Introducción a la programación en C

Informática I - Instituto Unviersitario Areonáutico

May 2, 2021

Etapas de un programa en C

1. Escritura del código fuente: se pueden utilizar diferentes editores de texto: gedit, VIM, Zinjal, Atom, Emacs, etc.

   ▸ Ver en github

Etapas de un programa en C

1. Escritura del código fuente: se pueden utilizar diferentes editores de texto: gedit, VIM, Zinjal, Atom, Emacs, etc.
   ▸ Ver en github
2. Pre-procesamiento: procesa directivas como #include, #define e #if, remueve los comentarios, etc. En general, luego del preprocesamiento, el archivo resultante contiene una gran cantidad de líneas de código.
   ▸ Ver en github

Etapas de un programa en C

1. Escritura del código fuente: se pueden utilizar diferentes editores de texto: gedit, VIM, Zinjal, Atom, Emacs, etc.
   ▸ Ver en github
2. Pre-procesamiento: procesa directivas como #include, #define e #if, remueve los comentarios, etc. En general, luego del preprocesamiento, el archivo resultante contiene una gran cantidad de líneas de código.
   ▸ Ver en github
3. Compilado: el compilador de C traduce el código del apartado anterior a assembler.
   ▸ Ver en github

Etapas de un programa en C

1. Escritura del código fuente: se pueden utilizar diferentes editores de texto: gedit, VIM, Zinjal, Atom, Emacs, etc.
   ▸ Ver en github
2. Pre-procesamiento: procesa directivas como #include, #define e #if, remueve los comentarios, etc. En general, luego del preprocesamiento, el archivo resultante contiene una gran cantidad de líneas de código.
   ▸ Ver en github
3. Compilado: el compilador de C traduce el código del apartado anterior a assembler.
   ▸ Ver en github
4. Ensamblado: el ensamblado transforma el programa escrito en lenguaje ensamblador a código objeto.
   ▸ Ver en github

Etapas de un programa en C

1. Escritura del código fuente: se pueden utilizar diferentes editores de texto: gedit, VIM, Zinjal, Atom, Emacs, etc.
   ▸ Ver en github
2. Pre-procesamiento: procesa directivas como #include, #define e #if, remueve los comentarios, etc. En general, luego del preprocesamiento, el archivo resultante contiene una gran cantidad de líneas de código.
   ▸ Ver en github
3. Compilado: el compilador de C traduce el código del apartado anterior a assembler.
   ▸ Ver en github
4. Ensamblado: el ensamblado transforma el programa escrito en lenguaje ensamblador a código objeto.
   ▸ Ver en github
5. Enlazado: reune uno o más módulos en código objeto con el código existente en las bibliotecas.
   ▸ Ver en github

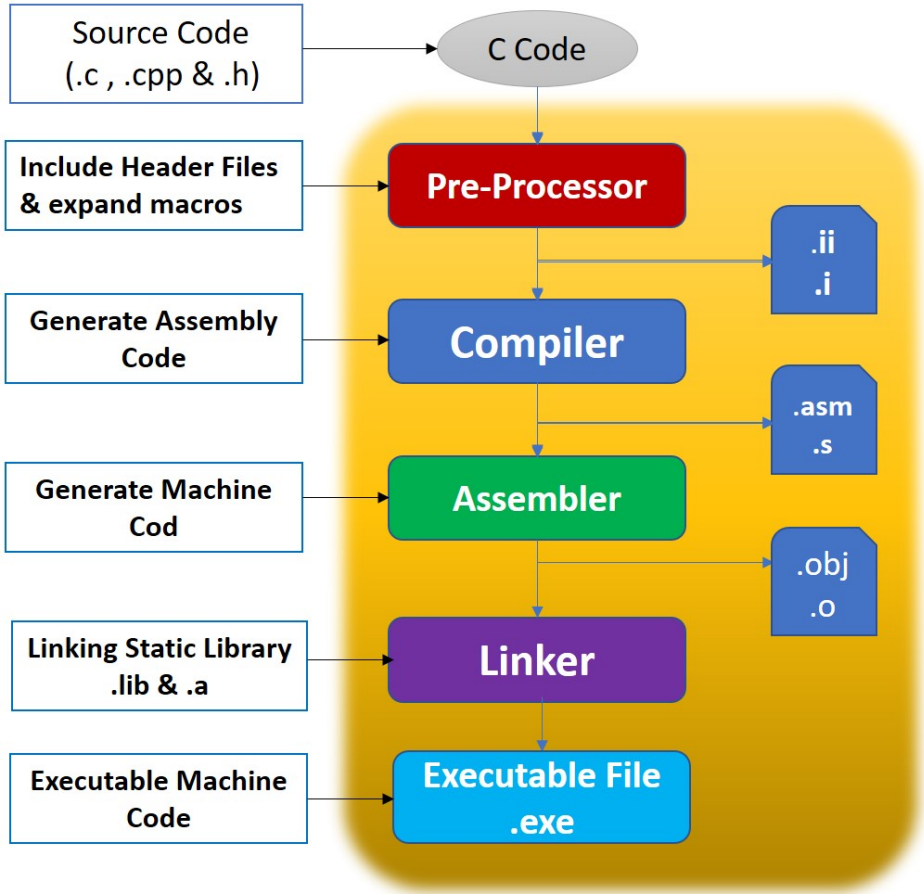


Figure: Etapas de un programa en C.

Estos ejemplos fueron generados agregando el flag "-save-temps" al compilador gcc.

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online
  - Linux kernel 2.2.0

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)
- Proyectos con aproximadamente más de 50 millones

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)
- Proyectos con aproximadamente más de 50 millones
  - Age of empires - Versión online

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)
- Proyectos con aproximadamente más de 50 millones
  - Age of empires - Versión online
  - Linux kernel 2.2.0

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)
- Proyectos con aproximadamente más de 50 millones
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)
- Proyectos con aproximadamente más de 50 millones
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)
- Proyectos con aproximadamente más de 2 mil millones de líneas de código

Buenas practicas de programación

- Proyectos con aproximadamente 300.000 lineas de código o menos
  - Aplicación promedio para smartphone
  - Photoshop v1.0 (1990)
  - Quake 3 (1999)
  - Transbordador espacial STS (1981)
- Proyectos con aproximadamente menos de 10 millones de líneas de código
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)
- Proyectos con aproximadamente más de 50 millones
  - Age of empires - Versión online
  - Linux kernel 2.2.0
  - Windows 3.1 (1992)
- Proyectos con aproximadamente más de 2 mil millones de líneas de código
  - Google Internet Services

Buenas practicas de programación: reglas de estilo

- Variables

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código
  - Debe realizarse con tabulaciones, nunca con espacios

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código
  - Debe realizarse con tabulaciones, nunca con espacios
  - Deben colocarse llaves para demarcar gráficamente el código que ejcutará cada estructura

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código
  - Debe realizarse con tabulaciones, nunca con espacios
  - Deben colocarse llaves para demarcar gráficamente el código que ejcutará cada estructura
  - Las llaves deben escribirse siempre

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código
  - Debe realizarse con tabulaciones, nunca con espacios
  - Deben colocarse llaves para demarcar gráficamente el código que ejcutará cada estructura
  - Las llaves deben escribirse siempre
  - Las líneas de código no deben superar la longitud de 80 caracteres

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código
  - Debe realizarse con tabulaciones, nunca con espacios
  - Deben colocarse llaves para demarcar gráficamente el código que ejcutará cada estructura
  - Las llaves deben escribirse siempre
  - Las líneas de código no deben superar la longitud de 80 caracteres
- Instrucciones que se deben evitar

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código
  - Debe realizarse con tabulaciones, nunca con espacios
  - Deben colocarse llaves para demarcar gráficamente el código que ejcutará cada estructura
  - Las llaves deben escribirse siempre
  - Las líneas de código no deben superar la longitud de 80 caracteres
- Instrucciones que se deben evitar
  - goto

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código
  - Debe realizarse con tabulaciones, nunca con espacios
  - Deben colocarse llaves para demarcar gráficamente el código que ejcutará cada estructura
  - Las llaves deben escribirse siempre
  - Las líneas de código no deben superar la longitud de 80 caracteres
- Instrucciones que se deben evitar
  - goto
  - Break

Buenas practicas de programación: reglas de estilo

- Variables
  - Los nombres deben ser cortos, descriptivos y concretos
  - Si el nombre contiene dos o más palabras, se las debe separar por _
  - Deben inicializarse a un valor conocido al momento de la declaración
- Constantes definidas con # define
  - Su identificador debe escribirse con mayúsculas
- Indentación del código
  - Debe realizarse con tabulaciones, nunca con espacios
  - Deben colocarse llaves para demarcar gráficamente el código que ejcutará cada estructura
  - Las llaves deben escribirse siempre
  - Las líneas de código no deben superar la longitud de 80 caracteres
- Instrucciones que se deben evitar
  - goto
  - Break
  - Continue

Tipos de datos en C

El lenguaje de programación C es **fuertemente tipado**, es decir que cada vez que se necesite declarar u operar con una variable, se debe definir y tener presente el tipo de la misma.
Nota: ver los tipos de datos *Unsigned*

Tipo de dato	Descripción	Rango
short	Valor entero de 2 bytes	-2^{16} a $2^{16}-1$
int	Valor entero de 4 bytes	-2^{32} a $2^{32}-1$
long	Valor entero de 8 bytes	-2^{64} a $2^{64}-1$
char	Caracteres ASCII	-128 a 127
float	Valor decimal de 4 bytes	3.4×10^{-38} a 3.4×10^{-38}
double	Valor decimal de 8 bytes	1.7×10^{-308} a 1.7×10^{-308}
bool	Valor binario	True o False
void	Tipo de dato nulo	
string	Cadena de char	

Table: Tipos de datos en C

Entrada y salida de datos I

Para imprimir por pantalla o ingresar datos a un programa en C, se debe informar **expresamente** el tipo de dato que se espera imprimir y/o recibir. Esto se realiza mediante el uso de **especificadores de formato**.

Tipo de dato	Especificador de formato
short	%*hd*
int	%*d*
long	%*li*
char	%*c*
float	%*f*
double	%*lf*

Table: Especificadores de formato en C

Entrada y salida de datos II

Scanf

Es una función de la librería de entrada/salida de C que permite tomar información desde el teclado.
Sintaxis: scanf("especificador de formato", &variable);
Ejemplo : scanf("%d", &edad);

Printf

Es una función de la librería de entrada/salida de C que permite imprimir información por pantalla.
Sintaxis: printf("especificador de formato", variable);
Ejemplo: printf("Su edad es: " %d , edad);

Nota: para estos ejemplos se supone que la variable se ha declarado de tipo int. Ver ejemplos siguientes.

Entrada y salida de datos III

```
1  #include <stdio.h>
2  int main()
3  {
4     int edad_actual=30;
5     int edad_anio_prox=0;
6     edad_anio_prox= edad_actual+1;
7      return (0);s
8  }
```

Operadores en C I

Operadores de asignación:

Operador	Acción	Ejemplo	Resultado
=	Asignación básica	$x = 10$	x vale 10
*=	Asignación producto	$x* = 10$	x vale 100
/=	Asignación división	$x/ = 2$	x vale 50
+=	Asignación suma	$x+ = 5$	x vale 55
-=	Asignación resta	$x- = 7$	x vale 48

Table: Operadores de asignación

Operadores en C II

Operadores aritméticos:

Operador	Acción	Ejemplo	Resultado
-	Resta	$x = 12 - 3$	x vale 9
$+$	Suma	$x = 12 + 3$	x vale 15
*	Multiplicación	$x = 12 * 3$	x vale 36
/	División	$x = 12/3$	x vale 4
–	Decremento	$x = 12; x--$	x vale 11
$++$	Incremento	$x = 12; x++$	x vale 13
%	Modulo	$x = 13\%2$	x vale 1

Table: Operadores de asignación

Precedencia de operadores

El lenguaje C evalúa las expresiones aritméticas en una secuencia precisa, por lo general son las mismas que aplicaríamos en el álgebra:

1. Las operaciones de multiplicación, división y módulo se resuelven primero. Si en una misma operación aparecen varias de ellas, se resuelven de izquierda a derecha

Precedencia de operadores

El lenguaje C evalúa las expresiones aritméticas en una secuencia precisa, por lo general son las mismas que aplicaríamos en el álgebra:

1. Las operaciones de multiplicación, división y módulo se resuelven primero. Si en una misma operación aparecen varias de ellas, se resuelven de izquierda a derecha
2. Las operaciones de suma y resta se aplican después. Si hubiese varias de estas, C separará en términos al igual que se haría en el álgebra

Precedencia de operadores

El lenguaje C evalúa las expresiones aritméticas en una secuencia precisa, por lo general son las mismas que aplicaríamos en el álgebra:

1. Las operaciones de multiplicación, división y módulo se resuelven primero. Si en una misma operación aparecen varias de ellas, se resuelven de izquierda a derecha
2. Las operaciones de suma y resta se aplican después. Si hubiese varias de estas, C separará en términos al igual que se haría en el álgebra
3. Luego de resueltas todas las operaciones, se procede a la asignación

Precedencia de operadores: ejemplo

Ecuación de una recta en forma algebraica

$$y(x) = ax + b \tag{1}$$

Ecuación de una recta en C

$$y = a * x + b \tag{2}$$

Precedencia de operadores

1. Operación $a * x$

Precedencia de operadores: ejemplo

Ecuación de una recta en forma algebraica

$$y(x) = ax + b \tag{1}$$

Ecuación de una recta en C

$$y = a * x + b \tag{2}$$

Precedencia de operadores

1. Operación $a * x$
2. Operación $+b$

Precedencia de operadores: ejemplo

Ecuación de una recta en forma algebraica

$$y(x) = ax + b \tag{1}$$

Ecuación de una recta en C

$$y = a * x + b \tag{2}$$

Precedencia de operadores

1. Operación $a * x$
2. Operación $+b$
3. Asignación del resultado a la variable y

Precedencia de operadores: ejemplo en C

```
1  #include <stdio.h>
2  int main()
3  {
4     int numero_1=0;
5     int numero_2=0;
6     int numero_3=0;
7
8     int suma=0;
9     int multiplicacion=0;
10    float promedio=0;
11    printf("Ingrese el numero 1 \n");
12    scanf("%d",&numero_1);
13    printf("Ingrese el numero 2 \n");
14    scanf("%d",&numero_2);
15    printf("Ingrese el numero 3 \n");
16    scanf("%d",&numero_3);
17
18
19    suma=numero_1+numero_2+numero_3;
```

Estrucutra selectiva múltiple: Switch

Etapas de un programa en C

Aplicación

Permite que un programa en C tome diferentes caminos en función del valor que tome una determinada instrucción.

Pseudocódigo

Según sea (variable de control)
- Caso 1:
  - Instrucciones caso 1
  - Instrucciones caso 1
  - frenar
- Caso 2:
  - Instrucciones caso 2
  - Instrucciones caso 2
  - frenar
- Caso por descarte:
  - Instrucciones caso por descarte
  - frenar

Estrucutra selectiva múltiple: Switch

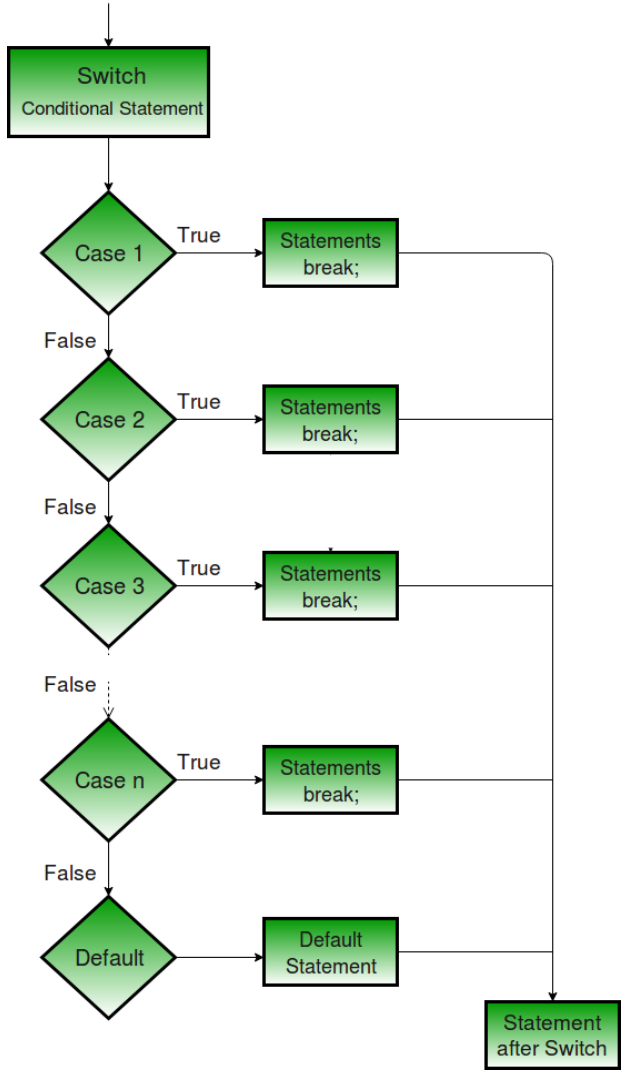


Figure: Diagrama de flujo para la estructura switch

Estrucutra selectiva múltiple: Switch

```
1  #include <stdio.h>
2  int main()
3  {    int opcion=0;
4       printf("Ingrese una numero entre 0-2\n");
5       scanf("%d",&opcion);
6       switch(opcion){
7        case 0:
8                printf("Usted ingreso la opcion 0 \n");
9                break;
10       case 1:
11               printf("Usted ingreso la opcion 1 \n");
12               break;
13       case 2:
14               printf("Usted ingreso la opcion 2 \n");
15               break;
16       default:
17               printf("Usted ingreso una opcion invalida\n"
18               break;}
19
```

Estrucutra selectiva múltiple: Switch - ejemplos

1. Diseñar y codificar un programa que permita simular una calculadora de numeros enteros. Luego de recibir dos operandos enteros, se deben poder realizar las siguientes opciones:
    1. Sumar
    2. Restar
    3. Multiplicar
    4. Dividir Si el usuario ingresa una opción inválida, esta debe ser informada.

    ▸ Ver en github

2. Diseñar y codificar un programa que permita conocer el estado de un alumno en función de la nota de su parcial. Si la nota ingresada es menor a cuatro, se debe imprimir reprobado. Entre cuatro y diez, aprobado. Cualquier otra opción, imrpimir mensaje indicano que la nota es incorrecta. Se debe usar una estructura selectiva switch.

    ▸ Ver en github

Operadores relacionales

Los operadores relacionales comparan el primer operando con el segundo y prueban la validez de la relación expresada. El resultado de una operación relacional es siempre verdadero o falso.

Operador	Relación
$<$	Primer operando menor que el segundo operando
$>$	Primer operando mayor que el segundo operando
$<=$	Primer operando menor o igual que segundo operando
$>=$	Primer operando mayor o igual que segundo operando
$==$	Primer operando igual a segundo operando
$!=$	Primer operando no igual a segundo operando

Table: Operadores relacionales en C

Operadores lógicos

Los operadores lógicos proporcionan un resultado a partir de que se cumpla o no una cierta condición. Sus operandos y su resultado son valores booleanos (verdadero o falso).

Operador	Relación
&&	operador AND (y) lógico
||	operador OR (o) lógico
! =	Operador NOT (no) lógico

Table: Operadores lógicos en C

Estructura selectiva simple: Diagrama de flujo

Definición

Se definen un conjunto de acciones a realizar, sólo si la condición de evaluación resulta VERDADERA. Para el caso contrario, no se realiza ninguna acción.

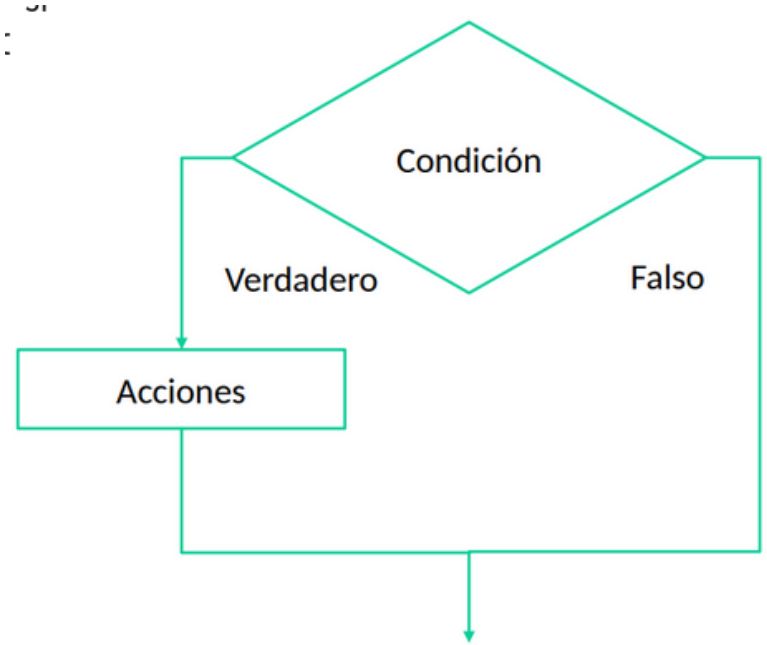


Figure: Diagrama de flujo estructura selectiva simple

Estructura selectiva simple: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar, sólo si la condición de evaluación resulta VERDADERA. Para el caso contrario, no se realiza ninguna acción.

Inicio del algoritmo

Estructura selectiva simple: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar, sólo si la condición de evaluación resulta VERDADERA. Para el caso contrario, no se realiza ninguna acción.

Inicio del algoritmo

    si (condición es verdadera)

Estructura selectiva simple: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar, sólo si la condición de evaluación resulta VERDADERA. Para el caso contrario, no se realiza ninguna acción.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
```

Estructura selectiva simple: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar, sólo si la condición de evaluación resulta VERDADERA. Para el caso contrario, no se realiza ninguna acción.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
```

Estructura selectiva simple: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar, sólo si la condición de evaluación resulta VERDADERA. Para el caso contrario, no se realiza ninguna acción.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
```

Estructura selectiva simple: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar, sólo si la condición de evaluación resulta VERDADERA. Para el caso contrario, no se realiza ninguna acción.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
```

Estructura selectiva simple: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar, sólo si la condición de evaluación resulta VERDADERA. Para el caso contrario, no se realiza ninguna acción.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
Fin del algoritmo
```

Estructura selectiva simple: Ejemplo en C

```c
#include <stdio.h>
int main() {
   int nota;
   printf("Ingrese la nota del examen\n");
   scanf("%d", &nota);

   if( nota < 4)
   {
      printf("El alumno no aprobo el examen\n");
   }


   if( nota >= 4)
   {
      printf("El alumno aprobo el examen\n");
   }
    return (0);
}
```

Estructura selectiva doble: Diagrama de flujo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

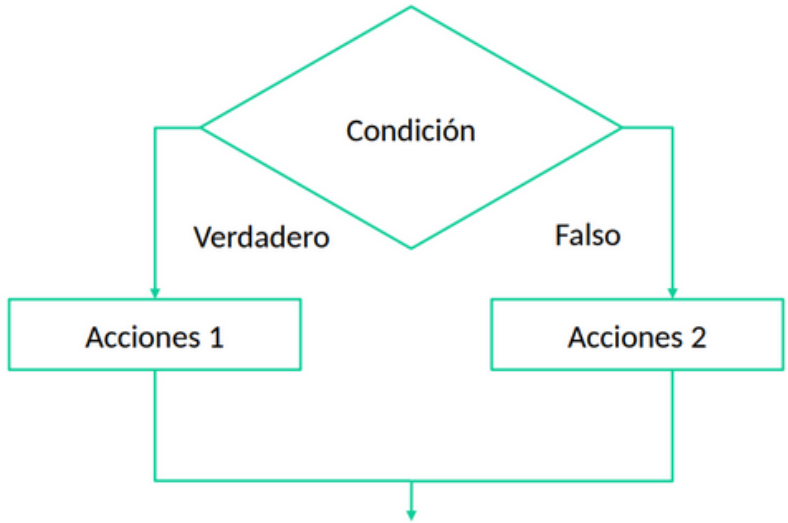


Figure: Diagrama de flujo estructura selectiva doble

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

Inicio del algoritmo

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

Inicio del algoritmo

    si (condición es verdadera)

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
    si no (condición falsa)
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
    si no (condición falsa)
        acción 1;
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
    si no (condición falsa)
        acción 1;
        acción 2;
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
    si no (condición falsa)
        acción 1;
        acción 2;
        acción 3;
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
    si no (condición falsa)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
```

Estructura selectiva doble: pseudocódigo

Definición

Se definen un conjunto de acciones a realizar para el caso en que la condición dentro del rombo de decisión de la figura sea verdadera, como así también para el caso contrario. Es decir, la condición en el rombo de decisión resulte falsa.

```
Inicio del algoritmo
    si (condición es verdadera)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
    si no (condición falsa)
        acción 1;
        acción 2;
        acción 3;
        acción 4;
Fin del algoritmo
```

Estructura selectiva doble: Ejemplo en C

```
1  #include <stdio.h>
2  int main()
3  {
4     int nota=0;
5     printf("Ingrese la nota del examen\n");
6     scanf("%d",&nota);
7
8     if(nota<4)
9     {
10       printf("El alumno no aprobo el examen\n");
11    }
12    else
13    {
14       printf("El alumno aprobo el examen\n");
15    }
16     return (0);
17 }
```

Estructura selectiva doble: ejemplos

1. Diseñar y codificar un programa que tome un número entero por teclado e indique si el número es positivo, negativo o cero.
   ▸ Ver en github
2. Diseñar y codificar un programa que tome dos números enteros por teclado e indique cual es el mayor. Si los números son iguales, se debe informar esta condición.
   ▸ Ver en github
3. Diseñar y codificar un programa que tome las coordenadas de un punto en el plano cartesiano (x;y) e imprima por pantalla a que cuadrante pertenece el punto.
   ▸ Ver en github
4. Diseñar y codificar un programa que tome por teclado el valor de cada uno de los lados de un triángulo y determine si es equilátero, isosceles o escaleno. También debe imprimir el perímetro del mismo.
   ▸ Ver en github

Ciclo repetitivo for

Definición

Es una estructura repetitiva controlada por contador. En general, se la utiliza cuando se conoce previamente la cantida de veces que las acciones seran ejecutadas.

Esta compuesto por:

- Valor inicial de la variable de control
- Condicion de continuacion de ciclo
- Incremento de la variable de control

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

i=0 definicion de la variable control anidacion

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

i=0 definicion de la variable control anidacion
Para i=0 hasta N

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

i=0 definicion de la variable control anidacion
Para i=0 hasta N
acción 1;

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

i=0 definicion de la variable control anidacion
Para i=0 hasta N
acción 1;
acción 2;

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

i=0 definicion de la variable control anidacion
Para i=0 hasta N
acción 1;
acción 2;
acción 3;

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

i=0 definicion de la variable control anidacion
Para i=0 hasta N
acción 1;
acción 2;
acción 3;
acción 4;

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

i=0 definicion de la variable control anidacion
Para i=0 hasta N
acción 1;
acción 2;
acción 3;
acción 4;

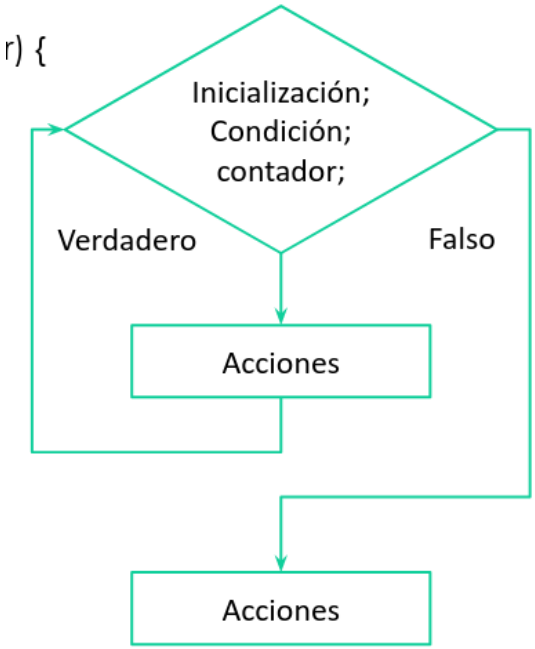
Fin del algoritmo

Ciclo repetitivo for: pseudocodigo y diagrama de flujo

Inicio del algoritmo

i=0 definicion de la variable control anidacion
Para i=0 hasta N
acción 1;
acción 2;
acción 3;
acción 4;

Fin del algoritmo



Ciclo repetitivo for: Codificacion en C

```
1  #include <stdio.h>
2  int main()
3  {
4     int ii=0;
5
6     for (ii=0; ii < 10; ii++)
7     {
8        print("%d\n",ii);
9
10    }
11
12   return (0);
13 }
```

Ciclo repetitivo for: Tabla de verificación I

ii	ii $<$ 5	printf(”%d”,i)	ii++
0			

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación II

ii	ii <5	printf("%d",i)	ii++
0	✓		

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación III

ii	ii <5	printf("%d",i)	ii++
0	✓	0	

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación IV

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1

```
1  #include <stdio.h>
2  int main()
3  {
4    int ii=0;
5
6    for (ii=0; ii < 10; ii++)
7    {
8      print("%d\n",ii);
9
10   }
11 }
12
13
14
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación V

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1			

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación VI

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación VII

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación VIII

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación IX

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2			

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación X

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓		

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XI

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XII

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XIII

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3			

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XIV

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3	✓		

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XV

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3	✓	3	

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XVI

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3	✓	3	4

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XVII

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3	✓	3	4
4			

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XVIII

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3	✓	3	4
4	✓		

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XIX

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3	✓	3	4
4	✓	4	

```
1  #include <stdio.h>
2  int main()
3  {
4    int ii=0;
5
6    for (ii=0; ii < 10; ii++)
7    {
8      print("%d\n",ii);
9
10   }
11 }
12
13
14
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XX

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3	✓	3	4
4	✓	4	5

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: Tabla de verificación XXI

ii	ii <5	printf("%d",i)	ii++
0	✓	0	1
1	✓	1	2
2	✓	2	3
3	✓	3	4
4	✓	4	5
5	X		

```c
#include <stdio.h>
int main()
{
  int ii=0;

  for (ii=0; ii < 10; ii++)
  {
    print("%d\n",ii);

  }
}
```

Figure: Estructura repetitiva for: ejemplo

Ciclo repetitivo for: anidacion

Inicio del algoritmo

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N
    Para j=0 hasta M

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N
    Para j=0 hasta M
        accion dentro del ciclo II

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N
    Para j=0 hasta M
        accion dentro del ciclo II
        accion dentro del ciclo II

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N
    Para j=0 hasta M
        accion dentro del ciclo II
        accion dentro del ciclo II
        accion dentro del ciclo II

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N
    Para j=0 hasta M
        accion dentro del ciclo II
        accion dentro del ciclo II
        accion dentro del ciclo II
    accion dentro del ciclo I

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N
    Para j=0 hasta M
        accion dentro del ciclo II
        accion dentro del ciclo II
        accion dentro del ciclo II
    accion dentro del ciclo I
    accion dentro del ciclo I

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N
    Para j=0 hasta M
        accion dentro del ciclo II
        accion dentro del ciclo II
        accion dentro del ciclo II
    accion dentro del ciclo I
    accion dentro del ciclo I
    accion dentro del ciclo I

Ciclo repetitivo for: anidacion

Inicio del algoritmo

i=0 definicion de la variable control ciclo repetivivo I
j=0 definicion de la variable control ciclo repetivivo II
Para i=0 hasta N
    Para j=0 hasta M
        accion dentro del ciclo II
        accion dentro del ciclo II
        accion dentro del ciclo II
    accion dentro del ciclo I
    accion dentro del ciclo I
    accion dentro del ciclo I

Fin del algoritmo

Ciclo repetitivo for: anidacion

```
1  #include <stdio.h>
2  int main()
3  {
4     int var_control          =0;
5     int positivos_cantidad=0;
6     int negativos_cantidad=0;
7     int numero               =0;
8     for (var_control=0; var_control < 10; var_control++)
9     {
10          printf("Ingrese el numero %d\n",var_control);
11          scanf("%d",&numero);
12          if(numero>=0)
13          {
14             positivos_cantidad=positivos_cantidad+1;
15          }
16          else
17          {
18             negativos_cantidad=negativos_cantidad+1;
19          }
```

Ciclo repetitivo for: ejemplos

1. Diseñar y codificar un programa en C que imprima la suma de todos los números comprendidos entre 0 y N. Donde N debe ser ingresado por teclado. ▸ Ver en github
2. Diseñar y codificar un programa en C que reciba 10 numeros ingresados por teclado e imprima la cantidad de positivos y negativos.El cero es considerado positivo. ▸ Ver en github
3. Diseñar y codificar un programa en C que reciba la cantidad de filas y columnas de una matriz y luego imprima dicha matriz donde cada elemento vale la suma de su posicion en filas y columnas ▸ Ver en github

Estructura repetitiva while

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

Estructura while: pseudocódigo

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

Inicio del algoritmo

Estructura while: pseudocódigo

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

Inicio del algoritmo

Mientras (condición de ciclo==Verdadera)

Estructura while: pseudocódigo

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

Inicio del algoritmo

Mientras (condición de ciclo==Verdadera)

acción 1;

Estructura while: pseudocódigo

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

Inicio del algoritmo

Mientras (condición de ciclo==Verdadera)

acción 1;
acción 2;

Estructura while: pseudocódigo

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

Inicio del algoritmo

Mientras (condición de ciclo==Verdadera)

acción 1;
acción 2;
acción 3;

Estructura while: pseudocódigo

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

Inicio del algoritmo

Mientras (condición de ciclo==Verdadera)

acción 1;
acción 2;
acción 3;
...

Estructura while: pseudocódigo

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

```
Inicio del algoritmo
    Mientras (condición de ciclo==Verdadera)
        acción 1;
        acción 2;
        acción 3;
        ...
        acción n;
Fin del algoritmo
```

Estructura while: diagrama de flujo

Es una estructura repetitiva controlada por centinela. Es decir, que las acciones dentro del bloque de repetición serán ejecutadas **si y sólo si** la expresión lógica evaluada en la condición de ciclo es verdadera.

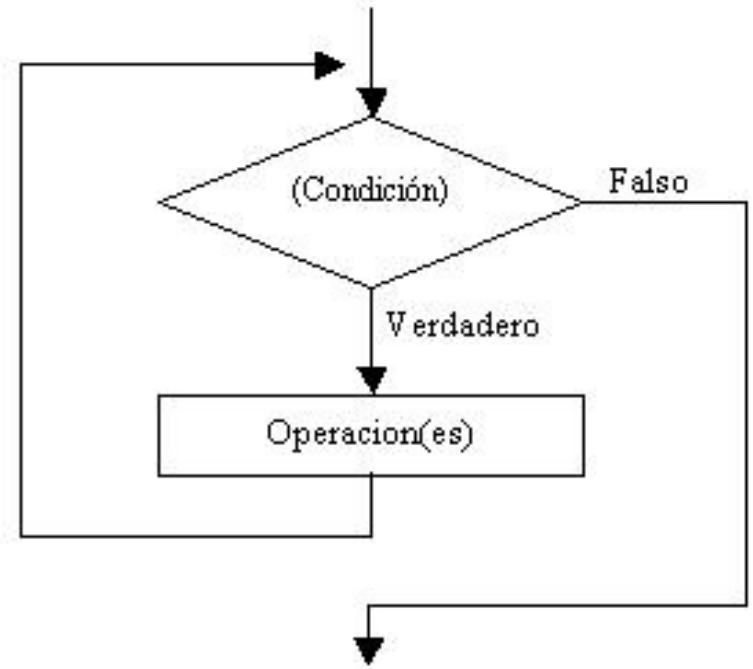


Figure: Diagrama de flujo estructura repetitiva while.

```c
#include <stdio.h>
int main()
{
    int acumulador=0;
    int numero=0;
    while(numero!=-1)
    {
        printf("Ingrese un numero entero\n");
        printf("Para salir ingrese -1\n");
        scanf("%d",&numero);
    }
    printf("La suma de los numeros es: %d",acumulador);
  return (0);
}
```

Estructura repetitiva do-while

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

Inicio del algoritmo

Hacer

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

Inicio del algoritmo

Hacer

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

Inicio del algoritmo

    Hacer

        acción 1;

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

```
Inicio del algoritmo
    Hacer
        acción 1;
        acción 2;
```

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

```
Inicio del algoritmo
    Hacer
        acción 1;
        acción 2;
        acción 3;
```

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

```
Inicio del algoritmo
    Hacer
        acción 1;
        acción 2;
        acción 3;
        ...
```

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

```
Inicio del algoritmo
    Hacer
        acción 1;
        acción 2;
        acción 3;
        ...
        acción n;
```

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

Inicio del algoritmo

Hacer

acción 1;
acción 2;
acción 3;
...
acción n;

Mientras (condición de ciclo==Verdadera)

Estructura repetitiva do-while: pseudocódigo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.

Inicio del algoritmo

Hacer

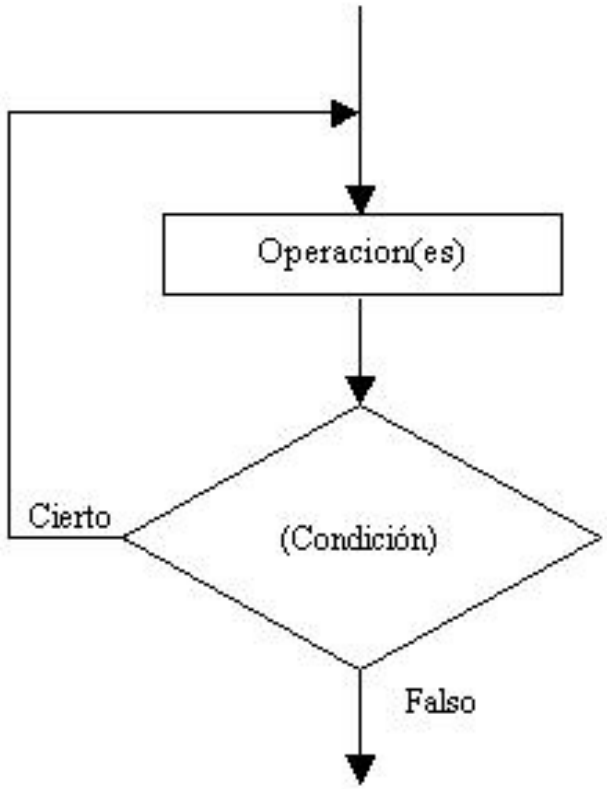
acción 1;
acción 2;
acción 3;
...
acción n;

Mientras (condición de ciclo==Verdadera)

Fin del algoritmo

Estructura repetitiva do-while: diragrama de flujo

La instrucción do...while es controlado por centinela pero evalúa la condición de continuación de ciclo **después de ejecutar el cuerpo del ciclo**; por lo tanto, el cuerpo del ciclo siempre se ejecutará al menos una vez.



```c
#include <stdio.h>
int main()
{
    int acumulador=0;
    int numero=0;
    do
    {
        printf("Ingrese un numero entero\n");
        printf("Para salir ingrese -1\n");
        scanf("%d",&numero);
        acumulador+=numero;
    }while(numero!=-1);
    printf("La suma de los numeros es: %d",acumulador);
    return (0);
}
```

Sentencias break y continue

Sentencia break

Cuando una sentencia break se encuentra dentro de un bucle, este termina su ejecución y el programa continua con las líneas de código que sucedan al ciclo repetitivo.

Sentencia continue

Cuando una sentencia continue se encuentra dentro de un ciclo repetitivo, el programa regresa al comienzo del mismo, evitando todas las líneas que se encuentren después de haber encontrado la sentencia mencionada.

Nota: tratar de evitar el uso masivo de estas sentencias, puesto que no son consideradas buenas prácticas de programación.

Sentencias break y continue: ejemplos

Funciones