Punteros a función

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Mg. Mauricio Dávila*

Revisores: *Esp. Ing. Ernesto Gigliotti*

Versión : 1.1

Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0

Internacional.

Índice de contenido

1 Introducción

Los punteros a función son uno de los recursos más potentes y flexibles de C, permitiendo técnicas de programación eficientes y avanzadas .

Para hablar de punteros a funciones, previamente hay que dejar en claro que las funciones tienen "dirección". Del mismo modo que en un array se asume que su dirección es la del primer elemento, se asume también que la dirección de una función es la del segmento de código ("Code segment") donde comienza el código de dicha función. Es decir, la dirección de memoria a que se transfiere el control cuando se la invoca (su punto de comienzo). El nombre de la función es un puntero a dicha función, cuando se quiere obtener la dirección de memoria de una función, solo basta con nombrarla sin los parentesis.

Una vez establecido esto, no tiene que extrañar que puedan definirse variables de un tipo especial para apuntar a estas direcciones. Técnicamente un puntero-a-función es una variable que guarda la dirección de comienzo de la función.

1.1 Definición

Un puntero a función es una variable que almacena la dirección de una función, permitiendo invocar a dicha función a través del puntero. Este tipo de construcción es útil pues encapsula comportamiento, que puede ser llamado a través de un puntero. Veamos cómo funciona mediante un ejemplo sencillo que crea un puntero a una función de saludar y lo invoca:

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

void saludar()
{
	printf("Hola\n");
}

int main(void)
{
	void (*pFuncion)(void);
	pFuncion =  saludar;
	pFuncion();
	return EXIT_SUCCESS;
}
```

1.2 Puntero a funciones con parámetros

Si la función a la que se quiere apuntar recibe algún parámetro, hay que indicar sus tipos al declarar el puntero. En el siguiente ejemplo la función recibe un parámetro por valor, otro por referencia y muestra sus valores. Cuándo se declara el puntero se escriben los tipos de los argumentos entre paréntesis.

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

void mostrar(int parametroA, float* pParametroB)
{
	printf("A: %d - B: %f \n",parametroA,*pParametroB);

}

int main(void)
{
	float auxiliarFloat = 3.14;
	void (*pFuncion)(int,float*);

	pFuncion =  mostrar;
	pFuncion(22,&auxiliarFloat);
	return EXIT_SUCCESS;
}
```

1.3 Puntero a función como parámetro de otra función

Las funciones también pueden pasarse como parámetros a otras funciones. En el siguiente programa se define una función principal "Calcular" que recibe tres parámetros, los dos operandos y el puntero a la función que realiza la operación.

Con este ejemplo puede verse que los punteros a funciones sirven para eliminar la redundancia en el código: al pasar la función como parámetro nos ahorramos tener que escribir variantes de la función principal para realizar cada una de las operaciones.

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

void sumar(int parametroA, int parametroB, int* pResultado)
{
	*pResultado = parametroA + parametroB;
}

void restar(int parametroA, int parametroB, int* pResultado)
{
	*pResultado = parametroA - parametroB;
}

int calcular( int parametroA, int parametroB, void(*pFuncion)(int,int,int*) )
{
	int auxResultado;
	pFuncion(parametroA , parametroB , &auxResultado);
	return auxResultado;
}

int main(void)
{
	int auxiliar;

	auxiliar = calcular(10 , 5 , restar);
	printf("El resultado de la resta es %d\n",auxiliar);

	auxiliar = calcular(10 , 5 , sumar);
	printf("El resultado de la suma es %d\n",auxiliar);

	return EXIT_SUCCESS;
}
```

1.4 Simplificar el uso de punteros de función

Podemos utilizar typedef para simplificar el uso de punteros de función en un caso como el anterior, la sintaxis para declarar un typedef para un puntero de función es:

```
typedef tipoRetorno (*nombreTipoPuntero)(tipoParametros);
```

Veamos cómo queda el ejemplo anterior al implementar esta solución:

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

typedef void(*tFuncionOperacion)(int,int,int*);

void sumar(int parametroA, int parametroB, int* pResultado)
{
	*pResultado = parametroA + parametroB;
}

void restar(int parametroA, int parametroB, int* pResultado)
{
	*pResultado = parametroA - parametroB;
}

int calcular( int parametroA , int parametroB , tFuncionOperacion pFuncion)
{
	int auxResultado;
	pFuncion(parametroA , parametroB , &auxResultado);
	return auxResultado;
}

int main(void)
{
	int auxiliar;

	auxiliar = calcular(10,5,restar);
	printf("El resultado de la resta es %d\n",auxiliar);

	auxiliar = calcular(10,5,sumar);
	printf("El resultado de la suma es %d\n",auxiliar);

	return EXIT_SUCCESS;
}
```