Guía práctica de estudio 02: Aplicaciones de apuntadores

Elaborado por:
M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Autorizado por:
M.C. Alejandro Velázquez Mena

Guía práctica de estudio 02: Aplicaciones de apuntadores

Objetivo:

Utilizar apuntadores en lenguaje C para acceder a las localidades de memoria tanto de datos primitivos como de arreglos.

Actividades:

- Crear apuntadores.
- Leer y modificar datos a través de apuntadores.

Introducción

Un apuntador es una variable que contiene la dirección de memoria de otra variable. Los apuntadores se utilizan para dar claridad y simplicidad a las operaciones a nivel de memoria.

Lenguaje C es un lenguaje de alto nivel porque permite programar a bajo nivel. La programación a bajo nivel se refiere a la manipulación de los recursos físicos de un equipo computacional. Los apuntadores permiten manipular de manera directa las localidades de memoria RAM de la computadora.

Licencia GPL de GNU

El software presente en esta guía práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * Author: Jorge A. Solano
 *
*/
```

Apuntadores

Un apuntador es una variable que contiene la dirección de memoria de otra variable, es decir, hace referencia a la localidad de memoria de otra variable. Debido a que los apuntadores trabajan directamente con la memoria, a través de ellos se accede con rapidez a la información almacenada.

Para declarar un apuntador se debe definir el tipo de dato y el nombre de la variable apuntador precedida de un asterisco (*). Una variable de tipo apuntador debe tener el mismo tipo de dato de la variable a la que va a apuntar:

```
TipoDeDato *apuntador, variable;
```

Para asignarle un valor al apuntador, se debe acceder a la localidad de memoria de la variable a través de un ampersand (&):

```
apuntador = &variable;
```

Los apuntadores solo deben apuntar a variables del mismo tipo de dato con el que fueron declarados.

Código (apuntador a carácter)

```c
#include <stdio.h>

/*
    Este programa crea un apuntador de tipo carácter.
*/

int main () {
    char *ap, c;
    c = 'x';
    ap = &c;

    // imprime el carácter de la localidad a la que apunta
    printf("Carácter: %c\n",*ap);
    // imprime el código ASCII de la localidad a la que apunta
    printf("Código ASCII: %d\n",*ap);
    // imprime la dirección de memoria de la localidad a la que apunta
    printf("Dirección de memoria: %d\n",ap);

    return 0;
}
```

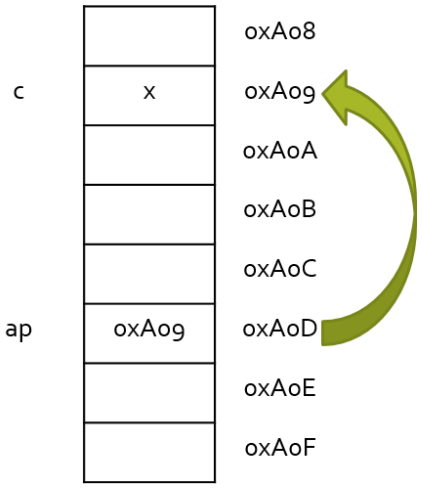


Figura 1. Apuntador a carácter.

Código (apuntador a entero)

```c
#include <stdio.h>

/*
    Este programa crea un apuntador de tipo entero
    y modifica la información a través del mismo.
*/

int main () {
    short a = 5, b = 10;
    short *apEnt;
    apEnt = &a;
    // imprime el valor entero de a
    printf("a = %i\n", a);
    b = *apEnt;     // b = 5
    // imprime el valor de lo que apunta apEnt
    printf("b = %i //*apEnt\n", b);
    b = *apEnt+1;   // b = 6
    // imprime el valor de lo que apunta apEnt + 1
    printf("b = %i //*apEnt+1\n", b);
    *apEnt = 0;     // a = 0
    // le asigna el valor de 0 a la variable al que apunta apEnt
    printf("a = %i //*apEnt = 0\n", a);

    return 0;
}
```

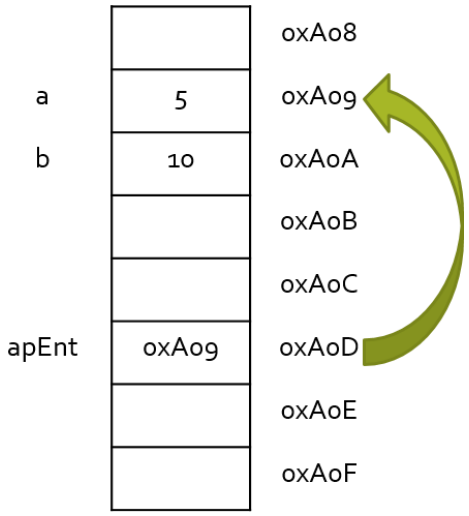


Figura 2. Apuntador a entero.

Código (apuntador a arreglo)

```c
#include <stdio.h>

/*
    Este programa crea un apuntador de tipo entero
    que apunta al inicio de un arreglo.
*/

int main () {
    short arr[5], *apArr;
    apArr = &arr[0];
    // imprime la dirección de memoria del arreglo en la posición [0]
    printf("Dirección del arreglo en la primera posición: %x\n",&arr[0]);
    // imprime la dirección de memoria del arreglo
    // (el nombre del arreglo es un apuntador)
    printf("Dirección del arreglo: %x\n",&arr);
    // imprime la dirección de memoria del apuntador apArr
    printf("Dirección del apuntador: %x\n",apArr);

    return 0;
}
```

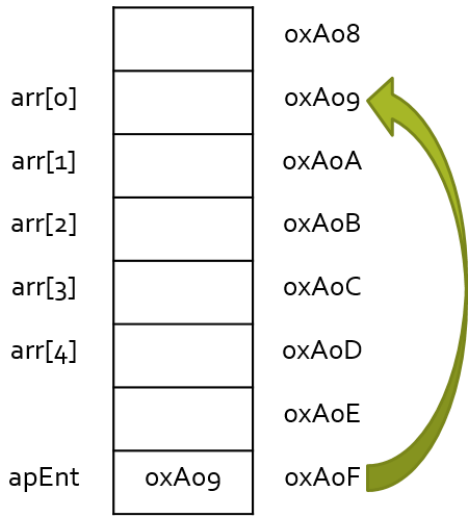


Figura 3. Apuntador a arreglo.

Como se puede observar en el código anterior, el nombre del arreglo es, en sí, un apuntador a la primera localidad del mismo.

Parámetros de las funciones

Dentro de los parámetros o argumentos que define una función, se pueden trabajar con los valores o con las referencias de las variables. Cuando se trabaja con los valores (lo que se conoce como paso de variables por valor) en realidad se envía una copia del valor original a la función de tal manera que, si ésta modifica el contenido de la variable, el valor original no se verá afectado. Por otro lado, cuando se trabaja con las referencias (lo que se conoce como paso de variables por referencia) en realidad se envía un apuntador hacia el valor original y, por ende, en realidad se está trabajando con dicho valor todo el tiempo.

Código (Paso de variables por valor y por referencia)

```c
#include<stdio.h>

/*
    Se ejemplifica el paso de variables por valor y por referencia.
*/

void pasarValor(int);
void pasarReferencia(int *);

int main(){
    int nums[] = {55,44,33,22,11};
    int *ap, cont;
    ap = nums;
    printf("Pasar valor: %d\n", *ap);
    pasarValor(*ap);
    printf("Pasar referencia: %d\n", *ap);
    pasarReferencia(ap);
    printf("Valor final: %d\n", *ap);
    return 0;
}

void pasarValor(int equis){
    printf("%d\n", equis);
    equis = 128;
    printf("%d\n", equis);
}

void pasarReferencia(int *equis){
    printf("%d\n", *equis);
    *equis = 128;
    printf("%d\n", *equis);
}
```

Debido a que el nombre de un arreglo es, en realidad, un apuntador a la primera localidad del arreglo, cuando se envía un arreglo como argumento de una función, dentro de la función se está trabajando directamente con el arreglo original.

Aplicaciones de apuntadores

Trabajar a nivel de memoria genera muchas ventajas, entre ellas la rapidez y la sencillez para manipular los datos. Dentro de las aplicaciones más útiles de los apuntadores se encuentran las que tienen que ver con arreglos.

Como ya se mencionó, el acceso a un arreglo se puede hacer mediante un índice a cada localidad del mismo. Sin embargo, otra forma de recorrer un arreglo es mediante un apuntador, haciendo más eficiente el acceso a los datos por la rapidez que proporciona éste, esto debido al concepto aritmética de apuntadores.

Dentro de la aritmética de los apuntadores, suponiendo que apArr es un apuntador a algún elemento de un arreglo, entonces:

- apArr++: Incrementa apArr para apuntar a la siguiente localidad de memoria.
- apArr+=i: Incrementa apArr para apuntar a la i-ésima localidad de memoria a partir del valor inicial de apArr.

Código (aritmética de direcciones)

```c
#include<stdio.h>

/*
    Se imprimen 3 valores de un arreglo a través
    de aritmética de direcciones.
*/
int main () {
    short arr[5] = {91,28,73,46,55};
    short *apArr;
    apArr = arr;
    // apunta al inicio del arreglo
    printf("*apArr = %i\n",*apArr);
    // suma una localidad al inicio del arreglo e imprime su valor
    printf("*(apArr+1) = %i\n",*(apArr+1));
    // suma dos localidades al inicio del arreglo e imprime su valor
    printf("*(apArr+2) = %i\n",*(apArr+2));

    return 0;
}
```

Código (arreglo unidimensional)

```c
#include<stdio.h>

/*
    Se recorre un arreglo unidimensional a través de un apuntador
*/

int main(){
    short nums[] = {55,44,33,22,11};
    short *ap, cont;
    ap = nums;

    for (cont = 0; cont < 5 ; cont++)
        printf("%x\n",(ap+cont));

    return 0;
}
```

Código (arreglo bidimensional)

```c
#include<stdio.h>

/*
    Se recorre un arreglo bidimensional a través de un apuntador
*/

int main(){
    int *ap, indice;
    int nums[3][3] = {{99,88,77},
        {66,55,44},
        {33,22,11}};
    ap = nums;
    for (indice = 0; indice < 9 ; indice++){
        if ((indice%3)==0)
            printf("\n");
        printf("%x\t",(ap+indice));
    }
    return 0;
}
```

El cifrado César

En el siglo I antes de Cristo, Julio César el célebre militar y político, usó éste cifrado para enviar órdenes a sus generales en los campos de batalla. El método consiste en desplazar el abecedario 3 posiciones, es decir, en lugar de iniciar en la letra A, el abecedario inicia en la letra D.

En claro	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cifrado	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

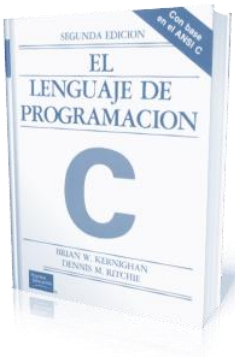
De tal manera, si se quiere enviar el mensaje en claro RETIRADA, el mensaje cifrado será UHWLUDGD.

Código (Cifrado César)

```
#include<stdio.h>

/*
    Programa que realiza la implementación del cifrado César
*/

#define TAM_PALABRA 20
#define TAM_ABC 26

char abecedarioEnClaro[TAM_ABC] =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T'
,'U','V','W','X','Y','Z'};
char abecedarioCifrado[TAM_ABC] =
{'D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W'
,'X','Y','Z','A','B','C'};

void cifrar(char *textoEnClaro);
void descifrar(char *textoCifrado);

int main(){
    short opcion = 0, contador;
    char palabra[TAM_PALABRA];

    while (1){
        printf("\n\t*** CIFRADO CÉSAR ***\n");
        for (contador=0 ; contador<26; contador++)
            printf("%c", *(abecedarioEnClaro+contador));
        printf("\n");
        for (contador=0 ; contador<26; contador++)
            printf("%c", *(abecedarioCifrado+contador));
        printf("\nElegir una opción:\n");
        printf("1) Cifrar\n");
        printf("2) Descifrar.\n");
        printf("3) Salir.\n");
        scanf("%d", &opcion);
```

```
        switch(opcion){
            case 1:
                printf("Ingresar la palabra a cifrar (en mayúsculas): ");
                scanf("%s", palabra);
                cifrar(palabra);
                break;
            case 2:
                printf("Ingresar la palabra a descifrar (en mayúsculas): ");
                scanf("%s", palabra);
                descifrar(palabra);
                break;
            case 3:
                return 0;
            default:
                printf("Opción no válida.");
        }
    }

    return 0;
}

void cifrar(char *textoEnClaro){
    printf("El texto %s cifrado es: ", textoEnClaro);
    int contadorAbcedario, contadorPalabra, indice = 0;
    for (contadorPalabra=0 ; contadorPalabra<textoEnClaro[contadorPalabra] ;
contadorPalabra++)
        for (contadorAbcedario=0 ; contadorAbcedario<TAM_ABC ;
contadorAbcedario++)
            if (abecedarioEnClaro[contadorAbcedario] ==
textoEnClaro[contadorPalabra]){
                printf("%c", abecedarioCifrado[contadorAbcedario]);
                contadorAbcedario = 26;
            }

    printf("\n");
}

void descifrar(char *textoCifrado){
    printf("El texto %s descifrado es: ", textoCifrado);
    int contadorAbcedario, contadorPalabra, indice = 0;
    for (contadorPalabra=0 ; contadorPalabra<textoCifrado[contadorPalabra] ;
contadorPalabra++)
        for (contadorAbcedario=0 ; contadorAbcedario<TAM_ABC ;
contadorAbcedario++)
            if (abecedarioCifrado[contadorAbcedario] ==
textoCifrado[contadorPalabra]){
                printf("%c", abecedarioEnClaro[contadorAbcedario]);
                contadorAbcedario = 26;
            }

    printf("\n");
}
```

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.