Universidad Nacional Autónoma de México

Facultad de Ingeniería

Estructura de Datos y Algoritmos I

Actividad 1

Bautista Corona Yahir

04/03/2021

Elementos sintácticos del lenguaje C

Los elementos sintácticos del lenguaje C son:

1. Operadores: permiten realizar alguna actividad específica, se clasifican en:
   a) Algebraicos
      + suma
      - resta
      % modulo
      / división
      * multiplicación
      = asignación

      NO EXISTE un operador para la potencia en el lenguaje C.

   b) Agrupación
      ( ) paréntesis

   c) Comparación
      < menor que
      <= menor igual que
      > mayor que
      >= mayor igual que
      == igual
      != diferente

NOTA: Observe que la asignación cuenta con un igual y la igualdad tiene dos iguales.

   d) Lógicos
      && (doble ampersand que indica Y)
      || (doble pipe que indica O)

Para resolver una operación es indispensable seguir la jerarquía de operadores como se sigue en matemáticas.
Y considerar que cuando dos operadores tienen el mismo nivel de prioridad o jerarquía dentro de una expresión, estos se evalúan de izquierda a derecha.
Para asegurar la realización de una operación en específico, deberá agruparse empleando paréntesis.

2. Constantes: son las cantidades que se pueden emplear y se clasifican en:
   a) Reales: pueden ser positivas o negativas y cuentan con parte entera y parte decimal. Existen también:
      i) Reales cortas, simples o sencillas (cantidades pequeñas)
      ii) Reales largas o de doble precisión (cantidades largas)
   b) Enteras: pueden ser positivas o negativas y cuentan con parte entera solamente. Existen también:
      i) Enteras cortas, simples o sencillas (cantidades pequeñas)
      ii) Enteras largas (cantidades largas)

c) Alfanuméricas
   i) Carácter (solo un símbolo, emplea apostrofes para indicar que es un carácter, por ejemplo: ‘u’)
   ii) Cadena (varios símbolos, más de uno, emplea comillas para indicar que es una cadena, por ejemplo: “12r:”)

3. Identificadores: permiten almacenar constantes solamente del mismo tipo del que son dichos identificadores. Existen los siguientes tipos:
   a) Variables: solo almacenan un valor a la vez. Las variables pueden ser de los siguientes tipos:
      i) Reales cortas, simples o sencillas (almacenan cantidades pequeñas)
      ii) Reales largas o de doble precisión (almacenan cantidades largas)
      iii) Enteras cortas, simples o sencillas (almacenan cantidades pequeñas)
      iv) Enteras largas (almacenan cantidades largas)
      v) Alfanumérico carácter

   b) Arreglos: almacenan varios datos del mismo tipo: los arreglos pueden ser de los siguientes tipos:
      i) Reales cortas, simples o sencillas (almacenan cantidades pequeñas)
      ii) Reales largas o de doble precisión (almacenan cantidades largas)
      iii) Enteras cortas, simples o sencillas (almacenan cantidades pequeñas)
      iv) Enteras largas (almacenan cantidades largas)
      v) Alfanumérico cadena

4. Declaraciones: permiten indicar cuales son los identificadores y el tipo de valores que pueden almacenar, es importante declararlos porque solo asi serán reconocidos en el programa. Existen dos tipos de declaraciones:
   a) Declaraciones locales; los identificadores son reconocidos en solo una parte del programa.
   b) Declaraciones globales: los identificadores son reconocidos en todo el programa.

5. Funciones de biblioteca: son aquellas que están programadas para realizar actividades puntuales, solamente son invocadas por el nombre y los parámetros correspondientes.

Programa básico en el lenguaje C

El lenguaje C contiene ya vistats en el algoritmo; secuencias, iterativas y condicionales. El programa básico en el lenguaje C sigue la siguiente plantilla:
NOTA: la sintaxis NO puede cambiarse, debe escribirse tal cual como se indica en cada caso.
La mayoría de las sentencias se escriben con minúsculas y esto no se puede modificar.

a) Directivas de pre-procesamiento (también llamadas bibliotecas o librerías)

   Sintaxis
   #include<archivo.h>

   - Archivo: se sustituye por el archivo que contiene las indicaciones necesarias que permiten que una estructura funcione correctamente.

b) Declaraciones globales
   Si algún identificador o función debe reconocerse en todo el programa debe declararse en esa parte

c) Función principal
   Sintaxis

```
int main()
        {
         Declaraciones locales
         Sentecias
         Return numero Entero;
         }
```

   NOTA: no hay espacio dentro de los paréntesis
   Declaraciones locales: si algún identificador debe reconocerse solo en la función principal debe declarase en esa parte
   Sentecias: son aquellas que veremos en clase
   Numero Entero: se sustituye por cualquier numero entero
   Al final de la sentencia return se coloca un; que NUNCA debe omitirse

   NOTA 2: todos los programas en el lenguaje C basados en el estándar ANSI C (que es el que manejamos en clase) se encuentran con formados por esta estructura, pueden o no existir declaraciones globales y locales de manera simultánea.

Sentencias o instrucciones básicas en el lenguaje C

La mayoría de las sentencias se escriben con minúsculas y esto no se puede modificar.

a) Declaraciones
   Para declarar los identificadores (por el momento empezaremos usando variables, más adelante veremos los arreglos) debemos detectar el tipo de dato o constante que van a manejar dichos identificadores (datos que van a almacenar) y declararlos empleando la palabra reservada correspondiente en el lenguaje C.
   La sintaxis para declarar es la siguiente:

   Tipo identificador;

   Donde tipo se sustituye por la palabra reservada correspondiente en el lenguaje C.

El ; NO se puede omitir.

Tipo de dato	Palabra reservada en el lenguaje C	Especificación de formato	Ejemplo de identificador	Declaración en el lenguaje C de varios identificadores
Entero corto	int	%d, %i	x, y, g	int x, y, g;
Entero largo	long	%d, %li	cont, h2	long, cont, h2;
Real corto	float	%f	g23, n, m	float g23, n, m;
Real largo	doublec	%lf	a, b, c	double a, b, c;
Alfanumérico	char	%c, %s	d, e	char d,e;

b) Lectura de datos (almacenamiento de datos cuando estos son pedidos)

Sintaxis
scanf(“Especificación de formato”,&identificación);

Por ejemplo:
Se darán 2 datos de tipo entero corto, consideremos la tabla vista y utilicemos la sentencia anterior para almacenarlos, es importante saber cómo y el orden en el que serán almacenados, si se hace separándolos con comas la instrucción quedaría:

scanf(“%d,%d”,&x,&y);

Se escoge de manera indistinta cualquiera de las dos especificaciones de formato, pero siempre debe haber una por cada dato que se va a almacenar. Se pone una coma separando dichas especificaciones de formato porque se indicó que se almacenaran separándolos por coma. Debe haber un & antes de cada identificador, dichos identificadores siempre deben listarse usando una coma.

Otro ejemplo:

Se darán 3 datos, uno de tipo real corto, otro de tipo real largo y por ultimo un entero corto, considerándose la tabla vista y utilicemos la sentencia anterior para almacenarlos, es importante saber cómo y el orden en el que serán almacenados, si se hace separándolos con enter la instrucción quedaría:

scanf(“%f%lf%d”, &n,&a,&g);

Deben existir tres especificaciones de formato, porque se almacenarán tres datos y se deben colocar en el orden en el que se va guardando los datos, pero siempre debe haber una por cada dato que se va a almacenar. Dichas especificaciones de formato van una tras otra sin espacio ni coma porque se indicó que se almacenarían separándolos con enter. Debe haber un & antes de cada identificador, dichos identificadores siempre deben listarse usando una coma.

c) Escritura de texto o datos (Impresión de texto o datos)

Se debe decidir lo que se imprimirá

1. Impresión de texto

   Se puede usar alguna de las siguientes sentencias:

   Sintaxis
   puts(“Texto”);

   o

   printf(“Texto”);

   El texto se escribe como se quiere ver en la pantalla

   Por ejemplo:
   puts(“Hola Amigooooos”);
   printf(“Nos vemos muy prontooooo…..”);

2. Impresión de datos almacenados en un identificador

   Sintaxis
   printf(“especificación de formato”, identificador);

   Ejemplo:
   Quiero ver el dato almacenado en la variable d, como es un carácter quedaría:
   printf(“%c”,d);

   Si deseo ver varios datos entonces debo poner especificación de formato por cada identificador cuyo dato almacenado en este voy a ver en la pantalla y debe decidirse como los quiero ver, si todos juntos, separados con comas, con espacios, con puntos, etcétera. Veré los datos en a,b,c separados con un espacio:

   printf(“%lf%lf%lf”,a,b,c);

3. Impresion de datos almacenados en un identificador y texto

   El texto se puede colocar antes o después del dato que quiero ver en la pantalla

   Sintaxis
   printf(“Texto especificación de formato”,identificador);

   Quiero ver el texto Los valores son: y posteriormente lo que tienen almacenado los identificadores x,y,g

   Ejemplo:
   printf(“Los valores son:%i,%i,%i”,x,y,g);

1. Dar dos números de tipo real y almacenarlos en N1 y en N2 respectivamente
2. Realizar N3= N1+N2
3. Mostrar N3

Pseudocódigo

```
INICIO
        N1,N2,N3: reales
        ESCRIBIR “Dar dos números reales”
        LEER N1, N2
        HACER N3:= N1+N2
        ESCRIBIR N3
FIN
```

Uso de comentarios al momento de programar

En el lenguaje C es muy común hacer uso de comentarios para colocar notas que nos permitan entender el código fuente que se está programando.

Existen dos tipos de comentarios:

a) Comentarios en una sola línea
   Solo permiten hacer un comentario en una sola línea en el código fuente, para esto se anteponen dos diagonales.

   Por ejemplo:

```
// este programa calcula una suma
```

b) Comentarios en varias líneas consecutivas
   Permiten hacer comentarios en un bloque (varias líneas consecutivas) en el código fuente, para esto se coloca /* antes de empezar los comentarios y se escribe */ cuando se termina el bloque de comentarios.

   Por ejemplo:

```
/* este programa calcula una suma y para ello
Es necesario introducir dos números
Que finalmente darán un resultado*/
```

Instrucciones de selección (parte 1)

En el lenguaje C es muy común hacer uso de las instrucciones de selección, también llamadas instrucciones de condición o condicionales.

Una instrucción de selección permite que el control del programa se transfiera a un determinado punto de ejecución, y esto depende de cierta condición sea verdadera o no.
Existe tres tipos básicos de instrucciones capaces de controlar el flujo de ejecución de un programa: las sentencias (if-else, switch-case) y la expresión condicional (también llamada temaría) ?

1. If-else
   Hace referencia a una condicion simple ya que solamente tiene dos posibles respuestas, cuando se cumple la condicion evaluada o cuando no se cumple.

   Sitaxis
   ```
   if (conclusión)
   {
       Sentencias
   }
   else
   {
         Sentencias
   }
   ```

   La condición debe estar bien construida siguiendo la sintaxis:

   variable1 operadorComparación variable2_o_valor1

   Donde variable1 puede ser cualquier variable que se desee compararse contra otra variable (variable2) o contra un valor especifico (valor1).

   El operadorComparación utilizado solo puede ser cualquiera de los vistos en clase y que se muestran en la tabla siguiente:

Operador	Significado
<	Menor que
>	Mayor que
<=	Menor igual que
>=	Mayor igual que
==	Igual
!=	Diferente

   Si la condición evaluada se cumple, es decir, es verdadera entonces se ejecutan las sentencias que están dentro de las llaves del if, si dicha condición NO se cumple entonces se ejecutan las sentencias que están dentro de las llaves del else, pues else hace referencia al caso contrario.

   La estructura if-else solo requiere la directiva de pre-procesamiento #include<stdio.h>

NOTA: si al evaluar las condiciones NO hay sentencias en el else este puede quedar sin sentencias dentro de las llaves u omitirse completamente.

NOTA: si el if o el else solo cuentan con UNA sentencia pueden omitirse las llaves.

NOTA: si se omiten las llaves se puede colocar esta única sentencia en la misma línea que el if y el else solo separado por un espacio.

NOTA: antes de una llave que abre jamás va a haber un punto y coma.

Sintaxis

```
if (condición)
{
     if (condición)
     {
          if (condición)
           {
              Sentencias
           }
          else
           {
           Sentencias
            }
          }
      else
       {
        Sentencias
       }
     }
else
{
 Sentencias
}
```

Donde la condición debe estar bien construida siguiendo la sintaxis ya vista. Y el anidamiento puede estar en el if o el else.

NOTA: si hay un anidamiento y en el hay un if con una sola sentencia y así sucesivamente, el código puede quedar como se muestra a continuación

Switch

Este tipo de sentencia te permite contar con múltiples alternativas como posibles opciones, pero solamente una de ellas puede llegar a realizarse. Debe considerarse que solamente se puede probar un identificador que sea una variable de tipo entero o alfanumérica entera, pero NUNCA real ni alfanumérico cadena.

Sintaxis

```
   switch (identificador)
          {
            Case valor1:
          Sentencias
          break;
```

```
    Case valor2:
Sentencias
break;
    Case valorN:
Sentencias
break;
     default;
sentencias
break;
}
```

La palabra default es otra palabra reservada que pertenece al switch, indica que, si ningún valor anterior coincidió con el valor del identificador comparado, se deberán ejecutar las sentencias que se encuentren inmediatamente debajo de default. Obsérvese que también los dos puntos acompañan a default. El default NUNCA tiene valor que se prosiga.

La estructura switch solo requiere la directiva de pre-procesamiento #include<stdio.h>

Expresión condicional ?:

Esta es la alternativa a la instrucción if-else y tiene la siguiente sintaxis:

Expresión de prueba? Expresión 1: expresión 2;

Si la expresión de prueba es verdadera, es decir, distinta a cero, se evalúa la primera expresión y el resultado permanece como efecto de toda la expresión condicional. En el caso contrario, cuando la expresión de prueba es falsa, es decir, si su valor es cero, se evalúa la segunda expresión y el resultado permanece como efecto de toda expresión condicional.

Instrucciones de repetición

En el lenguaje C es muy común hacer el uso de instrucciones, también llamadas instrucciones de iteración, bucles o ciclos.
Una instrucción de repetición permite repetir un bloque de instrucciones mientras se conserve verdadera alguna condición. Las instrucciones que se repiten forman lo que se llama cuerpo del ciclo.

Para elaborar un bloque de código iterativo se requieren cuatro elementos:

1. Una instrucción de repetición que delimita la sección repetitiva de código y controla su ejecución. Existen tres formas de instrucción de repetición en el lenguaje C: while, do-while y for.
2. Cada una de las instrucciones requieren de una condición que debe evaluarse. Las condiciones validas son semejantes a las utilizadas en las instrucciones de selección donde el código se ejecuta, solamente si la condición es verdadera.

3. Una expresión que establece, inicialmente, la condición y debe colocarse antes de que esta sea evaluada por primera vez, para asegurar la correcta ejecución del código iterativo la primera vez que se evalúa.
4. Dentro de la sección del código iterativo debe existir una instrucción que permita que la condición se vuelva falsa, esto es necesario para garantizar que en algún momento la iteración se detenga.

En el lenguaje C se manejan dos tipos de repeticiones:

a) Repetición controlada o repetición definida: antes de que inicie la ejecución del ciclo el número de iteraciones es conocido.
b) Repetición controlada por un valor (centinela) que se utiliza para determinar la continuidad o no del ciclo o repetición indefinida: el número de iteraciones no es conocido.

Operadores de incremento y decremento

En el lenguaje C existen operadores de incremento y decremento que soportan una sintaxis abreviada para añadir (incrementar) o restar (decremento) una unidad al valor de una variable.

Estos operadores se utilizan mucho en las instrucciones de repetición y permiten abreviar las operaciones de incremento o decremento que normalmente se emplean en la sintaxis:

Identificador = identificador + incremento
Identificador = identificador – incremento

Los operadores de incremento y decremento se utilizan como una forma abreviada de modificar el valor almacenado en una variable y asi tener acceso a este.

En la tabla se observa la forma de abreviar estas operaciones empleando los operadores de incremento y decremento.

Operación completa	Operación abreviada u operador	Decremento
Identificador=identificador+incremento Si incremento =1 Identificador=identificador +1	Identificador ++	Valor de identificador antes de incrementar. Devuelve identificador y luego incrementa identificador en uno
	++ identificador	Valor de identificador tras incrementar

		Incrementar identificador en uno y luego devuelve identificador
Identificador = identificador + incremento Si incremento= 1 Identificador = identificador + 1	Identificador +=n	Incremento identificador en n incrementar Devuelve identificador y luego incrementa identificador en uno
	++ identificador	Valor de identificador tras incrementar Incrementa identificador en uno y luego devuelve identificador
Identificador = identificador + incremento Si incremento ≠ 1 Por ejemplo: incremento = n	Identificador +=n	Incrementa identificador en n unidades y luego ejecuta el contenido
Identificador = identificador – decremento Si decremento = 1 Identificador = identificador – 1	Identificador --	Valor de identificador antes de disminuir Devuelve identificador, luego decrementa identificador en uno
	--identificador	Valor de identificador tras disminuir Decrementa identificador en uno, luego devuelve identificador
Identificador = identificador – decremento Si decremento ≠ 1 Por ejemplo: decremento = n	Identificador - =n	Decrementa identificador en n unidades y luego ejecuta el contenido

NOTA: solamente se puede hacer uso de la operación completa o la operación abreviada, nunca de manera simultánea ni combinándolas
El identificador debe ser el mismo cuando se emplea en la operación completa para que sea operación de incremento o decremento. Por ejemplo: x=x+6, y=y+1
P.e. en forma abreviada x+=6, y++

NOTA: el operador del incremento o decremento NUNCA se representa en un pseudocódigo o diagrama de flujo.

1. **Ciclo for o bucle for**

   El ciclo for permite repetir el conjunto de instrucciones que se encuentren entre sus llaves y ejecuta un número determinado de veces una secuencia de instrucciones, por eso el ciclo for es una repetición definida.

Ciclo while

El ciclo while permite repetir el conjunto de instrucciones que se encuentren entre sus llaves y ejecuta un número indeterminado de veces una secuencia de instrucciones, por eso el ciclo while es una repetición indefinida, ya que repite un conjunto de instrucciones mientras sea verdadera la condición o las condiciones que evalúa, si la condición devuelve un falso, finalizara su ejecución.

Para poder ingresar al conjunto de instrucciones dentro del while será necesario que la condición sea verdadera, al variar la condición por medio del incremento, se verificará nuevamente la condición para poder volver a ejecutar el conjunto de instrucciones. Cuando la condición sea falsa se romperá el ciclo de repetición.

Sintaxis

```
while (condición)
{
        Instrucciones
 }
```

**Ciclo while dentro de un ciclo while es un ciclo while anidado

Mientras que la expresión tenga un valor verdadero, representado por un valor numérico diferente de cero, se ejecutara rápidamente la secuencia de instrucciones evaluando en cada iteración el valor de la expresión. Si la expresión toma un valor falso, representado por un valor numérico igual a cero, la ejecución de la sentencia while finalizara.

La condición asociada al while debe ir encerrada entre paréntesis, esta expresión se formula de la misma manera que en if y utiliza, únicamente, los operadores que también emplean en if.

Para emplear la estructura del ciclo while basta con incluir #include<stdio.h> en el código fuente.

NOTA 1: puede ser que la condición sea falsa en el ciclo while, por lo que NUNCA se ingresaría a dicho ciclo.

Ejemplo

En los diagramas de flujo para identificar que no es un ciclo for, no hay operación de incremento, o no hay valor inicial o final.

Para saber si es un ciclo while la condición debe estar evaluada antes de la rama del sí, como se muestra en el diagrama.

Para declarar un identificador de tipo estructura

a) Declaración de una variable de tipo estructura

   Debe decidirse si se declara de manera global o de manera local, estos identificadores NO pueden llamarse como la estructura.

   1. Global
      Se coloca la declaración de la estructura y después de la llave que cierra y antes de las comillas se listan las variables que podemos usar.

Sintaxis

```
struct nombreEstructura
   {
      Tipo miembro1;
      Tipo miembro2;
      Tipo miembroN;
      }variable1,variable2,variableN;
```

Ejemplo

```
struct alumno
  {
       char nombre[50];
       float promedio;
       int numlista;
   }alumno1,alumnoB,alumnito;
```

Ejemplo:

```
Tipo identificador;
int x,y,h,j;
struct alumno alumno1,alumnoB, alumnito;
```

2. Local

   Sintaxis

   struct nombreEstructura variable1,variable2,variable3,variableN;

   ejemplo

   struct alumno, alumno2alumnoB55,alumnitos;

b) Declaración de un arreglo estático unidimensional de tipo estructura

Debe decidirse si se declara de manera global o de manera local, estos arreglos NO pueden llamarse como la estructura.

1. Global
   Se coloca la declaración de la estructura y después de la llave que cierra antes

   Sintaxis

```
struct nombreEstrucutra
    {
      Tipo miembro1;
      Tipo miembro2;
      Tipo miembro 3;
      Tipo miembro N;
     }arreglo1[tamaño],arreglo2[tamaño],arregloN[tamaño];
```

Ejemplo

```
struct alumno
   {
        char nombre[50]
         float promedio;
```

2. Local

   Sintaxis

   struct nombreEstructura

   arreglo1[tamaño],arreglo2[tamaño],arregloN[tamaño];

   Ejemplo

   struct alumno |a[6],h[10];

Para emplear un identificador de tipo estructura se sigue la siguiente sintaxis:

1. Guardar datos
   Es importante hacer notar que deben guardarse los datos en los miembros pertenecientes al identificador de tipo estructura (variable o arreglo) declarado NO en la estructura y para eso debemos ser específicos e indicar en que miembro y a que identificador pertenece dicho miembro.

   Sintaxis

   a) Si es una variable de tipo estructura

      scanf(“especificación de formato”,&nombreVariableEstructura.nombreMiembro);

   b) Si es un arreglo de tipo estructura

      scanf(“especificación de formato”,&nombreArregloEstructura[posicion].nombreMiembro);

      Ejemplo

      scanf(“%f”,&alumno2.promedio);

      scanf(“%i”,&A[0].numlista);

2. Mostar datos
   Es importante hacer notar que deben imprimir los datos de los miembros pertenecientes al identificador de tipo estructura (variable o arreglo) declarado NO de la estructura y para eso debemos ser específicos e indicar a que miembro y a que identificador pertenece el miembro.

Sintaxis

a) Si es una variable de tipo estructura
   printf(“especificación de formato”,nombreVariableEstrucutra.nombreMiembro);

b) Si es un arreglo de tipo estructura

   printf(“especificación de formato”,nombreArregloEstructura[posición],nombreMiembro);

NOTA 4: para realizar sumas, restas, etcétera, con el contenido de los miembros se sigue la misma metodología, empleando nombreArregloEstructura[posición].nombreMiembro o nombreVariableEstructura.nombreMiembro y aplicándolo como sea conveniente para realizar ciertas operaciones.

NOTA 5: las estructuras NO se representan en un algoritmo, pseudocódigo ni diagrama de flujo el programador decide si la emplea al programar para eficientar procesos. En un mismo programa puede haber varias estructuras.

NOTA 6: en un programa es posible que una estructura tenga declarados identificadores variados (que haya variables y arreglos de tipo estructura) para poder emplearla.

Yahir Bautista **C**orona

Lenguaje C

El lenguaje de programación C fue creado por Brian Kernighan y Dennis Ritchie a mediados de los años 70. La primera implementación del mismo la realizó Dennis Ritchie sobre un computador DEC PDP-11 con sistema operativo UNIX. C es el resultado de un proceso de desarrollo que comenzó con un lenguaje anterior, el BCPL, el cual influyó en el desarrollo por parte de Ken Thompson de un lenguaje llamado B, el cual es el antecedente directo del lenguaje C. El lenguaje C es un lenguaje para programadores en el sentido de que proporciona una gran flexibilidad de programación y una muy baja comprobación de incorrecciones, de forma que el lenguaje deja bajo la responsabilidad del programador acciones que otros lenguajes realizan por sí mismos. Así, por ejemplo, C no comprueba que el índice de referencia de un vector (llamado array en la literatura informática) no sobrepase el tamaño del mismo; que no se escriba en zonas de memoria que no pertenecen al área de datos del programa, etc.

El lenguaje C es un lenguaje estructurado, en el mismo sentido que lo son otros lenguajes de programación tales como el lenguaje Pascal, el Ada o el Modula-2, pero no es estructurado por bloques, o sea, no es posible declarar subrutinas (pequeños trozos de programa) dentro de otras subrutinas, a diferencia de como sucede con otros lenguajes estructurados tales como el Pascal. Además, el lenguaje C no es rígido en la comprobación de tipos de datos, permitiendo fácilmente la conversión entre diferentes tipos de datos y la asignación entre tipos de datos diferentes, por ejemplo, la expresión siguiente es válida en C:

```
float a; /*Declaro una variable para números reales*/
int b; /*Declaro otra variable para numero enteros*/
b=a; /*Asigno a la variable para entera el número real*/
```

Todo programa de C consta, básicamente, de un conjunto de funciones, y una función llamada main, la cual es la primera que se ejecuta al comenzar el programa, llamándose desde ella al resto de funciones que compongan nuestro programa.

Desde su creación, surgieron distintas versiones de C, que incluían unas u otras características, palabras reservadas, etc. Este hecho provocó la necesidad de unificar el lenguaje C, y es por ello que surgió un standard de C, llamado ANSI-C, que declara una serie de características, etc., que debe cumplir todo lenguaje C. Por ello, y dado que todo programa que se desarrolle siguiendo el standard ANSI de C será fácilmente portable de un modelo de ordenador a otro modelo de ordenador, y de igual forma de un modelo de compilador a otro, en estos apuntes explicaremos un C basado en el standard ANSI-C

Bibliografía:

Bonet Esteban, E. V. (s. f.). *El lenguaje de programación C*. El lenguaje de programación C. Recuperado 4 de marzo de 2021, de http://informatica.uv.es/estguia/ATD/apuntes/laboratorio/Lenguaje-C.pdf