**TEMA #2:** Elementos Básicos del lenguaje de Programación

# Elementos de un Programa.

## Identificadores y alfabeto

Los identificadores son cadenas alfanuméricas, o sea, cadenas de letras y dígitos, que sirven para identificar o nombrar, algún elemento de un programa. En la mayoría de los lenguajes de programación estas cadenas deben comenzar con letra y pueden contener el símbolo subrayado, “\_”. Se usan para denotar variables, constantes, módulos, etc.

La cantidad de caracteres que se admite para formar dichas cadenas puede variar de un lenguaje a otro e incluso de una versión a otra de un mismo lenguaje. Por lo general no se permiten letras acentuadas como parte de los identificadores.

Algunos compiladores de Pascal o cualquiera de sus versiones no diferencian las minúsculas de las mayúsculas dentro de cadenas usadas como identificadores, otros como los de C y C++ sí hacen distinción entre mayúsculas y minúsculas.

Ejemplos de identificadores válidos en C++

1. A
2. Alfa
3. Radio1
4. CentroDeMasa
5. Perimetro\_Del\_Cuadrado

Note que como C++ es sensible a la mayúscula, el primer identificador no sería el mismo que “a”, ni el segundo sería el mismo que “alfa” o “alFa”, etc.

Los estándares de codificación más difundidos actualmente entre los programadores usan el estilo mostrado en el ejemplo válido no. 4, o sea, cada palabra que compone el identificador comenzando con mayúscula y el resto de las letras minúsculas.

Ejemplos no válidos en C++

1. 1Beta ← no puede comenzar con número
2. Aα3 ← α es un carácter no permitido
3. BaseDeTriángulo ← no se permiten caracteres acentuados (á)
4. Longitud-Cilindro ← no se permite el carácter –

Recomendaciones para el uso de identificadores.

1. Elegir identificadores que al leerse refieran el contenido del elemento en cuestión.
2. Elegir identificadores que sean pronunciables y entendibles.
3. No usar abreviaturas a menos que sean muy claras.
4. Evitar usar caracteres numéricos que puedan confundirse con letras como son los casos de 1(uno) y l (ele minúscula), 2 y Z, 0 y o, etc.

# 

## Palabras reservadas

Las palabras reservadas son cadenas de caracteres alfanuméricos cuyo uso, como su nombre lo indica, está reservado para el compilador. Esto significa que no se pueden utilizar como identificadores en los programas.

Ejemplos

int, if, else, for, do, while, char, struct, void, return

Existen otras muchas palabras reservadas en el lenguaje. Para poder diferenciarlas de los demás identificadores en los documentos se usarán en negritas. Así se diferencian, generalmente, en los editores de código de los ambientes de desarrollo.

# Tipos de datos

Un tipo de dato define:

* un tipo de valor (numérico, de caracteres, lógico, etc.)
* un rango de posibles valores de ese tipo, y
* la cantidad de memoria que se necesita para almacenarlo.

Los tipos de datos se utilizan para construir otros identificadores contenedores de datos.

Algunos ejemplos de tipos de datos son:

* el tipo definido por todos los caracteres alfanuméricos incluyendo letras mayúsculas, minúsculas, dígitos y el resto de los caracteres posibles y que necesitan sólo un byte de memoria para almacenarse,
* el tipo definido por los valores lógicos verdadero y falso y que necesitan sólo un byte de memoria para almacenarse,
* el tipo definido por los valores numéricos enteros, con o sin signo, entre –2147483648 y 2147483647 y que necesitan 4 bytes de memoria para almacenarse,
* el tipo definido por los valores numéricos reales entre 5.0 x 10^–324 y 1.7 x 10^308 de 15 a 16 dígitos significativos y que necesitan 8 bytes de memoria para almacenarse,
* etc.

A cada tipo de dato se le asocia un identificador a través del cual se hace referencia al mismo.

Los lenguajes de programación definen algunos tipos de datos que los programadores pueden utilizar en sus programas, los cuales se conocen como tipos ***estándar***. Estos pueden variar de un lenguaje a otro e incluso entre las distintas versiones de un mismo lenguaje, por lo que es responsabilidad del programador conocer exactamente cuáles tipos de datos estándar existen y las características de aquellos que requiera utilizar. La información sobre los tipos estándar de cada compilador puede encontrarse en la ayuda de éstos.

También se utilizan en programación los términos tipo de dato ***simple*** y tipo de dato ***estructurado***. Los tipos ***simples*** son aquellos que hacen referencia, en cada instante, a un solo valor de los posibles. Para él se reserva una sola localización de memoria en la que se puede almacenar un valor o leer el valor que allí esté almacenado. Los ***estructurados***, como su nombre lo indica, son estructuras de datos que agrupan varios valores simples. Por esta razón los tipos simples constituyen la base de cualquier otro tipo de dato estructurado. En la primera parte de la asignatura se trabajará solamente con tipos de datos simples.

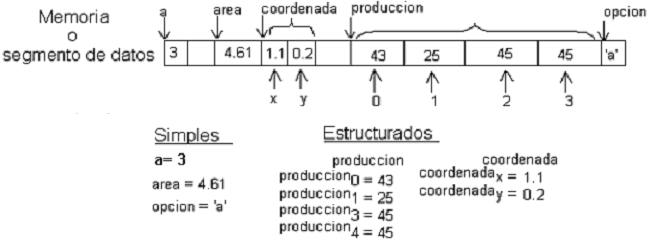
Se dice que un tipo es **ordinal**, si sus valores responden a un orden dado, por lo que es posible hablar del predecesor y sucesor de cualquiera de ellos. Nótese que el concepto de ordinal no es inherente a los números reales.

Algunos de los tipos de datos simples estándar definidos en C++

* **short, int** y **long** para enteros
* **float** y **double** para Reales. Se diferencian en rangos y precisión
* **char** para Caracteres
* **boolean** para Lógicos (valores posibles: false, true. Es ordinal, false precede a true)
* **byte** para datos binarios

# Variables

En los lenguajes de programación se le llama variable a todos los identificadores a través de los cuales se pueden almacenar datos en la memoria durante la ejecución de los programas y acceder a ellos. Toda variable tiene un identificador, un tipo y un valor. Las variables pueden referenciar datos simples o datos estructurados lo cual depende de la forma en que haya sido declarada la variable. El siguiente esquema ilustra de manera gráfica el almacenamiento de algunas variables en memoria:



Nótese que a cada variable se asocia una zona de memoria de un tamaño suficiente, según el tipo de dato, para almacenar sus valores.

Para declarar las variables se usa el identificador de tipo de la variable seguido del identificador de variable. La sintaxis (regla de escritura) es:

<identificador de tipo> <identificador de variable>

donde:

identificador de tipo: identificador de tipo estándar o algún otro tipo previamente definido por el programador, que permite al compilador conocer la memoria que debe reservar para el uso de dicha variable y además hacer chequeos de compatibilidad durante la ejecución del código.

identificador de variable: es el nombre, escogido por el programador para hacer referencia, durante el cuerpo del código, al valor almacenado en la zona de memoria correspondiente a la variable.

La declaración de variable, en C++, se realiza donde se usa por primera vez la variable o en cualquier parte del cuerpo previo a que sea usada por primera vez. Una misma variable debe ser declarada una sola vez. El alcance de una variable empieza en su declaración y se extiende hasta la llave derecha de cierre (**}**) del bloque donde se declaró. Por lo tanto, enunciados anteriores a la declaración de la variable no pueden hacer referencia a ella, aun si están dentro del mismo bloque. De la misma manera no pueden ser referenciadas por enunciados posteriores al cierre del bloque.

Cuando se hace la declaración de una variable se está indicando al compilador que en tiempo de compilación reserve en la memoria interna el espacio necesario para guardar el valor de dicha variable. El espacio reservado sólo será usado para este fin durante la ejecución de la aplicación y a través del identificador de la variable se podrá acceder a él ya sea para guardar o recuperar la información. A este tipo de reservación se le llama **reservación estática**.

A diferencia de las constantes, las variables pueden tomar diferentes valores durante la ejecución del código. Cada vez que una variable cambia su valor se actualiza la zona de la memoria donde ella se almacena, por lo que el valor almacenado allí anteriormente se pierde.

Ejemplos

**float** DaimetroInterior;

**int** CantidadTrabajadores;

**char** Consonante;

**bool** EsVisible;

En la variable *DiametroInterior*, se pueden almacenar valores reales, mientras que en *CantidadTrabajadores* se pueden almacenar números enteros. La variable no numérica *Consonante* permitirá manipular caracteres (uno en cada instante de tiempo). *EsVisible* podrá guardar, uno de los valores *true* o *false*.

Si en algún momento de la ejecución del código se trata de almacenar en una de estas variables un valor no permitido el compilador generará un mensaje de error y se detendrá la ejecución. A este chequeo realizado por el compilador se le denomina chequeo de compatibilidad de tipos.

Las variables en C++ pueden ser inicializadas con un valor en la declaración como se muestra a continuación:

**int** CantidadTrabajadores= 100;

En una declaración pueden declararse más de una variable, en ese caso se usa la coma “,” para separar las variables entre sí. Todas las variables serán del tipo especificado al inicio de la declaración. Ejemplo:

**float** r, s= 1.5;

En este ejemplo se reserva espacio para una variable flotante *r* y para otra variable flotante *s*. Esta última se inicializa con el valor *1.5*.

Ejemplo del uso de variables Memoria

…

**int** a = 20, b = 10;

… Localización Localización

… reservada para **a** reservada para **b** es

… es inicializada en 20 inicializada en 10

…

**int** suma;

…

… Localización

… reservada para **suma**

Nótese del ejemplo que la declaración es la que indica que se haga la reservación de espacio en la memoria. Por tanto:

* No se puede usar una variable que no se haya declarado previamente.
* Se reservarán tantos bytes como requiera el tipo especificado en la declaración.

Es necesario explicar que aunque el programador no asigne ningún valor a una variable, una vez que ésta se ha declarado, el espacio de memoria quedará reservado y no se puede asumir que está vacío. Por el contrario, allí siempre habrá algún valor. Una forma de comprobar esto es declarando una variable e inmediatamente mostrar su contenido. Hay algunos compiladores que inicializan las variables en valores por defecto, por ejemplo las numéricas en 0, las lógicas en false, etc., pero muchos no lo hacen, por lo que una buena práctica es asumir como responsabilidad del programador inicializar las variables siempre que sea necesario. En el ejemplo anterior se ha representado que el programador no ha asignado ningún valor a la variable *suma*, no que su localización esté vacía.

# Constantes

En muchas situaciones se utilizan valores constantes para realizar diferentes cálculos. Tal es el caso, por ejemplo, de la constante Π tan utilizada en las matemáticas para cálculos trigonométricos. Como su nombre lo indica, las ***constantes*** son valores que no varían en ningún cálculo.

Declarar o definir una constante en un programa significa declarar un identificador a través del cual se hará referencia a un valor constante en el programa, o sea, un identificador que se asocia a un valor que no cambia durante la ejecución.

Los lenguajes de programación incluyen muchas constantes ya definidas, que el programador puede utilizar y generalmente brindan a éstos la posibilidad de declarar sus propias constantes.

Una constante puede ser declarada aplicando el modificador **final** a un campo o una variable local. Una vez asignado un valor a un campo declarado como **final**, no es posible modificarlo. C++ permite declarar un campo **final** sin asignarle valor alguno, sin embargo el compilador se encarga de garantizar que un campo **final** no sea utilizado sin antes haber sido inicializado, para ello fuerza a inicializarlo en cada constructor. Ejemplo de constante:



# Operadores

Los operadores, según la cantidad de operandos que combinan, pueden ser:

* Unarios, operan sobre un solo operando. Tal es el caso del cambio de signo en los números.
* Binarios, operan sobre dos operandos. Ejemplo, el operador de suma, el de multiplicación, etc.
* Ternarios, operan sobre tres operandos.

Y según el resultado de la operación en:

* Aritméticos: operan sobre operandos numéricos.

Suma: +

Diferencia: –

Multiplicación: \*

División: / (entre enteros el resultado siempre será un entero, la parte entera del resultado. Si al menos uno de los operandos es algún tipo de número real, el resultado de la expresión es real)

Módulo (Resto de la división): %

Cambio de signo: – (unario)

int a =3;

int b = a++;

int c = ++a;

* Lógicos: Estudiar en el punto 3 del Estudio Independiente.

Los operandos de una expresión pueden ser cualquier identificador contenedor de datos como constantes, variables y otros que se verán más adelante, así como otras expresiones. Los identificadores de tipos no pueden formar parte de una expresión, pues no son contenedores de datos.

En el caso particular de C existe un único operador ternario (que opera sobre tres operandos) que es el Operador Condicional que se estudiará en próximas conferencias.

Precedencia de los operadores:



|  |  |
| --- | --- |
| **Operadores aritméticos** | **Prioridad** |
| - (Cambio de signo) | 0 |
| \*, / , % | 1 |
| +, - | 2 |

Al evaluar una expresión se realizan las operaciones de izquierda a derecha, y según la precedencia de los operadores que aparecen en ella. Para violar la precedencia de los operadores, se pueden utilizar paréntesis () dentro de las expresiones, con las mismas reglas conocidas del álgebra. No se pueden usar para esto los corchetes, pues están reservados para otro uso en el lenguaje.

**Asignación.**

La sentencia de asignación es aquella a través de la cual se asigna un valor a una variable. Se utiliza para ella el operador de asignación = y responde a la siguiente sintaxis:

<identificador de variable>= <expresión>;

Esta instrucción asigna a la variable, especificada en la parte izquierda, el valor resultante de la evaluación de la expresión dada. O sea, primero se evalúa la expresión y luego el resultado se deposita en la localización de memoria asociada a la variable en cuestión. El sentido de la asignación siempre es de derecha a izquierda, por lo que la variable que debe recibir el valor debe encontrarse siempre a la izquierda de la asignación, que se representa por “=”. Es importante resaltar que una asignación no es una igualdad, aunque a veces por comodidad se lea de esa forma.

El tipo de la variable de la izquierda tiene que ser compatible al del resultado de la evaluación de la expresión de la derecha, de lo contrario el compilador generará un error de incompatibilidad de tipos.

Ejemplos

1. x= x + 1;
2. **float** AreaCirculo = PI \* Radio\* Radio;
3. **float** AreaTriangulo = Base \* Altura;
4. **int** Resto:= k % n;
5. **float** AreaSemiCirculo:= AreaCirculo / 2;

Todos los ejemplos anteriores requieren, por supuesto, que los identificadores que no aparecen declarados hayan sido previamente declarados, tal es el caso de  *x*, *Radio*, *Base*, *Altura*, *k*, *n*.

Combinaciones:

|  |  |  |
| --- | --- | --- |
| ***Operador*** | ***Uso*** | ***Descripción*** |
| += | op1 += op2 | Adiciona los operandos op1 y op2, y asigna el resultado a op1. Esta expresión es equivalente a  op1 = op1 + op2. |
| -= | op1 -= op2 | Substrae el operando op2 del operando op1, y asigna el resultado a op1. Esta expresión es equivalente a op1 = op1 – op2. |
| \*= | op1 \*= op2 | Multiplica los operandos op1 y op2, y asigna el valor de resultado a op1. Esta expresión es equivalente a op1 = op1 \* op2. |
| /= | op1 /= op2 | Divide el operando op1 por el operando op2, y asigna el valor del resultado a op1. Esta expresión es equivalente a op1 = op1 / op2. |
| %= | op1 %= op2 | Asigna el residuo de la división del operando op1 por el operando op2 al operando op1. Esta expresión es equivalente a op1 = op1 % op2. |

## Conversiones entre tipos de datos

(funciones ValueOf)

# Expresiones.

*Expresión*: conjunto de operandos combinados entre sí, a través de operadores. Los operadores indican la operación a realizar. A toda expresión se asocia un valor que depende del valor de cada operando en el momento de su evaluación y de las operaciones indicadas entre ellos por los operadores.

### Las expresiones según el resultado de su evaluación pueden ser:

### Numéricas, al evaluarlas se obtiene un valor numérico producto de la combinación de operandos numéricos a través de operadores aritméticos.

### Lógicas, al evaluarlas se obtiene un valor lógico, producto de la combinación de operandos del mismo tipo a través de operadores de relación, o de otras expresiones combinadas a través de operadores lógicos. Estudiar el contenido para el Taller # 1, en el punto 1 del Estudio Independiente y por el libro de texto.

Ejemplos de expresiones

1. x + y \* z
2. Alfa \* 22 – h
3. PI \* Radio \* Radio
4. (h + s) / (n \* q)

# Comentarios

Para aumentar la claridad del código es una buena práctica documentarlo con comentarios. Un comentario no es más que algún texto que se coloca dentro del código, para hacer éste más explícito, y que es ignorado por el compilador durante la ejecución, o sea, no hacen que la computadora realice ninguna acción. Los comentarios no generan código objeto en lenguaje de máquina.

En C++ los comentarios se pueden ser:

De bloque:

/\* aquí va el comentario..

que puede abarcar tantas líneas como se desee \*/

De línea:

**//** esto también es un comentario pero que termina con el fin de línea

Ejemplos de comentarios

* // Este programa calcula potencia entera
* /\*Último cambio realizado el 23-10-2005

se incluyó el campo PrecioUnitario que se utiliza

en el cálculo del importe final\*/

--------------Clase 2-------------------------------------------------------------------------------------------------

# Estructuras básicas de la programación

Al inicio de la década del 60 comenzaron a aparecer trabajos sobre la programación estructurada. De manera general, todos de una forma u otra coincidían en afirmar que para escribir códigos claros y correctos sólo se necesitaban tres estructuras de programación:

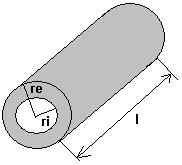
1. secuencial
2. alternativa
3. repetitiva

La *estructura secuencial*, es simplemente la idea de que las sentencias de un código son ejecutadas en el orden en que aparecen en él. A través de estructuras secuenciales se implementan fragmentos lineales de los algoritmos.

**Ejemplo de estructura secuencial**

Problema: Veamos como programar una secuencia que calcule el volumen de un cilindro hueco conocidos su radio interior, su radio exterior y longitud.

Primero analicemos el modelo matemático:

**Estructuras de selección o Condicionales**

Ver documento de Alternativas

--------------Clase 3-------------------------------------------------------------------------------------------------

**Estructuras de control o Ciclos**

Ver documento de Repeticiones

**Conclusiones**

En esta conferencia se han estudiado conceptos y elementos básicos de la programación que son independientes del lenguaje y para ejemplificarlos se ha utilizado la sintaxis del C/C++.

La comprensión de los conceptos expuestos en esta conferencia es de vital importancia para la compresión del resto del contenido del tema.

Una vez culminada la exposición los estudiantes deben ser capaces de responder:

¿Qué es un lenguaje de programación?

¿Qué es un programa?

¿Qué son los compiladores e intérpretes?

¿Qué es un programa fuente y un programa objeto?

¿Qué son y cómo se implementan en C++, los identificadores, las variables y las constantes? ¿Cuál es la regla de la precedencia de la declaración?

Los operadores y las expresiones aritméticas

La sintaxis y significado de la sentencia de asignación

Escribir alguna estructura secuencial sencilla en C++ con expresiones aritméticas y asignación

**Estudio independiente**

1. Estudiar para la comprobación de lectura del Taller # 1
2. Capítulo 1 del libro de texto
3. Notas de clases
4. Los operadores lógicos

Como se vio en la conferencia los operadores pueden ser numéricos y lógicos. A continuación se explican los operadores lógicos y los resultados de las expresiones lógicas según los operadores involucrados.

* Relacionales: operan sobre cualquier par de operandos que sean del mismo tipo.

Igual que: ==

Menor que: <

Menor o igual que: <=

Mayor que: >

Mayor o igual que: >=

Diferente de: !=

* Lógicos: operan sobre operandos lógicos.

Negación lógica: !

Conjunción lógica: &&

Disyunción lógica: **||**

El resultado de las expresiones que involucran a operadores lógicos se obtiene siguiendo lo especificado en las tablas de verdad de cada operador, las cuales se muestran a continuación:

&& (y lógico)

|  |  |  |
| --- | --- | --- |
| Operando 1 | Operando 2 | Resultado |
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

Note que sólo es **true** si ambos operandos son **true**

**||** (o lógico)

|  |  |  |
| --- | --- | --- |
| Operando 1 | Operando 2 | Resultado |
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

Note que sólo es false si ambos operandos son false

! (negación lógica)

|  |  |
| --- | --- |
| Operando | Resultado |
| true | false |
| false | true |

Note que el resultado siempre es lo contrario del operando.

Existen otros operadores que no aparecen en la relación anterior pues se utilizan entre operandos de tipos que no serán vistos por el momento.

Precedencia de los operadores, estudiados hasta ahora,teniendo en cuenta los operadores lógicos:

|  |  |
| --- | --- |
| **Operadores aritméticos y lógicos** | **Prioridad** |
| - cambio de signo | 0 |
| ! | 1 |
| \*, /, % | 2 |
| +, - | 3 |
| <,<=, >, >= | 4 |
| = =,!= | 5 |
| && | 6 |
| **||** | 7 |
| **=** | 8 |

### Recordemos que para violar la precedencia de los operadores, se pueden utilizar paréntesis dentro de las expresiones, con las mismas reglas conocidas del álgebra.

Al evaluar una expresión se realizan las operaciones de izquierda a derecha, y según la precedencia de los operadores que aparecen en ella.

Ejemplos de expresiones lógicas

1. EsVisible **&&** a > b /\*se supone que *EsVisible* es una variable **bool**  que ya ha

tomado valor y *a* y *b* son otras dos variables de cualquier tipo también previamente declaradas y con algún valor\*/

1. !(x = = z) **||** (b < 10) //Es equivalente a !(x = = z) **||** b < 10
2. !x = = z **||** b < 10 /\*Es diferente a la anterior aquí en lugar de negar el resultado de la igualdad

se niega el valor de x.\*/

1. !(x = = z) **||** b < 10 //Equivalente a 2
2. Ejercicios a resolver para el Taller #1
3. ¿Qué sucede con un virus si se aloja en RAM y usted apaga el equipo?
4. ¿Investigue cuál es la capacidad máxima en MB de un disquete de 3.5 pulgadas? Exprésela en KB.
5. Suponga que ha escrito 5 páginas del informe de programación o ha programado gran parte del código de una aplicación, cuando de repente se corta el suministro de energía eléctrica. ¿Qué sucede si usted no ha guardado la información en disco duro, o en algún otro medio de almacenamiento secundario? ¿Qué medidas adoptaría usted para evitar perder información?
6. En un bloque de un programa aparece el siguiente fragmento de código

{…..

**int** a = 10;

**int** b;

**int** c = a\* b;

…}

a) ¿En cuál memoria se reserva espacio para las variables a, b y c? \_\_\_Interna \_\_\_Externa

b) La memoria para las variables a, b y c se reserva durante la:

\_\_\_\_\_Compilación o \_\_\_\_\_ la Ejecución. Marque la respuesta correcta.

c) La memoria reservada para las variables a, b y c se libera cuando se muestra c.

\_\_\_\_\_Verdadero \_\_\_\_\_ Falso. Marque la respuesta correcta.

d)¿Qué resultado quedará almacenado en la variable *c*?

1. Declarar variables adecuadas para manipular los siguientes datos en una aplicación en C++:

* El resultado de la división (/) entre dos enteros
* Cantidad de estudiantes de una brigada
* Memoria ocupada por un fichero
* Cantidad de habitantes del planeta
* El nombre de una sala de hospital (desde la A hasta la J)
* La edad de una persona
* Si un estudiante está becado o no
* Si una cama de un hospital está ocupada o no
* El resto de la división entre dos números reales
* El factorial de un número

1. Escriba una secuencia lineal de código que conociendo la producción en unidades de cada una de las 3 líneas de producción de una fábrica, calcule la producción promedio, el por ciento que ésta representa de la norma conocida y la ganancia total de la fábrica conociendo el costo y precio unitario. Asuma que las tres líneas producen el mismo producto, y que la norma es igual para todas.

Indique con comentarios que operación se realiza en cada parte del algoritmo.

1. Diga el orden en que se realizan las operaciones en las siguientes expresiones:
2. i **<=** Cantidad **&&** **!**encuentra
3. (i **<=** Cantidad) **&&** **!**encuentra
4. existe **=** i **<=** cant-1
5. Evalúe las siguientes expresiones e indique el resultado

a) **bool** pertenece = x >= 10 **&&** x <= 20; para x con valor 15

b) **bool** pertenece = x > 5 **&&** x <= 10; para x con valor 5

c) **bool** pertenece = x < 5 **||** x > 10; para x con valor 15

d) **bool** equilatero = a == b **&&** a == c para a igual a 2,

b igual a 2 y c igual a 2

e) 7 + 3 \* 5 / 2 -1 == 13.5

f) 7 + 3 \* 5 / 2 -1 == 13

g) 2 % 2 + 2 \* 2 – 2 / 2 >= 3 **&&** (3 \* 9\* (3 + (9 \* 3 / (3)))) != 30

h) a == b **&&** a != b

1. Programe en C los siguientes enunciados
2. Una variable que recibe el resultado de una expresión que evalúa si tres variables tienen valor diferente.
3. Una variable que recibe el resultado de una expresión que evalúa si sólo dos de tres variables tienen el mismo valor.
4. Una variable que tome valor true si la variable c tiene almacenada una vocal ya sea mayúscula o minúscula, acentuadas o no.
5. Una expresión que devuelva true si entre las variables *x, y* y *z*, *x* tiene el mayor valor.
6. Una expresión que devuelva true si una variable tiene almacenado un número impar.
7. Una expresión que devuelva true si *x* es múltiplo de *y*
8. Programe en C fragmentos de código que permitan resolver los siguientes problemas. Simule la entrada de los datos con la orden ENTRAR dato1, dato2… y la visualización de los resultados con la orden MOSTRAR dato1, dato2,….
9. Un vendedor recibe cada semana un salario fijo más una comisión del 9% del total del promedio de las ventas realizadas en la semana. Conociendo el importe de cada una de las 5 ventas realizadas en la semana calcular lo que debe ganar en la semana y si está por encima de un valor dado almacenar true en una variable.
10. Almacenar true en una variable si el interés que se debe pagar por un préstamo en un banco es mayor que un valor dado. Se conoce la magnitud del préstamo, la taza de interés anual y la cantidad de días de vigencia del préstamo.

interés = préstamo \* taza \* días / 365