Universidad Naciona Autónoma de México

Facultad de Ingeniería

Estructuras de Datos y Algoritmos I

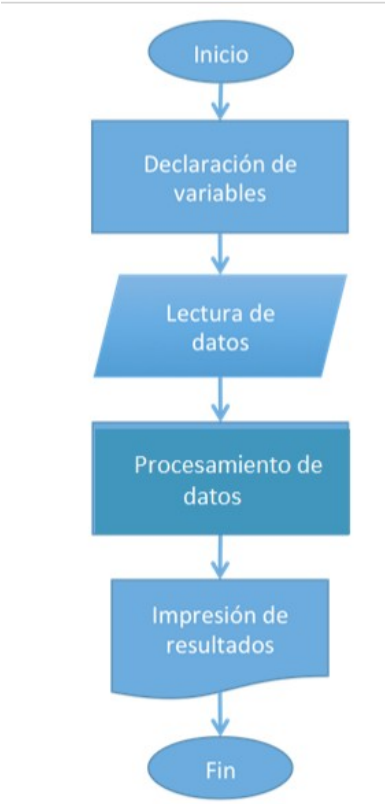
Actividad 1: Repaso

Sánchez Hernández Marco Antonio

Fecha: 09/junio/2021

Diagramas de flujo

Los diagramas de flujo, como su nombre indica, permiten representar gráficamente el flujo de la ejecución de un programa mediante el uso de ciertas figuras que representan un determinado proceso.



Diaragrama de flujo while, por Solano J., García E., Nakayama A. y Arteaga I., (2018), Salas A y B FI, (http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf)

Tipos de datos primitivos en C

Los datos primitivos son datos que están predefinidos dentro de un lenguaje de programación, en C existen principalmente 3 tipos de datos:

1. Números enteros **(int)**.
2. Números reales **(float).**
3. Caracteres **(char).**

Sin embargo, existen distintos tipos de estos, que principalmente difieren en el uso de memoria o el valor que pueden tomar:

Tipo de dato	Espacio en memoria	Rango
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615
float	4 bytes	3.4 E -38 to 3.4 E 38
double	8 bytes	1.7 E -308 to 1.7 E 308
long double	16 bytes	3.4 E -4932 to 3.4 E 4932

Definición de variables en C

Definir una variable sin inicializarla:

Tipo_de_dato Nombre;

Definir una variable inicializándola:

Tipo_de_dato Nombre = valor;

Definición de constantes en C

Una constante en lenguaje C puede ser definida de dos maneras diferentes, la primera es mediante el uso del preproceso ***#define***, y la segunda mediante el uso de la palabra reservada ***const***:

Definición de una constante mediante ***#define***:

#define *identificador valor;*

Definición de una constante mediante ***const***:
const *Tipo_de_dato Nombre = valor;*

La principal diferencia radica en que, el uso del preproceso ***#define*** creará la constante antes de iniciar la compilación del programa, mientras que el uso de la palabra reservada ***const***, definirá la constante durante el proceso de compilación.

Operadores aritméticos en C

Operador	Descripción
+	Realiza la suma de dos números (a+b).
-	Realiza sustración de dos números (a-b).
*	Realiza el producto de dos números (a*b).
/	Realiza el cociente de dos números (a/b, b != 0).
%	Indica el reciduo del cociente de dos números.
++	Incrementa el valor de un número a en uno.
--	Decrece el valor de un número a en uno.

Operadores de relación en C

Operador	Descripción
==	Evalua si dos valores son iguales o no. Si ambos valores son iguales, entonces la condición será verdadera.
!=	Evalua si dos valores son iguales o no. Si los valores no son iguales, entonces la condición será verdadera.
>	Evalua si el valor de la izquierda es mayor que el valor de la derecha. Si es mayor, entonces la condición será verdadera.
<	Evalua si el valor de la izquierda es menor que el valor de la derecha. Si es menor, entonces la condición será verdadera.
>=	Evalua si el valor de la izquierda es mayor o igual que el valor de la derecha. Si es mayor o igual, entonces la condición será verdadera.
<=	Evalua si el valor de la izquierda es menor o igual que el valor de la derecha. Si es menor o igual, entonces la condición será verdadera.

Operadores lógicos en C

Operador	Descripción
&&	AND: este operador regresa como valor verdadero si ambas condiciones se cumplen.
||	OR: este operador regresa como valor verdadero si al menos una condisión se cumple.
!	NOT: este operador regresa como valor verdadero si la condición es falsa, y regresa como valor falso si la condición es verdadera.

Comentarios en C

// Comentarios de una sola línea

/ Comentarios de múltiples líneas*/*

Entrada y Salida en C

La entrada y salida de datos en C, en alto nivel, se logra a paritr de incluir en el encabezado de un cierto código el prepoceso ***#include <stdio.h>***, el cual proporciona las funciones necesarias para leer e imprimir datos en la entrada estándar (por defecto, la entrada estándar es la terminal).

printf*(“datos a imprimir”);* //Impresión en la salida estándar de datos.

scanf*(%especificador_formato, variable);* //Lectura de datos de la entrada estándar.

Tipo de dato	Especificador de formato
char	%c
unsigned char	%c
signed char	%c
int	%d
unsigned int	%u
short	%hd

unsigned short	%hu
long	%ld
unsigned long	%lu
float	%f
double	%lf
long double	%Lf

Estructuras de selección en C

if-else

if validará si una determinada condición es verdadera o falsa, en caso de ser verdadera, se ejecutará un cierto bloque de código, en caso de ser falsa, el programa continúa con su ejecución omitiendo el bloque de código dentro de ***if***, u opcionalmente, puede ser añadido ***else***, el cual ejecutara otro bloque de código. La sintaxis para su implementación en C es la siguiente:

```
if (condición) {
    /*Bloque de código si la condición es verdadera*/
}

else {
    /*Bloque de código, si la condición es falsa*/
}
```

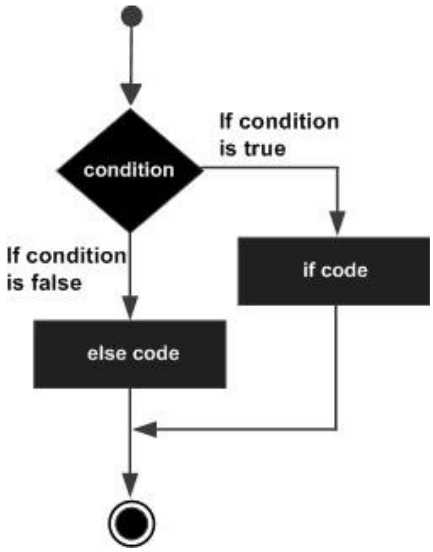


Diagrama de flujo de if-else, por tutorialspoint, s. f.
(https://www.tutorialspoint.com/cprogramming/if_else_statement_in_c.htm)

switch()

Cuando requerimos a partir de una condición dada ejecutar un determinado bloque de código, es mejor utilizar la estructura de selección ***switch()***. Esta estructura de selección permite definir *n*-posibles bloques de código a ejecutar dependiento de una condición almacenada en una variable. Cada posible bloque de código a ejecutar es contenido en un ***case***. La sintaxis para su implementación es la siguiente:

```
switch (condición) {
        case 1:
                /*Bloque de código*/
        break;

        …

        case n:
                /*Bloque de código*/
        break;
```

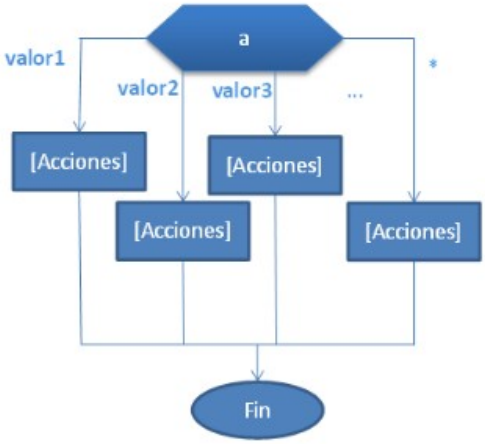
default:

*/*Bloque de código*/*

break;

}

El bloque dentro de ***default*** será ejecutado si la condición no satisface ninguna de las definidas para la ejecución de cada ***case***.



Diaragrama de flujo de switch, por Solano J., García E., Nakayama A. y Arteaga I., (2018), Salas A y B FI, (http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf)

Operador ternario

Para tomar decisiones a partir de condiciones simples, es conveniente el uso de este operador en lugar de ***if-else***, ya que será realizado con mayor eficiencia, ahorrando así líneas de código. La sintaxis para su implementación en C es la siguiente:

Condición ? SiEsVerdadera : SiEsFalsa

Ciclos en C

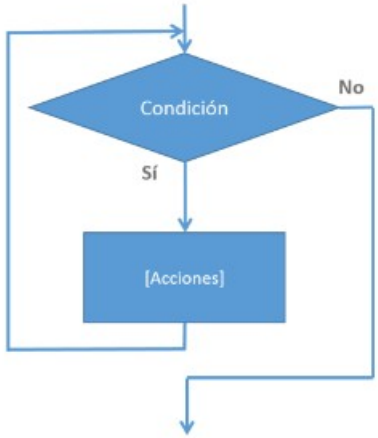
while

Un ciclo ***whlie*** repetirá un bloque de código mientras una cierta condición, la cual es verificada antes de ejecutar el bloque de código, sea verdadera. La sintaxis para su implementación es la siguiente:

while *(condición) {*

*/*Bloque de código*/*

}



Diaragrama de flujo while, por Solano J., García E., Nakayama A. y Arteaga I., (2018), Salas A y B FI, (http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf)

for

Al igual que el ciclo ***while***, el ciclo ***for*** repetirá un bloque de código *n*-veces hasta que una cierta condición sea falsa, para su declaración se requiere una variable de incremento (contador), por lo regular denotada por *i* o *j*, una condición para esta variable de incremento, y una instrucción para saber en cuanto debe incrementar la variable. La sintaxis para su implementación es la siguiente:

for *(variable_incremento, condición, incremento) {*

*/*Bloque de código*/*

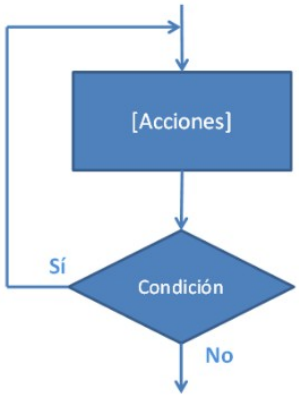
}

El ciclo ***whlie*** es mejor utilizarlo cuando desconocemos el número de veces que un ciclo puede repetirse, mientras que el ciclo ***for*** es mejor utilizarlo cuando conocemos el número de veces que un ciclo puede repetirse.

do-while

El ciclo ***do-while*** actúa exactamente igual que el ciclo ***while***, sin embargo, la condición es evaluada al final de la ejecución del bloque de código. La sintaxis para su implementación es la siguiente:

```
do {
        /*Bloque de código*/
}while (condición);
```



Diaragrama de flujo de do-while, por Solano J., García E., Nakayama A. y Arteaga I., (2018), Salas A y B FI, (http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf)

break, continue

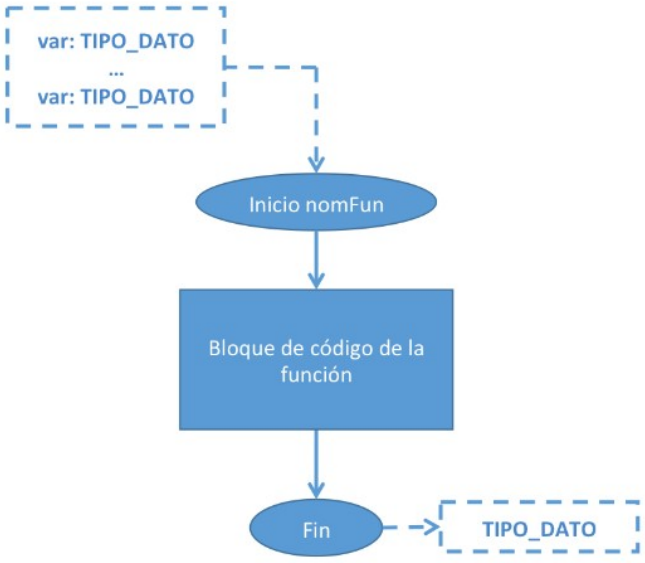
Estas palabras reservadas son utilizadas para controlar el flujo de ejecución dentro de ciclos. La palabra reservada ***break*** termina anticipadamente un ciclo, también es utilizada en ***switch()***. La palabra reservada ***continue*** continúa con la ejecución de un ciclo, si ***continue*** es ejectuado antes de llegar a la última línea de un ciclo, este omitirá las líneas de código restantes y comenzará su ejecución desde la primera línea.

Funciones en C

Todo código escrito en lenguaje C está constituido por al menos una funció (***main***). Una función es simplemente un bloque de código agrupado que permite ejecutar una tarea. Existen funciones predefinidas, como ***printf***, y funciones definidas por el propio usuario. La función ***main*** es la función donde se iniciará la ejecución de cualquier programa escrito en C. Las funciones pueden recibir parámetros (datos) con los cuales realizará las operaciones especificadas, una vez se terminan las operaciones, la función puede regresar o no algún resultado derivado de la ejecución de dichas operaciones. Para definir en C una función se utiliza la siguiente sintaxis:

```
Tipo_de_dato_de_retorno Nombre_de_la_función (n-parámetros) {
    /*Bloque de código a ejecutar*/
    return valor_de_retorno;
}
```

Si se desea que una función no regrese algún valor tras su ejecución, se debe usar el tipo de dato ***void*** el cual le indica a la función que no se espera que esta regrese algún valor, por ende, ***return*** puede ser omitida del código.



Diaragrama de flujo de una función, por Solano J., García E., Nakayama A. y Arteaga I., (2018), Salas A y B FI, (http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf)

Arreglos

Los arreglos son la estructura de dato más básica que existe y se definen como una colección de datos, del mismo tipo, los cuales son almacenados en espacios de memoria contiguos. Un arreglo puede ser unidimensional, es decir, los elementos del arreglo son almacenados en una fila (línea); o multidimensional, es decir, los elementos del arreglo son almacenados en representaciones de objetos de acuerdo con el número de dimensiones que este tenga, por ejemplo, un arreglo bidimensional será almacenado como una matriz (filas y columnas). Sin importar si se utiliza un arreglo de una sola dimensión o de más dimensiones, el almacenamiento en memoria es el mismo, contiguo.

Declaración de arreglos unidimencionales en C:

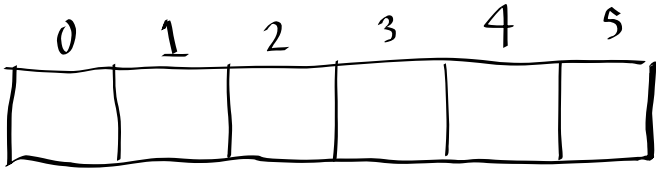
Tipo_de_dato Nombre_del_arreglo [tamaño del arreglo] = {elementos del arreglo};

Declaración de arreglos multidimensionales en C:

Tipo_de_dato Nombre_del_arreglo [tamaño dimensión 1] … [tamaño dimensión n] = {elementos dimensión 1}, …, {elementos dimensión n};

Existe una manera equivalente de para declarar los elementos de un arreglo multidimensional, sin embargo, esa forma puede generar mayor confusión al momento de querer acceder a un determinado elemento, por lo que, no será mencionada.

Para acceder a un determinado elementos de un arreglo se utiliza uno o varios índices, dependiendo del tipo de arreglo (unidimensiona o multidimensional). En lenguaje C, como en la mayoría de lenguajes de Programación, el índice del primer elemento de una dimensión de un arreglo es el número *0*, mientras que el último elemento es *n-1*, donde *n* es el tamaño de la dimensión del arreglo.



Apuntadores en C

En lenguaje C, los apuntadores permiten acceder a la memoria y dirección que ocupa una determinada variable. Su sintaxis para su implementación es:

*Tipo_de_dato **Nombre;*

Referencias

GeekforGeeks. (2020). *Data Types in C. Data Types in C – GeelforGeeks*. https://www.geeksforgeeks.org/data-types-in-c/

Kumar A. (2020). *A Practical Approach to Data Structures and Algorithm with Programming in C*. Arcler Press. http://bibliotex.ipublishcentral.com.pbidi.unam.mx:8080/pdfreader/practical-approach-to-data-structure-algorithm-programming-in-c

Solano J., García E., Nakayama A. y Arteaga I., (2018). *Manual de prácticas del laboratorio de Fundamentos de Programación*. Laboratorio de computación Salas A y B. http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf

Tutorialspoint. (s. f.). *C – Constants and Literals. Learn C Programming.* https://www.tutorialspoint.com/cprogramming/c_constants.htm

Tutorialspoint. (s. f.). *C - Data Types*. *Learn C Programming*. https://www.tutorialspoint.com/cprogramming/c_data_types.htm

Tutorialspoint. (s. f.). *C – Functions*. *Learn C Programming*. https://www.tutorialspoint.com/cprogramming/c_functions.htm

Tutorialspoint. (s. f.). *C – if … else statement*. *Learn C Programming*. https://www.tutorialspoint.com/cprogramming/if_else_statement_in_c.htm

Tutorialspoint. (s. f.). *C – Operators. Learn C Programming.* https://www.tutorialspoint.com/cprogramming/c_operators.htm

Tutorialspoint. (s. f.). *C – Preprocessors*. *Learn C Programming.* https://www.tutorialspoint.com/cprogramming/c_preprocessors.htm