Fundamentos de Programación

Bloque Práctico

Programación en lenguaje C

Índice de contenidos

Descripción del lenguaje de programación C

Descripción del lenguaje de programación C

- El lenguaje C fue diseñado por Dennis Ritchie como un lenguaje de propósito general en los laboratorios Bell en torno al año 1969

- Dichos laboratorios son los que produjeron las primeras versiones del sistema operativo UNIX

Descripción del lenguaje de programación C

- Su origen se debe a la necesidad de reescribir el sistema operativo Unix en una lenguaje de más alto nivel que el lenguaje ensamblador o máquina
- Debido a ello, C puede emplearse tanto para proyectos de bajo nivel (librearías, drivers, …) como para proyectos de alto nivel (utilidades de usuario, programas demonio, …)
- Es un lenguaje compilado, en el que los programas escritos en código fuente son transformados en código objeto binario y en ejecutables con la ayuda de un programa compilador (compilador) y un programa enlazador (linker)
- El tiempo que ha pasado desde su aparición inicial ha hecho que exista un gran número de compiladores para este lenguaje y que el código binario producido hoy en día esté muy optimizado

Descripción del lenguaje de programación C

- Dentro de los paradigmas de programación, el lenguaje C se enmarcaría en la familia de los lenguajes de programación estructurados y modulares

- En 1989, se adoptó el primer estándar (ANSI C) sobre este lenguaje. Posteriormente se han ido publicado revisiones del mismo hasta el día de hoy

  http://www.open-std.org/jtc1/sc22/wg14/www/projects#9899

  - Revisión C89 (1989 – **ANSI C**)
  - Revisión C99 (2000)
  - Revisión C11 (2011)
  - Revisión C2x (por publicar)

- La adopción de estándares es importante, porque garantiza que aunque sea un lenguaje compilado, si se siguen, se asegura la portabilidad de los programas

Descripción del lenguaje de programación C

- Actualmente hay varios proyectos de compiladores que nos permiten desarrollar programas escritos en C para multitud de plataformas. Algunos de ellos:

  - GCC (GNU C Compiler - https://gcc.gnu.org/). Compilador con licencia opensource. Es uno de los más extendidos. Parte importante del sistema operativo UNIX/Linux

  - Embarcadero (Borland) C/C++ Compile

    https://www.embarcadero.com/es/free-tools/ccompiler

    Uno de los compiladores más notables. Proviene de la antigua empresa Borland especializada en herramientas de programación

Descripción del lenguaje de programación C

- Clang (http://clang.llvm.org/). Otro compilador con licencia de código abierto. Apoyado por grandes empresas como Google y Apple

- Intel C/C++ Compiler

  https://software.intel.com/content/www/us/en/develop/tools/compilers/c-compilers.html

  Compilador no libre producido por Intel. Muy apreciado en entornos y proyectos donde se ha de producir un código objeto y unos ejecutables muy optimizados para plataformas basadas en el hardware de esa compañía.

Descripción del lenguaje de programación C

- Junto con el compilador, la otra pieza importante es el editor o IDE. Para C tenemos muchos y muy buenos. Algunos de los más empleados:
  - Sublime Text / Atom / Visual Studio Code
  - Codelite
  - CodeBlocks
  - Xcode
  - Netbeans
  - Eclipse (CDT)
  - CLion
  - Kdevelop
  - Anjuta IDE
  - QT Creator
  - vim / emacs ☺

Estructura de un programa en C

Estructura de un programa en C

- Un programa en C es un fichero o conjunto de ficheros de texto plano con código escrito en ese lenguaje de programación.

- Los ficheros que forman parte de un programa en C pueden ser de dos tipos:

  - Archivos de código fuente. Tienen extensión .c
  - Archivos de cabecera. Tienen extensión .h

- Los archivos de código fuente contienen principalmente la implementación de funciones que forman parte del programa

- Los archivos de cabecera suelen contener constantes, definiciones de tipos de usuario y declaración de funciones

Estructura de un programa en C

- Un programa en C que se vaya a transformar en un ejecutable debería contener al menos una función obligatoria llamada main()
- Esta función representa el algoritmo principal del programa
- Si el programa es grande, otro tipo de funciones podrían existir, pero siempre se empezará a ejecutar el programa por la primera línea de código de la función main()
- Como todas las funciones, main() puede devolver un valor (usualmente un número entero). Esta es la única función cuyo valor se devuelve al exterior del programa (tradicionalmente al sistema operativo para indicar si ha habido error o no)

Estructura de un programa en C

```
#include <stdio.h>
#define  N  100

int main(int argc, char *argv[])
{
   int i;

   for(i=0;i<N;i++) {
      printf("%d\n", i);
   }

   return 0;
}
```

- Este ejemplo es un programa que imprime por pantalla los 100 primeros números, desde el 0 al 99

- Consta de varias partes y elementos diferenciados:

Estructura de un programa en C

```
#include <stdio.h>                  ← Directivas para el preprocesador
#define  N  100

int main(int argc, char *argv[])    ← Función main con argumentos
{
   int i;                           ← Declaración de una variable

   for(i=0;i<N;i++) {               ← Estructura repetitiva (bucle)
      printf("%d\n", i);            ← Función para mostrar en pantalla
   }

   return 0;                        ← La función main devuelve un 0 al exterior
}
```

- El programa ha de compilarse. La compilación pasa por varias etapas (de forma simplificada): preprocesador → compilación → enlazado
- El preprocesador limpia y transforma el código antes de la compilación

Estructura de un programa en C

- El compilador transforma las instrucciones en un lenguaje de alto nivel como es C en otras que entiende la máquina (código ensamblador)
- Una vez que está transformado a ensamblador, se convierte a código objeto (código binario que ya puede ser llevado al hardware de la CPU, memoria, …)
- Finalmente, si ha de obtenerse un ejecutable el enlazado “pega” al código objeto todas las librerías necesarias para el correcto funcionamiento del programa y genera el ejecutable en el formato establecido para la plataforma

Estructura de un programa en C

```
#include <stdio.h>
#define  N  100

int suma(int a, int b, int c);                ← Declaración de una función

int main(int argc, char *argv[])
{
   int resultado;

   resultado = suma(2, 5, 6);                 ← Llamada a la ejecución de una función
   printf("a + b + c = %d\n", resultado)

   return 0;
}

int suma(int a, int b, int c) {               ← Implementación de una función
   return a + b +c;
}
```

- Los programas complejos pueden contener más de una función

Variables , constantes y operadores

Variables, constantes y operadores

- C es un lenguaje tipado. Todas las variables y constantes deben tener asociado un tipo válido.

- Los tipos van a afectar al espacio de almacenamiento, valores mínimos y máximo y operadores de esas constantes o variables.

- Los tipos predefinidos en C son (y por tanto palabras reservadas)

  - Números enteros: char, short, int, long, long long (> C99)
    - variante sin signo:
      unsigned char, unsigned char int, unsigned int, unsigned long, unsigned long long
  - Números reales: float, double, long double (varía según plataforma)
  - Números complejos (> C99): float _Complex, double _Complex, long double _Complex
  - Booleano (> C99): _Bool o bool (con stdbool.h)

Variables, constantes y operadores

Tipo char

- Se usa para almacenar números pequeños o caracteres individuales (porque se almacena su código ASCII)
- Ocupa 8 bits en memoria y el rango de valores es [-128,127] con signo y [0,255] sin signo
- El formato para imprimir por pantalla es %c . Ej: printf("%c", 'B');

Variables, constantes y operadores

Tipo short

- Se usa para almacenar números pequeños
- Ocupa 16 bits en memoria y el rango de valores es [−32767, +32767] con signo y [0, 65535] sin signo
- En algunas implementaciones no estándares (Windows API) recibe el nombre de WORD
- El formato para imprimir por pantalla es %hd o %hu.
- Útil para almacenar ciertas codificaciones de caracteres (ej: parte del UTF-8)

Variables, constantes y operadores

Tipo int

- Se usa para almacenar números enteros
- Ocupa 32 bits en memoria y el rango de valores es [−2147483647, +2147483647] con signo y [0, 4294967295] sin signo
- En algunas implementaciones no estándares (Windows API) recibe el nombre de DWORD
- El formato para imprimir por pantalla es %d o %u.
- Es el más empleado en todos los programas

Variables, constantes y operadores

Tipo long

- Se usa para almacenar números enteros grandes
- Ocupa 32 bits en memoria y el rango de valores es [−2147483647, +2147483647] con signo y [0, 4294967295] sin signo
- El formato para imprimir por pantalla es %ld o %lu.

Variables, constantes y operadores

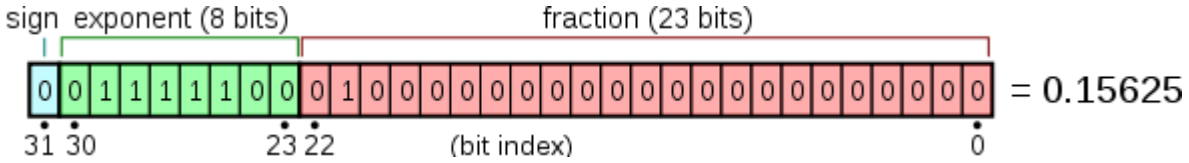
Tipo long long (requiere compilador C99 compatible)

- Se usa para almacenar números enteros enormes
- Ocupa 64 bits en memoria y el rango de valores es [−9223372036854775807, +9223372036854775807] con signo y [0, +18446744073709551615] sin signo
- El formato para imprimir por pantalla es %lld o %llu.
- Muy propio de aplicaciones informáticas en Ciencias (Física, Matemáticas, Química, Astronomía), Ingenierías o Finanzas

Variables, constantes y operadores

Tipo float

- Se usa para almacenar números reales. Sigue el estándar IEEE 754



- Ocupa 32 bits en memoria y el rango de valores es (excluyendo el 0)

  $[-3.40282347e+38, -1.175494351e{-}38] \cup [-1.175494351e{-}38, 3.40282347e+38]$

- El formato para imprimir por pantalla es %f o %e o %g.

- Hay que tener mucho cuidado con las conversiones entre este tipo y los enteros, ya que la representación interna es distinta

Variables, constantes y operadores

Tipo double

- Se usa para almacenar números reales. Sigue el estándar IEEE 754 (64 bits)
- Ocupa 64 bits en memoria y el rango de valores es (excluyendo el 0)

$$[-1.7976931348623157e+308, -2.2250738585072014e-308] \cup [2.2250738585072014e-308, 1.7976931348623157e+308]$$

- El formato para imprimir por pantalla es %lf o %le o %lg.
- Hay que tener mucho cuidado con las conversiones entre este tipo y el tipo float, por la pérdida de precisión

Variables, constantes y operadores

- C puede declarar variables de todos los tipos anteriores.
- Las variables se declaran o bien fuera de toda función (variables globales – no recomendadas) o al principio de cada función o bloque
- La forma de declarar una variable es una de éstas:

  [tipo] [identificador];
  [tipo] [identificador] = <valor>;

- Ejemplos

```
unsigned char letra = ‘Z’;
int numero = 12345, numero2 = -45;
float x = 1.34F;
double y = 123.45e5
```

Variables, constantes y operadores

- Las constantes son similares a las variables solo que son valores que no pueden cambiar.
- Hay dos maneras: mediante la directiva #define (constante simbólica) o mediante la palabra reservada const
- Ejemplos

```
#define PI  3.14159

const float PI = 3.14159F;
```

- En el primer caso es un símbolo que será reemplazado por el compilador. En el segundo es más similar a una variable cuyo valor no puede ser alterado.

Variables, constantes y operadores

- Cada tipo de variable tiene sus propios operadores. Los principales operadores son los de tipo aritmético, relacionales y lógicos

Operadores aritméticos

Operador	Símbolo	Ejemplo	Notas
Suma	+	x + y	
Resta	-	x - y	
Multiplicación	*	x * y	
División	/	x / y	Entre enteros es el cociente. y <> 0
Cambio de signo	-	-x	

- Para operaciones matemáticas más complejas (por ejemplo exponenciación) ha de recurrirse a la librería de funciones matemáticas math.h

Variables, constantes y operadores

Operadores relacionales (de comparación)

Operador	Símbolo	Ejemplo	Notas
Igualdad	==	x == 2	Devuelve 0 o distinto de 0
Desigualdad	!=	X != 2	
Menor	<	x < y	
Mayor	>	x > y	
Menor o igual	<=	x <= y	
Mayor o igual	>=	x >= y	

- Como no existe el tipo booleano en ANSI C, la comparación siempre devolverá un entero con valor 0 o 1

Variables, constantes y operadores

Operadores lógicos y de bit

Operador	Símbolo	Ejemplo	Notas
and	&&	x && y	1 si x e y son distintos de 0. 0 en caso contrario
or	||	x || y	0 si x e y son iguales a 0. 1 en caso contrario
not	!	!x	Negación lógica
xor	^	x ^ y	
Menor o igual	<=	x <= y	
Mayor o igual	>=	x >= y	
and a nivel de bit	&	x & y	Se hace la operación con cada bit de x e y
or a nivel de bit	|	x | y	Se hace la operación con cada bit de x e y
Desplazamiento der.	>>	x >> 2	
Desplazamiento izq.	<<	x << 2	

Variables, constantes y operadores

Operadores de autoasignación

Operador	Símbolo	Ejemplo	Notas
Autoincremento	++	x++ o ++x	x = x + 1. Prefijo: primero incrementar y luego usar
Autodecremento	--	x-- o --x	x = x - 1. Prefijo: primero incrementar y luego usar
Incremento y asig.	+=	x += 2	x = x + 2
Decremento y asig.	-=	x -= 2	x = x - 2
	*=	x *= 2	x = x * 2
	/=	x /= 2	x = x / 2
	>>=	x >>= 2	x = x >> 2
	<<=	x <<= 2	x = x << 2

Variables, constantes y operadores

Otros operadores

Operador	Símbolo	Ejemplo	Notas
Dirección de	&	&x	Devuelve la dirección de memoria de x
Indirección	*	*x	Devuelve el contenido en memoria de la dir. x
Tamaño de	sizeof	sizeof(x)	Devuelve el tamaño en bytes de x
Agrupación	()	(x+y)*2	Agrupa expresión para evaluarla primero
Índice	[]	valores[2]	En arrays se usa para acceder a una posición

- Todos estos operadores se pueden utilizar en expresiones.
- Los operadores implican una precedencia que conviene conocer para evaluar correctamente expresiones. Ej: () → * / → + -

Entrada y salida estándar

Entrada y salida estándar

- Se conoce como salida estándar a la posibilidad de mandar información a la pantalla de la computadora. Se representa en muchos lenguajes por el stream o manejador stdout

- Se conoce como entrada estándar a la posibilidad de recoger información por el teclado de la computadora. Se representa en muchos lenguajes por el stream o manejador stdin

- En el lenguaje de programación C, para trabajar con entrada o salida estándar hay que importar la librería stdio.h mediante una directiva de preprocesador de tipo include

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

Salida estándar

- Función printf. Imprime por pantalla con un determinado formato

```
printf("%c", 'a');
printf("%d\n", 345);
printf("%f", 3.45);
printf("%4.2f", 2.4567);
printf("%s", "Hola");
```

- Función putc. Imprime un carácter

```
putc('Z')
```

- Función puts. Imprime una frase almacenada en una cadena

```
puts("Hola mundo");
```

Entrada estándar

- Función scanf. Lee por teclado con un determinado formato

```
scanf("%d", &numero);
scanf("- %d\n", &numero);     Ej: - 123
scanf("%f", &peso);
```

- Función getc. Lee un carácter

```
ch = getc()
```

- Función gets. Lee una frase y la almacena en una cadena

```
gets(mensaje);
```

Ejercicios prácticos

1. Realizar un programa que imprima por pantalla los tipos básicos de C

2. Realizar un programa que imprima los 5 primeros números. Primero uno en cada fila y luego en una única fila separados por espacios

3. Realizar un programa que use los operadores aritméticos

4. Realizar un programa que utilice los operadores de autoasignación

Estructura selectiva simple

```
if( condición )
{
   …
}
```

- Se evalúa la condición. En ANSI C, cualquier valor entero distinto de 0 hará que se cumpla y se ejecutará el bloque de sentencias entre llaves { }
- La condición puede ser compleja, componiendo con los operadores and (&&), or (||) y not (!)

Estructura selectiva doble

```
if( condición )
{
   …
}
else
{
  …
}
```

- Se evalúa la condición. Si se cumple se ejecutará el primer bloque de sentencias. Si no se cumple se ejecutará el segundo bloque

- La condición puede ser compleja, componiendo con los operadores and (&&), or (||) y not (!)

Estructura selectiva múltiple (1ª forma)

```
if( condición )
{
   …
}
else if ( condición )
{
  …
}
…
else {
  …
}
```

- Se evalúa la condición. Si se cumple se ejecutará el primer bloque de sentencias. Si no se cumple se evalúa la segunda condición y así sucesivamente

- La condición puede ser compleja, componiendo con los operadores and (&&), or (||) y not (!)

Estructura selectiva múltiple (2ª forma)

```
switch(variable) {
   case <valor1>: <bloque sentencias> [ break; ]
   case <valor2>: <bloque sentencias> [ break; ]
   case <valor3>: <bloque sentencias> [ break; ]
   …
   default: <bloque sentencias>
}
```

- Se evalúa el valor de una variable. Si es igual a <valor1> se ejecuta el bloque de sentencias hasta encontrar un break, o se pasa a la siguiente condición
- La variable ha de ser de tipo entero

Estructura repetitiva 0+ veces

```
while( condición ) {
   <sentencias>
}
```

- Se evalúa la condición al principio. Si se cumple se ejecuta el bloque de sentencias. Al terminarlo, se vuelve al principio y se evalúa nuevamente la condición.

- Cuando se regula mediante un índice entero, la forma que suele adoptar en el caso de un bucle con contador creciente es:

```
i = INF
while( i < SUP ) {
   <sentencias>
   i = i + INC
}
```

Estructura repetitiva 1+ veces

```
do {
   <sentencias>
} while (condición)
```

- Se evalúa la condición al final. Si se cumple se ejecuta el bloque de sentencias otra vez.

- Cuando se regula mediante un índice entero, la forma que suele adoptar en el caso de un bucle con contador creciente es:

```
i = INF
do {
   <sentencias>
   i = i + INC
} while (i < SUP)
```

Estructura repetitiva N veces

```
for(i=INF; i<SUP; i = i + INC) {
   <sentencias>
}
```

- Se evalúa la condición al principio. Si se cumple se ejecuta el bloque de sentencias otra vez.
- La parte i = INF solo se ejecuta una vez
- La parte i<SUP (comparación) y i = i + INC (asignación) se ejecutan tantas veces como vueltas (iteraciones) de el bucle
- El bucle podría ser decreciente su INF > SUP y el valor de INC negativo (resta en vez de suma)

Arrays unidimensionales

```
#define N 10

int main() {
   int array[N];
   int i;

   /* Inicialización manual uno por uno */
   array[0] = 1;
   …
   array[N-1] = 1;

   /* Inicialización mediante bucle */
   for(i=0;i<N;i++) {
      array[i] = 1;
   }
}
```

Arrays bidimensionales

```
#define N 10

int main() {
   int matriz[N][N];
   int i, j;

   /* Inicialización manual uno por uno */
   matriz[0][0] = 1;
   …
   matriz[N-1][N-1] = 1;

   /* Inicialización mediante bucles */
   for(i=0; i<N; i++) {
      for(j=0; j<N; j++) {
         matriz[i][j] = 1;
      }
   }
}
```

Registros

```
struct punto2D {
   float x;
   float y;
}

int main() {
   struct punto2D p;

   p.x = 1.0F;
   p.y = 2.0F;
}
```

Cadenas de caracteres

```
#define N 80

int main() {
   char frase[N];

   /* Lectura por teclado */
   gets(frase);

   /* Imprimir por pantalla */
   printf("%s\n", frase);

   /* Acceso a las letras */
   frase[0] = frase[0] – 'a' + 'A'          /* Paso a mayúsculas la primera letra */
}
```

Cadenas de caracteres

- Cuando se trabaja con cadenas de caracteres es habitual usar las funciones definidas en <string.h>

  strcpy(cadena1, cadena2) – copia cadena2 en cadena1
  strcat(cadena1,cadena2) – concatena cadena2 en cadena1
  strlen(cadena) – cuenta el numero de letras de cadena

UEMC

Universidad Europea
Miguel de Cervantes

la universidad en persona

uemc.es