95.11 Algoritmos y Programación I (Electrónica)

Repaso de C

Sebastián Santisi

21 de marzo de 2019

Tipos

En C hay dos grandes grupos de tipos:

Enteros:

- Almacenan valores numerables
- Están limitados por la cantidad de bits de representación (2^b)
- Recupero el mismo valor que guardo
- Pueden irse de rango

Flotantes:

- Almacenan valores con decimales
- Utilizan una representación similar a la científica (mantisa y exponente)
- Guardan magnitudes, no valores exactos

Tipos enteros: Tamaño

Bits	Sin signo	Con signo	Cant.
8	$[0, 255]$	$[-128, +127]$	256
16	$[0, 65535]$	$[-32768, +32767]$	64K
32	$[0, 4294967295]$	$[-2147483648, +2147483647]$	4G
64	$[0, 18446744073709551615]$	$[-9223372036854775808, +9223372036854775807]$	16E
n	$[0, 2^n - 1]$	$[-2^{n-1}, +2^{n-1} - 1]$	

Capacidad según número de bits

Tipos enteros: Tamaño

Bits	Sin signo	Con signo	Cant.
8	$[0, 255]$	$[-128, +127]$	256
16	$[0, 65535]$	$[-32768, +32767]$	64K
32	$[0, 4294967295]$	$[-2147483648, +2147483647]$	4G
64	$[0, 18446744073709551615]$	$[-9223372036854775808, +9223372036854775807]$	16E
n	$[0, 2^n - 1]$	$[-2^{n-1}, +2^{n-1} - 1]$	

Capacidad según número de bits

Tipo	Bytes	Bits
`char`	1	8
`short`	2	16
`int`	4	32
`long`	8	64
`long long`	8	64

Ejemplo para GCC en 64 bits

Tipos flotantes

- No dependen de la arquitectura, definidos en IEEE-754
- `float`: 32 bits, 23 bits de mantisa por lo que representa $\log_{10}(2^{23}) \approx 7$ dígitos decimales
- `double`: 64 bits, 53 bits de mantisa, $\sim$16 dígitos decimales
- Existen los números IEEE-754 de cuádruple precisión (128 bits, 112 mantisa, $\sim$34 decimales) pero aún hay poco hardware que lo soporte
- Si bien existe `long double` en C99 no hay garantía de que sea mayor que `double` ni que resuelva sobre hardware

Tamaño de un tipo o variable

- Podemos consultar el tamaño de un tipo o variable (en bytes) con el operador `sizeof()` (eg. `sizeof(int)`)
- Para flotantes son los especificados por IEEE-754
- Para enteros sólo está especificado por el estándar `sizeof(char)== 1`
- Los demás son arbitrarios según el compilador y la plataforma
- Pero `sizeof(char)<= sizeof(short)<= sizeof(int)<= sizeof(long)<= sizeof(long long)`
- En `<stdint.h>` se definen tipos de tamaño fijo `int8_t`, `uint8_t`, `int16_t`, etc. para aplicaciones de bajo nivel

Tipos enteros: Signo

- Los tipos enteros pueden ser `signed` o `unsigned`
- Por omisión los tipos `short`, `int`, `long` y `long long` son siempre signados
- El tipo `char` (a secas) sirve para almacenar caracteres
- Si necesitamos una variable numérica de 1 byte podemos definir un `unsigned char` o `signed char` según corresponda

Literales

- En C todos los literales tienen tipo
- Por lo general no le prestamos mayor atención al tema, a menos que importe
- Ejemplos: `0`, `0U`, `0L`, `6.4`, `4.2F`, `'a'`

Límites

En `<limits.h>` encontramos:

- `CHAR_BIT`
- `INT_MAX`
- `INT_MIN`
- `UINT_MAX`
- etc.

(Hay una `<float.h>` análoga para flotantes.)

Elijamos tipos de variables

Si quisiéramos representar...

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`
- ... la temperatura del aula

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`
- ... la temperatura del aula: `float`

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`
- ... la temperatura del aula: `float`
- ... el dinero en una cuenta bancaria

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`
- ... la temperatura del aula: `float`
- ... el dinero en una cuenta bancaria: `long`

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`
- ... la temperatura del aula: `float`
- ... el dinero en una cuenta bancaria: `long`
- ... un factorial

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`
- ... la temperatura del aula: `float`
- ... el dinero en una cuenta bancaria: `long`
- ... un factorial: `double`

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`
- ... la temperatura del aula: `float`
- ... el dinero en una cuenta bancaria: `long`
- ... un factorial: `double`
- ... el valor de π

Elijamos tipos de variables

Si quisiéramos representar...

- ... la edad de una persona: `unsigned char`
- ... la temperatura del aula: `float`
- ... el dinero en una cuenta bancaria: `long`
- ... un factorial: `double`
- ... el valor de π: `double` (o `float`)

Declaración y definición de variables

- Cuando "declaramos" una variable le pedimos al compilador que reserve un espacio de memoria de un determinado tipo y le ponga un nombre
- La declaración de una variable no inicializa a la misma
- Cuando "definimos" una variable le asignamos un valor

```
int a, b;          // Declaramos a y b enteras
a = 5;             // Definimos el valor de a
float g = 9.81; // Declaramos y definimos a g
```

Conversiones de tipos

- Siempre se opera entre operandos del mismo tipo
- Si los operandos son distintos el *más chico* "se promueve" al tipo del *más grande*
- Si el resultado de una operación es *más grande* que el de la variable donde se almacena el mismo "se trunca"
- En principio no hay pérdida de información al promover pero puede haberla al truncar
- Es responsabilidad del programador asegurarse que algo quepa en una variable antes de truncarlo
- Puede imponerse el tipo de una variable utilizando un "casteo" (eg. `(int)peso` fuerza a tratar el valor de `peso` como si fuera de tipo `int` para esa expresión)

Redefinición de tipos

- Muchas veces queremos abstraernos de los tipos de C y "crear" nuevos
- Para esto utilizamos `typedef viejo nuevo;`
- Por ejemplo, si quisiéramos tener un tipo nuevo para representar edades:

```
typedef unsigned char edad_t;
...
edad_t mayoria_de_edad = 18;
```

- Solemos usar el sufijo `_t` para los tipos nuevos

Operadores

Asignación: `=`

Aritméticos: `+`, `-`, `*`, `/`, `%`

Booleanos: `==`, `!=`, `!`, `&&`, `||`, `<`, `<=`, `>`, `>=`

Otros: Los veremos más adelante

(Los aritméticos pueden combinarse con la asignación: `a += b;` es lo mismo que `a = a + b;`, en ese caso no sólo opera sino que además modifican el valor de la variable.)

Incrementos y decrementos

- Tanto `a++` como `++a` son "equivalentes" a `a += 1` o `a = a + 1` (por lo tanto operan y además asignan)
- Si el "postincremento" está dentro de una sentencia se usa el valor anterior y luego se incrementa

```
int b, a = 5;
b = a++;
// b == 5, a == 6
```

- Si el "preincremento" está dentro de una sentencia se incrementa y luego se usa el valor

```
int b, a = 5;
b = ++a;
// b == 6, a == 6
```

- Son análogos los operadores `a--` y `--a`

Booleanos

- En C todo tipo entero es booleano con la convención `0` es falso y cualquier otro valor es verdadero
- Todos los operadores booleanos devuelven `0` para falso y `1` para verdadero
- El encabezado `<stdbool.h>` define el tipo `bool` con los valores `true` y `false`
- Si bien `false == 0`, `true == 1` por lo que no cualquier entero "verdadero" es el booleano `true` del tipo `bool`

Funciones

- Las funciones encapsulan subrutinas de código
- Se comunican a través de los parámetros que reciben y del valor que devuelven
- La instrucción `return` se utiliza para terminar una función (y para devolver su valor)
- Se utiliza la palabra `void` para indicar una función que no devuelve nada ("procedimiento")
- Ejemplo:

```
int sumar(int a, int b) {
    return a + b;
}
...
    int i = sumar(1, 2);
    float f = sumar(1.9, 2.9);
```

Prototipos

- El prototipo declara los parámetros de una función
- Le sirve al compilador para resolver la llamada de una función que todavía no conoce
- Es el encabezado de la función seguida de un punto y coma
- Ejemplos:

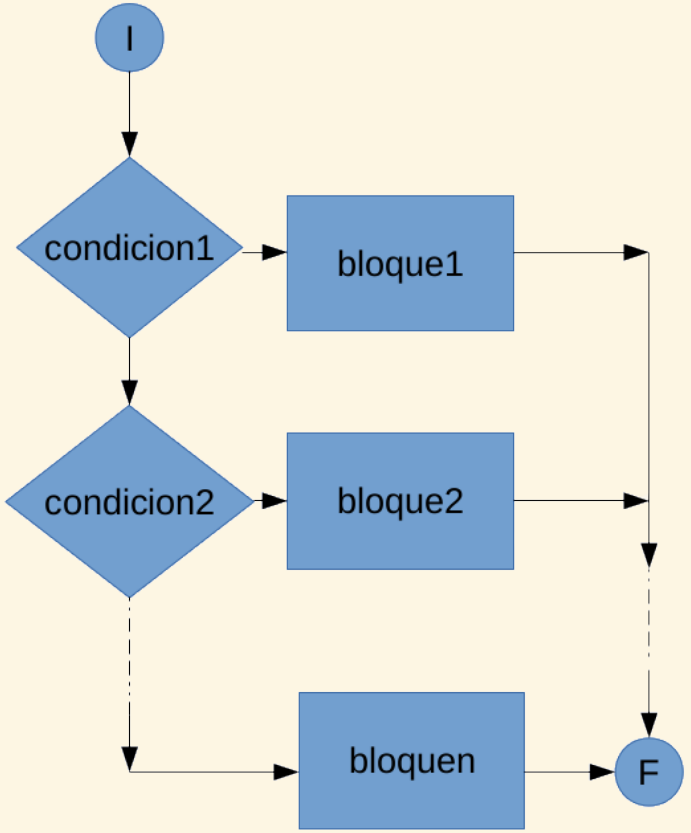
```
int sumar(int x, int y);
float celsius_a_fahrenheit(float temp);
int random();
void imprimir_angulo_en_gms(double rad);
```

Control: if

```
if(condicion1) {
    /* bloque1 */
}
else if (condicion2) {
    /* bloque2 */
}
/* ... */
else {
    /* bloquen */
}
```

Control: if

```
if(condicion1) {
    /* bloque1 */
}
else if (condicion2) {
    /* bloque2 */
}
/* ... */
else {
    /* bloquen */
}
```

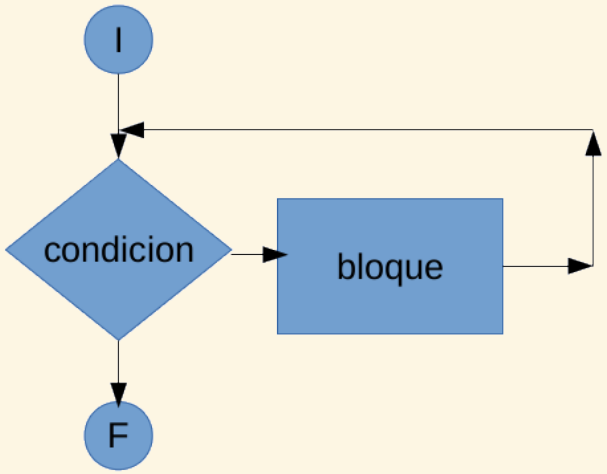


Control: while

```
while(condicion) {
    /* bloque */
}
```

Control: while

```
while(condicion) {
    /* bloque */
}
```



Control: for

```
for(inicializacion; condicion; incremento) {
    /* bloque */
}
```

Control: for

```
for(inicializacion; condicion; incremento) {
    /* bloque */
}
```