



UNIVERSIDAD DEL BÍO-BÍO

SISTEMAS
OPERATIVOS

El Lenguaje C

Aspectos Léxicos y Sintácticos

Fernando Santolaya
fsantolaya@ubiobio.cl

ESTRUCTURA DE UN PROGRAMA EN C

- Todos los programas escritos en C se componen de una o más funciones, no teniendo por qué estar todas en un mismo archivo.
- `main(...)` es la función obligatoria en todos los programas, ya que a través de ella se comienza la ejecución.
- Generalmente un programa en C tiene la siguiente forma:
 1. Comando para el preprocesador
 2. Definiciones de tipos de datos
 3. Prototipos de funciones
 4. Variables globales
 5. Funciones

Ejemplo:

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

comandos del preprocesador

```
typedef float balance;
```

definiciones de tipos

```
void imprime();
```

prototipos de funciones

```
int cont=3; balance b;
```

variable globales

```
int main(void) {  
    b=1000;  
    imprime();  
    return 0;  
}
```

funciones

```
void imprime(){  
    printf("cont = %d /n", &cont);  
    printf("balance = %f /n", &b);  
}
```

- Los bloques de código (2 o más instrucciones) se encuentran delimitados por llaves “{ }”.
- Las instrucciones compuestas pueden estar anidadas.
- Cada instrucción debe terminar con “;”.
- Los comentarios están delimitados por “/* */” y pueden ir en cualquier parte del programa.

No use “//” no pertenece al estándar ANSI C.

Ej: /* Aquí van los comentarios */

SENTENCIAS SIMPLES

- Una sentencia es cualquier expresión en C que termina con “;” (sin las dobles comillas).
- Pueden ser asignaciones, operaciones, llamadas a funciones o combinaciones de ellas.

Etiquetas

Sirven para etiquetar una sentencia de forma que el control del programa pueda ser transferido a ella.

Se separan de la sentencia por dos puntos “:”.

Forma general:

etiqueta: sentencia;

Por ejemplo: etiq100: x++

SENTENCIAS COMPUESTAS

Es un conjunto de sentencias simples que se encierran entre llaves “{” y “}” para formar un bloque de código.

Puede aparecer en cualquier lugar donde podría ir una sentencia simple.

Pueden contener declaraciones de nuevas variables (la declaración existe sólo dentro del bloque).

Forma general:

```
{  
    sentencia1;  
    sentencia2;  
    ...  
    sentenciaN;  
}
```

SENTENCIAS COMPUESTAS

Sentencia: if e if-else

Forma general:

```
if (expresión)  
    sentencia;
```

Si *expresión* es verdadera (valor mayor que 0) se ejecuta *sentencia*

La *expresión* debe estar entre paréntesis

```
if (expresión)  
    sentencia1;  
else  
    sentencia2;
```

SENTENCIAS COMPUESTAS

Sentencia: for

Forma general:

```
for (expresión1; expresión2; expresión3)  
    sentencia;
```

Expresión1: inicialización

Expresión2: condición para ejecutar sentencia. Se repite mientras no sea 0 (falso).

Expresión3: se utiliza para modificar el valor usado en la condición.

Expresión1 y expresión3 se pueden omitir.

Si se omite expresión 2 se asumiría el valor permanente de 1 (cierto) y el bucle se ejecutaría de forma indefinida.

SENTENCIAS COMPUESTAS

Sentencia: while

Forma general:

```
while (expresión)  
    sentencia;
```

Sentencia se ejecutará mientras el valor de expresión sea verdadero (distinto de cero).

SENTENCIAS COMPUESTAS

Sentencia: do-while

Forma general:

```
do  
    sentencia;  
while (expresión);
```

Sentencia se ejecutará mientras el valor de expresión sea verdadero (distinto de cero).

Sentencia siempre se ejecuta al menos una vez (diferencia con while)

SENTENCIAS COMPUESTAS

Sentencia: switch

Forma general:

```
switch (expresión) {  
    case constante1:  
        secuencia de sentencias;  
        break;  
    case constante2:  
        secuencia de sentencias;  
        break;  
    ...  
    ...  
    default:  
        secuencia de sentencias;  
}
```

SENTENCIAS COMPUESTAS

- Es una sentencia de selección múltiple.
- Se ejecuta el bloque asociado a la constante que es igual a la expresión.
- La sentencia default se ejecuta si no se ha encontrado ninguna correspondencia. Además es opcional.
- La sentencia break por su parte, permite salir de un bloque de código, si se coloca dentro de un ciclo permite salir del ciclo.

SENTENCIAS COMPUESTAS

Ejemplo:

```
void ver_opcion ( char c ) {  
    switch(c) {  
        case 'a': printf("Op A\n");  
                   break;  
        case 'b': printf("Op B\n");  
                   break;  
        case 'c':  
        case 'd': printf("Op C o D\n");  
                   break;  
        default: printf("Op ?\n");  
    }  
}
```

SENTENCIAS COMPUESTAS

Sentencia: continue

- Típicamente se usa dentro de un ciclo.
- Funciona de forma similar a break, pero en vez de terminar el ciclo fuerza una nueva iteración sin completar las instrucciones faltantes dentro del bloque.

Ejemplo:

```
int i;
for (i = 1; i <= 50; i++) {
    if (i % 2 == 0) /*si es impar imprimirlo*/
        continue;
    printf("%d", i);
}
```

SENTENCIAS COMPUESTAS

Sentencia: return

Formal general:

```
return expresión;
```

- Se utiliza para volver de una función.
- Retorna al punto donde se hizo la invocación.
- Expresión es opcional.

JUEGO DE CARACTERES DE C

El lenguaje C utiliza los caracteres del alfabeto inglés (mayúsculas A..Z, minúsculas a..z), los dígitos 0..9 y caracteres especiales.

+	-	*	_	=	%
&	#	!	?	^	"
/	~	\		<	>
()	[]	{	}
:	;	.	,		

IDENTIFICADORES

- Los identificadores se utilizan para nombrar a diversos tipos de elementos de un programa, tales como:
 - Variables de memoria
 - Constantes
 - Etiquetas (labels)
 - Funciones
- Un identificador consiste en una secuencia de letras o dígitos comenzando con una letra o “_”. Se pueden utilizar mayúsculas o minúsculas.

Ej correcto: cont, sum, p23, balance, _total

Ej incorrecto: 1cont, hola!, balance:

PALABRAS CLAVE

Existen ciertos nombres que son utilizados como palabras claves y no pueden ser utilizados como identificadores:

auto	do	for	short	typedef
break	double	if	signed	union
case	else	int	sizeof	unsigned
char	enum	long	static	void
continue	extern	register	struct	volatile
default	float	return	switch	while

TIPOS DE DATOS BASICOS

C soporta varios tipos de datos básicos, el tipo determina como se almacena el dato y su identificador.

Existen 5 tipos de datos atómicos:

Tipo	N° de bits	rango
char	8	0 a 255
int	16	-32.768 a 32.767
float	32	3,4E -38 a 3,4E +38
double	64	1,7 E -308 a 1,7 E +308
void	0	Sin valor

Void se utiliza para declarar funciones que no devuelven ningún valor o funciones sin parámetros de entrada.

DECLARACION DE VARIABLES

Todas las variables han de ser declaradas antes de ser usadas. Forma general:

<tipo> <lista de variables> Ej: `int i,j,cont;`

Existen 3 sitios donde se pueden declarar variables:

- Dentro de funciones (variables locales)
- En la definición de parámetros de funciones (parámetros)
- Fuera de todas las funciones (variables globales)

Modificadores

Los modificadores permiten cambiar el rango de valores almacenados por una variable o la forma en que esta es accesada o almacenada, por lo tanto, existen modificadores de tipos, modificadores de acceso y modificadores de almacenamiento.

Modificadores de tipos

- signed
- unsigned
- long
- short

Tipo	N° de bits	Rango
char	8	-128 a 127
unsigned char	8	0 a 255
signed char	8	-128 a 127
int	16/32	-32.768 a 32.767
unsigned int	16	0 a 65.535
signed int	16	-32.768 a 32.767
short int	16	-32.768 a 32.767
unsigned short int	16	0 a 65.535
signed short int	16	-32.768 a 32.767
long int	32	-2.147.483.648 a 2.147.483.647
signed long int	32	-2.147.483.648 a 2.147.483.647
float	32	3,4 E -38 a 3,4 E +38
double	64	1,7 E -308 a 1,7 E +308
long double	64	1,7 E -308 a 1,7 E +308

Modificadores de acceso

El típico modificador de acceso corresponde a “const” el cual no permite que el valor de una variable sea cambiado durante la ejecución de un programa (constante).

Ej: `const int a;`

Modificadores de almacenamiento

Se refiere a la permanencia de la variable y su ámbito (scope) dentro del programa, es decir, en qué parte será reconocida.

Ámbito corresponde a todo lo que está entre llaves {...}.

- Variables automáticas (auto).

Por defecto todas las variables son automáticas.

Ej: auto int suma= 0;

- Variables externas (extern).

Si una función situada en un archivo fuente desea utilizar una variable de este tipo declarada en otro archivo, la debe declarar con la palabra extern.

Ej:

Archivo 1	Archivo 2
<pre>int x,y; char ch; main() { ... }</pre>	<pre>extern int x,y; extern char ch; void func1() { x=120; x=y/10; ... }</pre>

- Variables estáticas (static).

Tienen memoria asignada durante toda la ejecución del programa. Su valor es recordado incluso si la función donde está definida termina y se vuelve a llamar más tarde.

Ej:

```
series (void) {  
    static int num;  
    num= num+23;  
    return (num);  
}
```

- Variables registro (register).

Son variables que se almacenan en los registros del procesador. Esto permite mayor velocidad de acceso.

Restricciones:

- Sólo se pueden usar con valores enteros.
- Sólo se puede aplicar a variables locales y a los parámetros formales de una función.

Ej:

```
pot_ent(int m, register int e) {  
    register int temp;  
    temp=1;  
    for(;e;e--)  
        temp= temp * m;  
    return (temp);  
}
```

OPERADORES ARITMETICOS

Operador	Acción	Ejemplo
-	Resta	<code>x= 5-3; x vale 2</code>
+	Suma	<code>x= 2+3; x vale 5</code>
*	Multiplicación	<code>x= 2*3; x vale 6</code>
/	División	<code>x= 6/2; x vale 3</code>
%	Módulo	<code>x= 5%2; x vale 1</code>
--	Decremento	<code>x= 1; x--; x vale 0</code>
++	Incremento	<code>x= 1; x++; x vale 2</code>

- **Incrementos y decrementos:**

Cuando un operador de incremento o decremento precede a su operador, se llevará a cabo la operación antes de utilizar el valor del operando.

Ej: Forma prefija

```
int x,y;    x e y valen 2005  
x = 2004;  
y = ++x;
```

Ej: Forma postfija

```
int x,y;  
x = 2004;  
y = x++;    y vale 2004, x vale 2005
```

OPERADORES RELACIONALES Y LOGICOS

Operador	Acción
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
==	Igual
!=	Distinto

El resultado de la evaluación en estas operaciones es de tipo entero:

Falso: 0

Verdadero: valor > 0

Operadores lógicos

Operador	Acción
&&	Conjunción (Y)
	Disyunción (O)
!	Negación (NO)


Cuidado && no significa lo mismo que &.

De igual forma || no significa lo mismo que |.

Otros operadores

- Operador ternario (...) ? ... : ...

Ej: cálculo del máximo entre a y b.

<code>z = (a > b) ? a : b;</code>		<pre>if (a > b) z = a else z = b;</pre>
	Es lo mismo	

- Operador unario sizeof:

Devuelve la longitud, en bytes, de la variable o el tipo especificado.

Ej:

```
double un_double;
printf("double: %d \n", sizeof(un_double));
printf("float: %d \n", sizeof(float));
printf("int: %d \n", sizeof(int));
```

- Operador unario cast

Permite hacer una conversión explícita (a la fuerza) del tipo de una expresión.

Ej:

```
int n;  
sqrt((double) n);
```

sqrt es una función de librería (math.h) y acepta sólo parámetros del tipo double.

- Operador coma “,”

Permite encadenar varias expresiones.

Ej:

```
x = ( y=9, z=0 );
```

x tendrá el valor 0.