Paso de argumentos por referencia

Hasta ahora hemos utilizado funciones que reciben argumentos y retornan resultados. El valor que nos han retornado las funciones se adquiere por medio de la instrucción *return* que se encuentra dentro de la función que es invocada.

En la llamada a una función, los valores que van dentro de los paréntesis se conocen como parámetros actuales o argumentos. En la cabecera de la función, los parámetros que se corresponden, uno a uno, con los de la llamada, se conocen como parámetros formales, o simplemente parámetros. Estos son variables locales de la función, al igual que las variables que se declararían dentro del cuerpo e la misma. Toda variable local de una función se destruye cuando la función finaliza y el control de ejecución regresa al punto de la llamada. La memoria RAM que esa variable utilizaba vuelve a estar disponible para el sistema operativo.

Los parámetros actuales **le entregan una copia** de su valor a los parámetros formales, y estos lo adquieren como valor inicial. En el transcurso de la ejecución de la función, los parámetros formales son utilizados como cualquier otra variable local, incluso su valor puede ser reasignado. Pero al finalizar la función, el último valor adquirido por los parámetros formales se pierde, dado que el parámetro es destruido, y al retornar al cuerpo de la función que hizo la llamada. De esta forma, los parámetros actuales conservan sus valores iniciales, aunque los parámetros formales hayan cambiado de valor. A esto se le conoce como paso de parámetros por valor.

Todos los ejemplos y ejercicios que hemos realizado hasta ahora con funciones que reciben valores por medio de sus parámetros, los han recibido por valor. Es decir, han recibido una copia del valor del parámetro actual correspondiente.

Por ejemplo:

```
#include "iostream"
void miFuncion(int);
using namespace std;

int main(void)
{
	int a = 5;

	cout << endl;
	cout << "POR VALOR" << endl << endl;

	cout << "El valor de a es: " << a << endl;

	miFuncion(a);

	cout << "El valor de a es: " << a << endl;

	cout << endl;
	return 0;
}
```

```
void miFuncion(int b)
{
	cout << "El valor de b es: " << b << endl;

	b = 10;

	cout << "El valor de b es: " << b << endl;
}
```

La corrida de este programa es la siguiente:

POR VALOR

El valor de a es: 5
El valor de b es: 5
El valor de b es: 10
El valor de a es: 5

Notar que el valor de la variable *a*, al regresar de la llamada a la función, sigue siendo el mismo que antes de la llamada, aunque su parámetro formal cambió de valor en la función.

En ocasiones es necesario que los valores reasignados a los parámetros formales se conozcan al retornar de la función invocada. Para ello C++ utiliza como parámetros formales lo que se conoce como punteros. Esto es, variables que, en lugar de almacenar valores específicos, almacenan las direcciones de memoria de las variables que contienen esos valores.

Para que un parámetro actual adquiera el valor de un parámetro formal, lo que se hace es enviar a la función la dirección del parámetro actual. Así que su parámetro formal correspondiente será un puntero. De esta forma, la localidad de memoria del parámetro actual es accedida desde la función y su valor cambiado cuando sea requerido. Al finalizar la función invocada, el parámetro actual contiene el último valor que les fue asignado a través del parámetro formal.

Recordemos que la declaración de los punteros en C++ se realiza de la siguiente manera:

tipo *nombre;

Note el símbolo de * a la izquierda del nombre. Esto indica que esa variable es un puntero.

Así que, si queremos asignarle a un puntero la dirección de memoria de una variable, escribimos algo como esto:

```
int *p;
int a = 5;

p = &a;
```

Notar que p se ha declarado como un puntero, así que lo que recibe es la dirección de la variable *a*. Recordar que el operador & es el "operador dirección de".

El operador * se conoce como "operador de indirección". Esto le indica a C/C++ que esta variable almacenará direcciones de memoria de variables de su mismo tipo.

Por ejemplo:

```
#include "iostream"
void miFuncion(int *);
using namespace std;

int main(void)
{
	int a = 5;
```

```
void miFuncion(int *b)
{
	cout << "El valor de b es: " << *b << endl;

	*b = 10;

	cout << "El valor de b es: " << *b << endl;
}
```

```
	cout << endl;
	cout << "POR REFERENCIA" << endl << endl;

	cout << "El valor de a es: " << a << endl;

	miFuncion(&a);

	cout << "El valor de a es: " << a << endl;

	cout << endl;
	return 0;
}
```

La corrida de este programa es la siguiente:

POR REFERENCIA

El valor de a es: 5
El valor de b es: 5
El valor de b es: 10
El valor de a es: 10

Notar que el valor de la variable a ha cambiado al retornar de la función porque fue enviada por referencia. En el cuerpo de *miFuncion*, siempre que se enuncia **b*, se refiere a la variable *a* de la función *main* que se ha enviado por referencia. Es decir, es su dirección la que se ha enviado.

Notar también el prototipo de la función, es la misma que el encabezado de la función, pero sin los nombres de las variables.

Ejemplos:

1) Elabore una función que intercambie los valores de dos variables.

```
#include <iostream>

using namespace std;

void intercambiar (int *, int *);

int main(void)
{
	int a, b;
	cout << endl;
cout << "INTERCAMBIAR LOS VALORES DE DOS VARIABLES"
<< endl << endl;

	cout << "Digite el valor de a: ";
	cin >> a;
	cout << "Digite el valor de b: ";
	cin >> b;

	cout << "El valor de a es: " << a << endl;
	cout << "El valor de b es: " << b << endl;

	intercambiar (&a, &b);
```

```
void intercambiar (int *x, int *y)
{
	int aux;

	aux = *x;
	*x = *y;
	*y = aux;
}
```

```
	cout << "El valor de a es: " << a << endl;
	cout << "El valor de b es: " << b << endl;

	cout << endl;
	return 0;
}
```

2) Dados dos números enteros determine, por medio de una función, el cociente y el residuo de la división entera entre ambos.

```
#include <iostream>

using namespace std;

void divisionEntera(int, int, int *, int *);

int main(void)
{
	int a, b, cociente, residuo;
	cout << endl;
	cout << "DIVISIÓN ENTERA DE DOS NÚMEROS"
		<< endl << endl;

	cout << "Digite el valor del dividendo: ";
	cin >> a;
	cout << "Digite el valor del divisor: ";
	cin >> b;

	divisionEntera(a, b, &cociente, &residuo);

	cout << "El cociente es: " << cociente << endl;
	cout << "El residuo es: " << residuo << endl;

	cout << endl;
	return 0;
}
```

```
void divisionEntera(int x, int y, int *c, int *r)
{
	*c = x / y;
	*r = x % y;
}
```

3) Escriba una función que, dado un entero, determine cuantas unidades, decenas, centenas y millares lo componen. Estos cuatro resultados deben ser recogidos a través de argumentos por referencia.

```
#include "iostream"
void descomponer(int, int *, int *, int *, int *);
using namespace std;

int main(void)
{
	int n, unid, dec, cent, mil;

	cout << endl;
	cout << "DESCOMPONER NÚMERO" <<
```

```
void descomponer(int n, int *unid, int *dec, int *cent, int *mil)
{
	*unid = n % 10;
	n = n / 10;
	*dec = n % 10;
	n = n / 10;
	*cent = n % 10;
	n = n / 10;
	*mil = n;
```

```
endl << endl;

	cout << "Digite un entero: ";
	cin >> n;

	descomponer(n, &unid, &dec, &cent, &mil);

	cout << n << " tiene:" << endl;
	cout << unid << " unidades" << endl;
	cout << dec << " decenas" << endl;
	cout << cent << " centenas" << endl;
	cout << mil << " millares" << endl;

	cout << endl;
	return 0;
}
```

```
}
```

La corrida de este programa es la siguiente:

DESCOMPONER NÚMERO

Digite un entero: 45861
45861 tiene:
1 unidades
6 decenas
8 centenas
45 millares

Observar que la función tiene cinco parámetros: el primero se recibe por valor y los otros cuatro se reciben por referencia. El primer argumento recibe el valor de *n*, el cuál, a pesar de ir cambiando en la función, conserva su valor original en *main*. Los otros cuatro argumentos permiten que las variables respectivas de *main* tomen los valores que se han calculado.

4) Construya una función que solicite carácteres desde teclado, hasta que el usuario ya no quiera seguir proporcionándolos. Su función deberá recoger la cantidad de vocales por medio de un argumento.

```
#include "iostream"
void contarVocales(int *);
using namespace std;

int main(void)
{
	int n;

	cout << endl;
	cout << "CONTAR VOCALES" << endl << endl;

	contarVocales(&n);

	cout << endl << endl;
	cout << "Cantidad de vocales digitadas: " << n << endl;

	cout << endl;
	return 0;
}
```

```
void contarVocales(int *cont)
{
	unsigned char car;

	*cont = 0;
	cout << "Digite un carácter, o Ctrl-z para finalizar: ";
	while(cin >> car){
		if (car == 'a' || car == 'e' || car == 'i' || car == 'o' || car == 'u')
			*cont = *cont + 1;
		cout << "Digite un carácter, o Ctrl-z para finalizar: ";
	}
}
```

La corrida de este programa es la siguiente:

```
CONTAR VOCALES

Digite un carácter, o Ctrl-z para finalizar: k
Digite un carácter, o Ctrl-z para finalizar: i
Digite un carácter, o Ctrl-z para finalizar: g
Digite un carácter, o Ctrl-z para finalizar: a
Digite un carácter, o Ctrl-z para finalizar: d
Digite un carácter, o Ctrl-z para finalizar: e
Digite un carácter, o Ctrl-z para finalizar: r
Digite un carácter, o Ctrl-z para finalizar: b
Digite un carácter, o Ctrl-z para finalizar: o
Digite un carácter, o Ctrl-z para finalizar: a
Digite un carácter, o Ctrl-z para finalizar:

Cantidad de vocales digitadas: 5
```

Notar que el parámetro por referencia de la función *contarVocales* es el que le facilita a la variable *n*, de la función *main*, el resultado del conteo.

Una cosa curiosa que se observa en este ejemplo es la forma de controlar el lazo. Pudo haberse controlado de cualquier otra manera, pero se hizo así para que observen una característica del comando *cin*, que se aprovecha en esta ocasión.

El comando *cin* devuelve cierto o falso, dependiendo de si tiene éxito en la lectura, o no. La combinación de teclas *Ctrl-z* genera el mismo byte que el sistema operativo Windows coloca al final de cada archivo almacenado en el disco, para saber dónde termina. Este byte es interpretado por C/C++ como el valor booleano falso, así que podemos aprovechar esto para utilizarlo en una expresión condicional. La forma de interpretar la instrucción *while(cin >> n)* es: "mientras haya algo que leer". Así que, cuando se digita *Ctrl-z*, eso le indica a C/C++ que ya se terminó la lectura.

En el sistema operativo Linux este byte puede generarse con la combinación de teclas *Ctrl-d*.