Introducción al lenguaje C

Programación 2

Profesorado de Informática
CeRP del Suroeste, Colonia, Uruguay

15 de marzo de 2016

Objetivos

- Mostrar las principales características del lenguaje que será utilizado en el curso

Objetivos

- Mostrar las principales características del lenguaje que será utilizado en el curso
- A partir de esta clase van a necesitar salir a practicar bastante de C para familiarizarse. Esta clase apunta a ser una ayuda inicial.

Objetivos

- Mostrar las principales características del lenguaje que será utilizado en el curso
- A partir de esta clase van a necesitar salir a practicar bastante de C para familiarizarse. Esta clase apunta a ser una ayuda inicial.
- Recordar que Programación 2 no es un curso sobre C, sino sobre Estructuras de Datos y Algoritmos.

Lenguaje Posible

- Lenguaje: C*
  - Es el lenguaje C, pero sumándole algunas (pocas) cosas de C++
  - Es “ficticio”

Hola mundo!

hola.cpp:

```
#include <stdio.h>

int main()
{
    printf("¡Hola, mundo!\n");
    return 0;
}
```

Hola mundo!

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

- Archivos con extensión **.cpp**

Compilación

- Archivos con extensión **.cpp**
- Se compila y enlaza con `g++` (compilador de C++)

Compilación

- Archivos con extensión **.cpp**
- Se compila y enlaza con `g++` (compilador de C++)
  - Compilo *programa.cpp*:
    `g++ -c programa.cpp`
    generando elarchivo *programa.o*

Compilación

- Archivos con extensión **.cpp**
- Se compila y enlaza con `g++` (compilador de C++)
  - Compilo *programa.cpp*:
    `g++ -c programa.cpp`
    generando elarchivo *programa.o*
  - Enlazo:
    `g++ programa.o -o programa`
    generando el ejecutable *programa*

Compilación

- Archivos con extensión **.cpp**
- Se compila y enlaza con `g++` (compilador de C++)
  - Compilo *programa.cpp*:
    `g++ -c programa.cpp`
    generando elarchivo *programa.o*
  - Enlazo:
    `g++ programa.o -o programa`
    generando el ejecutable *programa*
  - O directamente:
    `g++ programa.cpp -o programa`

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` (de C++)

Tipos de datos elementales

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

Ejemplos:

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

i = 1;
b = false;
```

Declaración de variables

- Las variables pueden ser declaradas en cualquier lugar.

Declaración de variables

- Las variables pueden ser declaradas en cualquier lugar.
- **Recomendación:** declararlas cuando se usan por primera vez.

Declaración de variables

- Las variables pueden ser declaradas en cualquier lugar.
- **Recomendación:** declararlas cuando se usan por primera vez.
- **Mito común:** pensar que declararlas fuera de bucles es más eficiente.

Declaración de variables

- Las variables pueden ser declaradas en cualquier lugar.
- **Recomendación:** declararlas cuando se usan por primera vez.
- **Mito común:** pensar que declararlas fuera de bucles es más eficiente.

```
int minimo(int tam, int[] arreglo) {
    int iMin = 0;
    for (int i = 1; i < tam; i++)
        if (arreglo[i] < arreglo[iMin])
            iMin = i;
    return arreglo[iMin];
}
```

Comentarios

```
/* comentario
de
varias
lineas */
int i = 1; /* asigno 1 a i */
char c; // comentario de una linea (C++)
float f;
// otro comentario
```

Expresiones I

- Operador de asignación: =

```
int a;
int b = 2;

a = 7;
a = b;
```

Expresiones I

- 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 otro Lenguaje, ejemplo Pascal

Expresiones II

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

Expresiones II

- Operadores de comparación: ==, !=, <, <=, > y >=
- Operadores lógicos: &&, || y !

Expresiones II

- Operadores de comparación: ==, !=, <, <=, > y >=
- Operadores lógicos: &&, || y !
- Operadores aritméticos: +, -, *, / y %

Expresiones II

- Operadores de comparación: ==, !=, <, <=, > y >=
- Operadores 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 III

- Incremento y decremento: ++ y --

Expresiones III

- Incremento y decremento: ++ y --
  - ++*a* incrementa el valor de *a* y retorna su valor **luego** del incremento

Expresiones III

- Incremento y decremento: ++ y --
  - ++*a* incrementa el valor de *a* y retorna su valor **luego** del incremento

Expresiones III

- Incremento y decremento: ++ y --
  - ++*a* incrementa el valor de *a* y retorna su valor **luego** del incremento
  - *a*++ incrementa el valor de *a* y retorna su valor **antes** del incremento
  - Análogo para decrementar

Expresiones III

- Incremento y decremento: ++ y --
  - *++a* incrementa el valor de *a* y retorna su valor **luego** del incremento
  - *a++* incrementa el valor de *a* y retorna su valor **antes** del incremento
  - Análogo para decrementar

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

Expresiones III

- Incremento y decremento: ++ y --
  - *++a* incrementa el valor de *a* y retorna su valor **luego** del incremento
  - *a++* incrementa el valor de *a* y retorna su valor **antes** del incremento
  - Análogo para decrementar

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

Valores finales:

Expresiones III

- Incremento y decremento: ++ y --
  - $++a$ incrementa el valor de a y retorna su valor **luego** del incremento
  - $a++$ incrementa el valor de a y retorna su valor **antes** del incremento
  - Análogo para decrementar

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

Valores finales:

- $a \rightarrow 3$

Expresiones III

- Incremento y decremento: ++ y --
  - $++a$ incrementa el valor de a y retorna su valor **luego** del incremento
  - $a++$ incrementa el valor de a y retorna su valor **antes** del incremento
  - Análogo para decrementar

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

Valores finales:

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

Expresiones III

- Incremento y decremento: ++ y --
  - $++a$ incrementa el valor de a y retorna su valor **luego** del incremento
  - $a++$ incrementa el valor de a y retorna su valor **antes** del incremento
  - Análogo para decrementar

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

Valores finales:

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

Constantes I

Se pueden definir utilizando `define`:

```c
#include <stdio.h>

#define BASE 10
#define ALTURA 5

int main() {
    int area = BASE * ALTURA;
    printf("Area: %d", area);
    return 0;
}
```

Constantes II

O usando `const`:

```c
#include <stdio.h>

int main() {
    const int BASE = 10;
    const int ALTURA = 5;
    int area = BASE * ALTURA;
    printf("Area: %d", area);
    return 0;
}
```

Constantes II

O usando `const`:

```c
#include <stdio.h>

int main() {
    const int BASE = 10;
    const int ALTURA = 5;
    int area = BASE * ALTURA;
    printf("Area: %d", area);
    return 0;
}
```

- La diferencia es que `define` es un reemplazo de texto antes de compilar y `const` utiliza variables (y por lo tanto tiene su espacio de memoria, su tipo, etc.) que no se pueden modificar.

Constantes II

O usando `const`:

```c
#include <stdio.h>

int main() {
    const int BASE = 10;
    const int ALTURA = 5;
    int area = BASE * ALTURA;
    printf("Area: %d", area);
    return 0;
}
```

- La diferencia es que `define` es un reemplazo de texto antes de compilar y `const` utiliza variables (y por lo tanto tiene su espacio de memoria, su tipo, etc.) que no se pueden modificar.
- Es buena práctica definir los nombres de las constantes en mayúsculas.

Estructuras de control I

- Selección
  - Sentencia `if`:

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

Estructuras de control II

- Selección
  - Sentencia `switch` :

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

Estructuras de control III

- Iteración

Estructuras de control III

- Iteración
  - Sentencia `while`:

```
while (condicion)
    cuerpo
```

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

Estructuras de control IV

- Iteración
  - Sentencia `for`:

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

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

Continuará