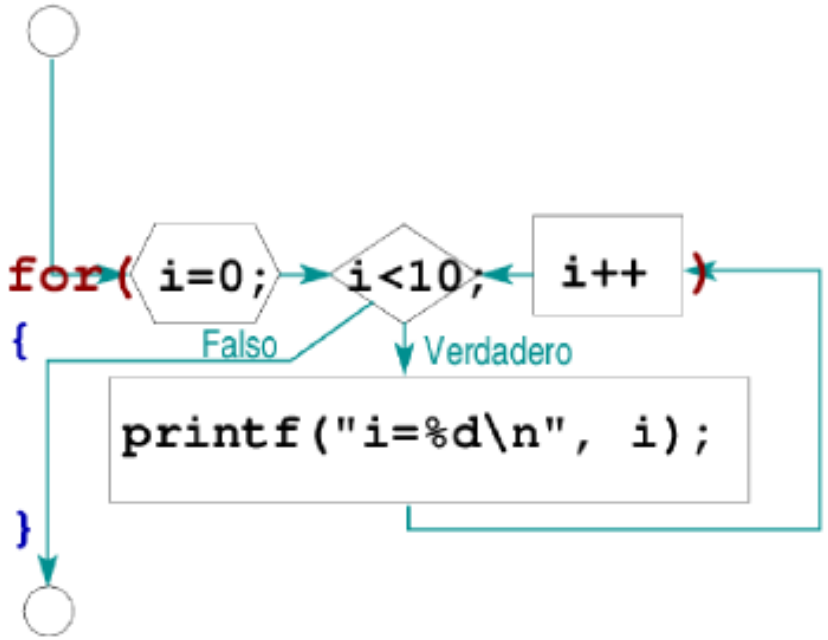
```
If (condición) {
        sentencia1;
} else {sentencia2}
```

Operadores de comparación: <, >, >=, <=, ==, !=

Operadores lógicos: ||, &&, ^, !

Ejemplo: Listado 11.4

11.4. Bucle “for”

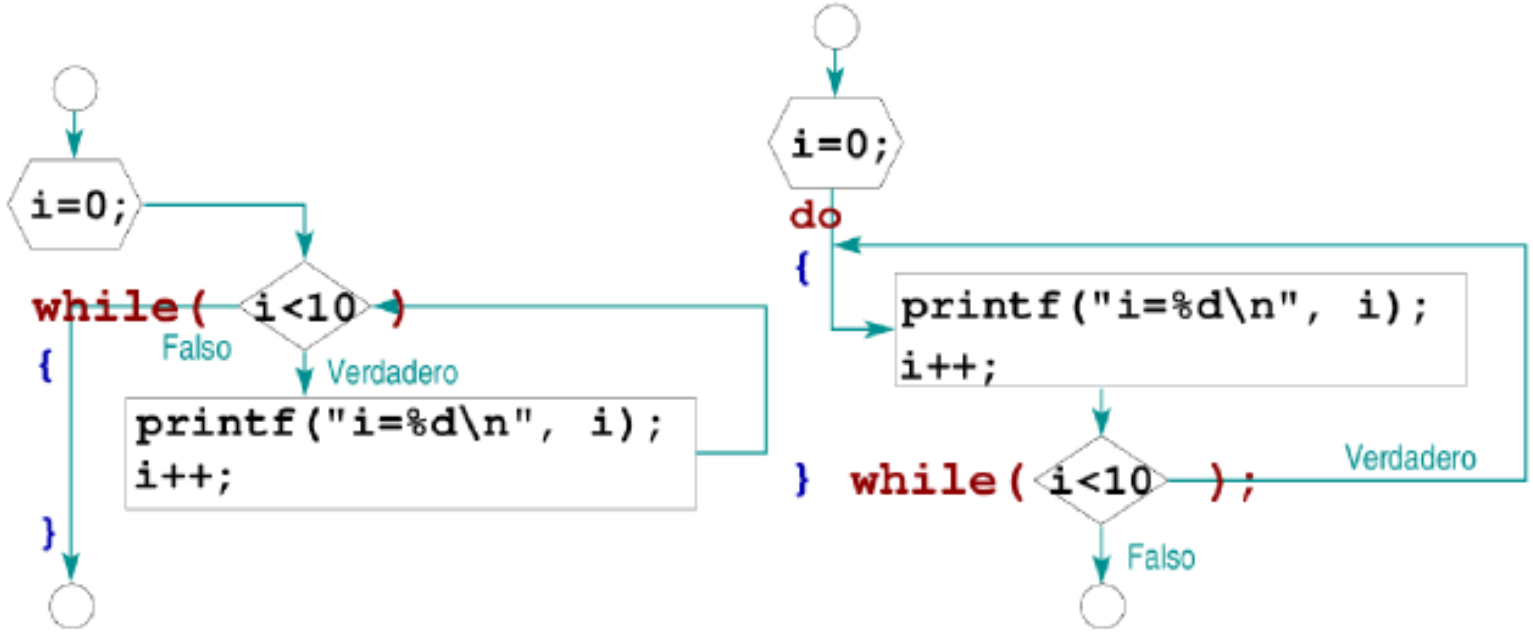


i++ → suma de 1 en 1

i+=m → suma de m en m

Ejemplo: Listado 11.5

11.5. Bucles “while” y “do…while”



do…while ejecuta al menos una vez

Ejemplo: Ejercico 11.9, Ejercicio 11.10

11.6. Selección con “switch…case…default”

```
if ( x == 1 ) {
      ...
} else if ( x==2 ) {
      ...
} else if ( x==3 ) {
      ...
} else {
      ...
}
```

```
switch( x ) {
case 1:
      ...
      break;
case 2:
      ...
      break;
case 3:
      ...
      break;
default:
      ...
}
```

En lugar de anidar else-if, se crean tablas de llaves-valores;

default → cuando ocurra un valor sin llave.

break → sirve para salir de la estructura y que no la siga recorriendo a ver si se cumple la condición.

Ejemplo: Listado 11.6

11.7. Vectores y matrices

tipo_de_datos nombre_vector [dimensión_vector]

Se pueden asignar los valores en la declaración, o en una asignación habitual.

Ojo: en C se empieza a contar por 0 al acceder a los elementos de un vector (en Maxima por 1).

Ejemplo: Listado 11.7

Igual para matrices:

tipo_de_datos nombre_matriz [dimensión1_matriz] [dimensión2_matriz]

Ejemplo: Ejercicio 11.11

11.8. Punteros

Un puntero es una variable que contiene una dirección de memoria. Por tanto, se va a poder acceder a una variable a través de su dirección en memoria (*), e igualmente se va a poder operar con la dirección de memoria de una variable (&).

- * → contenido en memoria de una dirección.
- & → dirección en memoria de la variable.
- int *q → q es un puntero a un valor int, o lo que hay en q es una variable tipo int.
- *(&var)==var; &(*q)==q;

“Para el propósito de este curso y el nivel de programación que se espera conseguir, el uso de punteros no es en absoluto necesario. En muchos casos pueden ser sustituidos por otros elementos del lenguaje C como los vectores o matrices. En otros casos basta con cambiar el modo de almacenamiento de la variable en cuestión. Sin embargo, su uso permite explotar aún más a potencialidad del C y conseguir diseños de programa mucho más elegantes.”

Ejemplo: Listado 11.8

11.8. Punteros

Los vectores elementos de vectores y matrices se almacenan consecutivamente en memoria (las matrices, por filas) → nos podemos desplazar por un vector o matriz con un puntero.

Al declarar un vector, int v[5], estamos creando un puntero v que apunta a v[0], es decir, v==&v[0], y el resto de valores se guardan en las casillas de memoria v+1, … , v+4.

En general: v+n=&v[n]; *(v+n)=v[n];

Ejemplo: Listado 11.9

Una matriz es un vector de vectores, por lo que:

M[i,j] = *(*(M+i)+j) = *(M+i)[j] = *(M[i]+j)

Ejemplo: Listado 11.10

Si pensamos en las cadenas de caracteres como vectores de caracteres, también les podemos dar un tratamiento con punteros.

Gracias!

Javier Carrasco Serrano

Física Computacional I, Las Tablas