Lenguaje C: conceptos generales

Fundamentos de Lenguajes de Programación

Universidad Nacional de la Patagonia Austral
Unidad Académica Rio Gallegos

Historia del Lenguaje C

- Fue inventado por Dennis Ritchie
  - en un DEC-PDP-11 en los Laboratorios BELL, basándose en el lenguaje B, creado por Ken Thompson.
- En 1983 el Instituto de Estándares Americanos estableció un estándar que definiera al lenguaje C, conocido como ANSI C.
- Los principales compiladores de C llevan implementado el estándar ANSI C.

Características del Lenguaje C

- Mainframes
- Se cataloga como un lenguaje de nivel medio
  - combina elementos de lenguajes de alto nivel (Fortran, Pascal, Basic, etc.) con la funcionalidad del lenguaje ensamblador.
- Pensado para escribir sistema operativo
- Permite el manejo de bits, bytes y direcciones de memoria.
- Posee sólo 32 palabras clave, definidas por el comité ANSI.

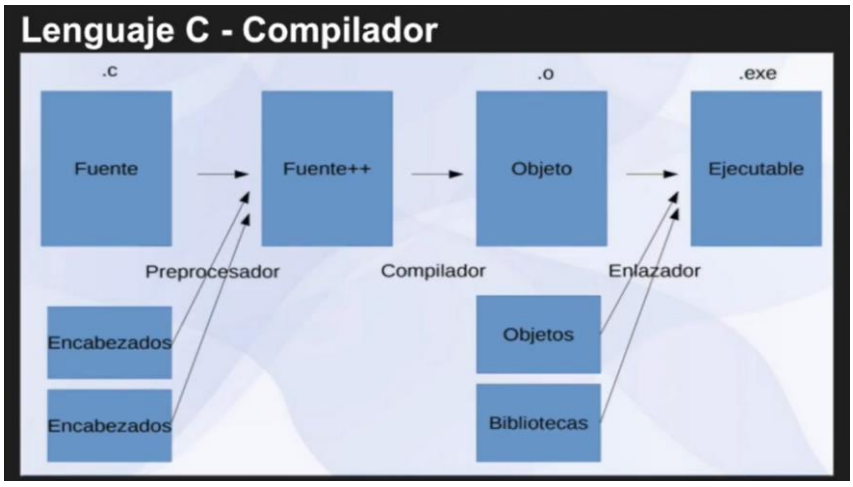
Lenguaje C

- Compilado
- Procedural
- Fuertemente tipado

32 Palabras Reservadas

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

Compilador



Elementos de un Programa

- Comentarios.
- Identificadores.
- Constantes.
- Variables.
- Operadores.
- Sentencias o instrucciones.
- Macros del pre-procesador.

Comentarios

- Comentarios multi-línea
  - Comienzan con **/*** y terminan con ***/**
  - No puede anidarse un comentario dentro de otro.
- Comentarios de una sola línea (C++, Java, Delphi).
  - Comienzan al principio de la línea con **//**
- **Ejemplo:**

```
//Esto es un comentario
/* Esto también es
       un  comentario */
```

Identificadores

- Se utilizan para nombrar variables, funciones, etiquetas y elementos definidos por el usuario.
- Los primeros seis caracteres deben ser significativos (distinguirse de otro similar) y máximo puede tener hasta 31 caracteres.
- El primer carácter debe de ser una letra o subguión. Posteriormente pueden ser letras, números, signos de subrayado.
- Existe diferencia entre mayúsculas y minúsculas.

Identificadores

- No pueden emplearse palabras reservadas como identificadores.
- No pueden emplearse nombres de funciones que ya existan en el programa o en la librería de funciones de C.
- No puede llamarse main.

Convenciones

- Empezar los nombres de funciones y de variables con una letra minúscula.
- Las constantes escritas con **#define** van con mayúsculas como `#define PI 3.1416`
- Las palabras intermedias comienzan con mayúsculas.

  `sumaMatrices`
- Utilizar el subguión para separar palabras intermedias.

  `suma_Matrices`
- Emplear nombres cortos para optimizar. **(i, j, k, cont)**

#define

- Se utiliza para asignar un identificador a una constante.

  #define PI 3.1416

  #define NCOLS 20
- El pre-procesador de C, sustituye la ocurrencia de PI por el valor 3.1416 en todo el programa antes de efectuar la compilación, del mismo modo se sustituye NCOLS por 2.

Variables

- Una variable es una localidad de memoria cuyo valor puede ser cambiado durante la ejecución del programa.
- Todas las variables deben de ser declaradas para se utilizadas.

  <tipo de dato> espacio(s) <identificador>;
- Ejemplo:

```
int a;
float area, radio, volumen;
```


const

- Es un modificador de acceso que me permite asignar a una variable un valor constante, es decir que una vez asignado a dicha variable su valor no podrá ser modificado durante el programa.

  const <tipo dato> esp <identificador> = valor;
- Ejemplo:

```
const int a=10;
const char pais[]="MÉXICO";
const char *nombre="VLADIMIR";
```

volatile

- Es un modificador de acceso que me permite cambiar el valor de una variable por medios no explícitamente especificados por el programa. Por ejemplo la dirección de una variable global que apunta a un puerto externo.

volatile <tipo dato> esp <identificador> = valor;

- Ejemplo:

```
volatile unsigned char *puerto = 0x30;
```


Operadores

- Son palabras o símbolos que implican una acción sobre ciertas variables. Pueden ser unarios (1 variable), binarios(2 variables) o ternarios (3 variables).
- Operadores Aritméticos
- Operadores Relacionales
- Operadores Lógicos
- Operadores de Asignación
- Operadores de Dirección
- Operadores de Bits

Operadores Aritméticos

Operador	Nombre	Descripción
+	Suma	5+2 → 7
-	Resta	5-2 → 3
*	Multiplicación	5*2 → 10
/	División	5/2 → 2
%	Módulo	5%2 → 1
(tipo de dato)	"Cast" forzado	(double)5 → 5.0

Operadores Relacionales

Operador	Nombre	Descripción
==	Igual a	if (a=='s')
!=	Diferente de	if (a!=null)
>	Mayor que	if (a>0.5)
<	Menor que	if (a<2l)
>=	Mayor o igual que	if (a>=2f)
<=	Menor o igual que	if (a<=3)

Operadores Lógicos

Operador	Nombre	Descripción
&&	Y (AND)	if ((a>3) && (a<9))
||	O (OR)	if ((a==2) || (a==3))
!	NEGADO (NOT)	if (!(a==3)) es igual a if (a!=3)

- Importante:

  FALSO es igual a cero.

  VERDADERO es diferente de cero.

Operadores de Asignación

Operador	Abreviado	No Abreviado
=	a=2;	a=2;
++	n++;	n=n+1;
--	n--;	n=n-1;
+=	n+=2;	n=n+2;
-=	n-=2;	n=n-2;
*=	n*=2;	n=n*2;
/=	n/=2;	n=n/2;
%=	n%=2;	n=n%2;

Operadores de Bits

Operador	Nombre	Descripción
<<	Corrimiento a la izquierda	b=a>>2;
>>	Corrimiento a la derecha	b=a<<3;
&	Y (AND) entre bits	c=a&128;
|	O (OR) entre bits	c=a|0x0a;
~	Complemento A1	c=~a;
^	O exclusivo (XOR)	c=^a;

Operadores de Asignación para bits

Operador	Abreviado	No Abreviado
<<=	n<<=2;	n=n<<2;
>>=	n>>=2;	n=n>>2;
&=	n&=0x0a;	n=n&0x0a;
|=	n|=7;	n=n|7;
^=	n^=0x03;	n=^0x03;
=	n=0x7f;	n=0x7f;

Nota:

0x7f, 0x0a, 0x03 son un números hexadecimales.

Operadores de Dirección

Operador	Nombre	Descripción
*	Operador indirección	Me da el valor que está almacenado en una dirección de memoria. También sirve para declarar una variable apuntador.
&	Operador dirección	Me da la dirección de memoria de una variable.

Variables Apuntador

- Sirven para almacenar una dirección de memoria.
- Utilizan el operador & para obtener la dirección.
- Se deben inicializar con NULL (equivale a cero).
- Se declaran como:

  <tipo de dato><*> espacio(s) <identificador>;

- Ejemplo:

```
int *a=NULL;
int b=2,c=1;
a=&b;  /*Guarda la direc. de b en a */
c=*a;  /*c vale 2 */
```

Precedencia de Operadores

() [] ->	Alta prioridad
! ~ + - ++ -- & * sizeof	Unarios
* / % + -	Aritméticos
<< >>	Corrimiento de bits
< <= > >= == !=	Relacionales
& ^ | && || ?:	Bits / Lógicos / Condicional
= *= /= %= += -= &= ^= |= <<= >>=	Asignación
,	Evaluación

Sentencias (Instrucciones)

- Una sentencia es una instrucción o expresión en C que tiene una consecuencia. Pueden ser asignaciones, operaciones, llamadas a funciones.
- Todas las sentencias terminan con el signo de punto y coma ;
- Pueden ser simples o compuestas. Las compuestas van entre llaves:

```
{
  sentencia1;
   sentencia2;
   :
   sentencian;
}
```

Sentencias (Instrucciones)

- Sentencias de Selección.
  - if – else, switch – case, ?:
- Sentencias de Repetición.
  - do – while, while, for
- Sentencias de Salto.
  - return, break, continue.

Estructura de un programa en C

I. Directivas del pre-procesador **#include y #define**
II. Declaración de Prototipos (Declaración de funciones)
III. Declaración de variables globales
IV. Funciones definidas por el usuario
V. Función Principal main( )

prototipos

Lenguaje C - Prototipos

- El compilador de C funciona en una sola pasada
- Para utilizar una función no hace falta que la misma esté definida previamente, pero sí hace falta que al menos esté declarada
- Para declarar las funciones se utilizan los prototipos:

```
void imprimir(int);
```

- Le informa al compilador que imprimir() es una función que recibe un entero y no devuelve nada. El compilador con esto puede ajustar las llamadas a la función aunque no conozca su implementación (ej. "imprimir(3.14)")

Ejemplo

```
#include <stdio.h>
#include <math.h>
#define VALOR 5.7

double modulo3D( double x, double y, double z );
double mod3; /* Variable global */

double modulo3D( double x, double y, double z ){
  return(sqrt(x*x+y*y+z*z);
}

int main( void ){
  int x, y, z;
  x=y=z=VALOR;
  mod3=modulo3D(x,y,z);
  printf("\nEl módulo es: %lf",mod3);
  return(0);
}
```

if-else

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

Nota: una expresión en C es todo aquello que regresa un valor. Como por ejemplo una condición lógica, operaciones aritméticas, llamadas a funciones, una variable, una constante (númérica, carácter, etc.).

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


Operador Condicional ?:

(expresión)? sentencia1 : sentencia2;

expresión? sentencia1 : sentencia2;

Se ejecuta:

sentencia1 si expresión = verdadero

sentencia2 si expresión = falso.

Es un operador ternario y puede utilizarse para asignar variables:

Var = (expresión)? sentencia1:sentencia2;

switch-case

```
switch(expresión)
 {
   case 1:  sentencias;
            break;
   case 2:  sentencias;
            break;
    :
   case n:  sentencias;
            break;
   default: sentencias_default;
            break;
 }
```


Arreglos Unidimensionales

- Los arreglos unidimensionales son secuencias de valores del mismo tipo que se almacenan en localidades contiguas de memoria, según el orden del índice.

  <tipo dato> esp <identificador>[tamaño];

- Ejemplo:

```
int valores[10];
float datos[5]={1.3,2.8,4.89,0.0,5.7};
char pais[]="MÉXICO";
char *nombre="VLADIMIR";
```

Conversión explicita e implícita de tipos

- Cuando se evalúa una expresión y el resultado se asigna a una variable, se debe tener en cuenta el tipo de todos los operandos y la variable en la que se almacena.
- **Conversion Implícita:** Si se asigna el valor de un entero **int** a una variable de menor tamaño de bytes como **char**, el entero corto cambia a un valor entero automáticamente.

Conversión explicita e implícita de tipos

- Ejemplo:

int	char
1000	-24
10000	16
-5000	120

- Se puede asignar un entero int a un char. C se encarga de hacer la conversión de tipos de ahí que se denomina **conversión implícita.**

Conversión explicita e implícita de tipos

- Pero en 1 byte (char) no caben los 4 bytes (long). C toma los 8 bits menos significativos del int y los almacena en el char. O sea preserva el valor solo si el número esta entre **-128 y 127**.

Operadores de conversión de tipos.

- Conversión Explícita: Puede convertir un valor de un tipo de datos a otro compatible indicándolo explícitamente y NO delegando la conversión a C.
- Ejemplo:

```
#include <stdio.h>
int main(void)
{
int       a = 1,b = 2;
float     x;

    x = a / ( float ) b;
    return ( 0 );
}
```

Operadores de entrada y salida.

- Para la salida de datos se utiliza habitualmente **printf**. Es una función disponible en la biblioteca **stdio.h**
- El uso mas simple es para mostrar una cadena por pantalla.

  printf (“un string”);

  Para incluir salto de línea se debe incluir \n.

  printf (“un string\n”);

Operadores de entrada y salida.

- Para mostrar números enteros o flotantes se debe usar necesariamente cadenas con formato.

Tipo	Marca
int	%d
unsigned int	%u
float	%f
char	%hhd
unsigned char	%hhu

Ejemplo.

- Si a es una variable de tipo **int** con valor 5, b es una variable de tipo **float** con valor 1.0, y c es una variable de tipo **char** con valor 100, esta llamada a la función printf :

```
printf ( "Un entero: %d, un flotante: %f, un byte:
  %hhd\n", a, b, c );
```

- Por pantalla…

```
Un entero: 5, un flotante: 1.000000, un byte: 100
```


Operadores de entrada y salida.

- Las marcas de formato para enteros aceptan modificadores.
- Un número positivo: reserva un número de espacios determinado (el que se indique) para representar el valor y muestra el entero alineado a la derecha.
  - Ejemplo: la sentencia

```
printf ( "[%6d]", 10 );
```

  - muestra en pantalla:

```
[    10]
```

Operadores de entrada y salida.

- Un número negativo: reserva tantos espacios como indique el valor absoluto del número para representar el entero y muestra el valor alineado a la izquierda.
- Ejemplo: la sentencia

printf ("[%-6d]", 10);

- muestra en pantalla:

[10]

Operadores de entrada y salida.

- Un número que empieza por cero: reserva tantos espacios como indique el valor absoluto del número para representar el entero y muestra el valor alineado a la izquierda. Los espacios que no ocupa el entero se rellenan con ceros.
- Ejemplo: la sentencia

printf ("[%06d]", 10);

- muestra en pantalla:

[000010]

Operadores de entrada y salida.

- El signo +: muestra explícitamente el signo (positivo o negativo) del entero.
- Ejemplo: la sentencia

printf ("[%+6d]", 10);

- muestra en pantalla:

[+10]

Operadores de entrada y salida.

- Hay dos notaciones alternativas para la representación de flotantes que podemos seleccionar mediante la marca de formato adecuada:

Tipo	Notación	Marca
float	Convencional	%f
float	Científica	%e

Operadores de entrada y salida.

- **La forma convencional** muestra los números con una parte entera y una decimal separadas por un punto.
- **La notación científica** representa al número como una cantidad con una sola cifra entera y una parte decimal, pero seguida de la letra e y un valor entero. Por ejemplo, en notación científica, el número 10.1 se representa con 1.010000e+01.

Operadores de entrada y salida.

- Los modificadores de formato que se han presentado para los enteros son válidos para flotantes, además la posibilidad de fijar la precisión:
- Un punto seguido de un número: indica cuántos decimales se mostrarán.
- Ejemplo: la sentencia

```
printf ( "[%6.2f]", 10.1 );
```

- muestra en pantalla:

```
[ 10.10]
```

Marcas de formato para texto.

- C distingue entre caracteres y cadenas:

Tipo	Marca
carácter	%c
cadena	%s

Marcas de formato para texto.

- Por ejemplo
  - printf ("[%c]", 97);
  - muestra en pantalla:
  - [a]
- El valor 97 también puede representarse con el literal ’a’, así que esta otra sentencia
- printf ("[%c]", ’a’);
- también muestra en pantalla:
- [a]

Marcas de formato para texto.

- printf ("[%s]", "un string");
- En pantalla se muestra esto:
- [un string]
- También puedes usar números positivos y negativos como modificadores de estas marcas.
- Su efecto es reservar los espacios que se indique y alinear a derecha o izquierda.

Entrada por teclado.

- La función **scanf**, disponible al incluir **stdio.h**, permite leer datos por teclado.
- Se usa de un modo similar a **printf**: su primer argumento es una cadena con marcas de formato.
- A éste le siguen **una o más direcciones de memoria**. Si se desea leer por teclado el valor de una variable entera a:
- scanf ("%d", &a);

Entrada por teclado.

- scanf recibe:
- Una cadena cuya marca de formato indica de **qué tipo es el valor que vamos a leer por teclado**:

Tipo	Marca
int	%d
unsigned int	%u
float	%f
char como entero	%hhd
char como carácter	%c
unsigned char como entero	%hhu
unsigned char como carácter	%c

Entrada por teclado.

- La dirección de memoria que corresponde al lugar en el que se depositará el valor leído.
- Se debe proporcionar una dirección de memoria por cada marca de formato indicada en el primero argumento.
- \n.

Arreglos Unidimensionales

- Un arreglo se identifica por su nombre, pero para el compilador este equivale a la dirección del primer elemento del arreglo, es decir:
- Ejemplo:

```
int num[50];

Para el compilador:
num  es igual a  &num[0]
/* La dirección del elemento 0 */
```


Arreglos Unidimensionales

- Los índices son necesarios para desplazarse a través del arreglo. El primer elemento tiene el índice cero y el último el índice (tamaño-1).
- Se deben utilizar variables enteras y para agilizar el acceso a memoria se pueden declarar con el modificador register.
- Ejemplo:

```
int register i, j, k;
```

Arreglos Unidimensionales

- Para guardar o leer los elemento en un arreglo es muy frecuente utilizar la sentencia for.
- Ejemplo:

```
int register i;
int num[50];

/* Asigno al arreglo valores del 0 al
   500 */
for (i=0; i<50; i++)
  num[i]= i*10;
```


Arreglos Unidimensionales

- Para introducir los elemento en un arreglo con la función scanf se recomienda utilizar una variable auxiliar dentro de un ciclo for.
- Ejemplo:

```
int register i;
double datos[100];
double temp;
for (i=0; i<100; i++)
 {
   scanf("%lf",&temp);
   fflush(stdin); /* stdin=entrada estándar */
   num[i]= temp;
 }
```

Arreglos Unidimensionales

- Un arreglo puede tener N dimensiones, dependiendo de las limitaciones de la memoria y su declaración es la siguiente:

<tipo dato> esp <identificador>[dim1] [dim2]…[dimN];

- Ejemplo:

```
double cubo[3][3][3];
```


Arreglos de Caracteres

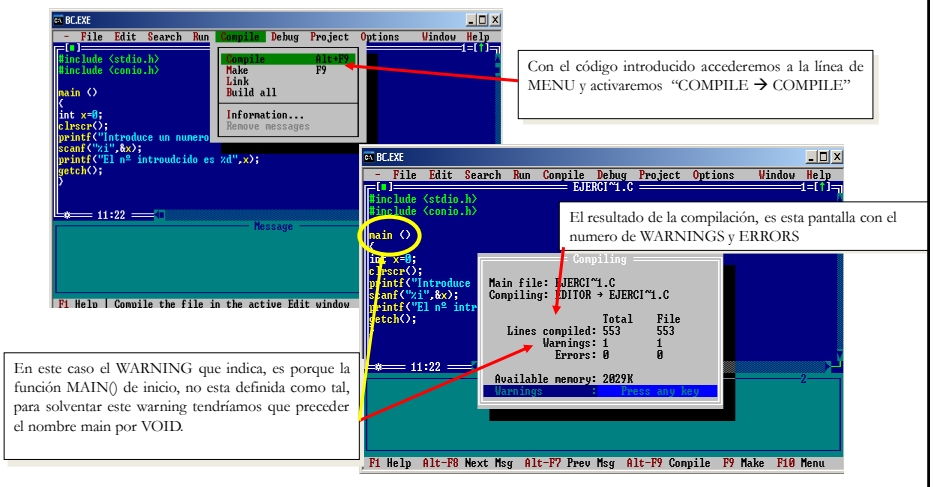
- **Una cadena o "String" se manipula en lenguaje C, mediante arreglos de caracteres. Los arreglos de caracteres terminan con el carácter nulo '\0' que en realidad es el valor cero.**
- **La única diferencia con los arreglos numéricos es que se requiere un carácter adicional para indicar cuando el final del arreglo.**
- Ejemplo:

```
char nombre[31]; /* Uso sólo 30 */
```

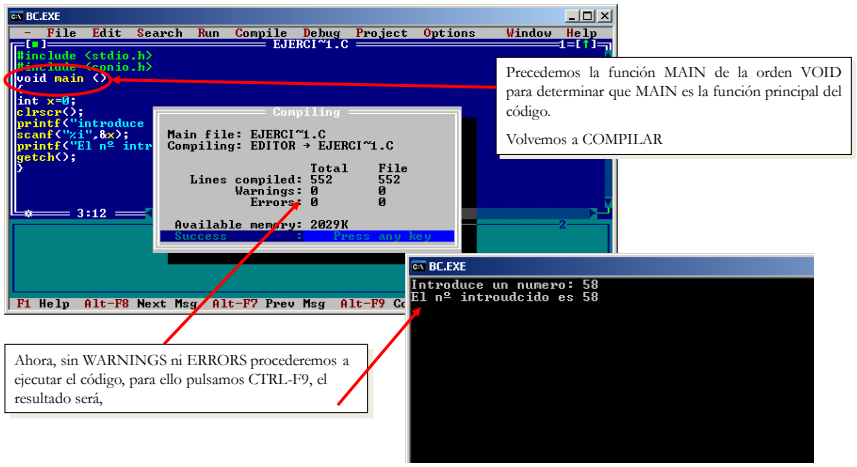
MODO DE EJECUCION DE UN PROGRAMA

- Desde el editor de “C”, cuando tenemos el código copiado tendremos que “compilarlo” para comprobar que no hay ningun error de sintaxis, etc…, ya que en el caso de haberlo, no se podrá ejecutar.
- El “C” controla dos tipos de errores, los WARNINGS y los ERRORS.
- Un programa con WARNINGS se podrá ejecutar ya que son errores leves, pero con ERRORS no podremos ejecutar el código.

Como compilar el código



Como compilar el código



Precedemos la función MAIN de la orden VOID para determinar que MAIN es la función principal del código.

Volvemos a COMPILAR

Ahora, sin WARNINGS ni ERRORS procederemos a ejecutar el código, para ello pulsamos CTRL-F9, el resultado será,

Lenguaje C

- Fin de clase

Próximas clases

- Clase práctica:
  - Viernes 18 y 22/08
- Clase teórica:
  - Jueves 28/08: Datos