Guía de Referencia Rápida de C

Programa C

```
// Inclusión de bibliotecas
// Definición de Constantes y Tipos
// Definición de subprogramas
int main()
{
    // cuerpo del programa principal
    return 0;
}
// gcc -Wall -Werror -o main main.c
```

Tipos Simples

```
char short int unsigned float double
enum Color { ROJO, VERDE, AZUL };
```

Constantes, Variables, Asignación

```
#define PI 3.1415
int main()
{
    char a, b;    // sin inicializar
    enum Color c; // sin inicializar
    int x = 1;    // inicialización
    x = x * 2;    // asignación
    a = 'z';      // asignación
    c = ROJO;     // asignación
    x = (int)PI;  // asig, conversión
}
```

Operadores: Precedencia, Asociatividad

Operador	Tipo de Operador	Asociatividad
[ ] -> .	Binarios	Izq. a Dch.
! ~ - *	Unarios	Dch. a Izq.
* / %	Binarios	Izq. a Dch.
+ -	Binarios	Izq. a Dch.
<< >>	Binarios	Izq. a Dch.
< <= > >=	Binarios	Izq. a Dch.
== !=	Binarios	Izq. a Dch.
&	Binario	Izq. a Dch.
^	Binario	Izq. a Dch.
|	Binario	Izq. a Dch.
&&	Binario	Izq. a Dch.
||	Binario	Izq. a Dch.
?:	Ternario	Dch. a Izq.

Asignación e Incrementos

Sentencia	Equivalencia
++vble;	vble = vble + 1;
--vble;	vble = vble - 1;
vble++;	vble = vble + 1;
vble--;	vble = vble - 1;
vble += expr;	vble = vble + (expr);
vble -= expr;	vble = vble - (expr);
vble *= expr;	vble = vble * (expr);
vble /= expr;	vble = vble / (expr);
vble %= expr;	vble = vble % (expr);
vble &= expr;	vble = vble & (expr);
vble ^= expr;	vble = vble ^ (expr);
vble |= expr;	vble = vble | (expr);
vble <<= expr;	vble = vble << (expr);
vble >>= expr;	vble = vble >> (expr);

*Secciones opcionales marcadas con (*)

‡Por cuestiones de espacio, en algunas ocasiones el código fuente no sigue un estilo de codificación adecuado.

Entrada / Salida Básica

```
#include <stdio.h>
int main()
{
    int a, x;
    double z;
    char nombre[30];
    scanf(" %d %lg", &x, &z);
    printf("Resultado: %d %g\n", x, z);
    a = getchar();
    fgets(nombre, sizeof(nombre), stdin);
}
```

Estructuras de Control

```
if ( expr_lógica ) {
    ...
} else if ( expr_lógica ) {
    ...
} else {
    ...
}
switch ( expr_ordinal ) {
case VALOR:
    ...
    break;
default:
    ...
    break;
}
for ( i = 0; i < NELMS; ++i ) {
    ...
}
while ( expr_lógica ) {
    ...
}
do {
    ...
} while ( expr_lógica );
```

Subprogramas

```
int func(int valor_1, const struct Fecha* ref_cte_2)
{
    ...
    return valor;
}
void proc(int* ref_1, struct Fecha* ref_2)
{
    ...
}
int main()
{
    struct Fecha f = { 29, 6, 2011 };
    int x = func(3, &f);
    proc(&x, &f);
}
```

Tipo Registro

```
struct Fecha {
    unsigned dia;
    unsigned mes;
    unsigned anyo;
};
int main()
{
    struct Fecha f = { 29, 6, 2011 };
    struct Fecha hoy = f;
    hoy = f;
    hoy.dia = f.dia;
    hoy.mes = f.mes;
    hoy.anyo = f.anyo;
}
```

Tipo Array

```
#define NELMS 20
typedef int Dat[NELMS];
int main()
{
    int i;
    Dat d = { 1, 2, 3 };
    for (i = 0; i < NELMS; ++i) {
        d[i] = d[i] * 2;
    }
}
```

Tipo Array 2 Dimensiones

```
#define NFILS 2
#define NCOLS 3
typedef int Fila[NCOLS];
typedef Fila Matriz[NFILS];
int main()
{
    int f, c;
    Matriz m = {
        { 1, 2, 3 },
        { 4, 5, 6 }
    };
    for (f = 0; f < NFILS; ++f) {
        for (c = 0; c < NCOLS; ++c) {
            m[f][c] = m[f][c] * 2;
        }
    }
}
```

Salida de Datos Formateada(*)

```
#include <stdio.h>
int main()
{
    int x = 12;
    double d = 3.141592;
    const char* s = "hola";
    printf("%4d", x);    // ##12
    printf("%04d", x);   // 0012 (Dec)
    printf("%04x", x);   // 000c (Hex)
    printf("%04o", x);   // 0014 (Oct)
    printf("%12.4g", d); // #######3.142
    printf("%12.4f", d); // ######3.1416
    printf("%12.4e", d); // ##3.1416e+00
    printf("%-6s", s);   // hola##
}
```

Cadenas de Caracteres

```
#include <stdio.h>
#include <string.h>
int main()
{
    int i;
    char* p;
    char nm[30] = "pepe luis";
    if (strlen(nm) == 0)
        strcpy(nm, "juan");
    }
    if (strcmp(nm, "lucas") <= 0) {
        strcat(nm, " lopez");
    }
    p = strchr(nm, 'p');
    p = strstr(nm, "pepe");
    for (i = 0; nm[i] != '\0'; ++i) {
        nm[i] = (char)(nm[i] + 1);
    }
    scanf(" %s", nm);
    fgets(nm, sizeof(nm), stdin);
    printf(" %s\n", nm);
}
```

Ficheros: Salida de Datos

```
#include <stdio.h>
int main()
{
    FILE* f_out;
    f_out = fopen("datos.txt", "w");
    if ( f_out != NULL ) {
        if (fprintf(f_out, " %s\n", "datos") < 0) {
            /* Error */
        }
        if (fputc('c', f_out) < 0) { /* Error */ }
        if (fputs("datos\n", f_out) < 0) {
            /* Error */
        }
        fclose(f_out);
    }
}
```

Ficheros: Entrada de Datos

```
#include <stdio.h>
int main()
{
    FILE* f_in;
    f_in = fopen("datos.txt", "r");
    if ( f_in != NULL ) {
        int datos;
        char line[126];
        if (fscanf(f_in, " %s", &datos) < 0) {
            /* Error */
        }
        int c = fgetc(f_in);
        if (c < 0) { /* Error */ }
        if (fgets(line,sizeof(line),f_in) == NULL) {
            /* Error */
        }
        fclose(f_in);
    }
}
```

Módulos

```
//--- modulo.h -----------------------
#ifndef modulo_h_
#define modulo_h_
#define PI 3.1415
struct Dato {
    int xx;
};
void put(struct Dato* d, int x);
int get(const struct Dato* d);
#endif
//--- modulo.c -----------------------
#include "modulo.h"
void put(struct Dato* d, int x)
{
    d->xx = x;
}
int get(const struct Dato* d)
{
    return d->xx;
}
//--- main.c -------------------------
#include <stdio.h>
#include "modulo.h"
int main()
{
    struct Dato d;
    put(&d, 5);
    printf("%d\n", get(&d));
}
//-----------------------------------
// gcc -Wall -Werror -c modulo.c
// gcc -Wall -Werror -c main.c
// gc -o main main.o modulo.o
```

Memoria Dinámica

```
struct Nodo {
    struct Nodo* sig;
    int dato;
};
int main()
{
    struct Nodo* ptr = malloc(sizeof(*ptr));
    struct Nodo* ptr2 = ptr;
    struct Nodo n = *ptr;
    struct Nodo* ptr3 = &n;
    ptr->sig = NULL;
    ptr->dato = 3;
    if ((ptr != NULL)&&(ptr == ptr2)) {
        ptr = ptr->sig;
    }
    free(ptr);
}
```

Asertos(*)

```
#include <assert.h>
int main()
{
    assert((x > 3)&&(y < 6));
}
```