TALEN

Funciones 2

Sobrecarga

La sobrecarga de funciones es una propiedad que tiene C++ para crear funciones con el mismo nombre pero diferentes parámetros. Se toma a los parámetros o argumentos distintos cuando: se tiene un tipo de dato distinto, cuando se tiene un número diferente de argumentos, cuando ocurre las dos cosas. Por ejemplo, supongamos que tenemos una función que devuelve el cuadrado de un número.

```
int cuadrado(int n){return (n*n); }
```

Sobrecarga

¿Qué pasa? Que esta función sólo acepta un número entero (int) así que, si queremos pasar como argumento un número de tipo real float o double no podríamos, a menos que hiciéramos dos funciones como sigue:

```
float fcuadrado(float n){return (n*n); }

double dcuadrado(double n){return (n*n); }
```

Sobrecarga

Pero como C++ soporta la sobrecarga, se puede implementar funciones sobrecargadas que tuvieran el mismo nombre y diferentes argumentos.

```
int cuadrado(int n){return (n*n); }

float cuadrado(float n){return (n*n); }

double cuadrado(double n){return (n*n); }
```

Introducción a Punteros

Punteros

Un puntero es una dirección que se refiere a una ubicación en la memoria. Se utilizan comúnmente para permitir que las funciones o estructuras de datos conozcan y modifiquen la memoria sin tener que copiar la memoria a la que se hace referencia. Los punteros se pueden utilizar con tipos primitivos (integrados) o definidos por el usuario.

Los punteros hacen uso de la "desreferencia" * , "dirección de" & , y "flecha" -> operadores. Los operadores '*' y '->' se usan para acceder a la memoria a la que se apunta, y el operador & se usa para obtener una dirección en la memoria.

Creando una variable de puntero

Se puede crear una variable de puntero utilizando la sintaxis específica * , por ejemplo, int *pointer_to_int; .

Cuando una variable es de tipo puntero (int *), solo contiene una dirección de memoria. La dirección de la memoria es la ubicación en la que se almacenan los datos del tipo subyacente (int).

Dirección de otra variable

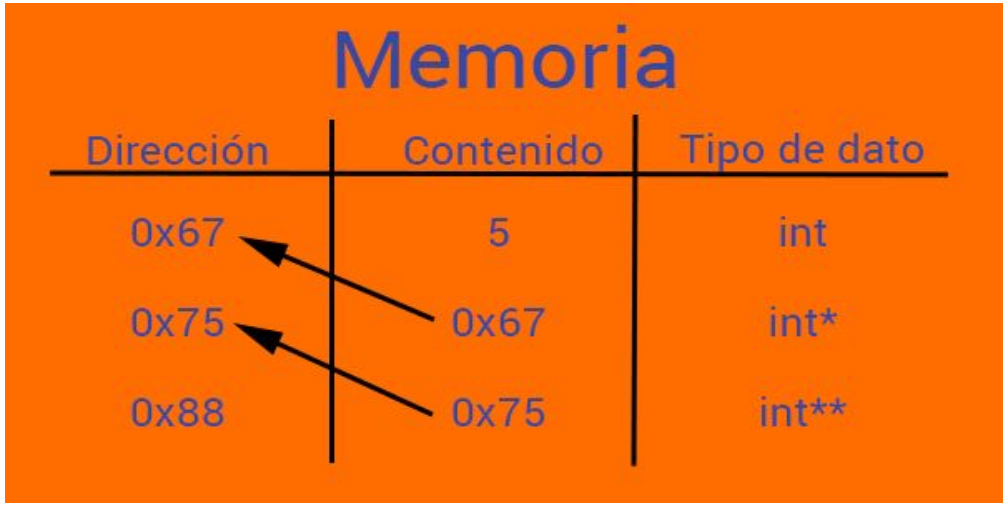
Los punteros se pueden asignar entre sí como variables normales; en este caso, es la dirección de memoria que se copia de un puntero a otro, no los datos reales a los que apunta un puntero.

Además, pueden tomar el valor nullptr que representa una ubicación de memoria nula. Un puntero igual a nullptr contiene una ubicación de memoria no válida y, por lo tanto, no hace referencia a datos válidos.

Puede obtener la dirección de memoria de una variable de un tipo dado prefijando la variable con la dirección del operador & . El valor devuelto por & es un puntero al tipo subyacente que contiene la dirección de memoria de la variable

Accediendo al contenido

Cómo tomar una dirección requiere & , también el acceso al contenido requiere el uso del operador de referencia * , como un prefijo. Cuando se hace referencia a un puntero, se convierte en una variable del tipo subyacente (en realidad, una referencia a él). Luego se puede leer y modificar.



Operaciones de puntero

Hay dos operadores para punteros: Dirección de operador (&): devuelve la dirección de memoria de su operando. Operador de contenido de (desreferencia) (*): devuelve el valor de la variable ubicada en la dirección especificada por su operador.

```
int var = 20;
int *ptr;
ptr = &var;

cout << var << endl;
//Outputs 20 (The value of var)

cout << ptr << endl;
//Outputs 0x234f119 (var's memory location)

cout << *ptr << endl;
 //Outputs 20(The value of the variable stored in the pointer ptr
```

Array

Array

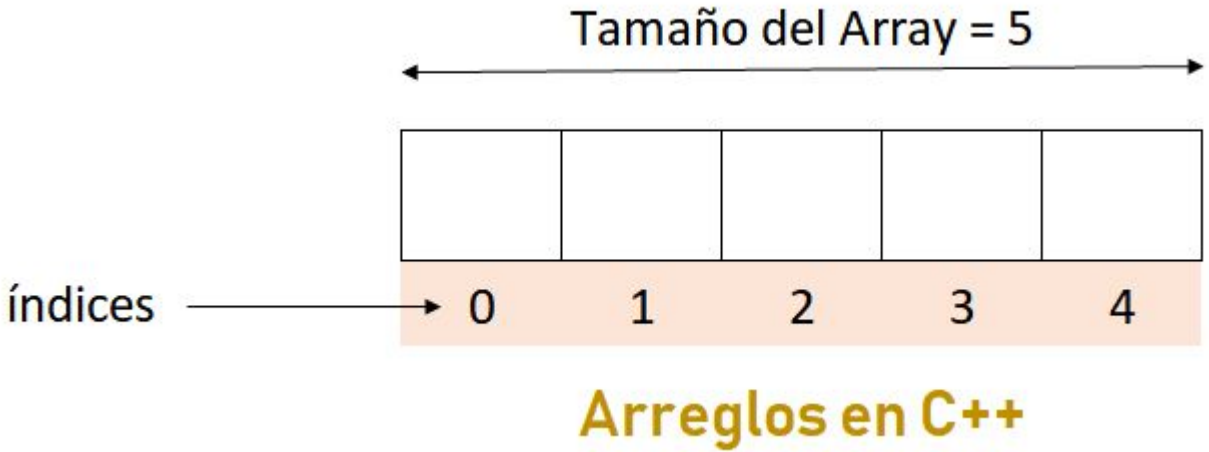
Con frecuencia tendremos necesidad de realizar una serie de cálculos u operaciones sobre un conjunto de datos del mismo tipo. La solución más simple es crear tantas variables como sean necesarias, por ejemplo si vamos a registrar 10 temperaturas debemos definir, por ejemplo.

Array

Expresado formalmente, un Array o Arreglo de datos, es una estructura de datos que permite almacenar un conjunto de datos del mismo tipo, definiendo el intervalo con un límite inferior y un límite superior, con un número denominado Índice hacemos referencia a cada elemento individual del arreglo. Se utilizará indistintamente el término Array o Arreglo

Array

Para comprender mejor los arrays, resulta muy útil la siguiente descripción gráfica.



La sintaxis para definir un arreglo en C++

TipoDato: Tipo de dato que tendrán los elementos del arreglo.

nombreArray: Nombre que asignamos para referirnos al arreglo.

TamañoArray: Valor entero que delimita el tamaño del arreglo.

Declarando un Arreglo

Si deseamos declarar un arreglo llamado arrTemp con 5 elementos de punto flotante, debemos hacerlo de esta forma:

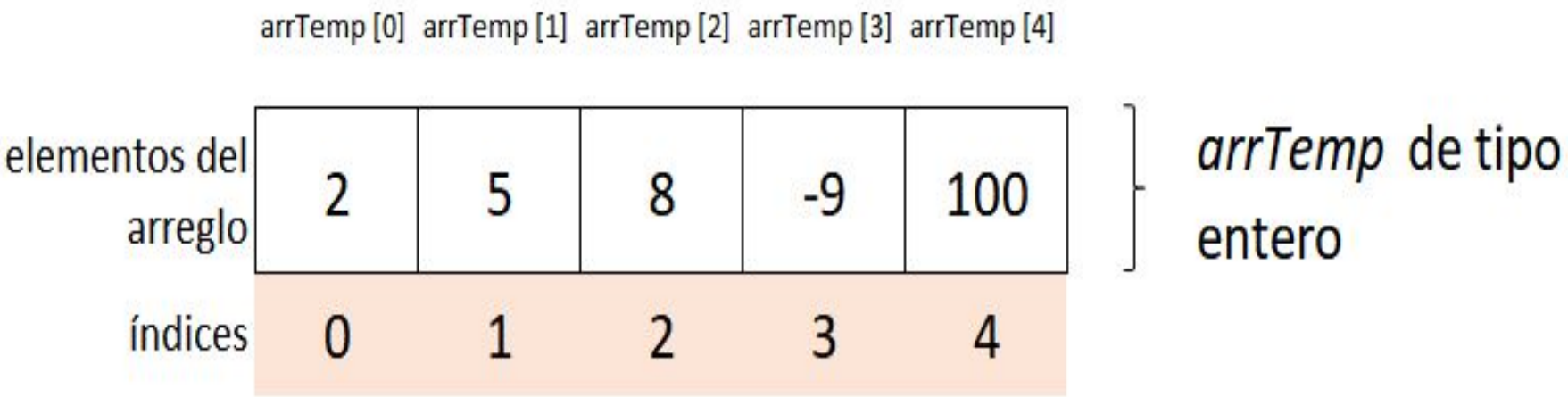
float arrCalif[5]; //Declara un arreglo de 5 elementos de punto flotante llamado arrCalif

Accediendo a los elementos del Arreglo

Para acceder a un elemento del arreglo debemos usar el nombre del arreglo utilizando el índice del elemento que nos interesa. Por ejemplo, si queremos asignar los valores 2, 5, 8, -9, 100 al arreglo arrTemp lo haremos de la siguiente forma:

```
arrTemp[0] = 2;

arrTemp[1] = 5;

arrTemp[2] = 8;

arrTemp[3] = -9;

arrTemp[4] = 100;
```

Para que visualices lo que hemos hecho hasta el momento, apóyate en la siguiente imagen



Inicialización de Arreglos

Un arreglo que no se inicializa también contiene basura, por lo que resulta conveniente inicializarlo al momento de declararlo de esta forma:

float arrTemp[5] = {2, 5, 8 ,-9, 100}

Array_Integer_Puntero

Sobrecarga de Array Integer, Ingreso por Puntero,Listado por Puntero.

https://replit.com/join/zzffxoszik-maximo-arielari

Bibliografía

https://learntutorials.net/es/cplusplus/topic/3056/punteros