Programación en C/Referencia

Esta parte pretende ser una referencia completa de los elementos del lenguaje C. Se pretende que aquí se presenten todos los datos disponibles de los tipos de datos del lenguaje C, toda la información relevante de los operadores del lenguaje C, las formas en la que se deben usar la principales estructuras e información acerca de algunas bibliotecas y funciones que son estándares. Todo esto con la mayor calidad posible. O sea que en un momento aquí va a existir algo así como una super síntesis del manual de programación en C con los datos que son importantes pero que son fáciles de olvidar.

Tipos de datos

`char`

- bytes = 1
- Máximo = 127 (255 cuando es `unsigned`)
- Mínimo = -128 (0 cuando es `unsigned`)

`int`

- bytes = 4
- Máximo = 2,147,483,647 (4294967295 cuando es `unsigned`)
- Mínimo = -2,147,483,648 (0 cuando es `unsigned`)

`float`

- bytes = 4
- Máximo Exponente = 10^{37}
- Mínimo Exponente = 10^{-37}

`long`

- bytes = 4
- Máximo = 2,147,483,647 (4294967295 cuando es `unsigned`)
- Mínimo = -2,147,483,648 (0 cuando es `unsigned`)

`long long`

- bytes = 8
- Máximo = 9,223,372,036,854,775,807 (18,446,744,073,709,551,616 cuando es `unsigned`)
- Mínimo = 9,223,372,036,854,775,808 (0 cuando es `unsigned`)

`short`

- bytes = 2
- Máximo = 32767 (65,535 cuando es unsigned)
- Mínimo = -32768 (0 cuando es unsigned)

`double`

- bytes = 8
- Máximo Exponente = 10^{308}
- Mínimo Exponente = 10^{-308}

`void` -sin tipo-

Operadores

Nombre del Operador	Símbolo o representación	Orden de agrupación
Operador de miembro de estructura	`nombre.miembro nombre->miembro`	izquierda -> derecha
incremento, decremento, positivo, negativo, NOT lógico, NOT de bits, derreferenciación, dirección, cast, tamaño	`++, --, +, -, !, ~, *`*`puntero`*`, &`*`nombre`*`, (`*`tipo`*`)`*`valor`*`, sizeof(`*`variable`*`)`	derecha -> izquierda
Multiplicación, División, Módulo	`*, /, %`	izquierda -> derecha
Suma y resta	`+, -`	izquierda -> derecha
Desplazamiento de bits izquierda y derecha	`<<, >>`	izquierda -> derecha
Comparaciones: Mayor que, Mayor o igual que, Menor que, Menor o igual que	`>, >=, <, <=`	izquierda -> derecha
Comparaciones: Igual que, Diferente de	`==, !=`	izquierda -> derecha
AND para bits	`&`	izquierda -> derecha
XOR para bits	`^`	izquierda -> derecha
OR para bits	`|`	izquierda -> derecha
AND para comparación	`&&`	izquierda -> derecha
OR para comparacion	`||`	izquierda -> derecha
Expresiones de condiciones	*`expr1`* `?` *`expr2`* `:` *`expr3`*	derecha -> izquierda
Incrementar **c** en **n**, Decrementar **c** en **n**, Multiplicar **c** por **n**, Dividir **c** entre **n**	`c+=n, c-=n, c*=n, c/=n, ...`	derecha -> izquierda

Todos los operadores están puestos en el orden en que están en la jerarquía del lenguaje C. En caso de que estén en una expresión todos los operadores juntos y de que no se use el operador (), que rompe la precedencia, entonces van a ser evaluados desde el primero de la tabla hasta el último en ese orden. En caso de que se encuentren varios operadores iguales o de la misma precedencia en una expresión sin paréntesis se van a resolver en el orden en que diga la columna de "Orden de agrupación". La precedencia es una cosa importante porque nos dice en que orden va a ser resuelta una expresión y el orden en que se resuelve una expresión influye en su resultado.

```
un ejemplito:
1 + 2 / 3 + 4 no es igual a (1 + 2) / (3 + 4)
eso porque según la precedencia:
1 + 2 / 3 + 4 es igual a 1 + (2 / 3) + 4)
(2/3 = 0 por ser una operación entre enteros, 1 + 4 = 5)
(en el otro caso 1 + 2 = 3 y 3 + 4 = 7 y 3 / 7 = 0 por ser una operación entera)
```

Como la división tiene mayor precedencia que la suma se resuelve primero la división.

```
otro ejemplito:
72 / 3 / 2 / 6 no es igual a (72 / 3) / (2 / 6)
eso porque según la precedencia:
72 / 3 / 2 / 6 es igual a ((72 / 3) / 2) / 6
(La división no es una operación asociativa y por eso cambia el resultado si
resolvemos en orden diferente, no es lo mismo cuando usamos la precedencia del
lenguaje C como en la tabla de precedencias, que cuando obligamos a que se
resuelva la operación en otro orden usando paréntesis).
```

Para los operadores de Incremento (++) y decremento (−−) hay que tener en cuenta el sitio en el que se encuentran con respecto a la variable que están influenciando. **Con esto me refiero a que siendo `a` una variable `++a` no es igual que `a++`, aunque en algunas ocasiones parezca lo contrario**. Para el caso de `++a` el incremento será realizado antes de evaluar la expresión. Y para `a++` se evaluara primero la expresión completa y luego se realizara el incremento.

En la tabla hay tres operadores que están identificados con un calificativo "para bits". Con esto me refiero a que influenciaran los bits de una variable. Si tengo dos valores `a` y `b` el resultado de un `a && b` será el valor del AND lógico entre los valores respectivos pero el `a & b` tendrá como resultado varios ANDs lógicos individuales entre cada uno de los bits de las dos variables.

Estructuras

Estructura general de un programa de lenguaje C:

```
inclusión de bibliotecas
...
definición de macros y constantes
...
definiciones de funciones o/y prototipos
...
definiciones de variables globales
...
int main(argumentos){
   sentencias
   ...
   return 0;
}
definiciones de cuerpos de funciones (cuyos prototipos ya fueron declarados)
...
```

Estructura general de un header file de lenguaje C:

```
inclusión de bibliotecas
...
definición de macros y constantes
...
definiciones de funciones o/y prototipos
...
definiciones de variables globales
...
definiciones de cuerpos de funciones (cuyos prototipos ya fueron declarados)
```

```
...
```

Estructura de un bloque/sentencia:

```
{
   sentencia/s
}
```

En cualquier momento un bloque de sentencias puede sustituir a una sentencia.

Estructura general de una función de lenguaje C:

```
tipo_de_retorno nombre_de_la_funcion (tipo argumento1, tipo argumento2) {
   variables
   ...
   sentencia/s
   ...
   return valor_de_retorno;
}
```

Estructura del `if`:

```
if (condicion)
   sentencia/s
else if (condicion)
   sentencia/s
else
   sentencia/s;
```

Estructura del `switch`:

```
switch (valor) {
   case valor:
      sentencias
      ...
   break;
   case valor:
      sentencias
      ...
   break;
   ...
   default:
      sentencias
      ...
}
```

Estructura del `while`:

```
while(condicion) {
   sentencias
   ...
}
```

Estructura del `do - while`:

```
do {
   sentencias
   ...
}while(condicion);
```

Estructura del `for`:

```
for (iniciacion; condicion; incremento) {
   sentencias
   ...
}
```

Estructura de un `struct`:

```
struct {
   tipo variable;
   tipo variable;
   ...
}
```

Estructura de un `unión`:

```
union {
   tipo variable;
   tipo variable;
   ...
}
```

Bibliotecas y funciones

Cuando no se especifica un tipo de dato es porque la función permite mas de uno ya sea a través de casting o de otra maña. Esta parte como ya se explico antes es solo para mostrar el formato de las funciones.

Funciones de <stdio.h>:

```
int printf("cadena de formato", variable1, variable2);  //Muestra por pantalla
int scanf("cadena de formato", variable1, variable2);  //Lee de pantalla
```

Funciones de <stdlib.h>:

```
int system("llamada");
```

Funciones de <math.h>:

```
double sin(valor);  //Calcula el sinus
double cos(valor);  //Calcula el cosinus
double tan(valor);  //Calcula la tangente
double asin(valor);
double atan(valor);
double acos(valor);
double sinh(valor);
double cosh(valor);
double tanh(valor);
```

```
  double log10(valor);
  double log(valor);
  double ldexp(valor1,valor2);
  double pow(valor);
  double sqrt(valor);  //Calcula raíces cuadradas
```

Funciones de <time.h>:

```
  struct tm time();
```

Funciones de <string.h>:

```
  int strcmp(cadena1, cadena2); //Compara dos cadenas de carácteres
  int strcat(cadena1, cadena2);
  int strcpy(cadena1, cadena2); //Copia la primera cadena en la segunda
  int strlen(cadena);  //Da la longitud de una cadena
```

Funciones de <ctype.h>:

Función toupper ANSI C Convierte un carácter, en un parámetro entero ch, a mayúscula.

Valor de retorno: ch debe estar en el rango 0 a 255, y si está entre a y z lo convierte a su equivalente en el rango A a Z, el resto de los valores no son modificados. El valor de retorno es el valor convertido si ch era una minúscula, o el valor original en caso contrario. Nota: los caracteres en acentuados, o con diéresis, en minúscula y la ñ no sufren modificaciones.

Ejemplo:

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

int main()
{
   char cadena[] = "esto es una cadena de prueba";
   int i;

   for(i = 0; cadena[i]; i++)
      cadena[i] = toupper(cadena[i]);

   printf("%s\n", cadena);
   return 0;
}
```

Preprocesador de C

#include <fichero>	incluir fichero de cabeceras
#include "fichero"	incluir fichero de usuario
#define nombre texto	sustitución de texto
#define nombre(var) texto	macro con argumentos. Ejemplo. #define max(A,B) ((A)>(B) ? (A) : (B))
#undef nombre	anular definición
#	entrecomillar al reemplazar
##	concatenar argumentos y reescanear
#if, #else, #elif, #endif	compilación condicional
#ifdef, #ifndef	¿nombre definido, no definido?
#if defined(nombre)	¿nombre definido?
****	carácter de continuación de línea

Fuentes y contribuyentes del artículo

Programación en C/Referencia *Fuente*: http://es.wikibooks.org/w/index.php?oldid=175770 *Contribuyentes*: Almorca, Fires, Gargo, ManuelGR, Wutsje, 24 ediciones anónimas

Licencia