Tema 1: Metodología

- **Introducción**

Informática se define como la ciencia o disciplina que estudia el transito automatico de la información. Estudia los medios para automatizar la información y poner dicha información a disposición de los usuarios.

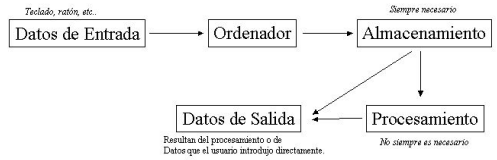
Algoritmo se define como una secuencia de acciones que describen la solución a un problema de manera general.

Sentencia se puede definir como cada una de las acciones que contiene o realiza un algoritmo.

Programa puede definirse como un algoritmo codificado en un lenguaje compatible con el ordenador.

Metodología: Se define como conjunto de técnicas que nos permiten representar algoritmos. Estudia los puntos de vista para resolver un problema. (Técnicas, Medios), son los puntos de vista que se deben adoptar para resolver algoritmos.

Un Programa debe constar siempre de las siguientes fases:



- La distancia entre la entrada y la salida (dependiendo del procesamiento), marcará la dificultad o la sencillez del programa.

- ***Fases de la Programación***

Fase 1: Planteamiento del problema

Fase 2: Análisis y comprensión del problema (Fase mas importante)

Fase 3: Resolución del Algoritmo (No pasar a la Fase 4 sin estar seguros de que el algoritmo vaya a funcionar)

Fase 4: Codificación del Algoritmo (Tecleado)

Fase 5: Pruebas de Verificación

Después de la fase de programación podrían incluirse las siguiente fases:

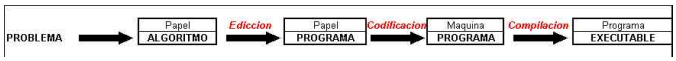
Fase de Optimización: Fase en la que una vez hecho el programa, y has verificado que funciona, deseamos cambiar el algoritmo del mismo para hacerlo mas rápido o introducir otras mejoras.

Documentación:

Documentación Interna: Introducción de texto en castellano, dentro del codigo fuente, para aclarar cuestiones referentes al propio programa.

Documentación Externa: Instrucciones para usar el programa.

Proceso gráfico para realizar un programa:



Un compilador es un programa que permite programar en un lenguaje de programación determinado. Cada lenguaje de alto nivel, tiene su propio compilador.

- ***Tipos de lenguaje de programación***

Existen dos tipos:

- El **lenguaje Maquina**: Es el lenguaje propio de la maquina, el único que el ordenador entiende.
- **Simbólicos**: Palabras simbólicas que especifican las instrucciones. Existen dos clases:

  - Ensamblador (BAJO NIVEL): Cada instrucción en este lenguaje equivale a una instrucción en el lenguaje maquina.
  - Lenguajes de Alto Nivel como pueden ser Cobol, Pascal, Fortran, Logo, Basic, Visual Basic, etc...

Cada instrucción de un lenguaje de alto nivel equivale a muchas instrucciones del lenguaje maquina.

Descripción breve de los tipos de lenguaje de alto nivel:

- **Cobol**: Hace unos 20 años que fue creado para gestionar información almacenada en ficheros. Un lenguaje orientado a gestionar negocios.
- **Fortran**: Es aun mas antiguo que Cobol, y fue creado para trasladar formulas con el objetivo de la resolución de problemas de tipo científico.
- **Pascal:** Lenguaje orientado a problemas científicos.
- **Logo:** Lenguaje educativo, pensado para enseñar a programar.
- **ADA:** Lenguaje desarrollado por una multinacional, para su uso privado, el cual esta pensado para programar muy estructuradamente.
- **C:** Esta pensado para funcionar a bajo y alto nivel, y por tanto tiene características propias de los lenguajes de bajo nivel, pensado principalmente para programas de tipo técnico, tale como sistemas operativos, etc...
- **Basic:** Lenguaje destinado al usuario, de facil comprensión, y pensado para ordenadores poco potentes.
- **Visual Basic:** Maneja la sintaxis normal de Basic, incorporando la capa de programación visual, desarrollado para el entorno Windows.

- ***Características de un programa***

  - **Legibilidad**: Debe estar hecho de manera que se pueda leer fácilmente. El lenguaje C no es un lenguaje excesivamente legible por su naturaleza.
  - **Portabilidad**: Esta característica se basa en utilizar recursos comunes en todos los lenguajes, para que sea facil trasladar el programa de un lenguaje a otro. Sera portable si esta elaborado con instrucciones comunes y sencillas.
  - **Modificabilidad**: El programa fuente debe poder ser cambiado lo mas fácilmente posible.

- Eficiencia: Dentro de lo posible, ya que por hacer un programa demasiado eficiente se nos puede venir abajo en otras cuestiones.
- **Modularidad**: El programa deberá estar divido en fragmentos donde se distinguen partes que estan dentro de un todo, que es el algoritmo.
- **Estructuración**: Un algoritmo esta estructurado cuando su confección esta basada en una pautas estructurales que no son otras cosa que el orden que hay que seguir al programar.

- ***Objetivos de un programa***

Son los objetivos o elementos que un programa va ha manejar, utilizar o gestionar. Estos elementos van a contener datos o lo van a ser directamente.

Podemos distinguir distintos *tipos de objetos*:

- **Numérico**: Objeto que representa un valor numérico con el que se pueden hacer operaciones aritméticas.

Ej: 234

- **Alfanumérico:** Un objeto que representa una cadena de caracteres.

Ej: 234chupachsus3

- **Caracter:** Un caracter es cualquier símbolo que se representa en forma de un solo símbolo.

Ej**:** `a'

- **Lógico:** Solo hay dos posibilidades, verdadero y falso, un valor lógico es un valor de verdad. Un valor lógico se verifica o no se verifica.

Otra *clasificación* de los objetos *más precisa* seria:

- **Constantes:**
- Normales:
- Numéricas
- Alfanuméricas
- Caracter
- Lógica
- Figurativas:
- Numérica
- Alfanumérica
- Caracter
- Lógica

· **Variables**

Definición de Constante: Unobjeto de tipo constante es un valor cuyo contenido no puede variar. Podríamos diferenciar:

- ◊ **Constante Normal**: Valor constante expresado en si mismo. Ej: Valor numérico 128
- ◊ **Constante figurativas**: Un nombre que de manera figurada simboliza un valor constante que no cambia. Ej: PI = 3,1416 ; en el que PI sería la constante.

Definición de Variable: Lugar en la memoria que esta reservado, y que posee un nombre, un tipo y un contenido. El valor de la variable puede cambiar y puede ser modificado a lo largo

del programa.

Al igual que las constantes pueden ser de ***tipo***:

- Numérica
- Alfanumérica
- Logica
- Carácter

La variable tiene ya de entrada un valor determinado, de forma que el tipo de variable, debe de ser uno en concreto. Por tanto el primer dato de la variable condicionara los posteriores datos que almacenará después.

La variable contiene tres cualidades:

- *Nombre*
- *Tipo*
- *Valor*

En el momento de declarar una variable deberemos definir el nombre y el tipo, pero el valor podrá declararse después.

Para facilitar y recordar fácilmente que valor tiene asignado cada variable, esta deberá tener un nombre acorde al valor que contenga.

El espacio en una unidad de caracteres (STRING) no existe, ya que este espacio es designado por un símbolo denominado carácter blanco. Ej.: Esto"es"una"cadena. De modo que a la hora de contabilizar los caracteres el espacio será incluido como tal, ocupando así un lugar en la memoria.

Tema 2: Realización de Algoritmos

♦ ***Diagramas de Flujo***

Definición de diagrama de flujo o también llamado ordinograma: Sistema de representación gráfica que utilizaremos para expresar los algoritmos.

◊ *Estructuras y acciones* en un diagrama de flujo:

· **Definición de variables**: Es obligatorio definir todas las variables que vayamos a utilizar durante el programa, en el inicio del diagrama de flujo.

Cuando nuestro programa se ejecuta, a nuestros efectos la memoria contendrá información basura (información que estaba cargada en la memoria antes de ejecutar nuestro programa)

Para definir una variable es imprescindible indicar cual será su nombre y su tipo. Pudiendo asignar a esa variable su valor posteriormente.

· **Asignación de valores a variables**:

♦ Existen dos tipos de asignaciones:

· *Directa*: Asignar un valor directamente que ya conocíamos a priori.

· *Indirecta:* El valor de estas variables vendrá dado una vez que se ejecuta el programa, y se produzcan datos de entrada.

La acción de asignación a una variable destruye cualquier valor inicial de esa variable.

· **Declaración de Expresiones numéricas:**

Definición de expresión numérica: Dos o mas valores numéricos operados entre si aritméticamente. Ej:

2x + y − 7x

2x + 3y = 10

Los operadores aritméticos siguen una *jerarquia de prioridad:*

− La multiplicación tiene prioridad respecto de la suma.

Asignación directa de variables en un Diagrama de Flujo

Se representa con este símbolo:

< Variante > = <Expresión>

Expresión: Consiste en un valor o más en el caso de que halla varios valores operados entre si mediante operadores del tipo que sea.

La expresión que da el valor a una variable puede ser de varios tipos:

· Aritmética: Uno o más valores numéricos operados entre si. Una expresión puede constar de variables y constantes, utilizando los operadores aritméticos:
  · Modulo (MOD)
  · Multiplicación (*)

· División (/)
· Suma (+)
· Resta (−)

Estos operadores tienen un nivel de prioridad:

Modulo − Multiplicación − División − Suma − Resta

El orden de evaluación es de derecha a izquierda. Resolviendo primero los paréntesis que tenga la expresión.

El operador Modulo (MOD), hace un resto entre enteros, es decir coge dos numeros enteros, los divide y nos da el resto de la división. Ej:

7 MOD 3 = 1 Ya que el resto de dividir 7 entre 3 es 1.

Resolver operaciones aritméticas en términos matemáticos

R = 18 − 3 * A + (A MOD B − C / 2) + A * C MOD 5

Siendo A = 5 B = 7 C =8

R= 18 − 3 * 5 + (5 MOD 7 − 8 / 2) + 5 * 8 MOD 5

R= 18 − 15 + (5 − 4) + 5 * 0

R= 3 + 1 + 0

R = 4

R = B − A + C * 2 + (C / 2 − 1) + A * 4 * B − 4

Siendo A = 2 B = 3 C = 6

R = 3 − 2 + 6 * 2 + (6/2 − 1) + 2 * 4 * 3 − 4

R = 1 + 12 + 2 + 20

R= 35

· Expresiones alfanumericas (STRINGS):

CADENA = Pepe + López PepeLopez

Si queremos que en el resultado salga un espacio en blanco, tendremos que indicarlo de la siguiente forma:

CADENA = Pepe + + López Pepe López

· Tambien puede haber expresiones de tipo lógico

Asignaciones indirecta por teclado de variables en un diagrama de flujo

Se representa con este símbolo:

El valor previo de la variable es destruido por el que se le asigna. El valor de esta variable no viene dado por el programa si no por un dispositivo de entrada. Esto se representa en un diagrama de flujo de la siguiente forma:

Ejercicio: Tomar por teclado dos numeros A y B y calcular la suma y la resta, guardando los resultados en variables.

Visualización por pantalla en un diagrama de flujo

Se representa con este símbolo:

Cada representación, abarcará una linea en la pantalla. Podemos indicar uno o mas valores dentro del mismo símbolo. No se pueden incluir expresiones dentro de este símbolo.

Otros tipos de variables

◊ **Variable Contador:**

- C = C + 1
- C = C + 5

La variable contador aumenta de manera constante cada vez que se ejecuta.

◊ **Variable Acumulador:**

- A = A + B

B es una variable que cambia de valor, por tanto cada vez que se ejecuta esta instrucción, la variable A aumenta de manera NO constante. Si B no cambiara de valor, A seria una variable contador.

◊ **Variable Bandera (Flag):**

- <Variable>=<Estado>
- Pers_Preferentes = <Si>

Este tipo de variables muestran el estado del valor de la variable, por ejemplo, si tenemos dos grupos de personas, preferentes o no preferentes, podríamos definir una variable de tipo Alfanumérico con el nombre de Preferentes y darla un valor Si, con lo que estaríamos declarando que las personas son preferentes.

Este tipo de variables, no tienen por que ser te tipo lógico, ya que pueden indicar mas de un estado. Ej:

Estado_civil = C (Casado)

V (Viudo)

S (Soltero)

En este caso una variable podria definirse como: significador de un estado dentro de un algoritmo.

ESTRUCTURAS DE PROGRAMACIÓN

Existen tres estructuras:

· Secuencial: Manera natural de organizar las accciones .
· Selectiva
· Iterativa

ESTRUCTURAS SELECTIVA

Posibilidad de tomar distintos caminos en la ejecución de las acciones. Podremos establecer caminos alternativos para la ejecución.

Podemos distinguir tres tipos:

· Selectiva simple
· Selectiva doble

· Selectiva múltiple (Se usa con poca frecuencia debido a su uso en casos selectivos)

◊ **Selectiva Simple:**

La estructura simple se representa de la siguiente forma:


NO

Dentro del rombo se expresa una condición lógica que se reduce a un valor lógico, verdadero o falso. Dentro del rectángulo se expresa una instrucción. Las estructuras simples, a diferencia con las dobles, solo representa instrucción en la rama del SI (Verdadero).

Cuando estas estructuras se ejecutan, se evalúa una condición, y si es verdadera el flujo del algoritmo sigue por la rama del SI, si la condición resulta ser falsa seguiría por la rama del NO.

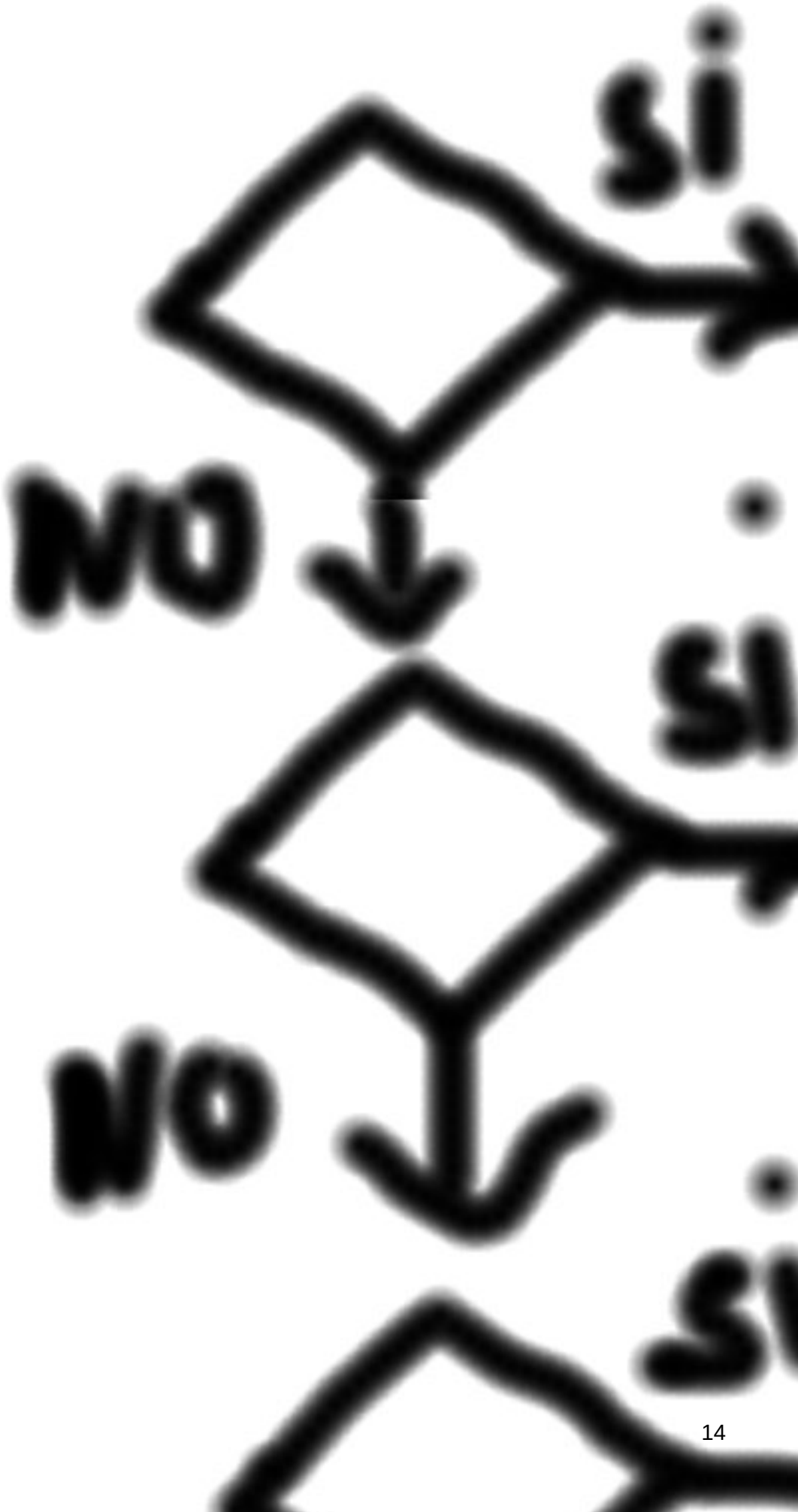
◊ **Selectiva Doble:**

Es el mismo tipo de estructura que la simple, con la diferencia que en la rama del NO, tambien se ejecuta una instrucción, esta estructura se representa de este modo:


NO

◊ **Selectiva Múltiple:**

Existen multiples estructuras seguidas con distintos caminos alternativos posibles. Se representaria de la siguiente forma:


Si
NO
Si
NO
Si

En este tipo de estructuras no puede haber instrucciones en la rama del NO, salvo en la ultima rama, como se señala en el gráfico. Si tras chequear todas las condiciones dentro de los rombos, todas resultan ser falsas, se ejecutaría la instrucción que tiene asignada la ultima rama del NO, aunque también puede ocurrir que dicha rama no tenga asignada ninguna instrucción, por lo que el programa seguiría el flujo normal del programa.

Si alguna de las condiciones resultara ser verdadera, se ejecutaría la instrucción correspondiente a su rama de SI, y depuse saldría de la estructura para seguir ejecutando el programa.

ESTRUCTURAS ITERATIVAS

Consisten en instrucciones que se van ha repetir de 0 a N veces.

No hay que confundir la idea de bucle con la idea de estructuras selectivas. (Una de las pocas paridas que dijo el Alfonso en Clase, ya que confundir estos dos tipos de estructuras es como confundir el tocino con la velocidad)

Tipos de estructuras iterativas:

- ◊ REPETIR MIENTRAS
- ◊ REPETIR HASTA
- ◊ DESDE HASTA

Repetir Mientras: Dentro del Rombo se usa una condición. El bucle no tiene ramas(aunque en la representación pueda parecerlo), se produce una entrada de bucle o una salida. Se representa de la siguiente forma:

La condicion de un bucle siempre va ha ser una condición lógica que se reducirá a un valor lógico como Verdadero o Falso, de modo que si la condición resulta ser verdadera el bucle repite hasta que la condición resulte ser falsa, entonces el bucle se rompe y se vuelve al flujo normal del programa.

Repetir Hasta:


si

Siendo el rectángulo una instrucción, y el rombo una condición, la instrucción se repetirá hasta que la condición de un valor de FALSO, entonces se romperá la estructura y se volverá al flujo del programa.

Desde Hasta:

Caso muy particular. El numero de repeticiones de una o mas instrucciones vendra marcado por una variable de control.

EXPRESIONES CONDICIONALES

Son las expresiones que utilizamos en los rombos, para deducir si dichas expresiones tienen un valor Verdadero o un valor falso. Podemos distinguir dos tipos de expresiones condicionales:

· EXPRESIONES DE RELACION
· EXPRESIONES LOGICAS

Expresiones de Relación: En una expresión de relación tenemos uno o más valores operados por los operadores correspondientes. Consiste en consecuencia en dos valores operados entre si con un operador de relación. Los operadores son:

> Mayor que

< Menor que

>= Mayor o igual que

<= Menor o igual que

= Igual que

<> Distinto que

Estos operadores, también pueden operar con caracteres(por medio de código ASCII). Ej.:

A < B Tendría un valor lógico de Verdadero pues el valor ASCII de A es menor que el de B (El código ASCII esta ordenado de acuerdo con el abecedario.)

Una expresión relacional obtiene una valor lógico (VERDADERO O FALSO).

Expresiones Logicas: Vamos a tener valores lógicos operados por operadores lógicos. Los operadores lógicos son los siguientes:

- NOT (No)
- AND (Y)
- OR (O)

Operador NOT: Operador que actúa sobre un solo valor. Este operador tiene la cualidad de invertir el valor lógico. Ej.:

NOT <Valor Lógico>

NOT VERDADERO ..... Tendría un valor FALSO

NOT FALSO..... Tendría una valor VERDADERO

Operador AND: La operación AND vale VERDADERO cuando los dos valores son verdaderos, de lo contrario vale FALSO. Ej.:

<Valor Lógico> AND <Valor Lógico>

Operador OR: Es similar a AND en el sentido de que llevará un valor lógico a izquierda y derecha, pero la diferencia estriba en que OR devuelve VERDADERO cuando cualquiera de los dos valores es VERDADERO.

Los operadores relacionales tienen prioridad sobre los lógicos, siempre y cuando ningún paréntesis indique lo contrario. La prioridad de los operadores lógicos es la siguiente:

- NOT
- AND
- OR

Resumen gráfico de los operadores lógicos:

VALOR A	VALOR B	NOT A	A AND B	A OR B
V	V	F	V	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	F

DIAGRAMAS DE LLAVES

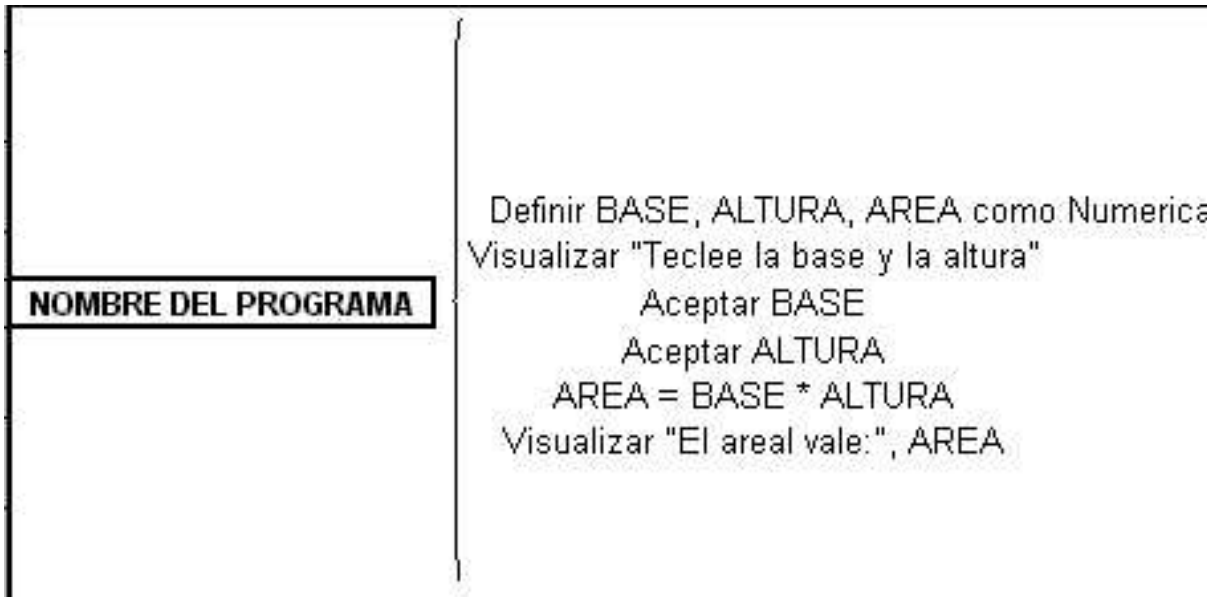
Para confeccionar un diagrama de llaves empezaremos poniendo el nombre del programa en mayúsculas y recuadrado. A continuación abriremos una llave, que será la llave principal y que abarcará todo el algoritmo desarrollado. El inicio del algoritmo se marcara con un punto de inicio en la parte superior de la llave, y el final se marcara con un punto de final en la parte inferior de dicha llave.

No se utilizarán símbolos gráficos para representar acciones, sino que se utilizarán expresiones cortas que describirán la instrucción a realizar. Si las instrucciones son secuenciales, se ejecutaran en TOP − Down, es decir de arriba abajo. Las variables se escribirán en mayúsculas y el resto del algoritmo en minisculas.

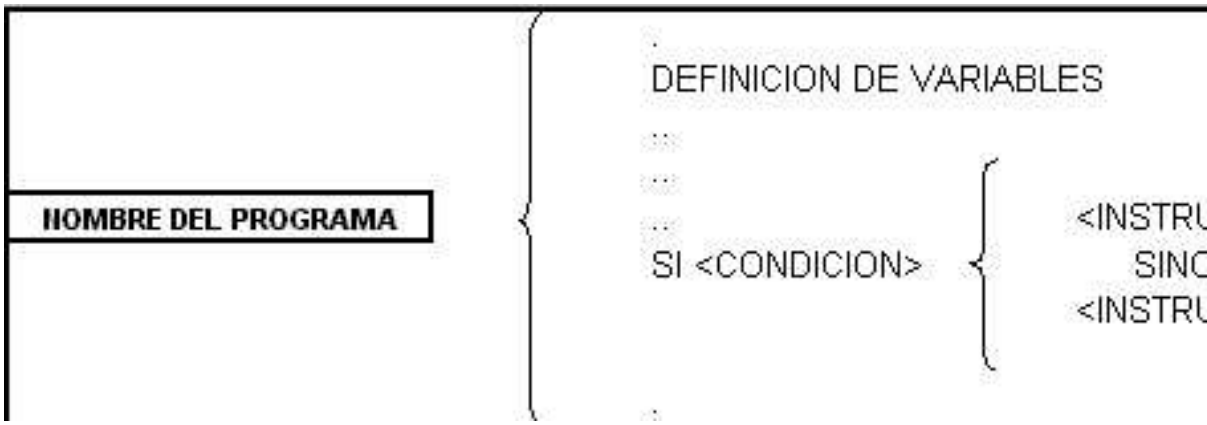
Estructuras en un diagrama de llaves

Se usan llaves dentro de la llave principal para definir nuevas estructuras condicionales o iterativas.

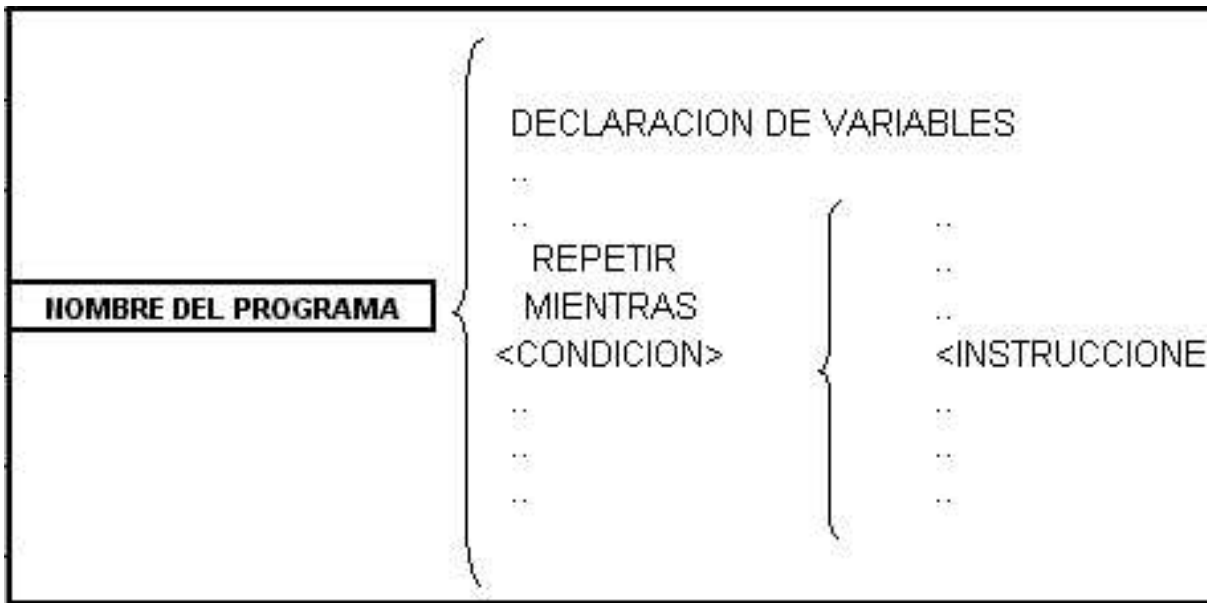
Ejemplo de diagrama de llaves en una estructura secuencial:



Ejemplo de diagrama de llaves en una estructura condicional:



Ejemplo de diagrama de llaves en una estructura iterativa:



TEMA 3: INTRODUCCIÓN AL LENGUAJE C

♦ *HISTORIA*

Fue desarrollado por la compañía Bell Telephone en 1972. Principalmente contribuyeron a su creación:

DENNIS RITCHIE y KEN THOMPSON

Antes de llamarse C existión una versión del lenguaje a la que se la denominó B. El lenguaje C es un lenguaje desarrollado para aplicaciones técnicas, tales como S.O, compiladores, ensambladores, editores de texto, etc.. Actualmente existen varios tipos de lenguaje C:

- C Standard (Utilizado en UNÍS SYSTEM V)
- C ANSI
- BORLAND C++ (Desarrollo de los lenguajes anteriores con mejores y programación orientada a objetos o eventos.)

♦ *CARACTERÍSTICAS DEL LENGUAJE C*

Se considera a nivel teórico, que es un lenguaje de propósito general, es decir que no esta desarrollado especialmente para aplicaciones explícitas. Si no que en este lenguaje se pude implementar cualquier tipo de algoritmo.

Se considera un lenguaje de alto nivel. Aunque roza al lenguaje de bajo nivel, ya que C porta instrucciones de bajo nivel, por lo que es posible programar en C a bajo nivel, esto hace de C un lenguaje mas potente que cualquier otro.

Es un lenguaje potente y eficiente.

Se considera un lenguaje muy portable. Esto significa que un programa escrito bajo UNIX, se podrás trasladar fácilmente a MSDOS.

Es complicado de aprender y de utilizar a fondo.

♦ E.I.D (Entorno Integrado de Desarrollo)

Software con el que podremos realizar todas las tareas propias de la programación y del desarrollo de aplicaciones informáticas como son las tareas de edición del código fuente, compilación, linkados, etc..

♦ COMPILACIÓN Y ENLAZADO

El programa fuente, que podemos guardarlo tanto en ficheros *.C como en *.CPP, no se puede ejecutar, antes hay que realizar un proceso previo en el que se codifican las instrucciones a lenguaje maquina, a este proceso le llamamos compilación. Después de este proceso realizamos el enlazado.

Cuando compilemos el código fuente, se creará un fichero con extensión *.OBJ, que contendrá las instrucciones en lenguaje maquina equivalentes al código fuente que nosotros editamos. Cuando realizamos el proceso de enlazado, se crea un fichero ejecutable con extensión EXE, que podremos ejecutar en el S.O.

♦ ESTRUCTURA DE UN PROGRAMA EN C

La estructura normal de un programa en C es la siguiente:

- Directrices del preprocesador.
- Definición de Constantes.
- Prototipo de Funciones.
- Definición de variables globales.
- Programa Principal:
  - Declaración de variables locales.
  - Sentencias.
- Desarrollo de las funciones definidas por el programador.

♦ ***Directrices del preprocesador***

En esta parte, definimos las pautas que ha de seguir el procesador antes de compilar el código fuente. Antes de compilar se han de incorporar las directrices. Antes de nada recordar que el lenguaje C es un lenguaje CASE SENSITIVE, es decir sensible a las letras mayúsculas y minúsculas.

Ej.:

#include <stdio.h>

#include <conio.h>

stdio.h es la abreviatura de Standard Input Output Headers.

Conio.h es la abreviatura de Console Standard Input Output Headers.

Los símbolos < > indican que los archivos de cabecera de van a buscar en el directorio Include, mientras que si usamos indicaríamos que antes de buscar los archivos de cabecera en el directorio include, los buscara en el directorio actual.

♦ ***Definición de constantes***

Definimos una constante del siguiente modo:

#define nombre_constante valor_constante

Ej.:

#define PI 3.1416

♦ ***Prototipo de funciones***

Además de las instrucciones o funciones propias del lenguaje C, nosotros podremos crearnos nuestras propias funciones. Ej.:

void PRESENTACIÓN (void) ;

♦ ***Declaración de variables globales***

Las variables globales, son variables que están disponibles en cualquier punto del programa. Las formas de definición son:

◊ int N1, N2; /* En este caso solo se definen */
◊ int E=0, F=0 /* Definimos y declaramos un valor */
◊ int G, H=0 /* Definimos 2, y declaramos el valor de una */

Preferencia de los operadores en la declaración de variables:

1.− ( )

2.− * / %

3.− + −

♦ ***Programa Principal***

```
void main ( )

{

instruccion;

instruccion;

}
```

Esta es la forma de declarar el programa principal. Dentro del programa principal tenemos también la declaración de variables locales para main. Ejemplo de declaración de variable local:

```
void main ( )

{

int x;

}
```

♦ ***Desarrollo de nuevas funciones***

Además de poder usar las funciones que ya incorpora el lenguaje C, nosotros podemos crear nuevas funciones para nuestros programas. Estas funciones las desarrollaremos fuera del programa principal. Pero las llamadas a estas funciones estarán dentro del programa principal.

REGLAS DE ESCRITURA

◊ El lenguaje C, como hemos dicho anteriormente es sensible a mayúsculas y a minúsculas. Todas las palabras clave o reservadas del lenguaje C deben escribirse en minúsculas. Hay que tener cuidado con las variables, pues podríamos definir una variable con x mayúscula que tuviese un valor totalmente distinto a otra variable llamada x minúscula.
◊ Cuando estamos editando un algoritmo en lenguaje C (el código fuente del programa

), podremos escribir en cualquier columna, ya que esto no alterará la ejecución del programa. Ej.:

Instrucción uno;

Instrucción uno;

◊ Para los nombres de variable:
- Pueden tener un numero máximo de 32 caracteres.
- Podemos incluir letras, dígitos y guión bajo.
- No pueden coincidir con palabras reservadas.
- Todas las definiciones se consideran instrucciones, y como toda instrucción en lenguaje C, se acaba con punto y coma.
- Los nombres deben empezar con letras.

◊ Con /* */ indicamos que lo que esta entre las barras y los asteriscos, es un comentario interno del programa (Documentación interna). También pueden usarse // , esta dos barras seguidas comunican al compilador que todo lo que las sigue en esa línea es un comentario, por lo que por cada línea de comentario habrá que poner las dos barras, la ventaja es que no hay que cerrarlas al final del comentario.

◊ Un BLOQUE, es una estructura que se abre con una llave, al final de la misma se cierra con otra. Ej.:

{

instrucción o función

instrucción o función

}

Cada bloque puede contener a su vez otro bloque.

TIPOS DE DATOS STANDARD

FLOAT : Ocupa 4 Bytes −−−−−− Coma Flotante de Simple Precisión −−−−

Máxima precisión de 6 dígitos.

INT: Ocupa 2 Bytes −−−−−−−− Numero Entero −−−− − 32768 a 32767

DOUBLE: Ocupa 8 Bytes −−−−− Coma Flotante de Doble Precisión −−−−

Máxima Precisión de 12 dígitos.

CHAR : Ocupa 1 Bytes −−−− Representa un Carácter −−− (−128 a 127)

FUNCIONES DE ENTRADA SALIDA

VISUALIZACION POR PANTALLA

Generalmente para visualizar los datos en la pantalla, utilizaremos la función ***PRINTF*** seguida de unos argumentos y modificadores de formato. Ej.:

printf (cadena de texto con modificadores de formato);

Este función esta incluida en el archivo de cabecera STDIO.H, por tanto antes de utilizarla deberemos cargar este archivo como #include <stdio.h>. Nos va ha servir para representar información en la pantalla.

Para escribir una variable dentro del texto, deberemos utilizar un código de formato, indicando el tipo de variable que se va ha representar, y después, fuera de las comillas que encierran la cadena de texto, indicar el nombre de la variable o variables que van ha ser representadas en pantalla. Ej.:

printf(El numero es: %i, ENTERO);

En la pantalla se visualizaría, suponiendo que ENTERO vale 2:

El numero es: 2

El símbolo % especifica que lo que viene a continuación es un modificador de formato.

Puede haber mas de un código de formato en la misma cadena.

Los distintos códigos de formato que se refieren a los tipos de variable son:

%i : Hace referencia a los números enteros.

%c : Cadena de caracteres (Tipo Alfanumérica)

%e : Numero en notación científica.

%f : Coma Flotante de simple precisión, numérico real en notación normal.

Si en un modificador de formato dentro de una función printf, colocamos un numero entero entre el símbolo de porcentaje y la letra que indica el tipo de variable, este indicará el tamaño mínimo de la información que va ha ser visualizada en pantalla, rellenando con blancos a la derecha los dígitos o caracteres que falten para llegar al numero entero especificado. Ej.:

printf(La letra %5c .,A);

La letra A ocupara en pantalla 5 caracteres, cuatro blancos y después la A.

En una función printf con FLOAT, podremos especificar también mediante otra mascara, el número de decimales que queremos representar en la pantalla. Si el numero de decimales reales, fuera mayor al especificado se redondearía la cifra. En el caso de redondear estaríamos perdiendo información en la pantalla, que no en el algoritmo. Ej.:

Printf(La cifra es %7,2f .,CIFRA);

Esta mascara lo que hace es representar un numero de 7 caracteres en total incluyendo la coma, y rellenado con blancos a la derecha si este ocupase menos, y de esos 7 caracteres los dos últimos son dos decimales. Por lo que en pantalla visualizaríamos un número de 4 dígitos de parte entera, una coma, y dos digitos de parte decimal.

Otros formatos de Impresión:

%h : Entero corto de un Byte.

%u : Enteros decimales sin signo de 2 Bytes.

%d : Enteros decimales con signo de 2 Bytes.

%ld : Imprime un entero largo de 2 Bytes

%f: Imprime valores con punto decimal de 4 Bytes.

%lf : Imprime valores con punto decimal de 8 Bytes.

%c: Imprime un carácter de 1 Bytes

%s: Imprime una cadena de caracteres o STRING

%o: Imprime un entero octal sin signo.

%x: Imprime un entero Hexadecimal sin signo.

%X: Imprime un entero Hexadecimal con signo.

%e: Imprime valores reales en notación científica.

%g: Usa %e o %f según el tamaño del valor a imprimir.

%%: Imprime el signo %.

%p: Muestra un puntero.

%i: Imprime enteros decimales con signo.

CAPTURA DE DATOS

La captura de datos puede hacerse mediante la función ***SCANF***, que esta declarada en el archivo de cabecera STDIO.H. Su forma de uso:

scanf(%?, <argumentos>);

Ej.:

scanf(%3i,&VAR);

Mediante esta instruccion, estamos diciendo que acepte tres dígitos de un numero entero y que los guarde en una variable con el nombre de VAR. El %3i es un código de formato que establece el numero máximo de dígitos que van ha ser guardados en la variable VAR. Y &VAR no es mas que una dirección que apunta a la variable VAR.

IMPORTANTE: La función SCANF no recoge los datos tecleados directamente del teclado, si no que lo hace mediante el BUFFER. De modo que si en el BUFFER existen datos residuales, es posible que en una futura llamada a la función SCANF, recoja esos datos residuales como valores introducidos por el usuario y produzca ejecuciones no deseadas. Para

evitar esto, es preciso y recomendable limpiar el buffer antes de cada SCANF.

Podemos limpiar el buffer mediante la función FFLUSH(STDIN); . Ej.:

fflush (stdin);

OTRAS FUNCIONES DE PANTALLA

El archivo de cabecera **CONIO.H**, tiene definidas una serie de funciones de pantalla.

La función **CLRSCR ( )**; borra todo lo que hay en la pantalla, y posiciona el cursor en la posición 1,1 (primera línea y primera columna). Ej.:

clrscr ( );

La pantalla esta compuesta de 1 a 22 líneas y de 1 a 80 columnas.

Con la función **CLREOL ( )**; borramos, desde donde esta situado el cursor hasta el final de la linea donde esta situado el mismo. Ej.:

clreol ( );

Con la función **GOTO X,Y** situamos el cursor en una posición de la pantalla especifica. Donde X son las columnas e Y las filas. Ej.:

goto 3, 3;

TEMA 4: SENTENCIAS DE CONTROL DE PROGRAMA

Entendemos por sentencia de control de programa un sentencia o función que nos permite establecer sentencias de control. Son las siguientes:

IF

Esta instrucción nos permite codificar decisiones. Su formato es:

if (CONDICION)

{

<instrucciones>

}

else

{

<instrucciones>

}

Un bloque de instrucciones consiste en una o varias instrucciones finalizadas cada una con punto y coma encerradas todas ellas entre una llave de apertura y otra de cierre.

Dentro de la condición del IF, pueden utilizarse los siguientes operadores de relación:

< , > , >= , <= Primer nivel de prioridad

= = , !=, Segundo nivel de prioridad

Tambien puede contener operadores lógicos:

! (NOT) Primer nivel de prioridad

&& (AND) Segundo nivel de prioridad

|| (OR) Tercer nivel de prioridad

VALORES BOLEANOS

En el lenguaje C no existen los valores boleanos en si mismos, aunque pueden usarse de manera diferente:

if (F + G)

printf(Esta es la rama de verdadero);

De este modo, si F + G es 0, ejecutará la rama de Falso, y si es cualquier valor diferente de 0 ejecutara la rama de verdadero.

Igual pasaría con el siguiente algoritmo:

if (R=A+B)

printf(Esta es la rama de verdadero);

Si R = 0 entonces se ejecutaria la rama de falso y si toma cualquier valor distinto de 0, toma la rama de verdadero.

Un bloque de instrucciones equivale a una instrucción terminada en punto y coma.

BLOQUE WHILE

Sirve para calificar bucles de tipo habitual. El formato de esta instrucción es el siguiente:

WHILE(<CONDICION>)

<INSTRUCCIÓN> ;

Si ponemos un ; en el final de la linea del while estaríamos dejando el bucle vacío.

BLOQUE FOR

FOR (<EXP1>; <EXP2>; <EXP3>)

<INSTRUCCION> ;

En el bucle for deberemos especificar tres opciones que son opcionales:

Expresión 1: Se ejecuta antes de entrar en el bucle una sola vez(Asignación de la variable que toma un valor inicial).

Expresión 2: Condición de control del bucle.

Expresión 3: Se ejecuta el final de cada una de las repeticiones.

BLOQUE DO−WHILE

DO

<INSTRUCCION>

WHILE(<CONDICION>);

Este bucle se ejecutara de 1 a n veces, es decir que mínimo se ejecutará una vez.

VARIABLES STRING

Podemos definir variables STRING de la siguiente forma:

Char CADENA[10];

Donde lo que esta entre corchetes es la longitud que tiene la cadena.

24