Funciones C++

Antonio Espín Herranz

Punteros void a funciones

- El uso más importante de punteros void en C++ es pasar la dirección de tipos de datos diferentes en una llamada a función cuando no se conoce por anticipado que tipo de dato se pasa.
- void mi_funcion(**void *p**);
- **p**: puede ser cualquier tipo de puntero.

Ejemplo

```
#include <iostream>
using namespace std;
enum dato{caracter,real,entero,cadena};
void ver(void *,dato);

int main(){
char a='b';
int x=3;
double y=4.5;
char *cad="hola";

ver(&a,caracter);
ver(&x,entero);
ver(&y,real);
ver(cad,cadena);
}
```

```
void ver(void *p, dato d){
    switch(d){

    case caracter: printf("%c\n",*(char *)p);
    break;

    case entero: printf("%d\n",*(int *)p);
    break;

    case real: printf("%ld\n",*(double *)p);
    break;

    case cadena: printf("%s\n",(char *)p);
    }
}
```

Compilación separada

maximo.h / maximo.cpp

```
//prototipo de la función
#ifndef MAXIMO_H
    #define MAXIMO
    int max(int,int);
#endif

//Archivo maximo.cpp
//definición de la función
int max(int x, int y) {
    if (x>y) return(x);
    return(y);
}
```

main.cpp

```
#include <iostream>
using namespace std;
#include "maximo.h“

int main(){
    int a=5,b=6;
    cout<<"mayor "<<max(a,b);
}
```

Variable referencia

- Una referencia o variable referencia en C++ es simplemente otro nombre o alias de una variable.

- En esencia una referencia actúa igual que un puntero (contiene la dirección de un objeto), pero **funciona de diferente modo**, ya que ***no se puede modificar la variable a la que está asociada la referencia***, pero sí se puede modificar el valor de la variable asociada.

Ejemplo

```
//Usando variable referencia
int i;
int &x=i; // x es un alias de i
x=40; // i vale 40

//Usando punteros
int i;
int *p=&i;
*p=40; //i vale 40
```

Parámetros por valor y por referencia

- En C++ el paso por valor significa que al compilar la función y el código que llama a la función, ésta recibe una copia de los valores de los parámetros que se le pasan como argumentos. Las variables reales no se pasan a la función, sólo copias de su valor.

- Cuando una función debe modificar el valor de la variable pasada como parámetro y que esta modificación retorne a la función llamadora, se debe pasar el parámetro por referencia. En este método, el compilador no pasa una copia del valor del argumento; en su lugar, pasa una referencia, que indica a la función dónde existe la variable en memoria.

- La referencia que una función recibe es la dirección de la variable. Es decir, pasar un argumento por referencia es, simplemente, indicarle al compilador que pase la dirección del argumento.

Ejemplos

```
void demo(int &valor){
    valor=5;
    cout<<valor<<endl;
}
void main(){
    int n=10;
    cout<<n<<endl;
    demo(n);
    cout<<n<<endl;
}
```

Una **limitación** del método de paso por referencia es que se pueden pasar sólo variables a la función. **No se pueden utilizar constantes ni expresiones en la línea de llamada a la misma**.

Modificadores const / volatile

- Tiene los mismo significados que en la declaración de variables.

- Si un parámetros viene marcado como **const**, el compilador no permitirá que se modifique.
  - Por ejemplo, la función **strcpy**.

- Con volatile, se entiende que puede ser modificado de forma externa y el compilador no lo optimizará,

Número de argumentos variable

- En **C++** hay un método para proporcionar argumentos a funciones que tienen listas de argumentos en número variable.

- Este método facilita al compilador toda la información necesaria en una lista con un número variable de argumentos, proporcionándole un conjunto de variables prescritas que le indican cuántos argumentos contiene la lista y qué tipo de datos son.

Número de argumentos variable

- Las macros que se pueden utilizar para este propósito están definidas en un archivo de cabecera denominado **stdarg.h** y
son: **va_start()**, **va_arg()** y **va_end()**.

- La macro **va_list** es un tipo de dato que es equivalente a una **lista de variables**. Una vez que se define una variable **va_list**, se puede utilizar como un parámetro de las macros: **va_start()** y **va_end()**.

Número de argumentos variable

- La sintaxis de la macro **va_start()** es:
  - **void va_start(va_list arg_ptr, prev_param);**

- **arg_ptr** apunta al primer argumento opcional en una lista variable de argumentos pasados a una función. Si la lista de argumentos de una función contiene parámetros que se han especificado en la declaración de la función, el argumento **prev_param** de **va_start()** proporciona el nombre del argumento especificado de la función que precede inmediatamente al primer argumento opcional de la lista de argumentos.
- Cuando la macro **va_start()** se ejecuta, hace que el parámetro **arg_ptr** apunte al argumento especificado por **prev_param**.

Número de argumentos variable

- La sintaxis de la macro **va_arg()** es:
  - **void va_arg(va_list arg_ptr,tipo);**
- La macro **va_arg()** tiene un propósito doble:
  - Primero, **va_arg()** devuelve el valor del objeto apuntado por el argumento **arg_ptr**.
  - Segundo, **va_arg()** incrementa **arg_ptr** para apuntar al siguiente elemento de la lista variable de argumentos de la función que se está llamando, utilizando el tamaño tipo para determinar dónde comienza el siguiente argumento.

- La sintaxis de la macro **va_end()** es: **void va_end(va_list arg_ptr);**
- La macro **va_end()** realiza las tareas auxiliares que son necesarias para que la función llamada retorne correctamente. Cuando todos los argumentos se leen, **va_end()** reinicializa **arg_ptr** a **NULL**.

Ejemplo

```
#include <iostream>
#include <stdarg.h>
using namespace std;

int calcular(int primero,...);

int main(){
cout<<calcular(2,15,-1)<<endl;
cout<<calcular(6,6,6,-1)<<endl;
cout<<calcular(8,10,1946,47,-1)<<endl;
}
```

```
int calcular(int primero,...){
    int cuenta=0,suma=0,i=primero;

    va_list marcador;
    va_start(marcador, primero);

    while (i!=-1){
        suma+=i;
        cuenta++;
        i=va_arg(marcador,int);
    }
    va_end(marcador);
    return suma;
}
```

Argumentos por defecto

- Una mejora de las funciones en C++ es que ***se pueden especificar los valores por defecto para los argumentos cuando se proporciona un prototipo de una función***. Cuando se llama a una función se pueden omitir argumentos e inicializarlos a un valor por defecto.

- Un argumento por defecto u omisión es un parámetro que un llamador a una función no ha de proporcionar. Los argumentos por defecto también pueden ser parámetros opcionales. Si se pasa un valor a uno de ellos, se utiliza ese valor. Si no se pasa un valor a un parámetro opcional, se utiliza un valor por defecto como argumento.

Ejemplo

```
void f(int ,int =2);
void main(){
      f(4,5);
      f(6);
}
void f(int i, int j){
      cout<<i<<","<<j<<endl;
}
```

- Al ejecutar el programa, se visualizará: **4,5 6,2**
- Los argumentos por defecto se pasan por valor.
- Todos los argumentos por defecto debe estar situados al final del prototipo de la función. Después del primer argumento por defecto, todos los argumentos posteriores deben incluir también valores por defecto.