Introducción al Lenguaje C

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

August 3, 2010

El Laboratorio

- 5 tareas individuales obligatorias

El Laboratorio

- 5 tareas individuales obligatorias
- Posibles puntajes por tarea:
  - 0 $\rightarrow$ Reprobado
  - 1 $\rightarrow$ Insatisfactorio
  - 2 $\rightarrow$ Satisfactorio
  - 3 $\rightarrow$ Muy Bueno

El Laboratorio

- 5 tareas individuales obligatorias
- Posibles puntajes por tarea:
  - 0 $\rightarrow$ Reprobado
  - 1 $\rightarrow$ Insatisfactorio
  - 2 $\rightarrow$ Satisfactorio
  - 3 $\rightarrow$ Muy Bueno
- Requisitos mínimos de aprobación:
  - Total de puntos mayor o igual a 10
  - Puntaje mayor a 0 en cada tarea
  - Fuentes entregados compilan y linkeditan
  - Trabajo individual

El Laboratorio

- 5 tareas individuales obligatorias
- Posibles puntajes por tarea:
  - 0 $\rightarrow$ Reprobado
  - 1 $\rightarrow$ Insatisfactorio
  - 2 $\rightarrow$ Satisfactorio
  - 3 $\rightarrow$ Muy Bueno
- Requisitos mínimos de aprobación:
  - Total de puntos mayor o igual a 10
  - Puntaje mayor a 0 en cada tarea
  - Fuentes entregados compilan y linkeditan
  - Trabajo individual
- Lenguaje: **C***
  - C con algunas cosas de C++

- Programación 3 es un curso de **Estructuras de Datos y Algoritmos**

- Programación 3 es un curso de **Estructuras de Datos y Algoritmos**
- Programación 3 **NO ES** un curso de C/C++

Comparación con Modula 2

hola.mod:

```
MODULE hola
FROM InOut IMPORT;


BEGIN
 WriteLine(hola mundo)
END hola.
```

hola.cpp:

```
#include <stdio.h>

int main()
{
 printf("hola mundo\n");
 return 0;
}
```

Comparación con Modula 2

hola.mod:

```
MODULE hola
FROM InOut IMPORT;


BEGIN
 WriteLine(hola mundo)
END hola.
```

hola.cpp:

```
#include <stdio.h>

int main()
{
 printf("hola mundo\n");
 return 0;
}
```

`main` es una función especial, a partir de la cual comienza la ejecución del programa.

Compilación

- Módulos de definición con extensión **.h**

Compilación

- Módulos de definición con extensión **.h**
- Módulos de implementación con extensión **.cpp**

Compilación

- Módulos de definición con extensión **.h**
- Módulos de implementación con extensión **.cpp**
- Se compila y linkedita con **g++** (compilador de C++)

Compilación

- Módulos de definición con extensión **.h**
- Módulos de implementación con extensión **.cpp**
- Se compila y linkedita con **g++** (compilador de C++)
  - Compilo mod1.cpp, mod2.cpp y mod3.cpp:

    ```
    g++ -c mod1.cpp
    g++ -c mod2.cpp
    g++ -c mod3.cpp
    ```

    generando los archivos mod1.o, mod2.o y mod3.o

Compilación

- Módulos de definición con extensión **.h**
- Módulos de implementación con extensión **.cpp**
- Se compila y linkedita con **g++** (compilador de C++)
  - Compilo mod1.cpp, mod2.cpp y mod3.cpp:

    ```
    g++ -c mod1.cpp
    g++ -c mod2.cpp
    g++ -c mod3.cpp
    ```

    generando los archivos mod1.o, mod2.o y mod3.o
  - Los linkedito:

    ```
    g++ mod1.o mod2.o mod3.o -o prog.exe
    ```

    generando el ejecutable prog.exe

Compilación

- Módulos de definición con extensión **.h**
- Módulos de implementación con extensión **.cpp**
- Se compila y linkedita con **g++** (compilador de C++)
  - Compilo mod1.cpp, mod2.cpp y mod3.cpp:

    ```
    g++ -c mod1.cpp
    g++ -c mod2.cpp
    g++ -c mod3.cpp
    ```

    generando los archivos mod1.o, mod2.o y mod3.o
  - Los linkedito:

    ```
    g++ mod1.o mod2.o mod3.o -o prog.exe
    ```

    generando el ejecutable prog.exe
  - Otra opción:

    ```
    g++ mod1.cpp mod2.cpp mod3.cpp -o prog.exe
    ```

Tipos de Datos Elementales

- Entero: `int`

Tipos de Datos Elementales

- Entero: `int`
- Caracter: `char`

Tipos de Datos Elementales

- Entero: `int`
- Caracter: `char`
- Real: `float`

Tipos de Datos Elementales

- Entero: `int`
- Caracter: `char`
- Real: `float`
- Booleano: `bool` (extensión de C++)

Tipos de Datos Elementales

- Entero: `int`
- Caracter: `char`
- Real: `float`
- Booleano: `bool` (extensión de C++)

Tipos de Datos Elementales

- Entero: `int`
- Caracter: `char`
- Real: `float`
- Booleano: `bool` (extensión de C++)

Ejemplos:

```
int i;
char c;
float f;
bool b;

i = 1;
b = false;
```

Expresiones

- Operador de Asignación: =

```
int a;
int b = 2;

a = 7;
a = b;
```

Expresiones

- Operador de Asignación: `=`

```
int a;
int b = 2;

a = 7;
a = b;
```

  - La asignación retorna un valor, por lo que es válido:

    ```
    a = b = 9;
    ```

Expresiones

- Operador de Asignación: `=`

```
int a;
int b = 2;

a = 7;
a = b;
```

  - La asignación retorna un valor, por lo que es válido:

    ```
    a = b = 9;
    ```

  - ERROR COMÚN: Confundir con comparación booleana de Modula 2

Expresiones (2)

- Operadores de Comparación: ==, !=, <, <=, > y >=

Expresiones (2)

- Operadores de Comparación: ==, !=, <, <=, > y >=
- Operadores de Lógicos: &&, || y !

Expresiones (2)

- Operadores de Comparación: ==, !=, <, <=, > y >=
- Operadores de Lógicos: &&, || y !
- Operadores Aritméticos: +, -, *, / y %

Expresiones (2)

- Operadores de Comparación: ==, !=, <, <=, > y >=
- Operadores de Lógicos: &&, || y !
- Operadores Aritméticos: +, -, *, / y %

Expresiones (2)

- Operadores de Comparación: `==`, `!=`, `<`, `<=`, `>` y `>=`
- Operadores de Lógicos: `&&`, `||` y `!`
- Operadores Aritméticos: `+`, `-`, `*`, `/` y `%`

Precedencia

```
a+1 < b && c == 9*d || e < 7
```

Expresiones (2)

- Operadores de Comparación: ==, !=, <, <=, > y >=
- Operadores de Lógicos: &&, || y !
- Operadores Aritméticos: +, -, *, / y %

Precedencia

```
a+1 < b && c == 9*d || e < 7
```

equivale a:

```
(((a+1) < b) && (c == (9*d))) || (e < 7)
```

Expresiones (3)

- Incremento y decremento: ++, --

Expresiones (3)

- Incremento y decremento: ++, --
  - ++a incrementa a en uno y retorna su valor luego del incremento

Expresiones (3)

- Incremento y decremento: ++, --
  - ++a incrementa a en uno y retorna su valor luego del incremento
  - a++ incrementa a en uno y retorna su valor antes del incremento

Expresiones (3)

- Incremento y decremento: ++, --
  - ++a incrementa a en uno y retorna su valor luego del incremento
  - a++ incrementa a en uno y retorna su valor antes del incremento

Expresiones (3)

- Incremento y decremento: ++, --
  - ++a incrementa a en uno y retorna su valor luego del incremento
  - a++ incrementa a en uno y retorna su valor antes del incremento

```
int a = 1;
int b,c;
b = ++a;
c = a++;
```

Expresiones (3)

- Incremento y decremento: ++, --
  - ++a incrementa a en uno y retorna su valor luego del incremento
  - a++ incrementa a en uno y retorna su valor antes del incremento

Valores Finales:

```
int a = 1;
int b,c;
b = ++a;
c = a++;
```

Expresiones (3)

- Incremento y decremento: ++, --
  - ++a incrementa a en uno y retorna su valor luego del incremento
  - a++ incrementa a en uno y retorna su valor antes del incremento

```
int a = 1;
int b,c;
b = ++a;
c = a++;
```

Valores Finales:

- a $\rightarrow$ 3

Expresiones (3)

- Incremento y decremento: ++, --
  - ++a incrementa a en uno y retorna su valor luego del incremento
  - a++ incrementa a en uno y retorna su valor antes del incremento

```
int a = 1;
int b,c;
b = ++a;
c = a++;
```

Valores Finales:

- a $\rightarrow$ 3
- b $\rightarrow$ 2

Expresiones (3)

- Incremento y decremento: ++, --
  - ++a incrementa a en uno y retorna su valor luego del incremento
  - a++ incrementa a en uno y retorna su valor antes del incremento

```
int a = 1;
int b,c;
b = ++a;
c = a++;
```

Valores Finales:

- a $\rightarrow$ 3
- b $\rightarrow$ 2
- c $\rightarrow$ 2

Sentencias

- Comparación

Sentencias

- Comparación
  - Sentencia `if`:

```
if (value >= 6 && value <= 12)
{
   printf ("Aprobado");
   cantidad_de_aprobados++;
}
else if (value >= 3)
   printf ("Examen");
else if (value >= 0)
   printf ("Reprobado");
else
   printf ("Valor incorrecto");
```

Sentencias (2)

- Sentencia `switch`:

```
switch (value)
{
  case 6: case 7: case 8: case 9:
  case 10: case 11: case 12:
      printf ("Aprobado");
      cantidad_de_aprobados++;
      break;
  case 3: case 4: case 5:
      printf ("Examen");
      break;
  case 0: case 1: case 2:
      printf ("Reprobado");
      break;
  default:
      printf ("Valor incorrecto");
}
```

Sentencias (3)

- Iteración
  - Sentencia `while`:

```
while condicion
    cuerpo
```

Sentencias (3)

- Iteración
  - Sentencia `while`:

```
while condicion
    cuerpo
```

```
int i = 0;
while (i < 10)
{
    printf ("*");
    i++;
}
```

Sentencias (3)

- Iteración
  - Sentencia `while`:

```
while condicion
    cuerpo
```

```
int i = 0;
while (i < 10)
{
   printf ("*");
   i++;
}
```

  - Sentencia `for`:

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

Sentencias (3)

- Iteración
  - Sentencia `while`:

```
while condicion
    cuerpo
```

```
int i = 0;
while (i < 10)
{
   printf ("*");
   i++;
}
```

  - Sentencia `for`:

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

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

Tipos de Datos Estructurados

- Enumerados: `enum`

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

mes este_mes = agosto;
```

Tipos de Datos Estructurados (2)

- Estructuras: `struct`

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

Tipos de Datos Estructurados (2)

- 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 = 4;
hoy.f_mes = agosto;
hoy.f_anio = 2010;

int dia_hoy = hoy.f_dia;
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;
```

Tipos de Datos Estructurados (3)

- Punteros
  `int * p;` $\rightarrow$ `p` es un puntero a `int`

Tipos de Datos Estructurados (3)

- Punteros

  `int * p;` → `p` es un puntero a `int`

```
int i;
p = &i;
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;
```
→ p es un puntero a int

```
int i;
p = &i;
```
→ p apunta a la dirección de i

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;
```
→ p es un puntero a int

```
int i;
p = &i;
*p = 10;
```
→ p apunta a la dirección de i

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;          → p es un puntero a int

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

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;
```
→ `p` es un puntero a `int`

```
int i;
p = &i;
```
→ `p` apunta a la dirección de `i`

```
*p = 10;
```
→ `i` toma el valor 10

```
int * p2;
p2 = p;
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;          → p es un puntero a int

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
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;
```
→ p es un puntero a int

```
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;
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;          → p es un puntero a int

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;

p = new int;
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;          → p es un puntero a int

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;

p = new int;      → p apunta una nueva dirección de enteros
delete p;
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;          → p es un puntero a int

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;

p = new int;      → p apunta una nueva dirección de enteros
delete p;         → p se libera la memoria apuntada por p
```

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;          → p es un puntero a int

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;

p = new int;      → p apunta una nueva dirección de enteros
delete p;         → p se libera la memoria apuntada por p
```

  - ERROR COMÚN: hacer `delete p2;`

Tipos de Datos Estructurados (3)

- Punteros

```
int * p;          → p es un puntero a int

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;

p = new int;      → p apunta una nueva dirección de enteros
delete p;         → p se libera la memoria apuntada por p
```

  - ERROR COMÚN: hacer `delete p2;`
  - `new` y `delete` son de C++

Tipos de Datos Estructurados (4)

- Arreglos

Tipos de Datos Estructurados (4)

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

Tipos de Datos Estructurados (4)

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

Tipos de Datos Estructurados (4)

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento tiene índice 0
  - Estáticos: memoria reservada en tiempo de compilación

```
int arr[2];
```

Tipos de Datos Estructurados (4)

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento tiene índice 0
  - Estáticos: memoria reservada en tiempo de compilación

```
int arr[2];
```

Tipos de Datos Estructurados (4)

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento tiene índice 0
  - Estáticos: memoria reservada en tiempo de compilación
    `int arr[2];` $\rightarrow$ valores posibles: `arr[0]` y `arr[1]`

Tipos de Datos Estructurados (4)

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento tiene índice 0
  - Estáticos: memoria reservada en tiempo de compilación

    `int arr[2];` $\rightarrow$ 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}};
    ```

Tipos de Datos Estructurados (4)

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento tiene índice 0
  - Estáticos: memoria reservada en tiempo de compilación

    `int arr[2];` $\rightarrow$ 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 (4)

- Arreglos
  - Varios objetos del mismo tipo puestos consecutivamente en memoria
  - El primer elemento tiene índice 0
  - Estáticos: memoria reservada en tiempo de compilación

    `int arr[2];` $\rightarrow$ 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: memoria reservada en tiempo de ejecución

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