ALGORITMOS Y ESTRUCTURA DE DATOS

X

Parámetros por referencia: comparación entre C, C++ y Pascal

Argumentos y parámetros

Cuando invocamos a una función, ya sea desde el programa principal o desde otra función podemos pasarle valores que serán los parámetros que la función tomará en cuenta para realizar su tarea.

Desde el punto de vista de quién invoca a la función (en adelante "el llamador") decimos que a la función le **pasamos argumentos**. En cambio, desde el punto de vista de la función que es invocada decimos que **recibimos parámetros**.

Si bien la diferencia es semántica, tenerla en cuenta nos ayudará a comprender mejor los conceptos que se pretenden explicar en este documento. Veamos el siguiente ejemplo:

```
int main()
{
   int a = 10;

   f(a,20); // invoco a una funcion

   return 0;
}

int f(int x, int y)
{
   // hago algo con los valores de x e y
}
```

En este fragmento de código, en el programa principal invocamos a la función `f` pasándole dos argumentos: la variable `a` y el valor literal 20. Pero desde el punto de vista de la función, ésta recibe dos parámetros: `x` e `y` cuyos valores seguramente tomará en cuenta para desarrollar su tarea.

Es importante aclarar que los argumentos que le pasamos a una función deben tener una correspondencia directa con los parámetros que la función espera recibir; tanto en la cantidad cómo en el tipo de datos.

Es decir: si una función espera recibir dos parámetros, ambos de tipo `int`, entonces al invocarla debemos pasarle dos argumentos de este mismo tipo de datos. Y si la función espera recibir un parámetro entero y luego otro de tipo `float` cuando la invoquemos tendremos que pasarle dos argumentos de esos tipos de datos, exactamente en el orden en que la función los espera recibir.

Parámetros por valor

Cuando le pasamos argumentos a una función esta recibe a través de los parámetros copias de sus valores actuales. Es decir que en el ejemplo del apartado anterior la función `f` recibirá en `x` el valor 10 y en `y` el valor 20.

Estos parámetros son copias que están almacenadas en la memoria interna de la función; por esto cualquier modificación que pretendamos hacerles solo tendrá validez dentro de la misma función pero cuando la invocación finalice todas sus variables (incluidos los parámetros) de destruirán y estas modificaciones quedarán sin efecto.

El caso típico para ilustrar este problema es el de una función que recibe dos parámetros e intenta permutar sus valores.

```
void permutar(int x, int y)
{
   int aux = x;  // resguardo el valor de x
   x = y;        // a x le asigno el valor de y
   y = aux;      // a y le asigno el valor que tenia x
}

// programa principal
int main()
{
   int a=10, b=20;

   // invocamos a la funcion permutar
   permutar(a,b);

   cout << "a vale: "<< a << ", b vale: " << b << endl;

   return 0;
}
```

Si le preguntamos a algún programador inexperto cual cree que será la salida que arrojará este programa seguramente dirá: "a vale: 20, b vale: 10". Claro que estará equivocando porque, como dijimos más arriba, los parámetros `x` e `y` sólo reciben copias de los valores que contienen las variables `a` y `b`; luego, al finalizar la llamada a la función `permutar` todas sus variables locales se destruirán y las variables `a` y `b` del programa principal no habrán sufrido ninguna modificación. Por esto, la salida que realmente arrojará el programa será: "a vale: 10, b vale: 20".

Claro que la intención al desarrollar la función `permutar` era que ésta efectivamente pueda permutar los valores de los parámetros que recibe y para lograrlo no tenemos que trabajar con sus valores sino con sus referencias.

Parámetros por referencia

Hemos llegado al punto central de este documento, que no es en sí mismo el concepto de "parámetro por referencia" sino explicar las diferencias que existen entre los lenguajes de programación C, C++ y Pascal para implementar dicho concepto.

Implementaciones C++ y Pascal

En estos lenguajes la implementación del concepto de parámetro por referencia es prácticamente idéntica. Para que una función trabaje con las referencias de los parámetros que recibe y no con las copias de sus valores debemos indicarlo explícitamente en su encabezado como veremos a continuación.

C++

```
void permutar(int& x, int& y)
{
   int aux = x;
   x = y;
   y = aux;
}

// programa principal
int main()
{
   int a=10,b=20;

   permutar(a,b);

   // la salida sera: 20,10
   cout << a << "," << b <<endl;

   return 0;
}
```

Pascal

```
procedure permutar(var x:integer; var y:integer)
var aux:integer;
begin
   aux := x;
   x := y;
   y := aux;
end;

// programa principal
var a,b:integer;
begin
   a := 10;
   b := 20;

   permutar(a,b);

   // la salida sera: 20,10
   writeln(a,',',b);
end.
```

Tanto en C++ como en Pascal el manejo de las referencias es "declarativo". Alcanza con declarar en el encabezado de la función que vamos a trabajar con referencias y no con copias de los valores de los argumentos. En C++ lo hacemos agregando el símbolo `&` (léase *unpersand*) al final del tipo de dato del parámetro. En Pascal, en cambio, lo hacemos anteponiendo la palabra `var` al identificador de la variable.

En ambos lenguajes el llamador de la función (que en este caso es el programa principal) no debe hacer nada para que los argumentos pasen a la función por referencia y no por valor. Es la función quién decide si tomará el valor o la referencia de sus parámetros.

Implementación C

El concepto de "parámetro por referencia" no existe como tal en el lenguaje de programación C. Esto no quiere decir que en C las funciones no puedan modificar los valores de los parámetros que reciben; sólo significa que la responsabilidad de administrar las referencias (punteros y direcciones de memoria) recaerá sobre el programador. Veamos la versión C de la función que permuta los valores de sus parámetros.

```
void permutar(int* x, int* y)
{
   int aux = *x; // asigno en aux el contenido de x
   *x = *y;      // asigno en el contenido de x el contenido de y
   *y = aux;     // asigno en el contenido de y el valor resguardado en aux
}
```

Como en C no existen los parámetros por referencia la única manera de hacer que una función modifique los valores de sus parámetros es recibiendo directamente sus direcciones de memoria. Por esto la función `permutar` recibe dos parámetros de tipo `int*` que significa "puntero a entero". Luego, las variables `x` e `y` no contienen valores `int` sino las direcciones de memoria en donde dichos valores se encuentran ubicados.

Dentro del código de la función `x` e `y` representan direcciones de memoria y si queremos referirnos a los valores referenciados por estas direcciones tendremos que hablar de `*x` y `*y` respectivamente. Decimos entonces que `*x` es "el valor que está siendo direccionado por `x`". También podríamos decir que es "el valor contenido en la dirección apuntada por `x`" o bien, simplemente diremos que `*x` es "el contenido de `x`".

Lo anterior es fundamental para comprender porqué en la primer línea de código de la función hacemos:

```
int aux = *x;
```

en lugar de hacer:

```
int aux = x;
```

Las variables `aux` y `x` son de diferentes tipos de datos. La primera puede contener un valor de tipo `int`; en cambio la segunda contiene una dirección de memoria en la cual podría existir un valor de tipo `int`. Por esto `*x` (léase "el contenido de `x`") representa a dicho valor. Entonces `*x` es un `int`.

Desde el punto de vista del llamador (el programa principal) esto no pasará inadvertido porque la función `permutar` ahora espera recibir dos direcciones de memoria que deben apuntan respectivamente a los dos valores enteros que queremos que sean permutados. Para esto utilizaremos el operador `&` que al aplicarlo a una variable retorna su dirección de memoria.

```
// programa principal
int main()
{
   int a=10,b=20;

   permutar(&a,&b);

   // la salida sera: 20,10
   printf("%d, %d\n", a, b);

   return 0;
}
```

Comparación entre C++ y C

La confusión resulta casi inevitable porque el mismo operador que en C++ utilizamos en el encabezado de la función para indicar que un parámetro lo recibiremos por referencia, en C se utiliza como operador de dirección que al anteponerlo a una variable nos permite acceder a su dirección de memoria.

C++

```
void permutar(int& x, int& y)
{
   int aux = x;
   x = y;
   y = aux;
}

// programa principal
int main()
{
   int a=10,b=20;

   permutar(a,b);

   // la salida sera: 20,10
   cout << a << "," << b <<endl;

   return 0;
}
```

C

```
void permutar(int* x, int* y)
{
   int aux = *x;
   *x = *y;
   *y = aux;
}

// programa principal
int main()
{
   int a=10,b=20;

   permutar(&a,&b);

   // la salida sera: 20,10
   printf("%d, %d\n",a,b);

   return 0;
}
```

Esta comparación nos permite reforzar el concepto que planteamos al inicio: En C++ utilizamos `&` para indicar, en el encabezado de la función, que queremos trabajar con la referencia al parámetro. Luego, todo el manejo de direcciones de memoria resulta transparente para el programador.

Como contrapartida, en C la responsabilidad manipular las direcciones de memoria para lograr el efecto "parámetro por referencia" es nuestra. Entonces utilizamos `&` cómo "operador de dirección" para obtener la dirección de memoria de las variables que pasaremos como argumentos a la función.