



Lenguaje de programación C

Objetivos



■ General:

- Programar un computador utilizando un lenguaje de programación estructurada, lenguaje C.

■ Específicos:

- Utilizar las instrucciones básicas de un lenguaje de programación estructurada: secuencia, selección y repetición.
- Utilizar las capacidades de manejo de estructuras de datos simples y compuestas.
- Diseñar programas modulares mediante subrutinas.
- Desarrollar programas para la manipulación de archivos de datos.

Planificación (I)

■ Algoritmos	Análisis y resolución de problemas, pseudo código
■ Estructura de un programa C	Encabezado, declaraciones, cuerpo del programa
■ Declaraciones	Identificadores, comentarios, definición de constantes, declaración de variables.
■ Tipos de datos simples	Tipos estándares, compatibilidad y conversión de tipos, reglas de precedencia de operadores, tipos definidos por el usuario, tipo enumerado
■ Sentencias básicas	Asignación, entrada de datos, salida de datos
■ Estructuras de control	Secuencia, acción simple, acciones compuestas; selección, sentencias if, if else, switch ; repetición, sentencias while, do-while, for .

Planificación (II)

■ Subprogramas	Diseño y estructura de programas, funciones (con y sin retorno de valores) , parámetros por valor, variables locales y variables globales.
■ Tipos de datos estructurados	Arreglos, manejo de strings (strcpy , strcat , strcmp , strlen , y otros) algoritmos de búsqueda, algoritmos de ordenación.
■ Tipos de datos estructurados	•Struct
■ Manejo de archivos	<ul style="list-style-type: none">•archivos de caracteres (texto)•archivos de enteros/reales•archivos de registros

Bibliografía



■ Libros:

- Fco. Javier Ceballos. *C/C++ Curso de programación*. Ra-Ma (2ª Ed.), 2001.

- Brian C. Kernighan and Dennis M. Ritchie. *El lenguaje de programación C*. Prentice Hall, 1988.

- Byron S. Gottfried. *Programación en C*. McGraw-Hill, 1991.

- D. R. Llanos. *Curso C bajo UNIX*. Paraninfo, 2001.

Enlaces Interesantes

- C GNU <http://gcc.gnu.org>
- C Programming Notes. Steve Summit.
<http://www.eskimo.com/~scs/cclass/notes/top.html>
- Programming in C: A Tutorial. Brian W. Kernighan.
<http://www.lysator.liu.se/c/bwk-tutor.html>
- Programming in C. Unix System Calls and Subroutines using C. A.D. Marshall <http://www.cs.cf.ac.uk/Dave/C/CE.html>
- C Programming. Steve Holmes.
<http://www.strath.ac.uk/IT/Docs/Ccourse/>
- Programming in C. Geocities.
<http://www.geocities.com/SiliconValley/Software/5562/>
- C Language Tutorial.
http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/c_tutorial.html



Lenguaje de Programación C

Acercas de C

- C fue desarrollado por Brian Kernighan y Dennis Ritchie en los laboratorios Bell.
- Valores son guardados en variables.
- Programas son estructurados por funciones.
- Flujo de Control es controlado usando bucles, sentencias y llamadas a funciones.
- Entrada y Salida son dirigidas a archivos o al terminal.
- Datos son almacenados juntos en arreglos o estructuras.
- C permite un control más preciso de entradas y salidas → Programas pequeños mas eficientes.
- C es un lenguaje amigable.
- ANSI C es proclamado la versión estándar del lenguaje.

Historia

- 1967, Martin Richard crea el lenguaje BCPL
- 1970, Ken Thompson crea el lenguaje B.
 - Transportable.
 - Lenguaje evolucionado e independiente de la máquina.
- 1972, Dennis Ritchie y Brian Kernighan en los Laboratorios Bell de AT&T crean el lenguaje C modificando el lenguaje B. Reescriben Unix en C.
 - Se diseñan tipos y estructuras de datos.
 - Incluye punteros y funciones.
 - Riqueza de operadores.
- 1983, el instituto de estándares americano (ANSI) crea un estándar que definiera el lenguaje C.
- 1990, el estándar es adoptado.

UNIX



- UNIX fue escrito en C.
- C fue “inventado” específicamente para implementar UNIX.
- Todos los comandos de UNIX mas todas sus facilidades (“revisión de password”, “colas de impresión” o controladores de dispositivos) son escritos en C.
- En lenguaje C y con las bibliotecas de UNIX se pueden desarrollar sistemas complejos.

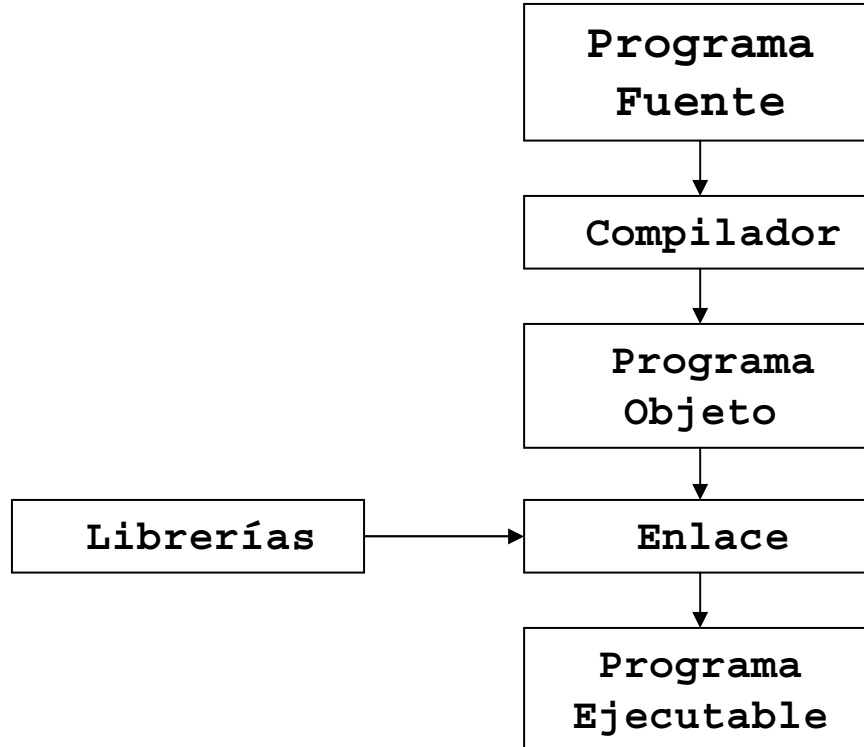
Escribir, Compilar, Ejecutar el Programa

- UNIX espera que su programa sea guardado en un archivo cuyo nombre finaliza en “.c”
- Para **editar**, utilizar un editor como “vi”, “jed”, “joe” o “emacs”
 - \$ vi prueba.c
- Para **compilar** se utiliza un compilador de la forma:
 - \$cc prueba.c -o prueba
- Para **ejecutar** bajo UNIX se debe escribir:
 - \$prueba ó \$./prueba

Compilador de UNIX

Programa ejecutable

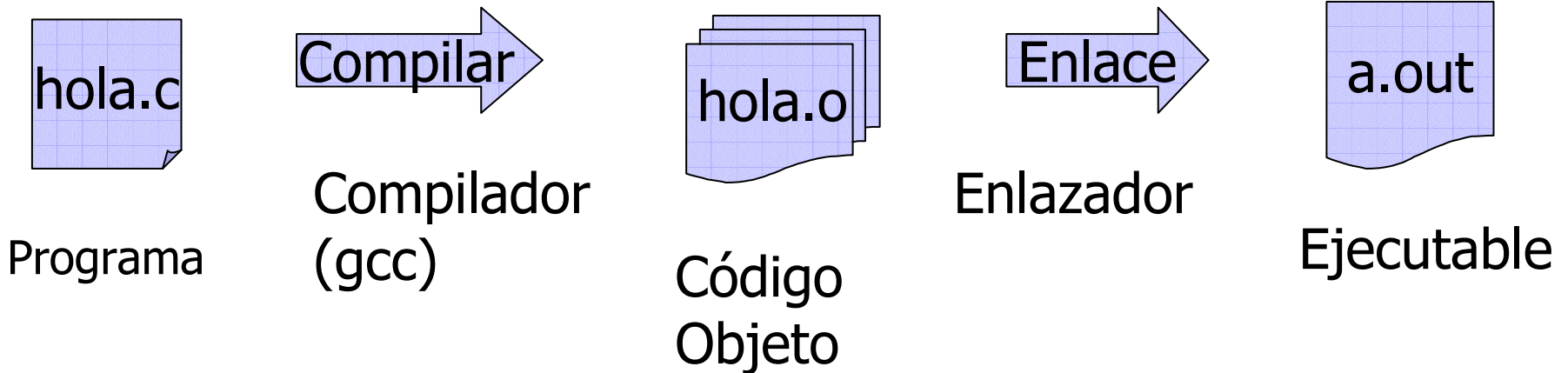
- Para obtener un programa ejecutable hay que seguir los siguientes pasos:



Ejemplo

```
/* Programa Hola Mundo*/  
#include <stdio.h>  
int main( )  
{  
    printf("¡Hola Mundo!");  
    return ( ) ;  
}
```

```
mipc$ gcc hello.c  
mipc$ ./a.out
```



Compiladores C

- Para el sistema operativo Linux:
 - lcc
 - gcc
- Para el sistema operativo Windows:
 - Turbo C/C++
 - Borland C/C++
 - Visual C++
- Editores de texto para Linux:
 - Vi, jed, emacs, joe.

Estructura de un programa

```
Declaraciones globales
f1()
{
    variables locales
    secuencia de sentencias
}
.
.
fN()
{
    variables locales
    secuencia de sentencias
}
main() {
    variables locales
    secuencia de sentencias
}
```

- ❑ Todos los programas en C constan de una o más funciones.
- ❑ La única función que debe estar absolutamente presente es la denominada `main()`, siendo la primera función que es llamada cuando comienza la ejecución del programa.



Tipos, Operadores y Expresiones.

Introducción



- **Variables y Constantes** son los objetos de datos básicos manipulados en un programa.
- Las **declaraciones** establecen las variables a ser usadas, su tipo y su valor inicial.
- **Operadores** especifican que se hace con las variables.
- **Expresiones** combinan variables y constantes para producir nuevos resultados.

Nombres de Variables

- Nombres son hechos de letras y dígitos.
- El símbolo “_” cuenta como letra.
- Letras mayúsculas y minúsculas son diferentes.
- Tradicionalmente se utiliza letras minúsculas para nombres de variables y letras mayúsculas para nombres de constantes.
- Nombres de **palabras reservadas** no pueden ser utilizadas:
 - if, else, int, float, char, switch, case, while, do, ...

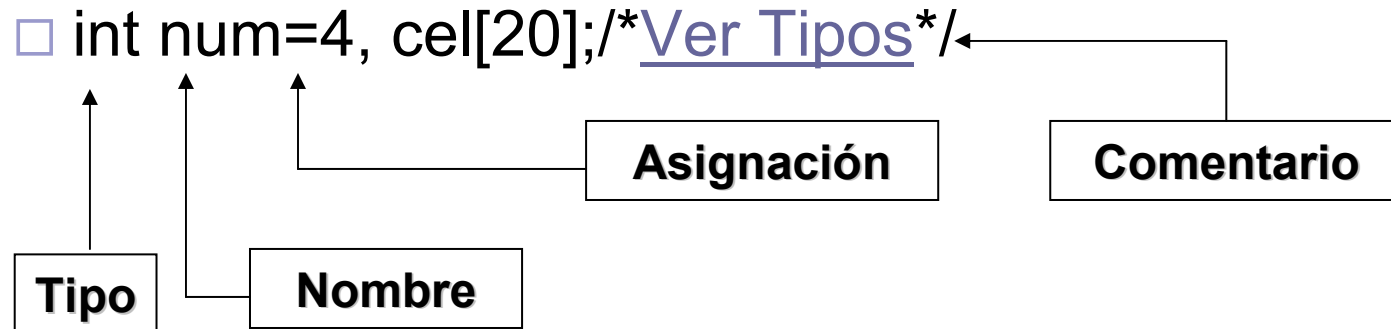
Palabras reservadas

- Es un identificador reservado para propósito específico. No pueden cambiarse. Siempre debe estar en minúscula.
- C tiene 32 palabras claves, 27 del estándar de Ritchie y Kernighan y 5 añadidas por ANSI.
- La lista completa de palabras reservadas de C estándar de ANSI:

auto	break	case	char	const	continue	default
do	double	else	enum	extern	float	for
goto	if	int	long	register	return	short
signed	sizeof	static	struct	switch	typedef	union
unsigned	void	volatile	while			

Tipos de Variables

- En C una variable debe ser declarada antes de ser usada.
- Pueden ser declaradas al comienzo de cualquier bloque de código, pero la mayoría son encontradas al comienzo de la función.
- Una declaración comienza con un tipo, seguido por el nombre de una o mas variables.



Tipos de Variables

int	Numero Entero	int a=3;
float	Numero Punto Flotante	float a=4.3;
char	Un byte de memoria suficiente para guardar un caracter	char a;
short	Entero de tamaño reducido	short int i;
long	Entero de tamaño aumentado	long int i;
unsigned	Entero sin rango negativo, → mayor rango positivo	unsigned int i;
double	Numero de punto flotante de doble precisión	double i;

Constantes

- Son valores fijos que no pueden ser alterados por el programa. Pueden ser de cualquier tipo de datos.
 - Ejemplos: 'a' '\n' '9' 1 123 35000 123.23 3e5
- Una constante en C es sólo la versión escrita de un número. Ejemplos: 4, 3e5
- Un carácter constante usualmente es solo un carácter encerrado entre comillas simples. Ejemplo: 'a'
- Valor de un carácter constante es el valor numérico en el conjunto de caracteres de la maquina.

\n	Newline	\'	Single quote
\t	Tab		
\0	Null		
\\	backslash		

Constantes

- **Expresión Constante** es una expresión que involucra solo constantes
- Para declarar constantes se puede utilizar la directiva **#define**.
 - Ejemplo: `#define MAX 1000`
- Son evaluadas en tiempo de compilación
- **String constante** es una secuencia de cero o más caracteres encerrado entre comillas.
 - Ejemplo: “Lenguaje de Programacion en C”
- Las comillas son delimitadores.
- El carácter nulo ‘\0’ es automáticamente colocado al final del string

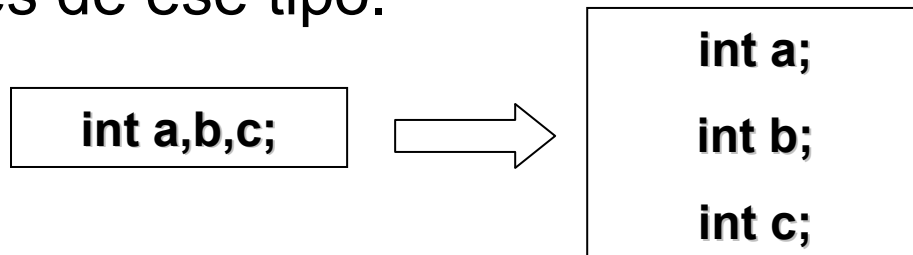
Declaración de variables

- Todas las variables han de ser declaradas antes de ser usadas y pueden cambiar su valor durante la ejecución del programa.

- Forma:

```
Tipo lista_de_variables;
```

- tipo: debe ser un tipo de datos válido de C.
 - lista_de_variables : puede consistir en uno o más identificadores separados por comas.
- Una declaración especifica un tipo y lista una o mas variables de ese tipo.



Variables Locales vs. Globales

- Variables **locales** son declaradas dentro del cuerpo de una función y **sólo pueden ser usadas dentro de esa función.**
- Estará disponible para otras funciones sólo si al ser llamadas desde la función actual, son pasados los valores de las variables.
- Una variable **global** estará disponible para todas las funciones.
- Prácticas modernas de programación recomiendan no abusar del uso de estas variables.

Inicialización de variables

- La asignación se realiza usando el signo '=', donde la constante del lado derecho se asigna al lado izquierdo.

```
Nombre_de_variable = constante;
```

Comentarios

- Los comentarios en C va entre los símbolos `/*` y `*/` o después de `//`.
 - Pueden ir ubicados en cualquier parte del programa.
- Ejemplo:

```
...  
/* Este es un comentario */  
...
```

Variables Externas

Archivo A

```
f0(){  
extern int a;  
...}  
f1(){...}  
int main(){...}
```

Archivo B

```
int a;  
g1(){  
    a=4;  
    ...  
}  
g2(){  
    ...  
}  
int main(){...}
```

- Cuando se utiliza desde una función en un archivo A una variable declarada en un archivo B, debe declararse en la función del archivo A como variable externa.

Variables Estáticas

- Son declaradas como Variables Locales, pero su declaración es precedida por la palabra `static`
- Puede ser accedida sólo desde la función en que fue declarada.
- Inicializada sólo una vez. No es destruida a la salida de la función, sino su valor es preservado, y estará disponible en la próxima llamada a la función.

```
int f1(){...
    static int a;
...}
int main(){...
    f1();
...}
```



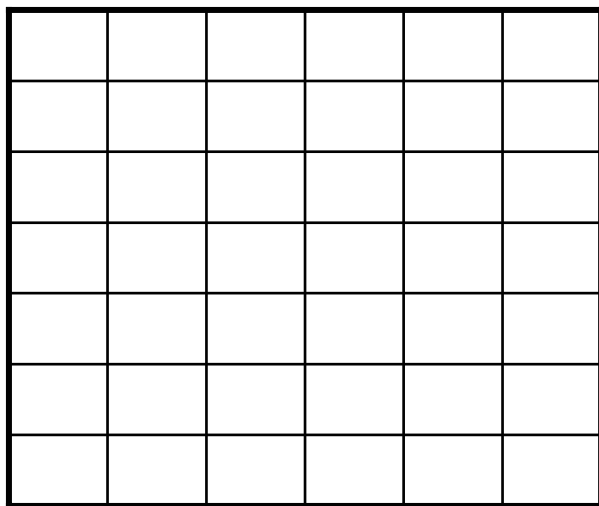
Arreglos

Arreglos

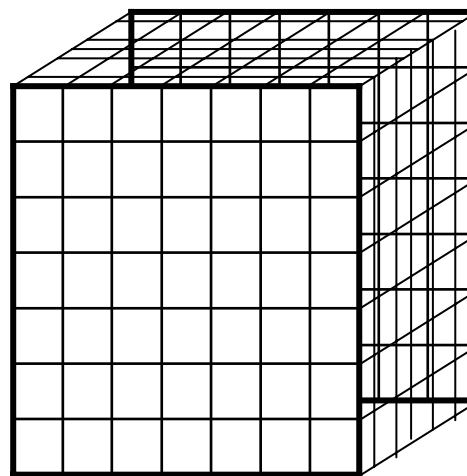
- Los arreglos son una colección de variables del mismo tipo.
- Los elementos individuales son identificados por un índice entero.
- En C el **índice comienza en cero** y siempre es escrito dentro de corchetes.
- Pueden ser de distintas dimensiones.



Unidimensional `int a[20];`



Bidimensional `int a[6][7];`



Tridimensional `float a[7][7][4];`



Operadores y Expresiones

Operaciones y Expresiones

- Una de las potencias de C es su amplio rango de operadores.
- Un **operador** es una función que es aplicada a valores para dar un resultado.
- Existen operadores aritméticos, comparación de valores, combinación de estados lógicos y manipulación de dígitos individuales unitarios.
- Los operadores y valores son combinados para formar **expresiones**.
- Sentencia de Asignación:
 - Una expresión es evaluada y el resultado es guardado en una variable → $y = a * x + b$

Operadores Aritméticos

Operador	Acción
+	Adición
-	Substracción
*	Multiplicación
/	División
%	Resto de división entera
++	Incremento
--	Decremento

Operadores Relacionales

Operador	Acción
>	Mayor que
>=	Mayor que o igual
<	Menor que
<=	Menor que o igual
==	Igual
!=	No igual

Operadores Lógicos

Tabla de Verdad

Operador	Acción
&&	Y
	O
!	NO
&&	Y

p	q	p&&q	p q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

- Los valores lógicos o booleanos no tienen un tipo especial.
- Se puede utilizar cualquier tipo integral que tenga un valor 0 representando falso y cualquier otro valor representando el verdadero.

Precedencias

- Si existen más de un operador perteneciente a un mismo nivel, la prioridad va de izquierda a derecha.

```
(x>=10 && x<20) || x<5  
if(!NULL) printf("...");
```

()
!, ++, --, -, (tipo)
*, /, %
+, -
<, <=, >, >=
==, !=
&&
=, +=, -=, *=, /=
,



Datos definidos por usuario

Tipos de Datos Simples



- Estándar:

- `char, int float, double, void.`

- Definido por usuario:

- `enum, struct.`

Tipos de Datos Simples Estándar

- `char`

- es un carácter (alfanumérico), ocupa 1 byte.

- `int`

- es un número entero, ocupa 2 bytes.
- [-32 768 , 32 767]

- `float`

- son números con decimales y/o exponentes, ocupa 4 bytes.
- [3.4e-38 , 3.4e38]

- `double`

- son números con decimales y/o exponentes, ocupa 8 bytes.
- [1.7e-308 , 1.7e308]

- `void`

- no posee valor, ocupa 0 bytes.

Tipos de Datos Simples Estándar

Ejemplo:

```
void main() {  
    /*Declaración de las variables*/  
    int n1,n2;  
    float r;  
    char ch;  
    unsigned int a;  
    const char p='a';  
    /*Realizar algo con la variables*/  
}
```

Modificadores de tipo y de acceso

- Los modificadores de tipo se utilizan para alterar el significado del tipo base para que se ajuste más precisamente a las necesidades de cada momento.
- Modificadores de tipo: `signed`
`unsigned`
`long`
`short`
- Los modificadores de acceso se usan para controlar las formas en que se acceden o modifican las variables. `const`
- Las variables de tipo `const` no pueden ser cambiadas durante la ejecución del programa.

□ Ejemplo: `const int cuenta = 100;`

Conversión de tipos

- Cuando en una expresión se mezclan constantes y variables de un mismo tipo y variables de distinto tipos, se convierten a un tipo único.
- El compilador de C convierte todos los operandos al tipo del mayor operando.
- Valores pasados a funciones como argumentos debe ser del tipo correcto.
- No hay problema en asignar un valor a una variable de diferente tamaño.
- Excepciones:
 - La variable es demasiado pequeña para sostener el valor.
CORRUPTO
 - Si la variable es tipo entera y se le asigna un valor real, el valor es redondeado.

Conversión de tipos

■ Regla:

- Cualquier char y long int es convertido a int. Cualquier float es convertido a double.
- Para todos los pares de operandos, si uno de los operandos es long double, el otro operando se convierte a long double.
- Si no, si un operando es double, entonces el otro se convierte a double.
- Si no, si un operando es long, entonces el otro se convierte a long.
- Si no, si un operando es unsigned, entonces el otro se convierte a unsigned.

Conversión de tipos, Ejemplo

```
char ch;  
int i;  
float f;  
double d;  
result= ( ch / i) + ( f * d) - ( f + i );
```

The diagram illustrates the type conversion process for the expression `(ch / i) + (f * d) - (f + i);`. It shows the following steps:

- `ch` (char) and `i` (int) are converted to `int` for the division operation.
- `f` (float) and `d` (double) are converted to `double` for the multiplication operation.
- `f` (float) and `i` (int) are converted to `double` for the addition operation.
- The results of the division, multiplication, and addition operations are all `double`.
- The final result of the entire expression is `double`.

Cast (forzado de tipos)

- Sirven para forzar que una variable sea de un cierto tipo.
- Forma general:

`(tipo) expresión`

- Ejemplo:

```
int i;  
float x;  
...  
x = (float) i/1;
```

Tipos definidos por el usuario

- El usuario puede definir sus propio tipos de datos, para ello puede utilizar:

`struct`

`union`

`enum`

Enumeraciones

- Una **enumeración** es un conjunto de constantes enteras con nombres que especifica todos los valores válidos que una variable de ese tipo puede tener.

- Declaración:

```
enum etiqueta {lista_de_enumeraciones} lista_de_variables;
```

- Ejemplo:

```
enum moneda {penique, niquel, dime, cuarto, medio_dólar, dolar };  
enum moneda dinero; /*declara la variable dinero de tipo moneda*/  
  
.....  
dinero=dime; /*Asigna el valor dime a dinero*/  
dinero=3; /*Asigna el valor cuarto a dinero*/
```


Shorthand

Shorthand	Equivalente
<code>i++</code> o <code>++i</code>	<code>i=i+1</code>
<code>i--</code> o <code>--i</code>	<code>i=i-1</code>
<code>i+=10</code>	<code>i=i+10</code>
<code>i*=10</code>	<code>i=i*10</code>
<code>i/=10</code>	<code>i=i/10</code>
<code>i%=10</code>	<code>i=i%10</code>



E/S

Asignaciones

- Es importante tener en cuenta que el signo '=' es utilizado para asignar un valor a la variable y '==' es utilizado para comparar dos variables.
- Modo de uso:

```
Variable_destino=var_1 operador var_2 operador var_3 ....
```

Entrada/Salida

- Para utilizar la entrada y salida de datos por teclado y pantalla respectivamente se debe incorporar la biblioteca:

```
#include <stdio.h>
```

- La instrucción utilizada para la salida de datos por pantalla es:

```
printf()
```

- La instrucción utilizada para la entrada de datos por el teclado es:

```
scanf()
```

printf()

■ Uso:

```
printf(const char *cadena, {lista de_variables});
```

Código	Formato
%c	Un único carácter
%d	Decimal
%i	Decimal
%e	Notación científica
%f	Decimal en punto flotante
%g	Usar el %e o %f, el más corto.
%o	Octal
%s	Cadena de caracteres.
%u	Decimales sin signo
%x	Hexadecimales
%%	Imprime %
%p	Muestra un puntero

printf()

Código	Significado
<code>\b</code>	Espacio atrás
<code>\f</code>	Salto de página
<code>\n</code>	Salto de línea
<code>\r</code>	Retorno de carro
<code>\t</code>	Tabulación horizontal
<code>\"</code>	Comillas dobles
<code>\'</code>	Comilla simple
<code>\0</code>	Nulo
<code>\\</code>	Barra invertida
<code>\v</code>	Tabulación vertical
<code>\a</code>	Alerta
<code>\o</code>	Constante octal
<code>\x</code>	Constante hexadecimal

scanf()

■ Uso:

```
scanf(const char *cadena, {&lista de_variables});
```

Código	Formato
%c	Un único carácter
%d	Decimal
%i	Decimal
%e	Notación científica
%f	Decimal en punto flotante
%g	Usar el %eo %f, el más corto.
%o	Octal
%s	Cadena de caracteres.
%u	Decimales sin signo
%x	Hexadecimales
%%	Imprime %
%p	Muestra un puntero

Ejemplo

```
#include <stdio.h>
void main(){
    int entero;
    float real;
    char ch;
    printf ("Introduzca los siguientes datos: \n");
    printf ("Introduzca un entero:\t");
    scanf ("%d",&entero);
    printf ("\nIntroduzca un real y un carácter:\t");
    scanf ("%f %c",&real, &ch);
    printf ("\nLos valores introducidos son: %d %f %c", entero, real,
    ch);
}
```




Comandos de Control de Flujo

Introducción

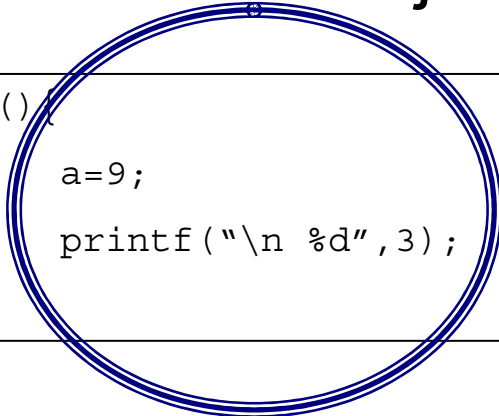
- Control de flujo de un lenguaje especifica el orden en que se ejecutan los cálculos.
- Básicamente son los pasos de un programa.
- La mayoría de las sentencias calculan y asignan valores o llaman a funciones.
- Por defecto una sentencia es ejecutada después de otra.
- Sin embargo se puede modificar la secuencia por usar flujos de control donde un grupo de sentencias son ejecutadas sólo si alguna condición es verdadera. O ejecutada una y otra vez en un bucle.
- Otro tipo de control de flujo ocurre cuando se llama una función; el llamador es suspendido mientras la función llamada procede.
- Una sentencia es un elemento dentro de un programa que puede aplicársele control de flujo.
- Controlar el flujo significa especificar el orden de las sentencias.

Sentencias y Bloques

- Una **expresión** tal como tal como `a=9` o `printf(...)` es una **sentencia** cuando es seguida por un *punto y coma* “;”.
- En C, el *punto y coma* es un terminador de sentencias.
- Las “llaves { }” son usadas para agrupar declaraciones y sentencias juntas en un **bloque**.

```
a=9;  
printf("\n %d",3);
```

```
f()  
{  
    a=9;  
    printf("\n %d",3);  
}
```



Estructura de Control Secuencial

- Es aquella en la cual una acción sigue a otra en forma secuencial.

```
void main() {  
    acción 1;  
    acción 2;  
    ...  
    acción n;  
}
```

Diagrama de FLUJO

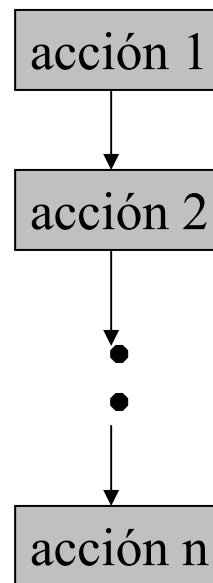
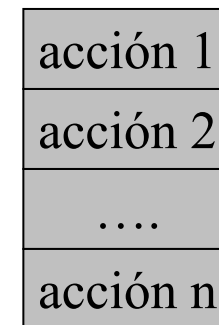


Diagrama de NASSI-SCHNEIDERMANN



Sentencias para Decisión y Control

■ Condicionales y Selección

- if
- If-else
- Switch

■ Bucles:

- for
- while
- do while

■ continue y break

■ goto

Estructura de Control Selectiva

■ Simple (if)

```
if (condición)
    acción;

if (condición) {
    acción_1;
    ...
    acción_n;
};
```

Diagrama de FLUJO

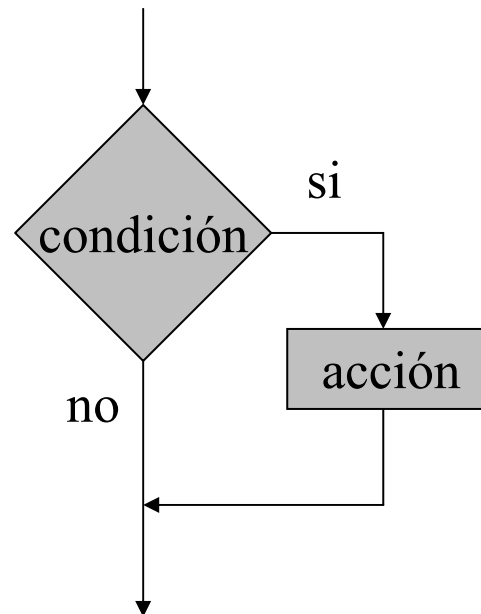
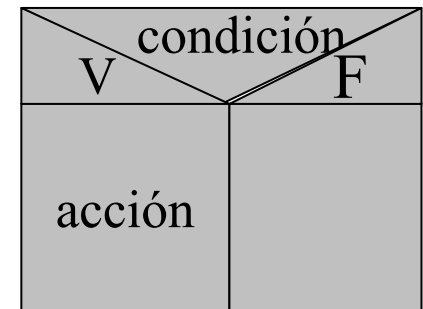


Diagrama de NASSI-SCHNEIDERMANN



Estructura de Control Selectiva Doble

- Doble (if/else)

```
if (condición)
    acción_1;
else acción_2;
```

Diagrama de FLUJO

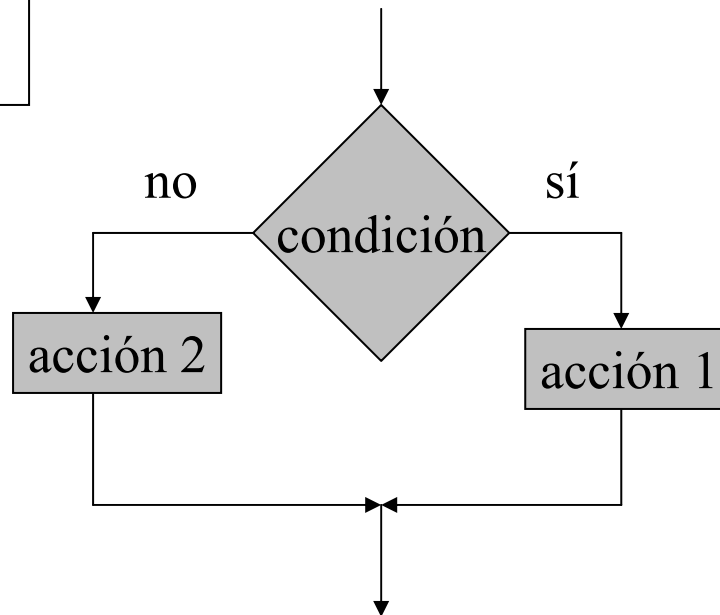
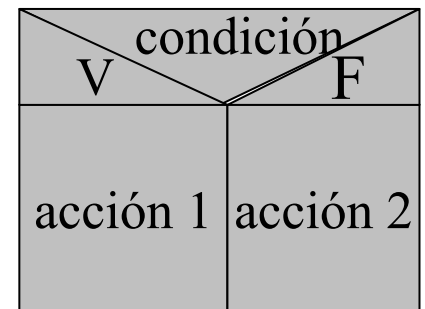
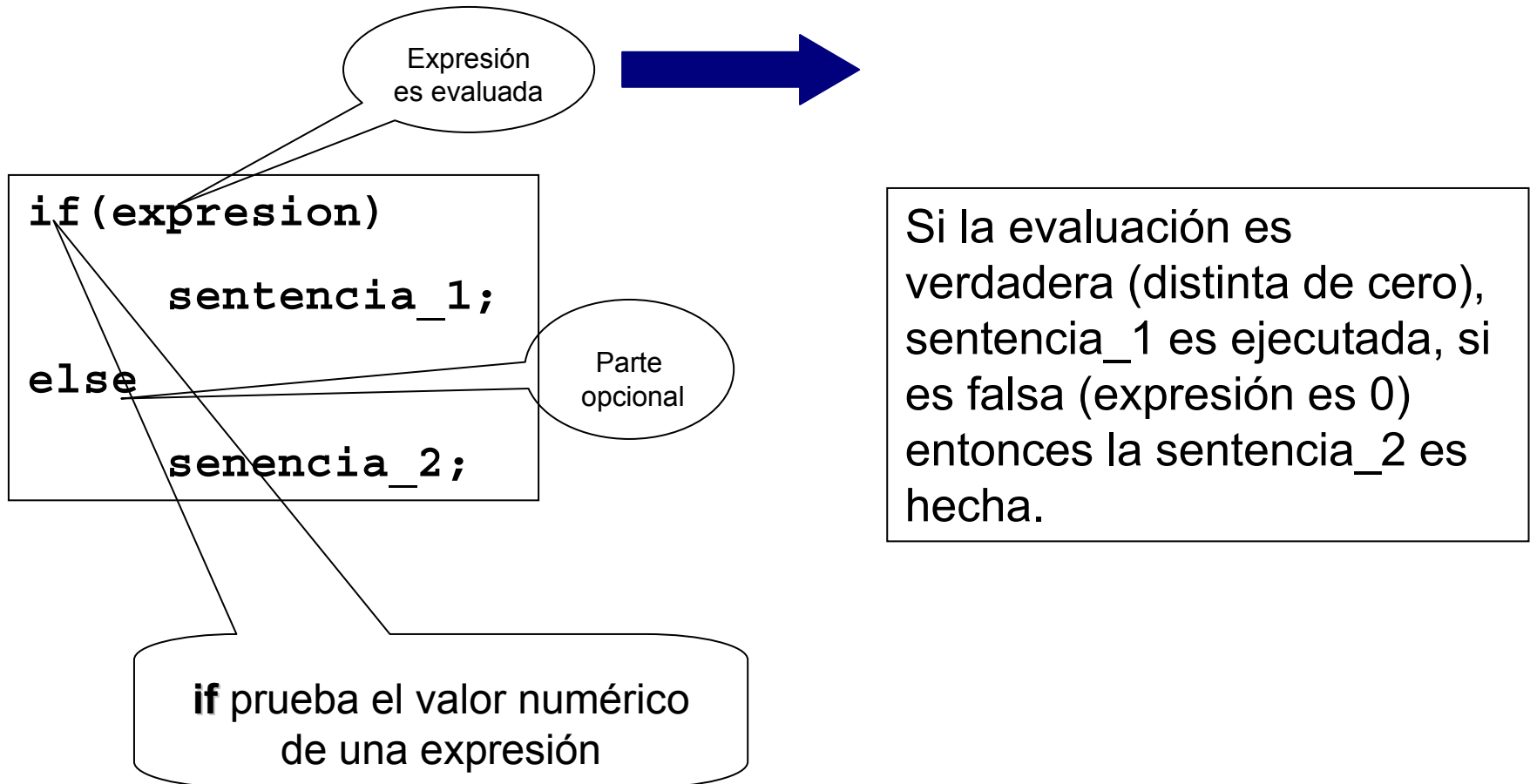


Diagrama de NASSI-SCHNEIDERMANN



if - else

- La sentencia if – else es utilizada para tomar decisiones.



Condicionales

■ Sentencia if, if – else

- if (expresion_condicional) sentencia
- if (expresion_condicional) sentencia
else sentencia

■ Condicional Binaria con una secuencia de sentencias:

- if(expresion_condicional){
 secuencia_de_sentencia
}
- if(expresion_condicional){
 secuencia_de_sentencia
}
else{
 secuencia_de_sentencias
}

Ejemplo

```
/*Dividir el primer numero por el segundo*/
#include <stdio.h>
void main() {
    int a,b;
    printf("Introduce dos numeros");
    scanf("%f %f", &a, &b);
    if(b!=0)printf("z/b=%d", a/b);
    else printf("division por cero");
}
```

```
/*Dividir el primer numero por el segundo*/
#include <stdio.h>
void main() {
    int a,b;
    printf("Introduce dos numeros");
    scanf("%f %f", &a, &b);
    if(b) printf("z/b=%d", a/b);
    else printf("division por cero");
}
```

Ejemplo

- If acepta cualquier expresión

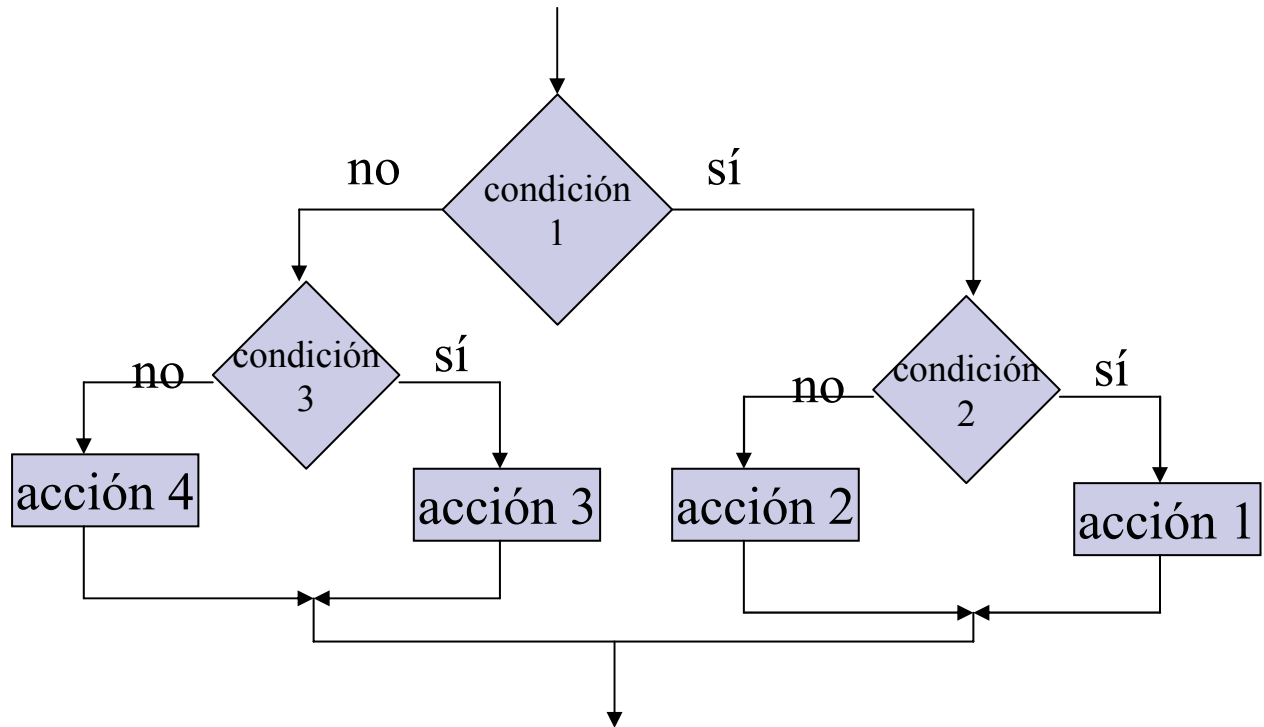
```
int k=1;
if (k=0)
    printf("es un cero");
else
    printf("es %d", k);
```

Estructura de Control Selectiva Anidada

Diagrama de FLUJO

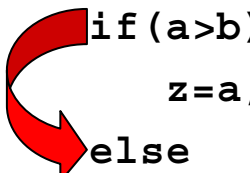
IF anidados:

```
if (condición_1)
  if (condición_2)
    acción_1;
  else
    acción_2;
else
  if (condición_3)
    acción_3;
  else
    acción_4;
```



Anidamiento if-else

```
if (n>0)
    if (a>b)
        z=a;
    else
        z=b;
...
```



?

```
if (n>0) {
    if (a>b)
        z=a;
}
else
    z=b;
...
```

- “else” se asocia con el if precedente (más interno).
- Si no se desea lo anterior, se deben usar llaves.

```
if (n>0)
    for (i=0; i<n; i++)
        if (a>b) {
            z=a;
            ...
        }
else
    printf("n es negativo");
...
```

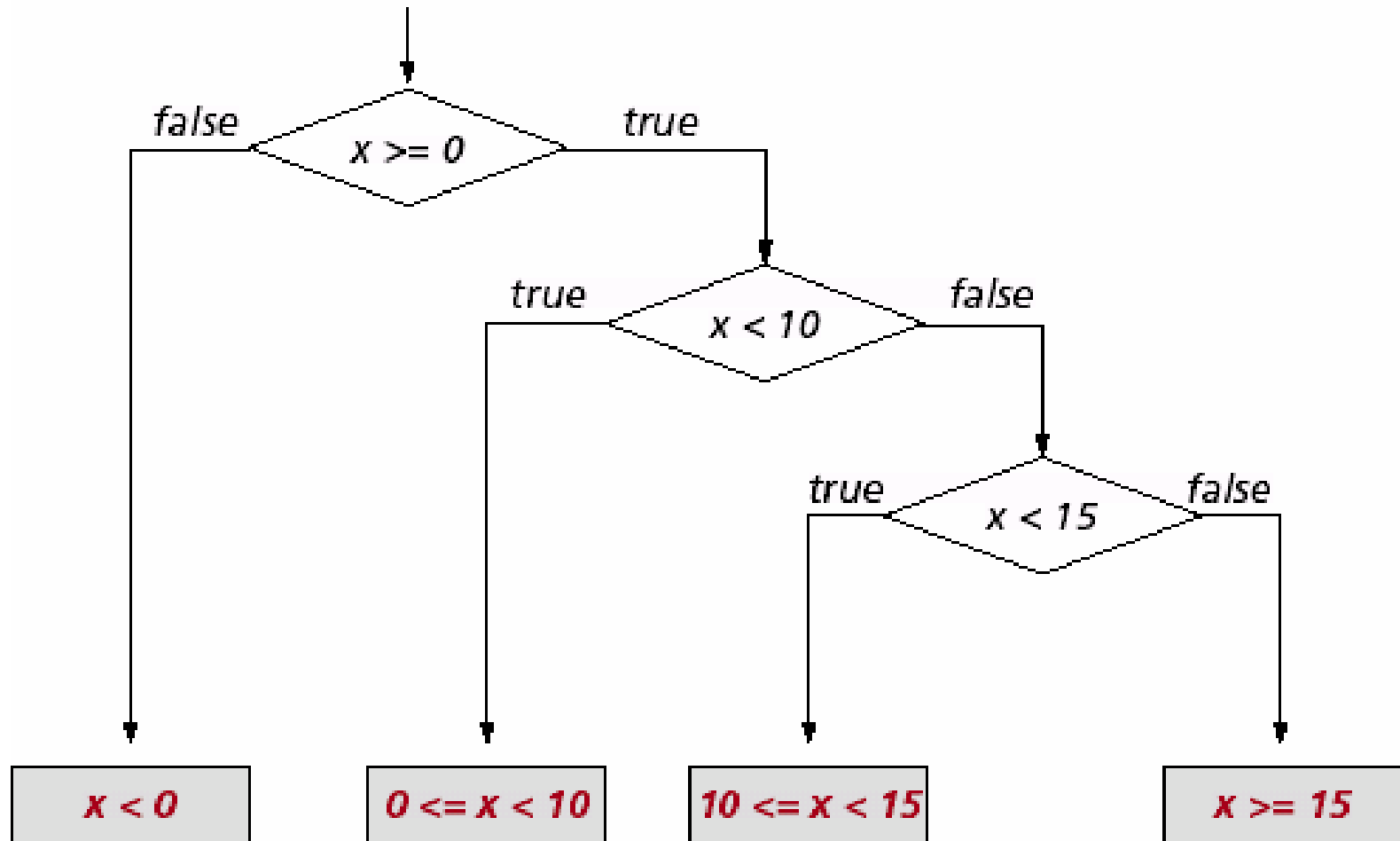
¿Cual es el error?
¿Que supuso el programador?

Anidamiento

```
if (c1) {
    if(c2) sentencia_1;    //c1 y c2
    if(c3) sentencia_2;    //c1 y c3
    else sentencia_3;      //c1 y no c3
}
else sentencia_4;        //no c1

if(x>=0) { //si x es no-negativo
    if(x<10) //... y si x<10
        printf("0<=%d<10",x);
    else{
        if(x>15) //si x está entre 10 y 15
            printf("10<=%d<15",x);
        }
    }
else{
    printf("x es negativo");}
```

Diagrama Comportamiento



Evaluación de Circuitos

- Tan pronto como una expresión compuesta determine completamente el valor de la expresión total, la evaluación se detendrá.

- Ejemplo:

```
if (n != 0)
```

```
    if (0 < x && x < 1/n) sentencia
```

- Mas eficiente

```
if ((n != 0) && 0 < x && x < 1/n)
```

```
    sentencia
```


if else if

- Una estructura de if else anidados

```
if (condicion)
    sentencia;
else
    if (condicion)
        sentencia;
    else
        if (condicion)
            sentencia;
        ...
        else sentencia;
```

Estructura if else con única sentencia

```
if (condicion)
    sentencia;
else if (condicion)
    sentencia;
else if (condicion)
    sentencia;
...
else
    sentencia;
```

```
if(x<0)
    ...// si x es negativo
else if(x>0)
    ...//si x es positivo
else ...//si x es cero
```

if else con secuencia de sentencias

```
if (condicion) {  
    secuencia_sentencias;  
}else{if (condicion) {  
    secuencia_sentencias;  
}  
...  
else{  
    secuencia_sentencias;  
}
```

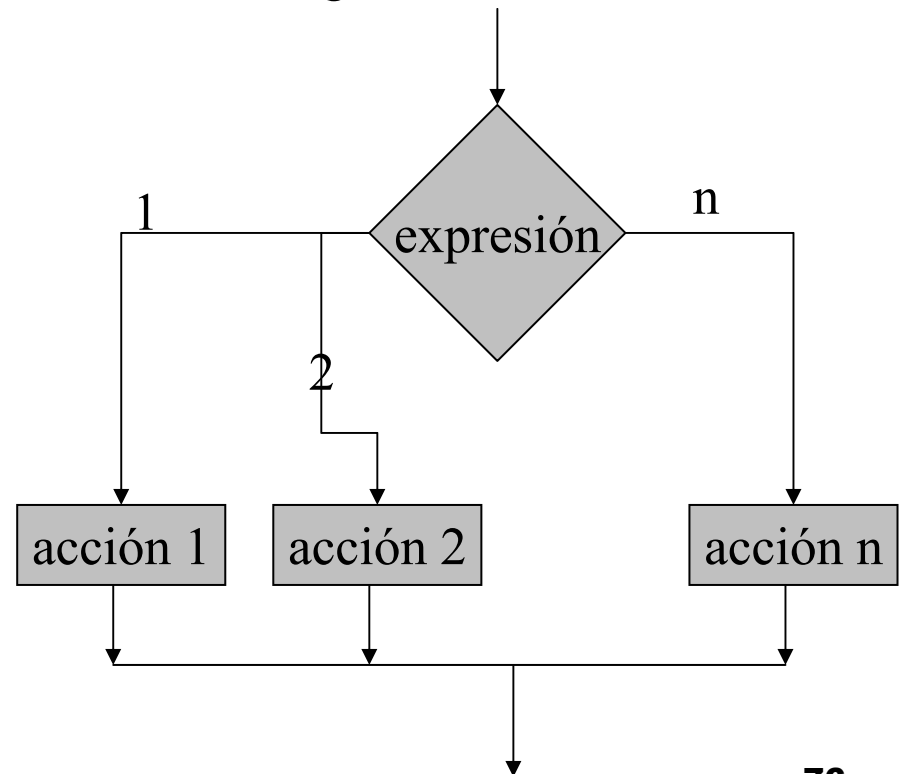
Estructuras de Control Selectivas

- Múltiple: Cuando se desea que existan más de dos decisiones (según sea).

CASE:

```
switch (alternativa) {  
    case alternativa_1:  
        <acción 1;>  
    case alternativa_2:  
        <acción 2;>  
        break;  
    ...  
    case alternativa_p:  
        <acción p;>  
    default:  
        acción_n;  
};
```

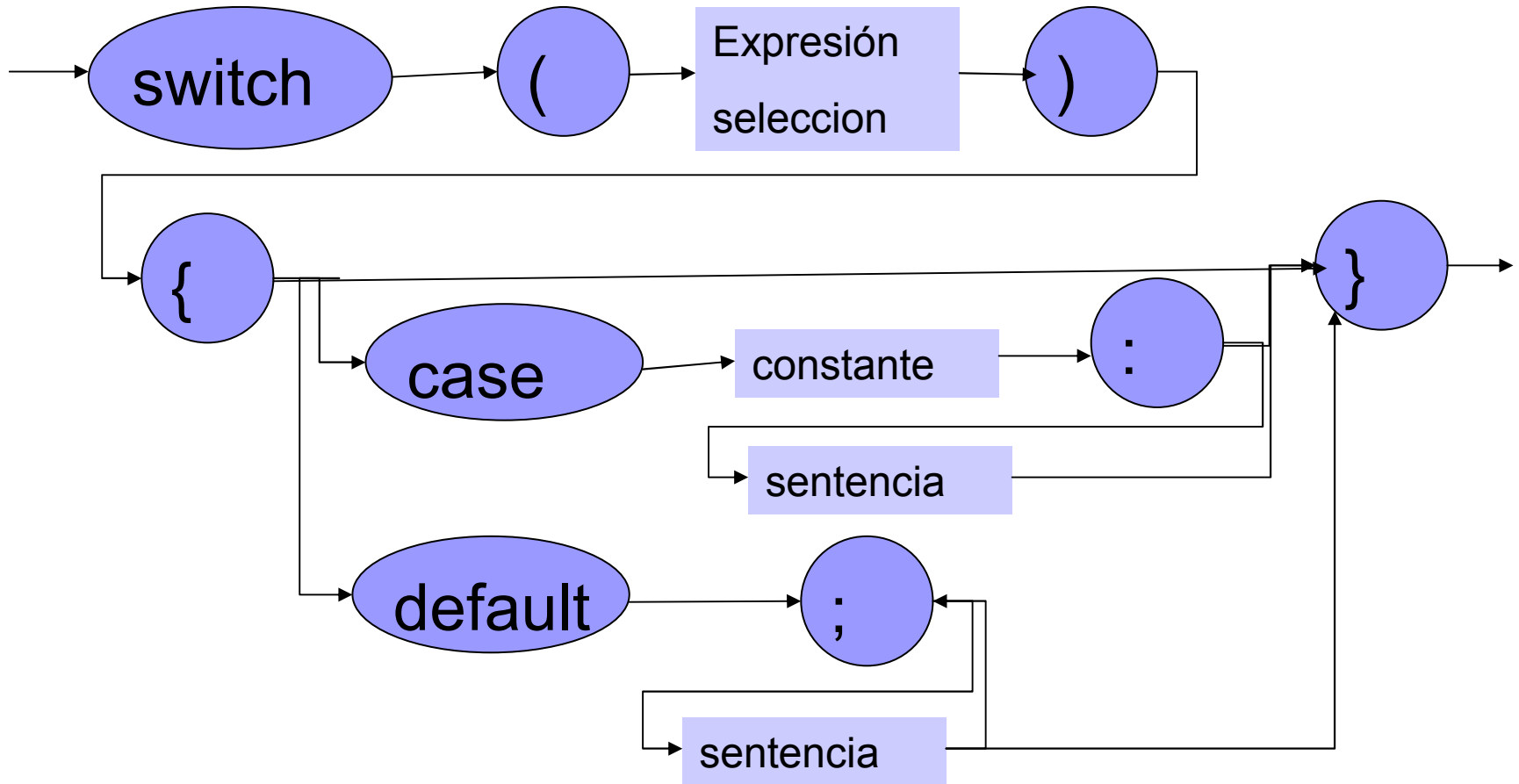
Diagrama de FLUJO



Switch

- La sentencia switch es una manera especial de tomar decisiones que prueba si una *expresión* toma alguno de los valores constantes.
- switch evalúa la expresión entera (integer) entre paréntesis.
- Cada caso debe ser etiquetado por un entero o carácter constante o expresión constante.
- El caso etiquetado por default (OPCIONAL) es ejecutado si ninguno de los otros casos son satisfechos.
- Los casos deben ser todos diferentes.
- *break* causa una salida inmediata del switch.

Sentencia Switch



Estructura Switch

```
switch (expresion_seleccion) {  
    case constante1:  
        secuencia_sentencia  
        break;  
    case constante2:  
        secuencia_sentencia  
        break;  
    ...  
    default:  
        secuencia_sentencia  
}
```

Comparación if-else vs. Switch-case

- Contar dígitos, espacios en blancos y otros caracteres.

```
main() {
int c,i,n_blanco,n_otro,n_digit;
n_blanco=n_otro=0;
for(i=0;i<10;++i) n_digit[i]=0;
while((c=getchar())!=EOF)
    if(c>='0' && c<='9')
        ++n_digit[c-'0'];
    else if(c==' '
||c=='\n' ||c=='\t')
        ++ n_blanco;
    else
        ++n_otro;
```

```
printf("digitos=");
for(i=0;i<10;i++)printf("%d",n_digit[i]);
```

```
while((c=getchar())!=EOF)
    switch(c) {
        case '0':case '1': case
'2':case '3': case
'4':case '5': case
'6':case '7': case
'8':case '9':
            n_digit[c-'0']++;
            break;
        case ' ':
        case '\n':
        case '\t':
            n_blanco++;
            break;
        default:
            n_otro++;
            break;
    }
```




Ciclos

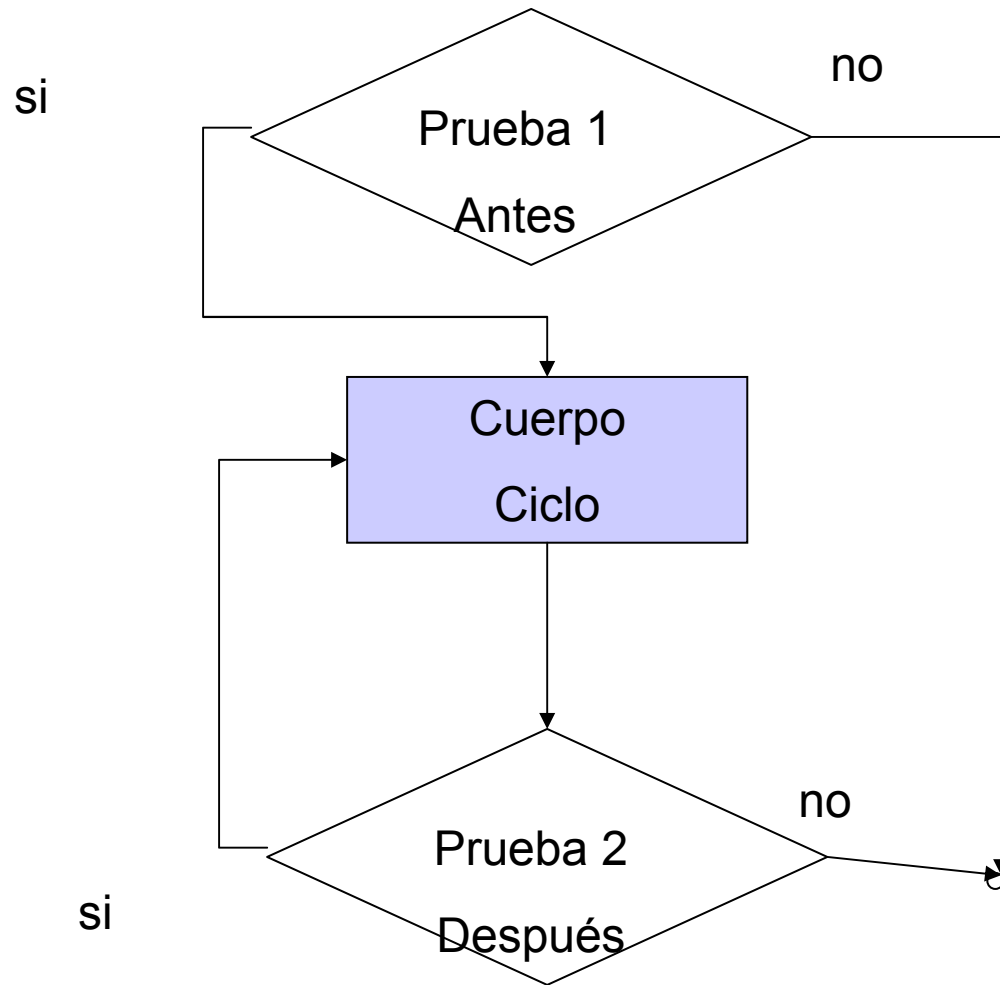
Ciclos – “Haciendo cosas una y otra vez”

- Ciclos son estructuras de control que repiten una serie de sentencias sin tener que reescribir
- Ciclos son usados comunmente para:
 - Contar
 - Sumar
 - Repetir multiplicaciones, incrementar, decrementar.
 - Mantener un registro de los valores.
 - Mantener una secuencia de comandos o acciones.

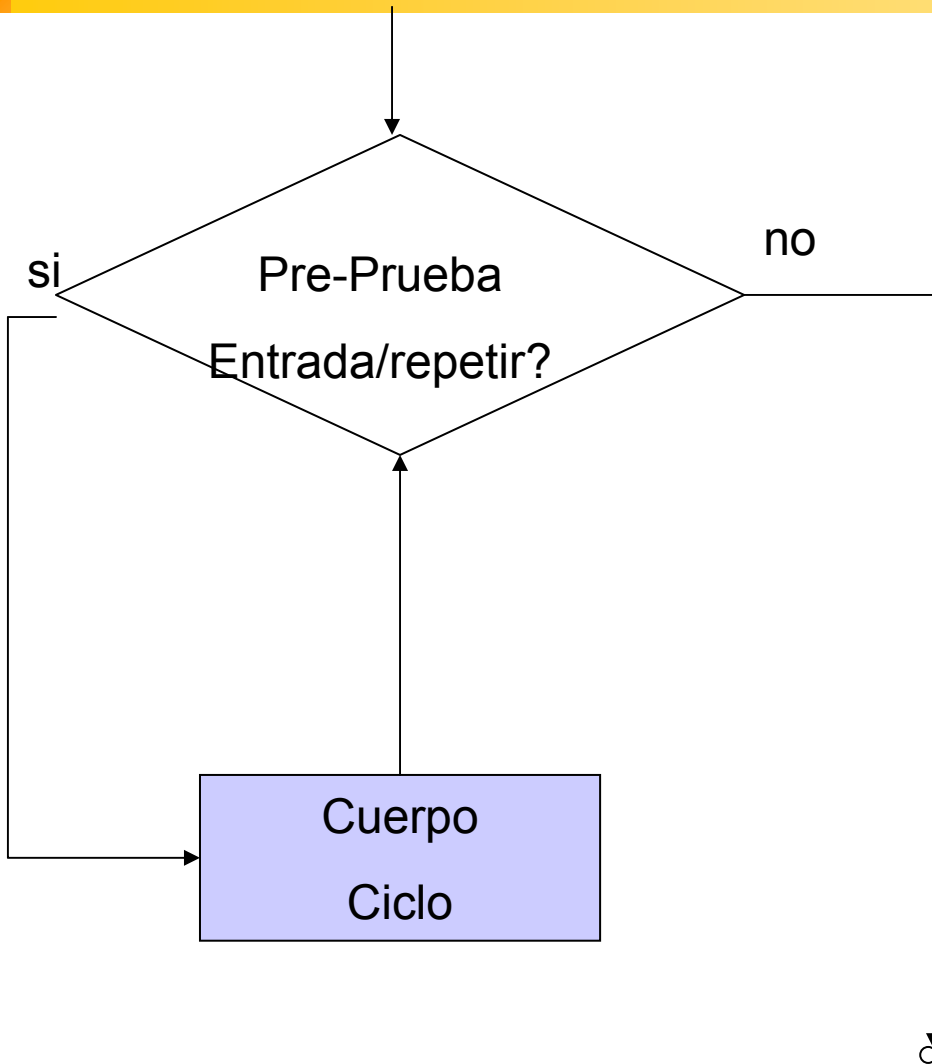
Definiciones

- **Entrada ciclo:** sentencia antes de entrar al ciclo.
- **Cuerpo del ciclo:** sentencias que son repetidas.
- **Condición del ciclo:** expresión a ser evaluada en orden de decidir si es que una nueva repetición debería ser ejecutada.
- **Salida del ciclo:** fin del ciclo, donde el flujo de control deja el ciclo.
- Cuando se escribe una instrucción de repetición se debería tener en consideración:
 - **Entrada:** las condiciones bajo las cuales se entra al ciclo.
 - **Continua:** condiciones para las cuales se debería continuar en el ciclo.
 - **Salida:** las condiciones bajo las cuales se quiere salir del ciclo.

Estructura General del Ciclo



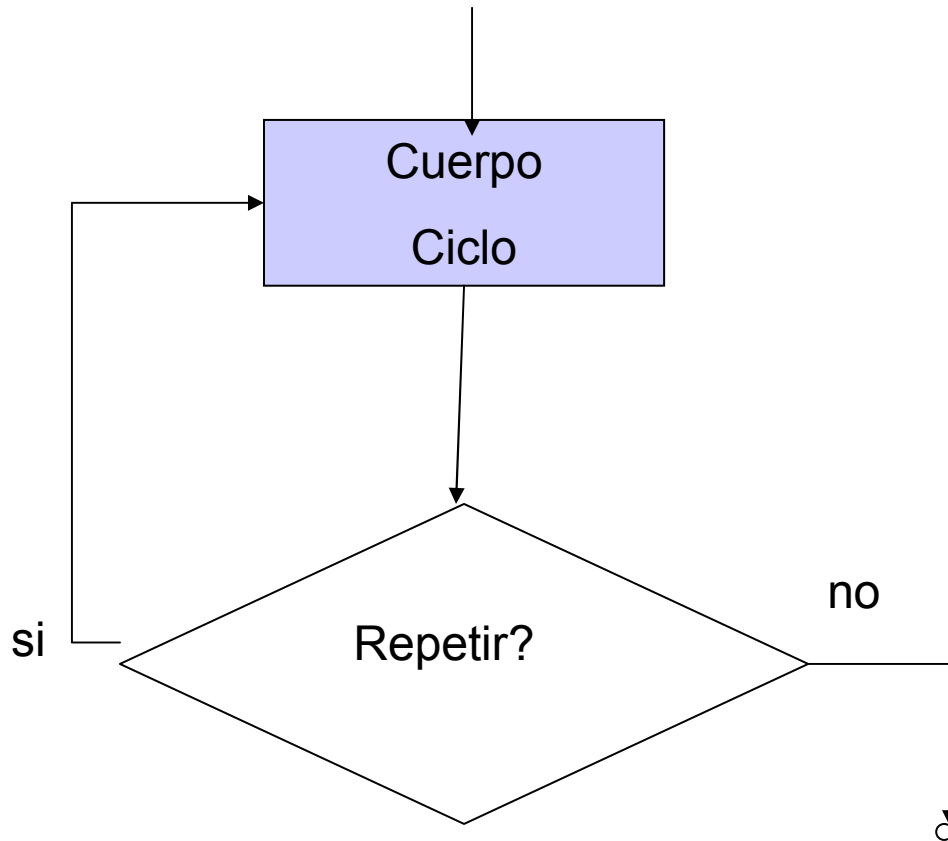
Ciclo Pre- Test (con condición de entrada)



```
for (n=0; n<10; n++)  
{ ... }
```

```
n=0;  
while (n<10)  
{ ...n++; ... }
```

Ciclo Post-Test (con condición de salida)



```
n=0 ;  
do {  
    ...n++; ...  
} while (n<10) ;
```

Estructuras de Control Repetitivas

- FOR: Cuando se conoce de antemano el número de veces que se quiere repetir el ciclo.

FOR

```
for (inicialización; condición;  
incremento )  
acción;
```

```
for (inicialización; condición;  
incremento ) {  
acciones;  
}
```

Diagrama de Flujo

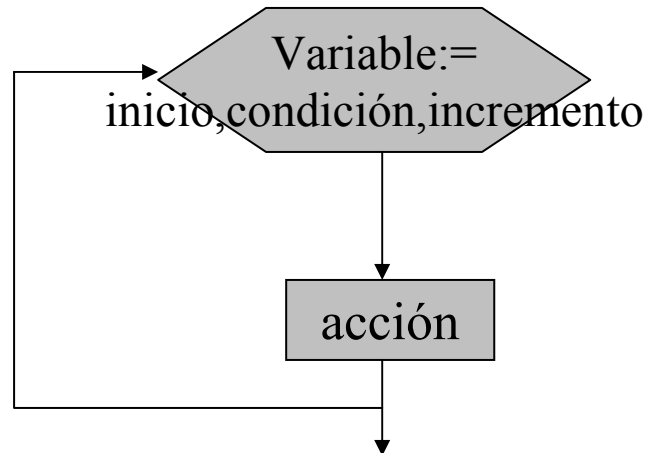
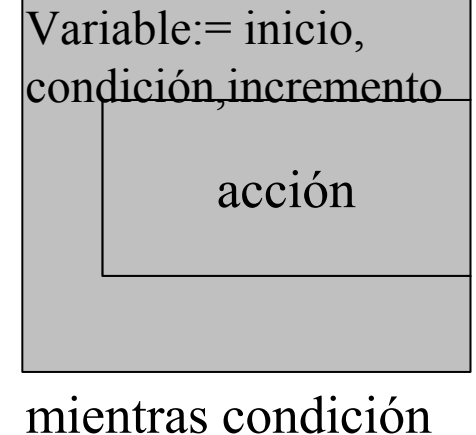


Diagrama NASSI-SCHNEIDERMANN



for

```
for (expr1; expr2; expr3)
    sentencia
```



```
expr1;
while (expr2) {
    sentencia
    expr3;
}
```

- expr1 y expr 3 son asignaciones o llamadas a funciones.
- Expr2 es una expresión relacional.

```
for ( ; ; ) {
    ...
}
```

- Si expr2 no esta presente se considera siempre verdadero → bucle infinito, unica salida **break** o **return**.

Ciclo for (Repetición Fija)

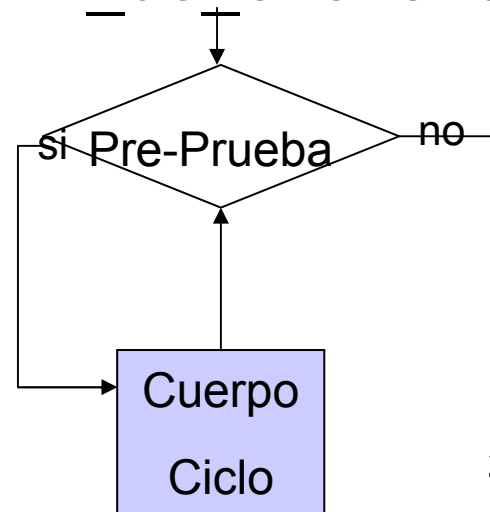
■ Repitiendo una única sentencia

```
for(entrada; prueba_salida; in_de_cremento)
    sentencia;
```

■ Repitiendo secuencias de sentencias

```
for(entrada; prueba_salida; in_de_cremento) {
    secuencia_sentencia;
}
```

Generalmente, ciclos for son controlados por contador



Desarrollo

Calcular los números de Fibonacci: $f_0=0$, $f_1=1$, $f_n=f_{n-1}+f_{n-2}$

Ejemplo: $f_4=f_3+f_2=2+1=3$

```
/*Calcular el n-esimo numero fibonacci */
int prev_fib=0; //=fn-2
int actual_fib=1; //=fn-1
int prox_fib; //=fn
int n,i;
main(){
    printf("Numeros Fibonacci\n");
    printf("Introduce numero n >=0: ");
    scanf("%d",n);
    for(i=0;i<n;i++){
        prox_fib=actual_fib+prev_fib;
        prev_fib=actual_fib;
        actual_fib=prox_fib;
    }
    printf("el %d-esimo fibonacci es %d",n, prev_fib);
    return 0;
}
```

Variaciones en el ciclo for

- Distintas expresiones de inicializaciones e incrementos:

```
for (x=0, y=0; x<=10; ++x, --y)
    printf ("x=%d, y=%d\n", x, y);
```

- Saliendo del ciclo cuando se presione una tecla (función kbhit())
- kbhit() retorna verdadero (es decir distinto de cero) si una tecla es presionada, sino retorna 0, es decir falso.

- Ejemplo:

```
int main() {
    int i;
    /*imprimir numeros hasta que se presione la tecla*/
    for(i=0; !kbhit(); i++)
        printf("%d ", i);
    return 0;
}
```

Estructuras de Control Repetitivas

■ Mientras

WHILE

```
while (<condición>)  
    acción;
```

```
while (<condición>){  
    acción_1;  
    ...  
    acción_n;  
}
```

Diagrama de Flujo

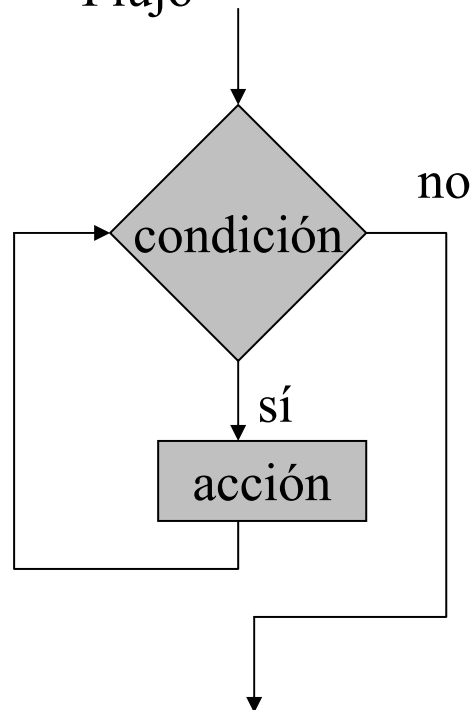
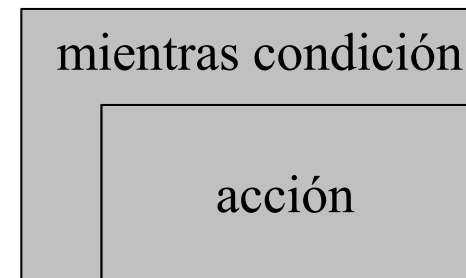


Diagrama NASSI-SCHNEIDERMANN



Ciclo While

- Simple sentencia a repetir:

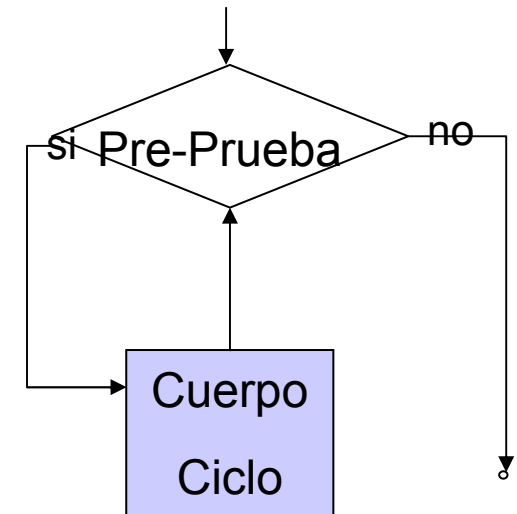
```
while (expresion) sentencia;
```

- Secuencia de sentencias a repetir:

```
while (expresion) {  
    secuencia_sentencia  
}
```

- Generalmente, ciclo while son controlados por eventos.

- La expresión es evaluada. Si es no-cero →sentencia es ejecutada. Este ciclo continua hasta que la expresión se hace cero.



Desarrollo

Calcular los números de Fibonacci: $f_0=0$, $f_1=1$, $f_n=f_{n-1}+f_{n-2}$

Ejemplo: $f_4=f_3+f_2=2+1=3$

```
int prev_fib=0; //fn-2
int actual_fib=1; //fn-1
int prox_fib; //fn
int n,i;
main() {
    printf("Numeros Fibonacci\n");
    printf("Ingrese numero n >=0: ");
    scanf("%d",&n);
    while(i<n) {
        prox_fib=actual_fib+prev_fib;
        prev_fib=actual_fib;
        actual_fib=prox_fib;
        i++;
    }
    printf("el %d-esimo fibonacci es %d",n, prev_fib);
    return 0;
}
```

Código mas eficiente

```
while(i++<n) {
    prox_fib=actual_fib+prev_fib;
    prev_fib=actual_fib;
    actual_fib=prox_fib;
}
```

■ Precauciones

```
main() {...
if(n==0 || n==1) {
    printf("fibonacci de %d es %d",n,n);
    return 0;
}while(...) {...}... return 0;}
```

```
main() {...
    if(n<=1) {//casos especiales
        printf("fibonacci de %d es %d",n,n);
    }
    else{
        while(...) {...}...
    }
    return 0;
}
```

Estructuras de Control Repetitivas

- Repetir: Permite repetir una acción o un bloque de acciones hasta que la condición sea verdadera.

```
DO/WHILE  
do{  
    acción;  
}while (condición);
```

Diagrama de Flujo

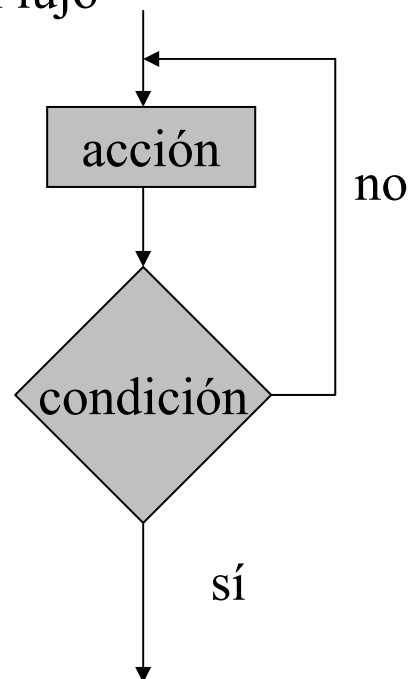
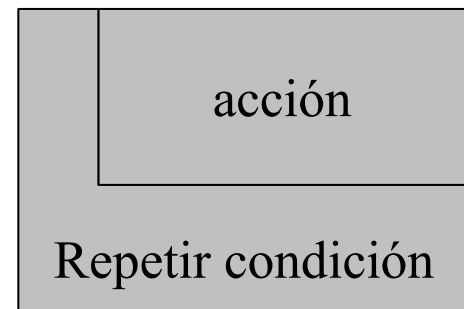


Diagrama NASSI-SCHNEIDERMANN



do while

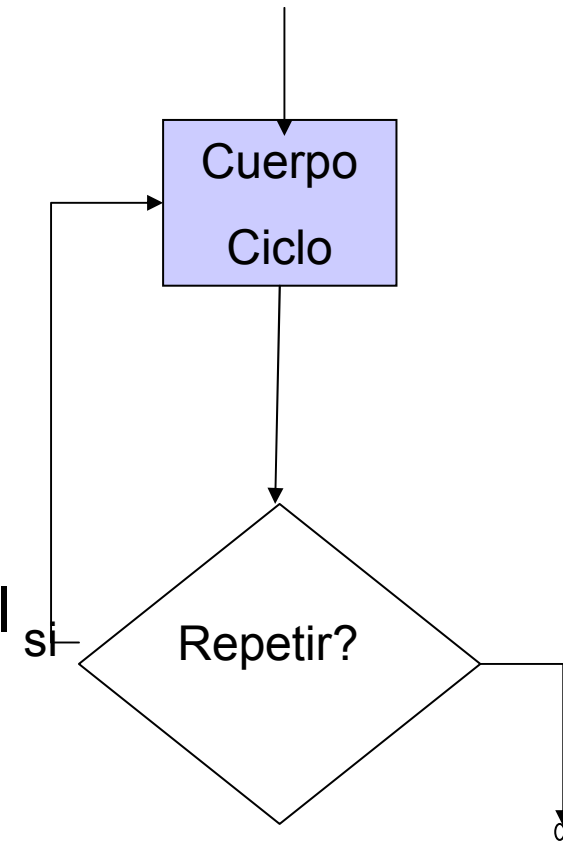
- Única sentencia

```
do sentencia; while (expresion) ;
```

- Secuencia de Sentencias a repetir

```
do{  
    sentencias  
}while (expresion) ;
```

- La condición de finalización está al final del bucle.
- El cuerpo es ejecutado al menos una vez.
- Primero la sentencia es ejecutada y luego la expresión evaluada.



Desarrollo

```
//inicializacion
void main() {
    printf("Ingrese numero n >0: ");
    scanf("%d", n);
do{
    prox_fib=actual_fib+prev_fib;
    prev_fib=actual_fib;
    actual_fib=prox_fib;
} while(i++<n)
    printf("el %d-esimo fibonacci es %d", n,
        (n==0)?0:prev_fib);
}
```

Ciclo Infinito

- Usando **for**

```
for ( ; ; ) { ... }
```

- Usando **while**

```
while (1) { ... }
```

```
while(1) {  
    printf("continua(s/n)\n");  
    scanf("%c", &respuesta);  
    switch(respuesta) {  
        case 's':case 'S': break;  
        case 'n':case 'N':  
            printf("Programa termina\n");  
            return 0;  
        default:  
            printf("Ingrese solo y o n \n"); }  
}
```

break vs. continue

■ break:

- Permite controlar las salidas de los bucles.
- Provee una salida temprana para for, while, do, switch.

```
for (i=0; i<1000; i++) {  
    //hacer algo  
    if (kbhit()) break;  
}
```

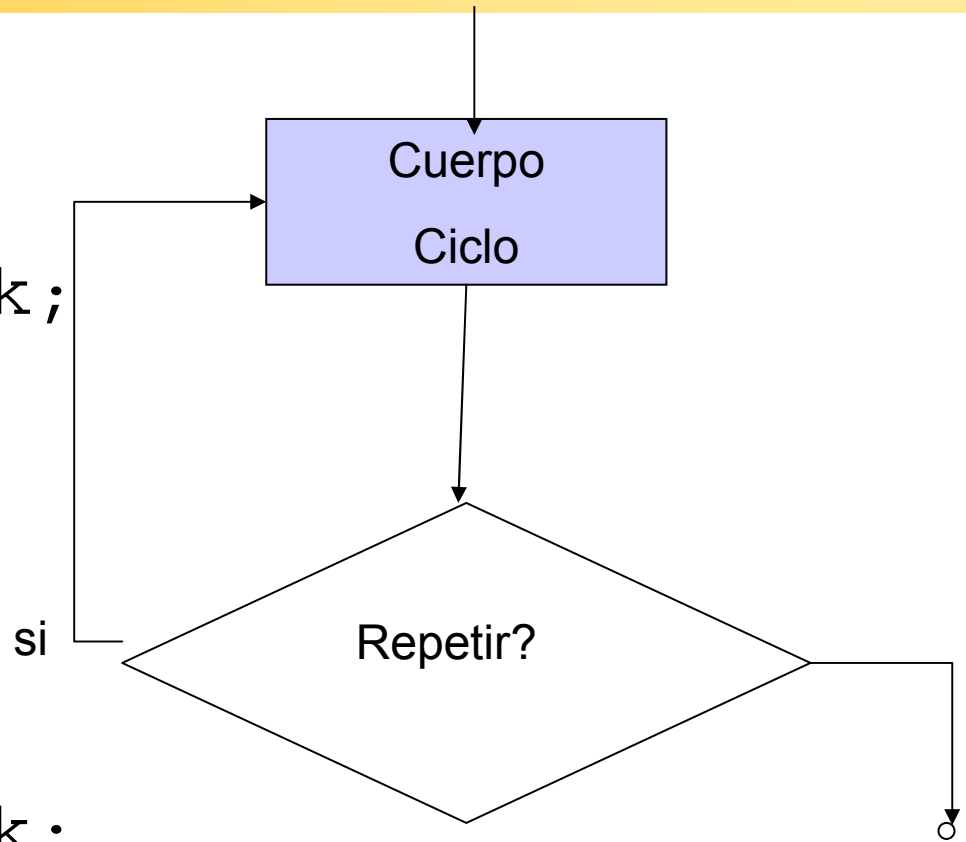
continue:

- Es utilizada cuando la parte del bucle que sigue es complicada.
- Provoca la próxima iteración del bucle cerrado a ser ejecutado inmediatamente.

Alternativa con bucle infinito

```
for(;;) {  
    //hacer algo  
    if(kbhit()) break;  
}
```

```
while(1) {  
    //hacer algo  
    if(kbhit()) break;  
}
```



break vs. continue

■ Break para salir de ciclos

```
int main() {
    int t, cont;
    for(t=0; t<100; t++) {
        cont=1;
        for(;;) {
            printf("contador=%d", cont);
            cont++;
            if(cont==10) break;
        }
        print("\n");
        return 0;
    }
}
```

■ Continue para saltar un ciclo

```
int main() {
    int x;
    for(x=0; x<=100; x++) {
        if(x%2) continue;
        printf("x=%d", x);
    }
    return 0;
}
```

Ejemplo continue

- Sumar valores positivos.
Valores negativos saltarlos.

```
#define N 10;  
...  
int a[N], S;  
...  
for (i=0; i<N; i++) {  
    if (a[i]<0)  
        continue;  
    S+=a[i];  
    ...  
}
```

goto

- Requiere una etiqueta para la operación

```
x=1;
```

```
start:
```

```
    x++;
```

```
    secuencia_sentencias
```

```
    if (x<100) goto start;
```



```
for (x=1;x<100;x++) {secuencia de  
    sentencias}
```


Guías para construir bucles

■ Como diseñar ciclos

□ Proceso

Cuál es el proceso a repetir

Cómo debería inicializarse

Cómo debería actualizarse

□ Condición


Cómo debería inicializarse

Cómo debería actualizarse

Cuál es la condición de termino

□ Después del Ciclo

Cuál es el estado del programa al salir del ciclo



Lenguaje de programación C