Fundamentos de Programación I

2. Elementos de un programa

Luis Rodríguez Baena (luis.rodriguez@upsam.es)

Universidad Pontificia de Salamanca
Escuela Superior de Ingeniería y Arquitectura

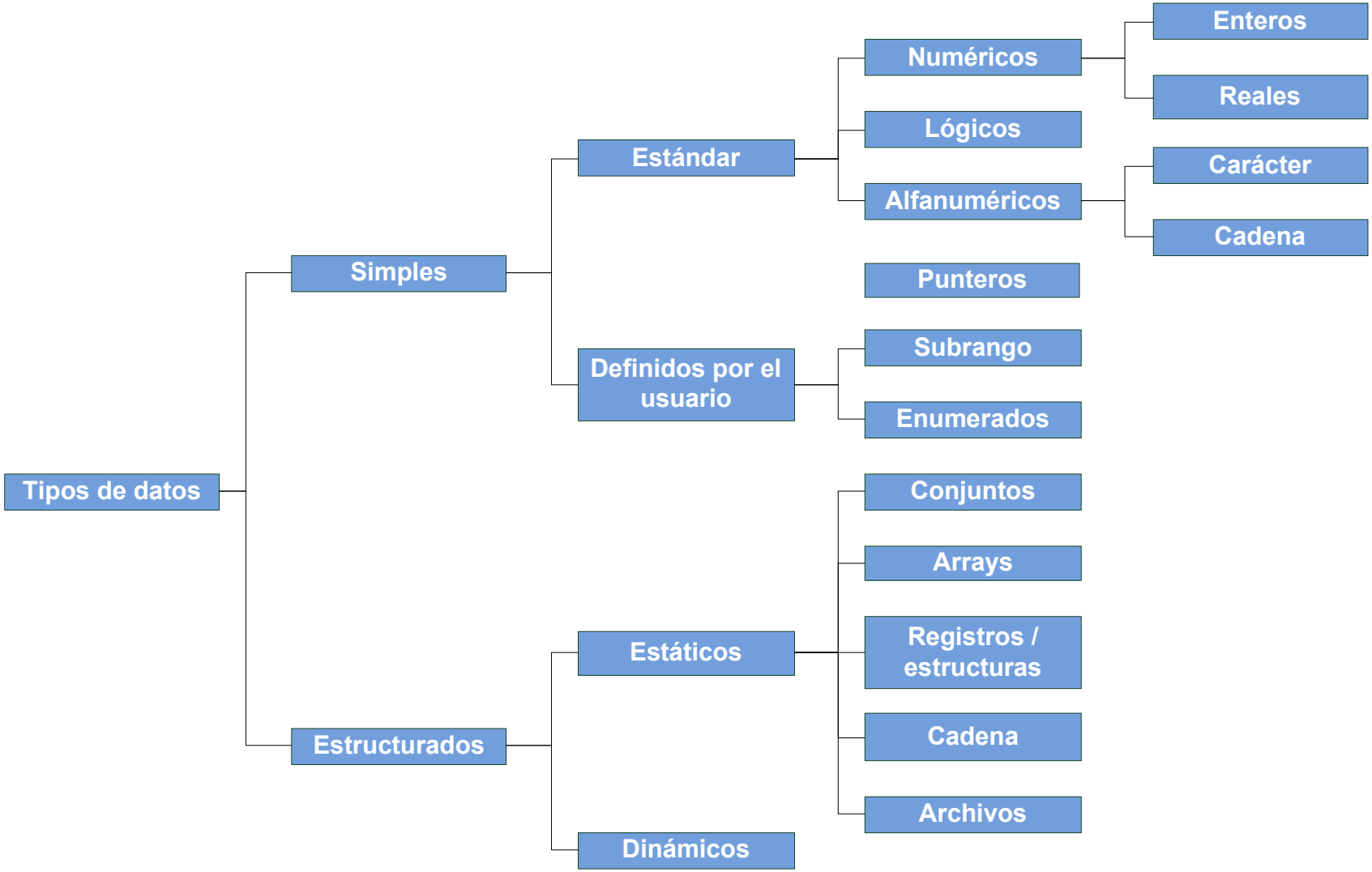
Datos y tipos de datos

- Los datos constituyen los elementos básicos sobre los que trabajará un algoritmo.
  - La elección de un tipo de dato determinará el algoritmo utilizado para procesarlos.
    - ✓ Algoritmos + estructuras de datos = programas (Nicklaus Wirth).
  - Por ejemplo:
    - ✓ Si un programa procesa una lista de alumnos…
      - ○ Requerirá un algoritmo con una estructura repetitiva que repita el mismo proceso para todos los alumnos hasta que finalice la lista.
    - ✓ Si un programa necesita hacer distinción entre las personas solteras o casadas…
      - ○ Requerirá de un algoritmo con una estructura selectiva que permita hacer la distinción entre los dos estados posibles que puede tomar el estado civil de una persona.

Datos y tipos de datos (II)

- Dato: unidad mínima de información con la que puede trabajar un programa.
  - El concepto de "información mínima" puede cambiar de un programa a otro en función de los **tipos de datos** que maneje.
    - ✓ Por ejemplo, mientras que FORTRAN puede trabajar con números complejos, las primeras versiones de BASIC sólo podían trabajar con números reales.
- Tipo de dato: conjunto de valores distintos que puede tomar un dato.
  - La mayoría de los lenguajes tienen un conjunto de tipos de datos similar.
  - Cada lenguaje puede nombrarlos o almacenarlos de forma distinta.
  - Aunque el procesador sólo trabaja con datos binarios, los lenguajes de alto nivel realizan una abstracción de la información.
    - ✓ El tipo de dato indica al programa como tiene que interpretar esa secuencia de bits.
      - Por ejemplo, la secuencia de bits `0000000001000001` puede ser interpretada por un programa como…
        - El valor numérico decimal 65 si se considera como un tipo de dato numérico.
        - El carácter A, si se considera como un tipo de dato `carácter`.

Clasificación de los tipos de datos



Datos simples

- Un dato simple es aquel que no se puede subdividir en otros más pequeños.
  - Por el contrario un dato estructurado estaría compuesto de otros datos que serán a su vez simples o estructurados.
- Datos estándar (datos primitivos o integrados).
  - Datos que son directamente soportados por el lenguaje de programación.
- Vamos a considerar los siguientes tipos de datos estándar.
  - Numéricos.
  - Lógicos.
  - Carácter.
  - Cadena.
  - Punteros.

Datos simples (II)

- Datos numéricos.
  - Entero.
    - ✓ Podrá contener cualquier valor numérico entero.
    - ✓ En C, según su tamaño en bytes pueden ser `int`, `short`, `long`.
  - Real.
    - ✓ Cualquier valor numérico con parte decimal.
    - ✓ En C podrán ser `float` y `double`.
- Datos carácter.
  - Podrán contener cualquier carácter válido.
  - En C se corresponde con el tipo de dato `char` y se puede considerar también como un dato entero.
- Datos lógicos.
  - Sólo podrán contener los valores si/no, verdad/falso.
  - En C no existen de forma directa y las expresiones lógicas devuelve 0 si es falsa o 1 si es verdadera.
    - ✓ El archivo de cabecera `stdbool.h` contiene especificaciones que enmascaran esto.
- Datos de tipo cadena.
  - Podrán contener una secuencia de caracteres.
  - En C no existen directamente y se consideran arrays de caracteres terminados en un carácter nulo.
- Datos de tipos puntero.
  - Contienen exclusivamente direcciones de memoria dónde se almacenen otros datos.
  - Se ven en profundidad en la asignatura de Fundamentos II

Datos simples (III)

- Datos definidos por el usuario.
  - El usuario (programador) decide el conjunto de valores que puede almacenar el tipo de dato.
  - Es necesario indicar el **nombre del tipo** y el **conjunto de valores**.
  - Según la forma en que se especifica el conjunto de valores.
    - ✓ Datos enumerados.
      - Se enumeran de forma explícita cada uno de los valores que puede contener.
        - Ejemplos:

```
díaSemana = {lunes,
   martes,miércoles,jueves,viernes,sábado,domingo}
```

        - Una dato de tipo `diaSemana` podrá contener cualquiera de estos valores.
      - En C se puede hacer algo parecido con las constantes enumeradas.
    - ✓ Datos subrango.
      - Son un subconjunto de algún tipo de dato ya definido.
      - Ejemplos:

```
sigloXX = 1900..1999
semanaLaboral = lunes..viernes
```

Datos estructurados

- Datos compuestos de otros datos.
  - Se podrán tratar tanto individualmente como en su conjunto.
- Según su forma de almacenamiento en memoria podrán ser:
  - Estáticos.
    - ✓ Se reserva un espacio fijo para almacenarlos y siempre ocupan la misma posición de memoria.
  - Dinámicos.
    - ✓ El espacio que ocupan y su posición pueden variar a lo largo de la ejecución del programa.
- En la asignatura de Fundamentos de Programación I se tratarán la siguientes estructuras de datos estáticas:
  - Arrays.
  - Registros.
  - Cadenas.
    - ✓ Se pueden considerar también un dato estructurado puesto que están compuestas de datos más simples (caracteres).
- Otros datos estructurados estáticos son:
  - Archivos (se verán en Fundamentos de Programación II).
  - Conjuntos.

Constantes

- Datos que no cambian su valor a lo largo de la ejecución del programa.
- Constantes literales.
  - Representación de la información que contiene el dato.
  - Según el tipo de información que contengan pueden ser.
    - ✓ Enteras.
      - Formadas por los dígitos de 0 a 1 y pueden estar precedidas por el signo -.
    - ✓ Reales.
      - Representación en coma fija.
        - Formadas por los dígitos de 0 a 1, el punto decimal y pueden estar precedidas por el signo – .
          - Por ejemplo, 567.56 o -34.678.
      - Representación en coma flotante.
        - Además de la mantisa, van seguidas por la base (representada por la letra E) y el exponente.
          - Por ejemplo, 2.34E02 (sería $2{,}34 \times 10^{2}$) o 1.034E-21 (sería $1{,}034 \times 10^{-21}$).

Constantes (II)

- Constantes literales *(continuación…)*.
  - Lógicas.
    - ✓ Representas mediante las palabras reservadas **`verdad`** o **`falso`**.
      - C no tiene este tipo de constantes, aunque el archivo de cabecera `stdbool.h` define las constantes `true` y `false` que valen 1 y 0 respectivamente..
  - Carácter.
    - ✓ Cualquier carácter del juego de caracteres utilizado encerrado entre comillas.
      - C distingue entre constantes de carácter y de cadena.
        - En C el delimitador de caracteres es la comilla simple.
  - Cadena.
    - ✓ Secuencia de caracteres del juego de caracteres utilizado encerrado entre comillas.
      - En C, el delimitador de cadenas es la comilla doble.
- Constantes simbólicas.
  - Constantes a las que se asigna un nombre nemotécnico (identificador).
  - El compilador sustituirá todas las apariciones de ese nombre por su valor asignado en la declaración de la constante.
    - ✓ En C se pueden declarar mediante el modificador `const`.

Variables

- Datos que pueden variar su contenido a lo largo de la ejecución de un programa.
- El nombre de la variable es un identificador válido.
- Las variables primitivas se almacenan en una zona de memoria que se reserva al arrancar el programa (pila) y queda inaccesible cuando termina.
  - Cuando en el programa se indica que se va a utilizar, por ejemplo, una variable entera.
    - ✓ Antes de la ejecución del programa se busca una zona de memoria lo suficientemente grande como para almacenar un dato entero.
    - ✓ A esa zona de la memoria se le asigna un identificador (un nombre).
    - ✓ Mientras que se ejecuta el programa al utilizar ese identificador se estará haciendo referencia a esa zona de memoria.
    - ✓ Al terminar el programa, el identificador "desaparece", por lo que no se puede utilizar esa zona de memoria, ya que no se puede hacer referencia a ella.

Expresiones

- Modifican los datos presentes en el algoritmo.
- Conjunto de variables, constantes, operadores y llamadas a funciones que siempre devuelven un valor.
  - El valor será de un tipo determinado que depende de los operadores y los operandos de la función.
  - Según el tipo de dato que devuelven habrá expresiones:
    - ✓ Numéricas.
    - ✓ Lógicas.
    - ✓ De cadena.
- En un algoritmo, las expresiones algebraicas comunes tendrán que ser traducidas a expresiones algorítmicas:

$$\frac{-a^2 - \frac{c}{d+5}}{2b}$$ se convertiría en (-(a*a) - c/ (d+5))/ (2*b)

Expresiones aritméticas

- Dan como resultado un valor de tipo numérico.
- Están compuestas de operandos numéricos y operadores aritméticos.
- El lenguaje algorítmico UPSAM utiliza los siguientes operadores aritméticos:

Operador	Significado	Ejemplo	Operador en C
**	Exponenciación	2 ** 3 equivale a 2^3	No existe en C. La función `pow(2,3)`. La función se encuentra en la biblioteca math.h
+	Suma aritmética	3 + 5 devolvería el valor 8	3 + 5
-	Resta. Como operador unario (con un solo operando) actuará como el signo negativo	8 – 3 devolvería el valor 5. -6 equivaldría al valor negativo de 6.	8 - 3
*	Multiplicación	6 * 4 devolvería 24	6 * 4
/	División real	5 / 2 devolvería 2.5	5 / 2
mod	Módulo (resto) de la división entera	5 **mod** 2 devolvería 1, el resto de dividir 5 entre 2	5 % 2
div	División entera	5 **div** 2 devolvería 2, el resultado de la división entera de 5 entre 2.	No existe. Si los operandos son enteros el operador / devuelve la división entera

Expresiones aritméticas (II)

- No todas las variables numéricas pueden aceptar cualquier expresión numérica.
  - Es importante determinar el tipo de dato (entero o real) devuelto por una expresión numérica.
- Valores devueltos por las expresiones aritméticas:
  - Dependen de los operadores y el tipo de dato de los operandos.
    - ✓ Los operadores +, - y *:
      - Si todos los operandos son enteros el resultado es entero.
      - Si algún operando es real el resultado es real.
    - ✓ Operador /.
      - El resultado es real.
    - ✓ Operador **.
      - El resultado es real.
        - Los compiladores suelen solucionar la exponenciación mediante logaritmos, por lo que el conveniente considerar el resultado como real.
    - ✓ Operadores **`mod`** y **`div`**.
      - El resultado es entero.

Operación	Resultado	Tipo de dato	Operación	Resultado	Tipo de dato
3+4	7	Entero	2*4.5	9.0	Real
5/2	2.5	Real	4/2	2.0	Real
15 mod 4	3	Entero	2**3	8.0	Real

Expresiones lógicas

- Devuelven un dato lógico (**`verdad`** o **`falso`**).
- Utilizan operadores relacionales y los operadores lógicos.
- Operadores relacionales.
  - Analizan la relación entre dos operandos del mismo tipo.
  - Pueden ser de cualquier tipo, pero siempre del mismo tipo.

Operador	Significado	Ejemplo	Operador en C
=	Igual que	2 = 3 devuelve **falso**	==
<	Menor que	3 < 5 devuelve **verdad**	<
>	Mayor que	8 > 3	>
<=	Menor o igual	4 <= 6 devuelve **falso**	<=
>=	Mayor o igual	5 >= 2 devuelve **verdad**	>=
<>	Distinto de	5 <> 2 devuelve **verdad**	!=

Expresiones lógicas (II)

- Operadores de relación en datos de tipo carácter y cadena.
  - Los operadores de relación se pueden utilizar en expresiones de tipo carácter y cadena.
  - Se compara el código numérico asociado a cada carácter que más o menos está ordenado alfabéticamente.
    - ✓ Hay que tener en cuenta que normalmente:
      - Cualquier letra mayúscula es menor que cualquier minúscula.
      - Cualquier dígito es menor que cualquier letra.
      - Algunos caracteres (por ejemplo la ñ o las vocales acentuadas) tienen un código mayor que los caracteres estándar.
    - ✓ Estas expresiones son verdaderas:

```
"A" < "B"
"c" > "A"
"ñ" > "z"
"MALAGA" > "MADRID"
```

  - En C se pueden comparar caracteres, pero para comparar cadenas hay que utilizar la función `strcmp` de la biblioteca `string.h`.

Expresiones lógicas (III)

- Operadores lógicos.
  - Evalúan dos expresiones lógicas.
  - Su resultado siempre es lógico.
  - Operadores:
    - ✓ `y` (en C, `&&`).
    - ✓ `o` (en C, `||`).
    - ✓ `no` (en C, `!`).
  - La tabla de verdad de estos operadores es:

a	b	a y b	a o b	no a
verdad	verdad	verdad	verdad	falso
verdad	falso	falso	verdad	falso
falso	verdad	falso	verdad	verdad
falso	falso	falso	falso	verdad

Expresiones de cadena

- Operan sobre cadenas o caracteres y utilizan operadores de cadena.
- Operador de concatenación: `&`, `+`.
  - El resultado de la concatenación de dos cadena es otra cadena formada por la unión de ambas.

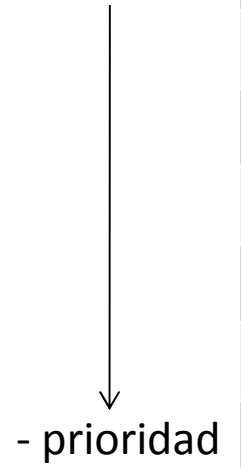
    `"Fundamentos" & "de Programación"` devolvería la cadena `"Fundamentosde Programación"`
  - C no tiene este operador. Para concatenar cadenas se utiliza la función `strcat` de la bilioteca `string.h`.

Prioridad de los operadores

- Una operación puede estar compuesta por varios operadores y operandos de distinto tipo.
  - Las operaciones siempre se hacen por parejas de operandos (en los operadores binarios), comenzando por el operador de mayor prioridad.
  - A igualdad de prioridad las operaciones se hacen de izquierda a derecha.
  - El paréntesis altera el orden natural de las operaciones.
    - ✓ Si existen varios paréntesis anidados se comienza por el más interno.
    - ✓ Si existen varios paréntesis al mismo nivel se evalúan de izquierda a derecha.
- Cada lenguaje puede tener un orden de prioridad propio.

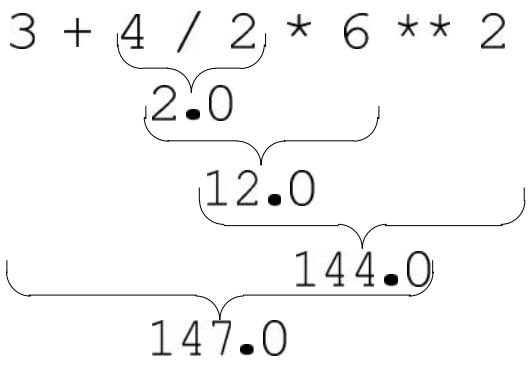
Prioridad de los operadores (II)

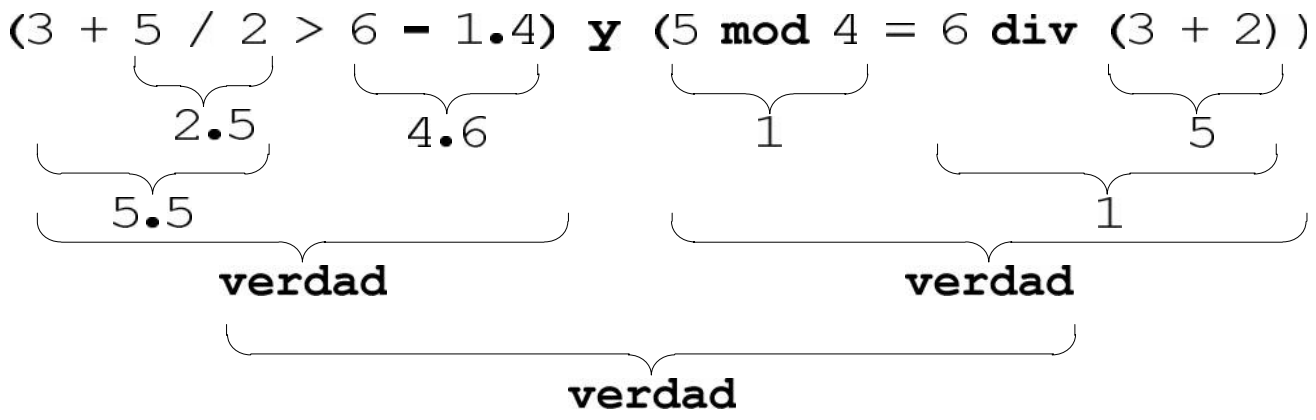
- Tabla de prioridades.



Lenguaje UPSAM	C
- (unario), **no**	- (unario), !
, *, /, **mod, **div**	*, / %
+, -, + (concatenación), &	+, -
<, <=, >, >=	<, <=, >, >=
=, <>	==, !=
y	&&
o	||

Prioridad de los operadores (III)





Instrucciones

- Un programa estará compuesto de una serie de proposiciones, sentencias o instrucciones que indican al ordenador las tareas a realizar y en el orden en que lo deben hacer.
- A pesar de la complejidad aparente que presentan los lenguajes de programación, los tipos de instrucciones se pueden resumir en tres categorías:
  - Instrucción de asignación.
  - Instrucciones de entrada y salida.
  - Instrucciones de control.

Instrucción de asignación

- La asignación modifica el valor de una variable.
  - Modifica el contenido de un área de memoria.
- El formato general de la instrucción de asignación sería el siguiente:

  nombreVariable ← *expresión*

  `subtotal ← precioProducto * unidadesProducto`
  - Evalúa la expresión que aparece a la derecha.
  - Asigna el valor de la expresión en la variable de la izquierda.
- En C, el operador de asignación es el símbolo `=`.
  - Además existen operadores de asignación que implican expresiones aritméticas como `+=`, `-=`, `*=`, `/=`, `%=`.
  - En C la asignación en sí misma se considera también una expresión.

Instrucción de asignación (II)

- El tipo de la expresión debe ser el mismo que el tipo de la variable.
- Conversión de tipos.
  - Algunos lenguajes realizan una conversión de tipos, convirtiendo automáticamente el valor de la expresión al tipo de la variable siempre que sea posible.
    - ✓ Esto no será posible en todas las expresiones.
  - Cuando se permite la conversión de tipos, puede haber casos de perdida de precisión cuando el tipo de dato de la variable tiene un tamaño menor que el tipo de dato de la expresión.
  - En C en las asignaciones se hace una conversión automática del tipo de dato si la variable tiene una capacidad mayor que la expresión.
    - ✓ En caso contrario el resultado queda indeterminado.

Instrucción de asignación (III)

- La variable puede aparecer a ambos lados del operador.
  - En la asignación…

    `a ← a + 5`

    ✓ Primero evalúa la expresión situada a la derecha.

    ✓ Después asigna el valor a la variable `a`.

    ○ El anterior valor de la variable `a` se pierde.

    ✓ En C el operador de asignación += haría esto.

    `a = a + 5` es equivalente `a a+=5`
  - Para realizar esto es necesario que la variable que aparece en la expresión tenga un valor inicial.

    ✓ Algunos lenguajes inicializan la variable a 0.

    ○ Esto no ocurre en todos los lenguajes.

    – C no inicializa las variables a 0.

    ○ Es una buena práctica inicializar siempre las variables.

Instrucciones de entrada y salida

- Permiten proporcionar información al algoritmo y devolver información al usuario.
- Instrucción de salida.
  - **`escribir`**`(`*`listaDeExpresiones`*`)`
  - La *`listaDeExpresiones`* son una o más expresiones separadas por comas.
- Instrucción de entrada.
  - **`leer`**`(`*`listaDeVariables`*`)`
  - La ejecución del programa se detiene hasta que se le proporcionan datos.
  - Cada uno de los datos proporcionados se asignan a una variable.
    - ✓ Es necesario que los datos proporcionados sean del mismo tipo que la variable.

Instrucciones de entrada y salida (II)

- En C las instrucciones de entrada y salida son funciones almacenadas en la biblioteca `stdio.h`.
- La instrucción de salida en C es…

  `printf(` *`cadenaFormato, listaExpresiones`*`)`

  ✓ Escribe el resultado de la lista de expresiones aplicando el formato de la cadena de formato.

  `printf("%i más %i es igual a %i",3,4,3+4)`

  ○ sacaría por pantalla

  ```
  3 más 4 es igual a 7
  ```

- La instrucción de entrada en C es…

  `scanf(`*`cadenaFormato, listaPunteros`*`)`

  - Lee por teclado una lista de valores separados por blanco y los trata de asignar a las variables de la lista de punteros.

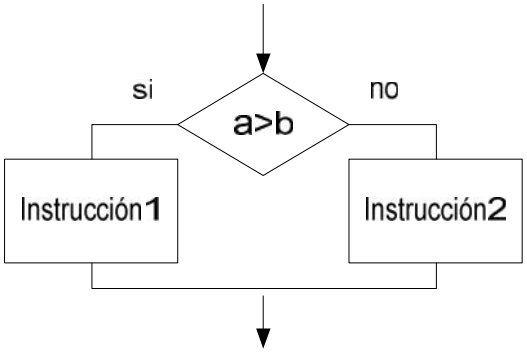
    `scanf("%i %i %i",&a,&b,&c)`

    ○ si la entrada de teclado es `45 65 78`…

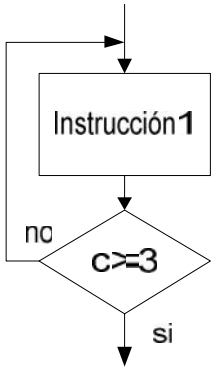
    – asignaría a la variable `a` el valor `45`, a la variable `b` el valor `65` y a la variable `c` el valor `78`.

Instrucciones de control

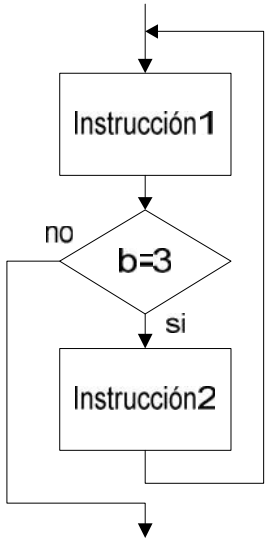
- Modifican el flujo normal (secuencial) de un programa.
- Todas utilizan una bifurcación.
  - Bifurcación condicional.
  - Bifurcación incondicional.



Bifurcación condicional. Si la expresión lógica es cierta ejecuta la instrucción1, en caso contrario la instrucción2



Bifurcación condicional. Si la expresión lógica es falsa se repite la instrucción1



Bifurcación incondicional. Después de la instrucción2 siempre se ejecuta la instrucción1

Elementos de un algoritmo

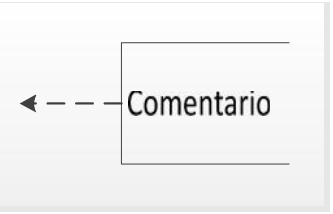
- Los algoritmos se pueden especificar mediante lenguaje natural o utilizando alguna herramienta de programación.
  - Algunas herramientas son flexibles (diagramas de flujo).
  - Otras son más rígidas y se acercan a un lenguaje de programación (pseudocódigo y Lenguaje Algorítmico UPSAM 2.1).
- Un algoritmo expresado en pseudocódigo estará compuesto de una serie de elementos que se deben utilizar mediante una sintaxis precisa.
  - Aunque no tienen por qué ser tan rígidos como un lenguaje de programación, respetar esa sintaxis ayudará a que el resto de las personas entiendan el algoritmo.
- Elementos de un algoritmo:
  - Variables y constantes, palabras reservadas, identificadores, comentarios, declaraciones.
- Palabras reservadas.
  - Cada lenguaje tiene un conjunto propio de palabras reservadas.
  - El compilador tratará esas palabras de forma especial.
  - Construcciones que forman parte del léxico del lenguaje y que tienen un significado especial en un programa fuente escrito en un lenguaje de programación determinado.
    - ✓ En nuestro lenguaje algorítmico se señalarán en negrita o subrayado.

Elementos de un algoritmo (II)

- Identificadores.
  - Palabras creadas por el programador con los caracteres permitidos por el lenguaje y que permiten identificar los distintos objetos creados por él como los nombres de variables.
  - Tienen que estar formados por caracteres del juego de caracteres permitidos por el lenguaje.
    - ✓ ASCII, Unicode.
      - Aunque hay implementaciones de C que permiten un juego de caracteres extendido, es más portable utilizar los caracteres ASCII estándar.
        - Estos no incluyen los caracteres específicos del idioma español como la ñ o las vocales acentuadas.
    - ✓ En nuestro caso: cualquier carácter alfabético del idioma español, caracteres numéricos o el guión bajo (_).
  - El lenguaje UPSAM 2.1 no es sensible a mayúsculas.
    - ✓ Otros lenguajes (C, Java) si lo son.
  - Deben comenzar por un carácter alfabético.
    - ✓ Se considera como carácter alfabético cualquier carácter o ideograma que describa un carácter alfabético o un dígito en cualquier idioma.
  - En la práctica, se consideran de longitud ilimitada.
    - ✓ C es capaz de admitir hasta 63 caracteres
  - No pueden ser igual que las palabras reservadas.

Elementos de un algoritmo (III)

- Comentarios.
  - Documentación interna del programa.
  - `{comentario}` , `/*comentario*/` o `//comentario`
    - ✓ En C:
      - //Comentario (comentarios de una sola línea).
      - /* Comentario */ (para comentarios de más de una línea).



- Instrucciones.
  - De asignación, de entrada/salida, de control.
- Declaraciones.
  - Instrucciones no ejecutables que definen los elementos utilizados en un algoritmo.
  - Declaración del nombre del algoritmo, tipos de datos, constantes y variables.
  - Declaración de subprogramas.
  - Declaración del inicio y el fin del código del algoritmo.

Estructura de un algoritmo en pseudocódigo

- La estructura de un algoritmo escrito en el lenguaje algorítmico UPSAM 2.1 sigue el esquema siguiente:

```
algoritmo NombreDelAlgoritmo
//Sección de declaraciones globales
[Declaración de tipos de datos]
[Declaración de constantes
   simbólicas]
[Declaración de variables]
//Cuerpo del algoritmo
inicio
   //Código del algoritmo
fin
```

Estructura de un algoritmo en pseudocódigo (II)

- Declaración de constantes.
  - Se utilizan para declarar las constantes simbólicas.

```
const
   identificador1 = expresión1
   identificador2 = expresión2
const
  pi = 3.141592                  //pi será una constante real
  radio = 23                     //radio será una constante entera
  existe = verdad                //Existe será una constante lógica
  ciudad = 'Madrid'              //ciudad será una constante de cadena
  longitudCircunfereciaFija = 2 * pi * radio
```

- Declaración de variables.
  - Se deben declarar todas las variables que utilice el algoritmo.

```
var
   tipoDeDato : ListaIdentificadores [= expresiónInicialización]
   ...
```

  - *tipoDeDato* podrá ser cualquier tipo de datos estándar o definido por el usuario, simple o estructurado.
    - ✓ De momento será **entero**, **real**, **lógico**, **carácter** o **cadena**.

Estructura de un algoritmo en pseudocódigo (III)

- Declaraciones en C.
  - C no tiene secciones especiales de declaraciones para constantes y variables.
  - Pueden declararse en cualquier lugar del programa.
  - Si es posible, conviene agruparlas al comienzo del programa y utilizar un comentario para indicarlo.
  - Declaraciones de variables.

```
tipoDato listaDeVariables
```

    - ✓ Cada variable puede llevar una expresión de inicialización.

```
//Declaración de variables
int a,b,c;
float d = 3.45;
boolean e;
char f,g,h;
char *i = "hola"
```

  - Declaración de constantes.
    - ✓ La palabra reservada **`const`** indica que un dato es constante.

```
//Declaración de constantes
const float PI = 3.141592      //PI será una constante real
```

Estructura de un algoritmo en pseudocódigo (IV)

- Ejemplo de algoritmo en lenguaje UPSAM 2.1.

```
algoritmo cálculoPrecioVenta
const
   IVA = 0.18
var
   real : total, precioProducto, subtotal, totalIVA , totalConIVA
   entero : unidadesProducto
inicio
   total ← 0
   leer(precioProducto)
   // precioProducto = 0 indica que no hay más productos
   mientras precioProducto <> 0 hacer
      //Calcula el subtotal de un producto
      leer(unidadesProducto)
      subtotal ← precioProducto * unidadesProducto
      total ← total + subtotal
      //Para salir, introducir un precioProducto = 0
      leer(precioProducto)
   fin_mientras
   totalIVA ← total * IVA
   totalConIVA ← total + totalIVA
   escribir('Total (sin iva):', total)
   escribir('IVA (16%):', totalIVA)
   escribir('Total a pagar:', totalConIVA)
fin
```

Estructura de un algoritmo en pseudocódigo (V)

- Ejemplo de programa en C.

```
/*
   El nombre del archivo fuente
   calculoPrecioVenta.c es el nombre del algoritmo
*/
#include <stdio.h>

int main(void){
   /*Declaracion de constantes */
   const float IVA = 0.18;

   /*Declaracion de variables */
   float total=0, precioProducto,subtotal,totalIVA,totalConIVA;
   int unidadesProducto;

   printf("%s","\nPrecio del producto (0 para terminar):");
   scanf("%f",&precioProducto);
```

Estructura de un algoritmo en pseudocódigo (VI)

- Ejemplo de programa en C.

```
   while(precioProducto !=0){
      printf("%s","\nUnidades producto:");
      scanf("%i",&unidadesProducto);
      subtotal = precioProducto * unidadesProducto;
      total += subtotal;
      printf("%s","\nPrecio del producto (0 para terminar):");
      scanf("%f",&precioProducto);
   }

   totalIVA = total * IVA;
   totalConIVA = total + totalIVA;
   printf("\nTotal (sin iva): %.2f\n",total);
   printf("IVA (18%%): %.2f\n",totalIVA);
   printf("Total a pagar: %.2f\n",totalConIVA);

   return 0;
}
```

Ejercicios

1. ¿Cuáles de los siguientes datos serían válidos para procesar por un lenguaje de descripción de algoritmos? ¿Por qué?.
   - En caso de que sean correctos ¿de qué tipo serían?

```
a) 45.6765

b) 6.34X-4

c) $Hola$

d) "Hola'

e) 'Hola'

f) 0,45

g) 4.06E-12

h)-3.12E+43

i)  verdad
```

Ejercicios (II)

2. ¿Cuáles de los siguientes identificadores serán válidos? ¿Por qué?

a) `x`

b) `A32`

c) `2000A`

d) `TotalxAño`

e) `Día`

f) `3B16`

g) `Nombre_Apellidos`

h) `Nombre-Apellidos`

i) `SueldoBase_Dia`

Ejercicios (III)

3. Convierta a expresiones algorítmicas las siguientes expresiones algebraicas.

a) $\frac{M}{N}+4$

b) $M+\frac{N}{P-Q^2}$

c) $M \cdot Q+\frac{\sqrt{4 \cdot D^{2 \cdot n}}}{\frac{P}{R}}$

d) $M+\frac{N}{P}-\frac{Q}{\sqrt{R}}$

Ejercicios (IV)

4. ¿Cuál sería el resultado de las siguientes expresiones?

a) ((5+4) **2 > 23/5) **o** (3 = 4)

b) 10 **mod** 13 + 4 / 2 + 1

c) (3 < 2 * 5) **o falso**

d) (55 **div** 2) ** (1/3)