UNIVERSIDAD ADOLFO IBÁÑEZ

2020/01

Lenguajes y Paradigmas de Programación

INGENIERÍA

Conozcamos C

Agenda

- Presentación del lenguaje C
- Estructura de un programa en C
- Tipos de datos escalares
- Operadores básicos
- Condicionales
- Iteraciones
- Introducción a las funciones E/S
- Arrays (vectores o matrices)
- Funciones y argumentos

Lenguaje C

- Creado en 1972 por D. Ritchie
- Lenguaje de propósito general
- Transportable (generalmente)
- Inicialmente de nivel medio (entre alto nivel y ensamblador)
- Pensado para (gestionar / programar) sistemas/comunicaciones
- Lenguaje compilado (compilar + enlazar)
- Modular (permite usar bibliotecas propias o estándar que se enlazan con nuestros programas principales)
- (Demasiado) conciso
- (Relativamente) sencillo de aprender

Palabras reservadas

Auto	double	if	static	break
Else	Int	struct	case	entry
long	Switch	char	extern	register
Typedef	continue	float	return	union
Default	for	sizeof	unsigned	do
Goto	short	while		

Estructura

Algoritmo principal es
<declaración tipos>
<declaración variables>
Inicio
  <composición secuencial acciones>
Fin

[int] main([<parámetros línea comando>])
[<declaración de parámetros>]
{
<declaración tipos>
<declaración variables>
<composición secuencial acciones>

Un programa en C es una colección de una o más funciones que se llaman unas a otras

La función principal siempre se llama main().

[<tipo>] nombre_función ([<lista parámetros>])
[<declaración de lista parámetros>]
{
<declaración tipos>
<declaración variables>
<composición secuencial acciones>
[return <expresión tipo de la función>]
}

Al finalizar una función ésta devuelve el control a la función que la ha llamado o al S.O. en el caso de la función main().

Reglas en C

- Formato libre de líneas (no hay límite de tamaño)
- Las sentencias deben separarse mediante punto y coma ;
- Las llaves {} agrupan conjuntos de sentencias lógicamente relacionadas (como inicio-fin, hacer-fin para,...)
- Los comentarios se ponen **/* así */**
- Todas las funciones llevan un tipo (int, por defecto)
- La instrucción return nos permite devolver explícitamente un valor. Si no se especifica, se devolverá un valor arbitrario, del tipo de la función.
- Es obligatorio definir todas las variables que se van a utilizar, es decir, equivale a IMPLICIT NONE.

Tipos de Datos

Tipos básicos

- char, carácter
- int, entero
- float, real de simple precisión
- double, real de doble precisión
- NO HAY LÓGICOS

Tipos de Datos

Tipos derivados de los básicos

- signed char, unsigned char --> como int
- short, signed short, unsigned, unsigned short:
- short int, signed short int, unsigned short int, signed int, unsigned int
- long, signed long, unsigned: long int, signed long int, unsigned long int

```
main() {
int entero1, entero2;
char caracter1, c, car2;
float real1, r2;
double d1, d2;

short int s;
long int entero_largo;
unsigned char cc;
return 0;
}
```

```
int entero = 1, otro_entero;
float r1, r2 = 1.2e-5;
double d=1.23456789;
char c = 'a', nueva_linea = '\n';

otro_entero = 12;
r1 = r2 / 0.5;
```

Declaración de constantes

```
#define PI 3.1416
#define TAMANO_MAX 200
```

```
#include <stdio.h>
int main()
{
int entero1, entero2;
char caracter1, car2, nueva_linea;
float real1;
double d1, d2;

short int s;
long int entero_largo;
unsigned char cc;
int entero = 1, otro_entero;

float r1, r2 = 1.2e-5;
double d=1.23456789;
char c = 'a';

otro_entero = 12;
r1 = r2 / 0.5;

printf("%f", r1);

/* Declaración de constantes*/
#define PI 3.1416
#define TAMANO_MAX 200
return 0;
}
```

```
int entero = 1, otro_entero;
float r1, r2 = 1.2e-5;
double d=1.23456789;
char c = 'a', nueva_linea = '\n';

otro_entero = 12;
r1 = r2 / 0.5;
```

Declaración de constantes

```
#define PI 3.1416
#define TAMANO_MAX 200
```

Operadores aritméticos

Suma	:	a + b	a = a + 1	a += 1
Resta	:	c - d	c = c - d	c -= d
Incremento	:	a=a+1	a++	
Decremento	:	b=b-1	b--	
Multiplicación	:	e * f	e = e * 2	e *= 2
División	:	g / h	g = g / 10	g /= 10
módulo/resto	:	i % 14	i = i % 2	i %= 2

Relacionales/Comparación

Mayor : j > k

Mayor o igual: ll >= m

Menor : n < op

Menor o igual: q <= r

Igual : s == t (es doble signo =, no confundir con =)

Distinto : u != v

Lógicos (falso == 0, cierto es != 0)

And	:	w && y
Or	:	w || y
Negación	:	! z

En las bibliotecas (#include <math.h>,...) existen otras funciones para realizar operaciones: sqrt(), sin() … etc que podrian necesitar.

CONDICIONAL SIMPLE

if (<condición != 0>) <sentencia_1>

if (<condición != 0>) { <composición_secuencial> }

CONDICIONAL COMPUESTO

```
if (<condición != 0>) {
  <sentencias_cierto>
}
else {
  <sentencias_falso>
}
```

CONDICIONAL GENERALIZADO

```
switch (<expresión_escalar>) {
  case <cte.1>: <sentencias>; [break;]
  case <cte.2>: <sentencias>; [break;]
  ...
  default: sentencias;
}
```

Iteraciones

for (equivalente al para algorítmico)

para <var> desde <v_ini> hasta <expr. fin> hacer

    <sentencias para>

fin para

```
for (i=1; i <= 10; i++)
  printf ("\n%d", i);
```

Iteraciones

while (equivale al mientras algorítmico)

mientras <condicion> sea cierta hacer <sentencias>

fin mientras

```
i=1;
while (i != 100)
   printf (“\n%d”, i++);

i=100;
while (i != 0)
{
   printf (“\n%d”, i);
  i = i-1;
}
```

Iteraciones

do while (equivale al repetir algorítmico)

repetir

<sentencias>

hasta que <condición> sea falsa

```
do {
scanf ("%d", &num);
} while (num != 0);
```

Funciones

Una función en C es un fragmento de código que se puede llamar desde cualquier punto de un programa. En C podemos diferenciar entre dos tipos de funciones:

a) Aquellas cuyo tipo de retorno es void (nulo), equiparables a lo que denominamos módulo genérico tipo procedimiento.

b) Aquellas cuyo tipo de retorno es un tipo de dato (como int, double o cualquier otro), equiparables a lo que denominamos módulo genérico tipo función.

Funciones de E/S

Escritura:

int printf (“<cadena control>”, <argumentos>);

La cadena de control especifica el formato y el número de argumentos

Los argumentos son las variables o expresiones a escribir

Devuelve nº de argumentos correctamente escritos

En la cadena de control pueden aparecer:

constantes carácter o cadena, que aparecen como tales,

constantes tipo carácter: \b, \n, \t, \’, \”,...

descriptores de formato, %?, que indican el formato con el que mostrarán los argumentos , donde ? es uno de los siguientes:

%c carácter sencillo

%d entero

%e real en notación científica

%f real simple precisión en notación científica

%g el más corto de %e, %f

%o octal

%x hexadecimal

%s cadena de caracteres

%u decimal sin signo

Los descriptores se pueden especificar mediante

%m.n?

Ejemplos: %10d %10.5f %20s

Ejemplos:

```
printf (“Un entero en una linea: %d \n”, entero);
printf (“Dos enteros en una linea: %d, %d\n”, entero1, entero2);
printf (“Una cadena %s de caracteres.\n”, cadena);
printf (“Varios %d tipos %c mezclados %s\n”, entero, caracter, cadena);

for (i=1; i <= 100; i++) {
   printf (“%d \t”, i);
   if (i % 25 == 0) printf (“\n”);
}
```

Lectura:

int scanf (“<cadena de control>”, <argumentos>);

- cadena de control: idem que en el printf
- Devuelve número de argumentos leídos correctamente
- Los argumentos son las variables o expresiones a leer.
- Los argumentos que sean de tipo dato-resultado o resultado y sean de tipos escalares, deben llevar delante el operador &:
- &: indirección, pasa la dirección de la variable y no su valor.

Esta es la forma en la que C modifica los valores de los argumentos de una función:

```
scanf (“%d”, &entero);                    /* lee un entero */
scanf (“%d, %d”, &entero1, &entero2);     /* ha leido dos enteros*/
scanf (“%s”, cadena);                     /* leida una cadena, que no es escalar*/
scanf (“%d %c %s\n”, &ent3, &c, string);  /* lee un entero, ent3, y un carácter, c,
ambos escalares, y una cadena, string, que no es escalar*/
```

Lectura:

Descriptores de formato para la lectura con scanf

%s	cadena de caracteres: string (sin &) porque es un array de caracteres
%c	carácter sencillo: &caracter
%d	entero: &entero
%e, %f, %g	real en notación científica, con signo y exponente opcionales: &real
%o	octal: &octal
%x	hexadecimal: &hexa
%u	decimal sin signo: &sin_signo

Arreglos:

Podemos tener en C vectores de cualquier tipo básico o compuesto:

```
<tipo> <variable> [<dimension>]+;

int enteros[10], matriz[10][20];
char cadena[20];

matriz[9][19] = 12;
cadena[0] = 'c';
for (i=0; i<10; i++) scanf ("%d", &enteros[i])
        scanf ("%s", cadena);
```

- Los índices variarán (0..N-1)
- C no comprueba que se exceda el número de elementos del vector
- Los nombres de los vectores (ej. cadena) son equivalente a la dirección del vector.
- &enteros[i] accede a la dirección del elemento i en el vector enteros (véase paso por referencia).

- En C sólo hay funciones para representar algoritmos con nombre:
  [<tipo>] nombre_función ([<lista parámetros>])
  [<declaración de lista parámetros>]
  {
  <declaración tipos>
  <declaración variables>
  <composición secuencial acciones>
  [return <expresión tipo de la función>]
  }

- Todas las funciones devuelven un valor (que define su tipo). Por defecto, devuelve un entero (int) que no hace falta declarar
- Mediante return se devuelve (un valor concreto y) el control a la función que la haya llamado.

Funciones y argumentos

Existen dos formas de pasar argumentos:

- por valor (equivalente a par. dato), y es la forma por defecto no modifica el valor del argumento
- por referencia (equivalente a par. dato-resultado) se puede modificar el valor del argumento.

- El paso por referencia implica pasar la dirección de la variable (direccionamiento indirecto) mediante el operador & (indirección).

Ejemplo, en el caso de printf y scanf, que son funciones de la biblioteca estándar de C, los argumentos se pasan:

- por valor a printf, ya que no los modifica
- por referencia (mediante &) a scanf porque si los modifica

Funciones y argumentos

```
* x e y son dos enteros que se pasan por valor a la función multiplica
    res es un entero, que se pasa por referencia, y se modifica con el resultado de la multiplicación
    x, y, resp son enteros que se pasan por valor a la función printf */
main() {
int x, y, res;
x = 10; y = 20;
multiplica (x, y, &res);
printf ("\nEl resultado de multiplicar %d por %d es %d", x, y, res);
}
```

Funciones y argumentos

```
/* a y b son enteros y se pasan por valor  no se modifican
   c es un puntero a entero (dirección a un entero).
Para modificarlo hay que acceder a su contenido mediante *c
*/
multiplica (a, b, c)
int a, b, *c;
{
  *c = a * b;
}
```

Tarea 1

- Instalar Python 3.10.2 (Disponible para Windows y MAC en Python Download
- Instalar Java 11 JDK (Disponible para Windows, MAC y Linux en Java Download
- Instalar GraalVM (GraalVM Website)
- Instalar GIT (Instrucciones en GIT Installation)
- Crear cuenta en GITHub (GITHub Website)
- Instalar client GITHub (GITHub Desktop)
- Instalar IDE de su preferencia (Visual Studio Code es bueno si no tienen alguno)
- https://www.youtube.com/watch?v=VdGzPZ31ts8 (Curso de GIT)

¿Preguntas?