Principios de Programación

El lenguaje C

1. Identificadores, constantes y variables

1.1. Conceptos de memoria

Los nombres de variable como x, y, suma corresponden a localizaciones o posiciones en la memoria de la computadora.

Cada variable tiene un nombre, un tipo y un valor.

Como ya vimos, tenemos restricciones en nombres de variables y constantes. Deben estar formados por letras o digitos o guión bajo (_) y el primer caracter debe ser una letra. Letras pueden ser minusculas o mayusculas y estas se consideran diferentes.

Hay palabras reservadas como por ejemplo: **if, else, int, float**, etc.

1.2. Declaración de constantes

Podemos declarar constantes de dos modos diferentes: con #define que asocia un valor a un identificador, o con la instrucción:

const float PI 3.1415926;

que declara que PI es una variable de tipo punto flotante y que es constante (no se puede modificar, cualquier intento de modificarla da error).

En los programas que vimos antes, cuando se ejecuta la sentencia:

scanf(" %d", &x);

el valor escrito por el usuario se coloca en la posición de memoria a la cual se le ha asignado el nombre x.

Si por ejemplo el usuario ingresa el valor 15 este se colocará en la posición de x.

Siempre que se coloca un valor en una posición de memoria, este valor sustituye cuaquier valor anterior que tuviera la variable.

1.3. Tipos de datos simples

Tenemos los siguientes tipos de datos en C con sus largos respectivos:

char	1 byte
bool	1 byte
int	2 bytes
short	2 bytes
long	4 bytes
unsigned	4 bytes
float	4 bytes
double	8 bytes

char almacena caracteres. bool es el tipo de los valores de verdad que son true y false. Podemos colocar en vez de true 1 y en vez de false 0.

las variables int pueden ser calificadas como short, long o unsigned. short y long se refieren a distintos tamaños, unsigned son siempre positivos.

Las declaraciones de estos enteros son:

```
short int x;
long int y;
unsigned int z;
```

la palabra int se puede omitir.

El tamaño de los tipos depende del sistema operativo.

1.4. Valores constantes

Para enteros se utiliza el número correspondiente, por ejemplo: 123, 5, -8. Una constante entera que es muy grande para entrar en un int se toma como long.

Para reales, se puede utilizar la notación científica : 123.45e-7 o 0.12E3. Las constantes punto flotante se almacenan como double.

Una constante de caracter se coloca entre comillas simples: 'a', 'B', '\ n'. El valor de una constante de caracter es el valor numérico del caracter en la máquina. Por ejemplo en la codificación ASCII el 0, '0' es 48. Por ejemplo, supongamos tengo el siguiente programa:

```
main ()
{
      char c='a';
      printf("El valor de c es: %c\n", c);
      printf("El valor numerico de c es: %d\n", c);
      system("PAUSE");
}
```

imprimira:

```
El valor de c es: a
El valor de c es: 97
```

Las constantes de caracter, representan también caracteres como ser, salto de linea, tabulares

1.5. Declaraciones

Todas las variables deben ser declaradas antes de su uso. Una declaración especifica un tipo y es seguida por uno o más nombres de variables.

Podemos inicializar las variables en su declaración, como por ejemplo en:

```
int x=5;
int y=7;
long f1=0,f2=3.5;
```

2. Asignación y aritmética en C

2.1. Asignación

Una asignacion en C es una instrucción de la forma var=e donde var es una variable y e es una expresión del mismo tipo que var.

Ejemplos:

- Supongamos declaramos int x,y; podemos realizar la asignación: x=2+y.
- Supongamos declaramos bool b1,b2; podemos realizar la asignación b1=!b2 donde ! representa el not.
- Supongamos declaramos float f1=9.3,g1=3; podemos realizar la asignación f1=g1/f1;
- La asignación es una expresión y varias asignaciones se asocian de derecha a izquierda. Puedo escribir:

  x=y=z+2

  esto lo que hace es: calcula z+2, lo asigna a y y luego asigna y a x. No podemos escribir por ejemplo x=y+1=z;

2.2. Operaciones aritméticas

La mayor parte de los programas C ejecutan cálculos aritméticos. Los operadores aritméticos de C son:

Operación	Operador	Ejemplo
Suma	+	f+7
Substracción	-	p-c
Multiplicación	*	b*m
División	/	x/y
Módulo	%	r

Utilizamos símbolos especiales, no necesariamente iguales a los que utilizamos en matemática.

El asterisco (*) indica multiplicación y el signo de porcentaje (%) denota módulo (resto de la división).

Los operadores aritméticos son operadores binarios (se aplican a dos argumentos), excepto el menos (-) unario que calcula el opuesto.

Estos operadores están *superpuestos* es decir tienen significado cuando se aplican a distintos tipos de datos como ser enteros y reales excepto módulo que tiene significado solo cuando se aplica a enteros.

En particular la división cuando se aplica a enteros *trunca* es decir redondea a un entero (la división de enteros da como resultado un entero).

ejemplos:

7/4 = 1
7,0/4 = 1,75
7,0/4,0 = 1,75
7 %4 = 3
17 %5 = 2

Un error común en programación es la división por 0.

Los paréntesis se utilizan del mismo modo qu en matemáticas. Por ejemplo para multiplicar a por (b+c) escribimos $a * (b + c)$.

C calculará las expresiones aritméticas en una secuencia precisa determinada por las reglas de precedencia de operadores que siguen y que en general son las mismas que en matemáticas, a saber:

1. Primero se calculan las expresiones contenidas en paréntesis. En el caso de paréntesis anidados se evalua primero la expresión en el par de paréntesis más interno.

2. A continuación se evalúan las operaciones de multiplicación división y módulo. Si hay varias se evalua primero de izquierda a derecha. Se dice que estas operaciones tienen el mismo nivel de precedencia.

3. Por último se calculan las operaciones de suma y resta. Igual que en el caso anterior si hay varias de estas operaciones se asocia de izquierda a derecha y se dice que tienen el mismo nivel de precedencia.

Ejemplo

- x= a*(b+c)/d+e*5

  primero evaluamos la expresión entre parentesis: (b+c) a continuación a* el resultado de la suma anterior a continuación dividimos por d luego calculamos e*5 y por ultimo sumamos el resultado de a*(b+c)/d a e*5.

  Podemos para simplificar agregar parentesis y luego evaluar. Obtenemos la siguiente expresión.

  ((a*(b+c))/d)+(e*5)

- z=p*r %q+w/x-y

  agreguemos parentesis, obtenemos:

  (((p*r) %q)+(w/x))-y

2.3. Operadores de igualdad y relacionales

Las expresiones booleanas (expresiones que tienen como valor true o false) se forman utilizando los operadores de igualdad y los operadores relacionales.

Los operadores relacionales tienen el mismo nivel de precedencia y se asocian de izquierda a derecha.

Los operadores de igualdad tienen un nivel de precedencia menor que los relacionales y también se asocian de izquierda a derecha.

Los operadores son los siguientes:

Operador de igualdad	significa	ejemplo de condición	es verdadero si
==	igualdad	x==y	x es igual a y
!=	distinto	x!=y	x es distinto a y

Operador relacional	significa	ejemplo de condición	es verdadero si
>	mayor	x>y	x es mayor que y
<	menor	x<y	x es menor que y
>=	mayor o igual	x>=y	x es mayor o igual a y
<=	menor o igual	x<=y	x es menor o igual a y

2.4. Operadores booleanos

Los operadores booleanos o conectivos lógicos son:

Operador	Significado	ejemplo	verdadero si
&&	AND	x && y	x e y son verdaderos, falso en otro caso
||	OR	x || y	verdadero si x o y son verdaderos, falso en otro caso
!	NOT	!x	verdadero si x es falso, falso si x es verdadero

2.5. Operadores de asignación

C dispone de operadores de asignación que *abrevian* las expresiones de asignación. Por ejemplo,

c=c+3;

se puede escribir en la forma

c+=3;

En general,

var = var op expr

donde $op \in \{+, -, *, /, \&\}$ puede ser escrito en la forma

var op = expr

Por ejemplo:

Operador	ejemplo	equivalente a
+=	c +=7	c=c+7
-=	d -=4	d=d-4
=	e*=5	e=e*5
/=	f/=3	f=f/3
%=	g %=9	g=g %9

2.6. Operadores incrementales y decrementales

C dispone de operadores incremental unario (++) y decremental unario (–). Podemos utilizar c++ en vez de c=c+1 o c+=1.

Si los operadores son colocados antes de la variable (++c,–c) se conocen como operadores de preincremento y predecremento, si se colocan después (c++,c–) se conocen como operadores de postincremento y postdecremento.

El preincrementar o decrementar una variable hace que la variable primero se incremente o decremente en 1 y a continuación el nuevo valor de la variable se utilizará en la expresión en la cual aparece.

por ejemplo:

```
int c;

c=5;
printf(" %d\n", ++c);
printf(" %d\n", c);
printf(" %d\n",–c);
```

el primer printf incrementa c, luego se imprime 6. El segundo printf imprime 6 y el tercero 5.

El postincrementar o decrementar una variable hace que el valor de la variable primero se utilize en la expresión en la cual aparece y luego se incremente o decremente en 1.

por ejemplo:

```
int c;

c=5;
printf(" %d\n",c++);
printf(" %d\n",c);
printf(" %d\n",c–);
printf(" %d\n",c);
```

el primer printf imprime c (5) y luego lo incrementa. El segundo printf imprime 6 y el tercero 6 y luego decrementa. El cuarto printf imprime 5.

2.7. Operador condicional

Es el único operador ternario de C (utiliza tres operandos). Los operandos junto con el operador condicional conforman una expresión condicional. Por ejemplo:

grado >= 60 ? printf("Aprobado\n") : printf ("Reprobado\n");

La idea es : se evalua la primer expresión que debe dar como resultado un booleano. Si la expresión evalua a verdadero se evalua la segunda expresión (la que está a la derecha de ?), si la expresión evalua a falso se evalua la tercer expresión (la que está a la derecha de ':').

2.8. Precedencia y Asociatividad

La precedencia y asociatividad de los operadores vistos hasta ahora es:

Operador	asociatividad	tipo
()	izq a der	paréntesis
++, –, -, !	der a izq	unario
,/, %	izq a der	multiplicativo
+,-	izq a der	aditivo
¡,¡=,¿,¿=	izq a der	relacional
==,!=	izq a der	igualdad
&&	izq a der	AND lógico
——	izq a der	OR lógico
?:	der a izq	condicional
=, +=, -=, *=, /=, %=	der a izq	asignación

Ejemplos de precedencia

- x==y+1>=z-3

  coloquemos paréntesis: (x==((y+1)>=(z-3)))

- x=y+1!=z+3

  coloquemos paréntesis:

  x=((y+1)!=(z+3))