Introducción al Lenguaje C

Introducción al Lenguaje y Estructuras de Control

Claudio Omar Biale
Facultad de Ciencias Exactas Químicas y Naturales
Universidad Nacional de Misiones
09/08/2012 y 10/08/2012

Introducción

- C fue inventado por Dennis Ritchie en el año 1972 en los laboratorios Bell.
- Fue originalmente desarrollado como un lenguaje para programación de sistemas.
- Es un lenguaje procedural, orientado a bloques.
  - No es un lenguaje orientado a objetos.
- Permite la programación modular en el que el programa puede escribirse en archivos separados que luego se compilan y enlazan para producir un archivo ejecutable único.
- Cercano al hardware.
- Es usado en la programación de:
  - sistemas embebidos,
  - sistemas operativos,
  - procesadores digitales de señales *(DSP)*.

Características

- No realiza comprobación de errores en tiempo de ejecución.
  - Ej.: no se comprueba que no se sobrepasen los límites de los arreglos.
- No hay procedimientos, sólo funciones.
  - Existen funciones que no devuelven ningún valor.
- Carece de manejo de excepciones y recolección de basura.
- El lenguaje diferencia mayúsculas de minúsculas.
- Todas las sentencias terminan con punto y coma.
  - Una sentencia es una instrucción o expresión en C que tiene una consecuencia. Pueden ser asignaciones, operaciones, llamadas a funciones.
- Los espacios en blanco y los tabuladores son ignorados por el compilador.
- Bloques de código *(Conjunto de sentencias)* delimitados con llaves.

Características

- Posee estructuras, uniones y tipos de datos compuestos.
- Permite el manejo de punteros a memoria y arreglos.
- Dispone de una biblioteca externa estándar.
- Compila a código nativo.
- Posee un macro procesador.

Características

- Solo permite el pasaje de parámetros por valor, para realizar el pasaje de parámetros por referencia se deben usar punteros.

- Número reducido de palabras clave:
  - 32 en C89,
  - 37 en C99 y
  - 44 en C11.

Comentarios

- De múltiples líneas:

```
/* comentario */
```

- De una sola línea:

```
// comentario hasta el final de línea
```

Nuestro Primer Programa en C

```
/* Esto es un comentario
de varias líneas */

int main (void) {
     return 0; // devolvemos 0
}
```

- Todo programa en C tiene una función `main()` que es el punto de inicio de la ejecución del mismo.
- Si la función no recibe ningún parámetro se escribe `void` en la lista de parámetros o se deja vacía la lista.
- La invocación de `return` dentro de la función `main()` finaliza la ejecución del programa.

Final de un Programa

- Un programa escrito en C finaliza:
  - Al llegar al final de la función `main()`.
  - Cuando la función `main()` invoca un `return`.
  - Si se ejecuta la función `exit()`[1] (definida en el archivo `stdlib.h`).
  - Se interrumpe externamente la ejecución de alguna manera.
  - El programa falla internamente.

[1] Existen variantes

Compilación y Ejecución

- Compilar un archivo fuente:

```
gcc -o nombre_binario fuente.c
```

- Compilar un archivo fuente con control de *warnings*:

```
gcc -Wall -o nombre_binario fuente.c
```

- Compilar un archivo fuente incluyendo en el binario información necesaria para depurarlo:

```
gcc -g -o nombre_binario fuente.c
```

- Ejecutar un binario:

```
./nombre_binario
```

Variables

- Concepto:
  - Es una posición de almacenamiento de datos de la memoria de la computadora que tiene nombre.

- Características:
  - El nombre de una variable puede contener letras, números y _.
  - Comienzan con letras o _.
  - No pueden usarse como nombre de variables las palabras reservadas o usar nombre de funciones.
  - Se deben declarar todas las variables antes de usarlas, establecer su tipo y de ser necesario darles un valor inicial.
  - El nombre de las variables puede tener hasta 31 caracteres.

Tipos de Datos

- Los tipos de datos básicos en C son:
  - Numéricos:
    - Enteros sin Signo.
    - Enteros con Signo.
    - Punto Flotante.
  - Lógicos.
  - Carácter.

- Determinados tipos básicos permiten modificadores:
  - `unsigned`: se utiliza para representar sólo valores positivos *(sin signo)*.
  - `signed`: para representar valores positivos y negativos *(por defecto)*.
  - `long`: para representar enteros o decimales largos.
  - `short`: para representar enteros cortos.

Tipos de Datos

- No existen las cadenas de caracteres como tipos elementales.
- `complex.h` permite manejar números complejos *(Definido en C99).*

Tipos de Datos

- Enteros sin Signo:
  - `unsigned char`
  - `unsigned short int` o `unsigned short`
  - `unsigned int` o `unsigned`
  - `unsigned long int` o `unsigned long`
  - `unsigned long long int` o `unsigned long long`
- Enteros con Signo:
  - `signed char`
  - `short int` o `short`
  - `int`
  - `long int` o `long`
  - `long long int` o `long long`
- Decimales:
  - `float`
  - `double`
  - `long double`

Tipos de Datos

- Lógicos:
  - Por lo general se utilizan los enteros para representar datos lógicos, utilizando el siguiente criterio:
    - `Verdadero`: valor distinto de cero.
    - `Falso`: valor igual a cero.
  - `_Bool`
  - `bool`
    - definido en el archivo `stdbool.h` permite utilizar los valores: `true` y `false`.

- Cararter:
  - `char`

Declaración de Variables

- Las variables se declaran de la siguiente forma:

```
tipo_de_datos nombre [ = valor_inicial ];
```

- Declaración simple:

```
int valor;
char caracter;
```

- Declaración múltiple:

```
int valor1, valor2;
float precio, iva;
```

- Declaración con asignación:

```
int x = 0;
char letra = 'a';
```

- Declaración combinada:

```
int v, y = 10, z;
```

Alcance de las Variables

- Global:
  - La variable declarada es visible desde todas las funciones del programa.

- Local:
  - La variable declarada sólo es visible dentro del bloque en la que es declarada.
  - Si existe una variable declarada globalmente con igual nombre que la variable local, tiene prioridad la variable local.

Expresiones Constantes

- Son valores que no se pueden modificar durante la ejecución del programa.

- Los tipos de constantes pueden ser:
  - Literales:
    - Enteras.
    - Coma flotante.
    - Caracter.
    - Cadena de caracteres.
  - Simbólicas:
    - Usando la directiva del preprocesador `#define`.
    - Utilizando la palabra reservada `const`.

Expresiones Constantes

- Una Constante Literal Entera puede ser:
  - `Decimal`: Es aquella que comienza con un dígito decimal distinto de cero al que sigue cualquier secuencia de dígitos decimales.
  - `Octal`: es aquella que comienza por un `0`, y va seguida de cualquier secuencia de dígitos octales.
  - `Hexadecimal`: es aquella que comienza por `0x` o `0X`, y va seguida de cualquier secuencia de dígitos hexadecimales.

Expresiones Constantes

- El tipo por defecto de las Constantes Literales Enteras es `int`.

- Si se quiere modificar este aspecto, se le pueden añadir uno de los siguientes sufijos:
  - `U`: indica que se corresponde con un `unsigned int`.
  - `L`: indica que se corresponde con un `long int`.
  - `LL`: indica que se corresponde con un `long long int`.
  - `UL` o `LU`: indica que se corresponde con un `unsigned long int`.
  - `ULL` o `LLU`: indica que se corresponde con un `unsigned long long int`.

Expresiones Constantes

- Una Constante Literal de Coma Flotante es aquella que tiene un punto decimal o un exponente o ambos.
- El tipo por defecto de las Constantes Literales de Coma Flotante es `double`.
- Si se quiere modificar este aspecto, se le pueden añadir uno de los siguientes sufijos:
  - `F`: indica que se corresponde con un `float`.
  - `L`: indica que se corresponde con un `long double`.

Expresiones Constantes

- Una Constante Literal de Caracter es un solo caracter encerrado con comillas simples.

- Almacena el valor entero que representa el caracter indicado según el código de representación utilizado.

Expresiones Constantes

- Algunos caracteres no se pueden representar, para ello se utilizan secuencias de escape, algunas son:
  - `'\''`: comilla simple.
  - `'\"'`: comilla doble.
  - `'\\'`: barra invertida.
  - `'\?'`: cierre de interrogación.
  - `'\t'`: tabulador horizontal.
  - `'\v'`: tabulador vertical.
  - `'\n'`: nueva línea.
  - `'\r'`: retorno de carro.
  - `'\a'`: sonido (campana).
  - `'\b'`: retroceso.
  - `'\0'`: nulo.

- Mediante secuencias de escape se puede expresar cualquier carácter ASCII indicando su código en octal (`\ooo`) o en hexadecimal (`\xhh`). Donde los símbolos `o` representan dígitos octales y los `h` dígitos hexadecimales.

Expresiones Constantes

- Una Constante Literal de Cadena de Caracteres es una secuencia de cero o más caracteres encerrados entre comillas dobles.

- La representación interna de una cadena tiene un carácter nulo (`'\0'`) al final, de modo que el almacenamiento físico es uno más del numeró de caracteres escritos en la cadena.

- No se puede comparar un caracter con una cadena de caracteres.

Expresiones Constantes

- Constantes Enteras Decimales

```
1U, 124, 124L
```

- Constantes Enteras Octales

```
01, 0102, 01L
```

- Constantes Enteras Hexadecimales

```
0x0, 0x11ef
```

- Constantes Decimales

```
32.0, 32e7
```

- Constantes de Caracter

```
'a', '\n', '\50', '\x9'
```

- Constantes de Cadena de Caracteres

```
"HOLA MUNDO\n", "El caracter barra invertida es \\"
```

Expresiones Constantes

- Constantes Simbólicas:

```
#define num 20
const int numero = 10;
```

Operadores Aritméticos

- `+` : suma.
- `-` : resta.
- `*` : multiplicación.
- `/` : división.
- `%` : módulo.

Operadores Aritméticos

- `++` : incremento *(Forma pre y posfija)*.

```
x++; // equivale a x = x + 1;
y = x++; // equivale a y = x; x = x + 1;
++x; // equivale a x = x + 1;
y = ++x; // equivale a x = x + 1; y = x;
```

- `--` : decremento *(Forma pre y posfija)*.

```
x--; // equivale a x = x - 1;
y = x--; // equivale a y = x; x = x - 1;
--x; // equivale a x = x - 1;
y = --x; // equivale a x = x - 1; y = x;
```

Operadores Relacionales

- == : igual.
- != : distinto.
- > : mayor.
- >= : mayor o igual.
- <= : menor o igual.
- < : menor.
  - `Recordar`: El resultado de una comparación es un entero, donde el cero denota falso y otro valor denota verdadero.

Operadores Lógicos

- `&&` : y.
- `||` : o.
- `!` : no.

Operadores de Asignación

- = : Operador de asignación.

- += :

```
x += y; // equivale a x = x + y;
```

- -= :

```
x -= y; // equivale a x = x - y;
```

- /= :

```
x /= y; // equivale a x = x / y;
```

- *= :

```
x *= y; // equivale a x = x * y;
```

- %= :

```
x %= y; // equivale a x = x % y;
```

Operador Condicional

- `?` : Es un operador ternario utilizado para escribir expresiones condicionales. El formato es:

```
exp1 ? exp2 : exp3
```

  - Si `exp1` es cierta la expresión completa evalúa al valor de `exp2`.
  - Si `exp1` es falsa, la expresión completa evalúa al valor de `exp3`.

```
valor = (x > 0) ? 1 : -1;

/* equivale a:
 si x > 0 entonces valor = 1 sino valor = -1 */
```

Otros Operadores

- `&` y `*` : operadores unarios utilizados para el manejo de punteros.
- `sizeof` : es un operador unario que devuelve el tamaño que ocupa una variable o un tipo de datos especificado.
- `(tipo)` : es un operador unario que realiza un conversión explicita del tipo de datos de una expresión.
- `.` y `->` : acceso a un campo de una estructura (registro).
- `,` : separador de evaluación de expresiones.

Otros Operadores

- `( y )` : permiten alterar el orden de evaluación por defecto. Paso de parámetros a una función.
- `[ y ]`: acceso a elementos de un arreglo.
- `& | ^ ~ << >>` : operadores a nivel de bit
- `&= ^= |= <<= =>>` : Operadores de asignación a nivel de bit.

Función printf()

- Utilizada para escritura de datos, su formato resumido es:

```
int printf (formato, argumentos);
```

  - Definida en el archivo `stdio.h`
  - `formato`: es una cadena que describe cómo mostrar la información.
  - `argumentos`: son las variables o expresiones a escribir.
  - En `formato` pueden aparecer:
    - constantes de cadena de carácter o
    - descriptores de formato, que indican el formato con el que se mostrarán los argumentos.

Función printf()

- Algunos descriptores de formato son:
  - `%c` : carácter sencillo.
  - `%d` o `%i` : entero.
  - `%e` : punto flotante en notación científica.
  - `%f` : punto flotante.
  - `%g` : de acuerdo al valor del exponente usa `%e` o `%f`.
  - `%o` : octal.
  - `%x` o `%X` : hexadecimal.
  - `%s` : cadena de caracteres.
  - `%u` : entero sin signo.

- Para otros tipos de valores enteros o de punto flotante podemos utilizar:
  - `%ld, %li, %lo, %lx, %lu`
  - `%lld, %lli, %llo, %llx, %llu`
  - `%Le, %Lf, %Lg`
  - `%hi, %hd, %ho, %hx, %hu`

Función printf()

```
#include <stdio.h>

int main (void) {

     int x=3;
     float y=3.3;
     char c ='A';

     printf("Ejemplo de uso de Printf!\n");
     printf("x  vale %d\n", x);
     printf("y vale %f,\n.., c vale %c.\n", y, c);

     return 0;
}
```

- `#include` le indica al preprocesador de C que tiene que agregar el archivo indicado a continuación de dicha sentencia al programa.
  - Semejante a las instrucción `uses` en Pascal

Función scanf()

- Utilizada para lectura de datos, su formato resumido es:

```
int scanf (formato, argumentos);
```

  - Definida en el archivo `stdio.h`
  - `formato`: igual que en la función `printf()`.
  - `argumentos`: son las variables o expresiones a leer.
  - Los `argumentos` que sean de tipo dato-resultado o resultado y sean de tipos escalares, deben llevar delante el operador `&`:
  - `&` : pasa la dirección de la variable y no su valor.
  - En el caso de las cadenas de caracteres no se utiliza `&`.

Función scanf()

```
#include <stdio.h>

int main (void) {

     int x, y;

     printf("Ingrese un valor: ");
     scanf("%d", &x);
     printf("Ingrese otro valor: ");
     scanf("%d", &y);

     printf("La división de los valores es: %f\n", (float) x / y);
     return 0;
}
```

Funciones getchar() y putchar()

- `getchar()`: Lee un carácter de la entrada estándar (teclado), su formato resumido es:

```
int getchar(void)
```

- `putchar()`: Escribe un carácter en la salida estándar (pantalla), su formato resumido es:

```
int putchar(caracter)
```

  - Están definidas en el archivo `stdio.h`

Funciones getchar() y putchar()

```
#include <stdio.h>

int main (void) {
     int c;
     while ( (c = getchar ()) != 'A' )
          putchar (c);
     return 0;
}
```

Sentencia Condicional

- La forma general de la sentencia `if` es:

```
if (condición)
    sentencia
```

  - Si la `condición` es verdadera *(distinta de cero)* se ejecuta `sentencia`.
  - La `condición` debe estar entre paréntesis.

Sentencia Condicional

- La forma general de la sentencia `if-else` es:

```
if (condición)
    sentencia1
else
    sentencia2
```

  - Si la `condición` es verdadera se ejecuta `sentencia1`, caso contrario, se ejecuta `sentencia2`.

  - Se permite la existencia de `if` anidados.

Sentencia Condicional

```
#include <stdio.h>

int main(void) {
     int a = 0;

     printf("Ingrese un valor: ");
     scanf( "%d", &a);

     if ( a > 0) {
          printf("El valor ingresado es mayor a cero.\n");
     } else {
          printf("El valor ingresado es menor o igual a cero.\n");
     }
     return 0;
}
```

Sentencia Repetitiva (while)

- La forma general de esta sentencia es:

```
while ( condición )
    sentencia
```

  - La `sentencia` se ejecuta una y otra vez mientras la `condición` sea cierta.

Sentencia Repetitiva (while)

```
#include <stdio.h>

int main (void) {

     int v = 1;

     while ( v < 5 ) {
          printf("%d al cubo es %d\n", v, v * v * v);
          v++;
     }

     return 0;
}
```

Sentencia Repetitiva (do-while)

- La forma general de esta sentencia es:

```
do
    sentencia
while ( condición )
```

  - La diferencia entre esta sentencia repetitiva y la anterior radica que nos aseguramos que la misma se ejecute al menos una vez.

Sentencia Repetitiva (do-while)

```
#include <stdio.h>

int main (void) {

     int valor;

     do {
          printf("Ingrese un valor entero (0 para salir) ");
          scanf("%d", &valor);
          printf("El valor ingresado es: %d\n", valor);
     } while (valor != 0);

     return 0;
}
```

Sentencia Repetitiva (for)

- La forma general de esta sentencia es:

```
for (inicial; condición; paso)
    sentencia
```

  - `inicial` se ejecuta antes de entrar en el bucle.
  - Si la `condición` es cierta, se ejecuta `sentencia` y después `paso`.
  - Luego se vuelve a evaluar la `condición`, y así se ejecuta la `sentencia` una y otra vez hasta que la `condición` sea falsa.

Sentencia Repetitiva (for)

```
#include <stdio.h>

int main(void) {

     int i, v;

     printf("Ingrese un valor entero: ");
     scanf("%d", &v);

     for (i = 0;  i <= 10; i ++) {
          printf("%d por %d es %d\n",v, i, v * i);
     }

     return 0;
}
```

break, continue y goto

- `break`
  - Se usa para salir de una sentencia `while`, `do-while`, `for` o `switch`.
  - Si se ejecuta se sale del bucle más interno o de la sentencia `switch` que se esté ejecutando.

- `continue`
  - Se interrumpe el ciclo actual de un bucle. Pasando a ejecutarse el siguiente ciclo del bucle.
  - Se puede usar en las sentencias `while`, `do-while` y `for`.

- `goto`
  - Equivalente a su análoga en Pascal, su formato es:

```
goto etiqueta;
```

  - Las etiquetas se definen de la siguiente forma:

```
etiqueta:
```

Sentencia Condicional Múltiple

- Se utiliza para ejecutar acciones diferentes según el valor de una expresión. La forma general de esta sentencia es:

```
switch (expresión) {
    case expresión1: sentencias;
    case expresión2: sentencias;
    ...
    default: sentencias;
}
```

  - La `expresión` se evalúa y si su valor coincide con el valor de alguna `expresión` indicada en los `case` se ejecutan todas las sentencias que le siguen.
  - Las expresiones deben ser de tipo entero o carácter.
  - Si el valor de `expresión` no se encuentra en la lista `case` se ejecutan las sentencias correspondientes a la opción `default`, si ésta no existe se continúa con la sentencia situada a continuación de `switch`.

Sentencia Condicional Múltiple

```
#include <stdio.h>

int main(void) {
     int a ;
     printf("Ingrese día de la semana: ");
     scanf("%d", &a);
     switch (a) {
          case 1:
          case 2:
          case 3:
          case 4:
          case 5:
               printf("Día laboral.\n");
               break;
          case 6:
          case 7:
               printf("Día no laboral.\n");
               break;
          default:
               printf("Error.\n");}
}
```

Introducción al Lenguaje C

Introducción al Lenguaje y Estructuras de Control

Claudio Omar Biale
Facultad de Ciencias Exactas Químicas y Naturales
Universidad Nacional de Misiones
09/08/2012 y 10/08/2012