Introducción al lenguaje C

Programación 2

Profesorado de Informática
CeRP del Suroeste, Uruguay

29 de marzo de 2016

Estructuras de control IV

- Iteración
  - Sentencia `for`:

```
for (inicio; condicion; paso)
    cuerpo
```

```
for (int i = 0; i < 10; i++)
    printf("*");
```

Tipos de datos estructurados I

- Enumerados `enum`:

```
enum mes {enero, febrero, marzo, abril, mayo, junio, julio,
    agosto, setiembre, octubre, noviembre, diciembre};
mes este_mes = marzo;
```

Tipos de datos estructurados II

- Estructuras `struct`:

```
struct fecha {
    int f_dia;
    mes f_mes;
    int f_anio;
};
```

Tipos de datos estructurados II

- Estructuras `struct`:

```
struct fecha {
    int f_dia;
    mes f_mes;
    int f_anio;
};
```

  - Se usa . para acceder a los miembros:

```
fecha hoy;

hoy.f_dia = 2;
hoy.f_mes = marzo;
hoy.f_anio = 2016;

int dia_hoy = hoy.f_dia;
```

Tipos de datos estructurados II

- Estructuras `struct`:

```
struct fecha {
    int f_dia;
    mes f_mes;
    int f_anio;
};
```

  - Se usa . para acceder a los miembros:

```
fecha hoy;

hoy.f_dia = 2;
hoy.f_mes = marzo;
hoy.f_anio = 2016;

int dia_hoy = hoy.f_dia;
```

  - Nota: esta es la forma de declarar estructuras y enumerados en C++ (y en C*). En C es ligeramente distinto.

Tipos de datos estructurados III

- Punteros

```
int * p; // p es un puntero a un número entero

int i;
p = &i; // p apunta a la dirección de i
*p = 10; // i toma el valor 10

int* p2;
p2 = p; // p2 apunta a la dirección de i

p = NULL; // así se deja en NULL un puntero

p = new int; // así se pide memoria
delete p; // se libera la memoria apuntada por p
```

Tipos de datos estructurados III

- Punteros

```
int * p; // p es un puntero a un número entero

int i;
p = &i; // p apunta a la dirección de i
*p = 10; // i toma el valor 10

int* p2;
p2 = p; // p2 apunta a la dirección de i

p = NULL; // así se deja en NULL un puntero

p = new int; // así se pide memoria
delete p; // se libera la memoria apuntada por p
```

  - Notar que estaría mal liberar la memoria asignada a *p2* ya que no es memoria asignada dinámicamente.

Tipos de datos estructurados III

- Punteros

```
int * p; // p es un puntero a un número entero

int i;
p = &i; // p apunta a la dirección de i
*p = 10; // i toma el valor 10

int* p2;
p2 = p; // p2 apunta a la dirección de i

p = NULL; // así se deja en NULL un puntero

p = new int; // así se pide memoria
delete p; // se libera la memoria apuntada por p
```

  - Notar que estaría mal liberar la memoria asignada a *p2* ya que no es memoria asignada dinámicamente.
  - `new` y `delete` son de C++

Tipos de datos estructurados IV

- Punteros

```
int *p3; // el asterisco puede ir junto a la variable (a la
    izquierda)

int* p4, p5;
```

Tipos de datos estructurados IV

- Punteros

```
int *p3; // el asterisco puede ir junto a la variable (a la ←
    izquierda)

int* p4, p5;
```

  - ¿`p5` es un puntero? **No**, la declaración anterior es equivalente a la siguiente:

```
int* p4;
int p5;
```

Tipos de datos estructurados IV

- Punteros

```
int *p3; // el asterisco puede ir junto a la variable (a la ←
    izquierda)

int* p4, p5;
```

  - ¿`p5` es un puntero? **No**, la declaración anterior es equivalente a la siguiente:

```
int* p4;
int p5;
```

  - Es como si el asterisco se pegara a la variable y no al tipo

Tipos de datos estructurados IV

- Punteros

```
int *p3; // el asterisco puede ir junto a la variable (a la ←
    izquierda)

int* p4, p5;
```

  - ¿`p5` es un puntero? **No**, la declaración anterior es equivalente a la siguiente:

```
int* p4;
int p5;
```

  - Es como si el asterisco se pegara a la variable y no al tipo
  - Si queremos dos punteros:

```
int* p4, * p5;
```

Tipos de datos estructurados V

- Punteros y estructuras

```
(*puntero_fecha).f_dia = 4;
puntero_fecha->f_dia = 4; // más fácil
```

Tipos de datos estructurados VI

- Arreglos

Tipos de datos estructurados VI

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria

Tipos de datos estructurados VI

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento está en el índice 0

Tipos de datos estructurados VI

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento está en el índice 0
  - Estáticos:

```
int arr[2]; // valores posibles: arr[0] y arr[1]

int vector[5] = {1, 2, 3, 4, 5};
int matriz[2][3] = {{1, 2, 3}, {4, 5, 6}};

int suma = 0;
for (int i = 0; i < 5; i++)
    suma += vector[i];
```

Tipos de datos estructurados VI

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento está en el índice 0
  - Estáticos:

```
int arr[2]; // valores posibles: arr[0] y arr[1]

int vector[5] = {1, 2, 3, 4, 5};
int matriz[2][3] = {{1, 2, 3}, {4, 5, 6}};

int suma = 0;
for (int i = 0; i < 5; i++)
    suma += vector[i];
```

  - Dinámicos:

```
int* vector = new int[10];
delete [] vector;
```

Tipos de datos estructurados VII

- Arreglos
  - Tener en cuenta que ni C ni C++ verifican que el índice esté dentro del rango permitido. Te deja así acceder a otra dirección de memoria, y **a veces puede dar segmentation default.**

Conversión de tipos I

- La mayoría de las conversiones son implícitas

```
float vf = 1.6;
int vi = 1 + vf; // vi = 2 (float se trunca)
vi = 1 + vf + vf; // vi = 4 (cast al "más grande")
vi = vi + true; // vi = 5 (true es 1)
vi = vi + false; // vi = 5 (false es 0)
vi = 'a' + 1; // vi = 98 (valor ASCII)
char vc = 'a' + 1; // vc = 'b'
vf = 1.5 + vi // vf = 99.500000
bool vb = 237; // vb = true (0 es false, otro true)
vf = 3 / 2; // vf = 1.000000
```

- Se puede hacer cast explícito

```
vf = (float)3 / 2; // vf = 1.500000
```

Conversión de tipos II

- ¿Cuál es el valor de `res`?

```
int res;
int i = 5 - 4.3;
bool b = 100.1;

if (i = 0)
    res = b + 100.9;
else
    res = b + i;
```

- ¿El resultado es 1?
- **Error común:** haber puesto = en lugar de ==. Aunque el Resultado es 1, si cambiamos por == obtenemos 101.

Funciones I

- Funciones

```c
int sumar(int a, int b) {
    return a + b;
}
```

  - Se puede invocar así:

```c
c = sumar(3, 8);
```

Funciones II

- Procedimientos

```c
void imprimirSuma(int a, int b) {
    int suma = a + b;
    printf("La suma es: %d\n", suma);
}
```

Funciones III

- Las funciones no se puede anidar
- En C todos los parámetros se pasan por valor
  - En C++ (y C*) existe el pasaje por referencia (&)

```
void sumarEnB(int a, int & b) {
    b = a + b;
}
```

    - No confundir con el operador & de punteros
  - En C el pasaje por referencia se simula utilizando punteros

```
void inc(int *i) {
    (*i) += 1;
}
```

    - Se puede invocar así: `inc(&valor);`

Entrada/Salida I

- `printf`: impresión en salida estándar

```
printf("hola mundo\n");
printf("-> %d, ", estructura->dato);
printf("( %d, %d)\n", pri, seg);
```

  - Función “especial” que recibe una cantidad variable de parámetros
  - El primer parámetro es la **cadena de texto de formato**
  - El resto depende de los **especificadores de formato** que se encuentren en el primero
  - Especificadores:

    `%d` `int`
    `%c` `char`
    `%f` `float`
    `%s` `char*`

  - Algunas secuencias especiales: `\'`, `\"`, `\\`, `\n`, `\t`

Entrada/Salida II

- `scanf`: lectura en la entrada estándar
  - La cadena de texto de formato es igual que en `printf`
  - Pero los parámetros tienen que ser punteros (para poder modificarlos)

```
int val, cant;
char str [10];
cant = scanf(" %d- %s", &val, str);
```

`89-bla` → cant = 2, val = 89, str = “bla”
`89bla` → cant = 1, val = 89, str = ??
`bla` → cant = EOF, val = ??, str = ??

Entrada/Salida III

- Ambas funciones están en la biblioteca estándar de C, en `stdio.h` (“Standard Input-Output” o Entrada y salida estándar)
- Para poder usarlas, se debe importar la biblioteca:

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

- En C++ existen `cin` y `cout` para la entrada y salida estándar, pero en este curso preferimos usar el estilo C

C*: C con algunas cosas de C++

En resumen, C* es C pero con las siguientes cosas de C++:

- `new` y `delete`

C*: C con algunas cosas de C++

En resumen, C* es C pero con las siguientes cosas de C++:

- `new` y `delete`
- Comentarios en línea

C*: C con algunas cosas de C++

En resumen, C* es C pero con las siguientes cosas de C++:

- `new` y `delete`
- Comentarios en línea
- Declaración de tipos como en C++ para registros y enumerados

C*: C con algunas cosas de C++

En resumen, C* es C pero con las siguientes cosas de C++:

- `new` y `delete`
- Comentarios en línea
- Declaración de tipos como en C++ para registros y enumerados
- Pasaje por referencia

C*: C con algunas cosas de C++

En resumen, C* es C pero con las siguientes cosas de C++:

- `new` y `delete`
- Comentarios en línea
- Declaración de tipos como en C++ para registros y enumerados
- Pasaje por referencia
- `bool`

Más información

1. Probar en la computadora qué pasa con cierto comportamiento que tengan duda

Más información

1. Probar en la computadora qué pasa con cierto comportamiento que tengan duda
2. Buscar en Internet (recuerden las cosas que sí usamos de C++)

Más información

1. Probar en la computadora qué pasa con cierto comportamiento que tengan duda
2. Buscar en Internet (recuerden las cosas que sí usamos de C++)
3. Consultar bibliografía del curso (ver en UruguayEduca)

Más información

1. Probar en la computadora qué pasa con cierto comportamiento que tengan duda
2. Buscar en Internet (recuerden las cosas que sí usamos de C++)
3. Consultar bibliografía del curso (ver en UruguayEduca)
4. Consultar en Foros del curso del Portal en UruguayEduca