Punteros, Referencias y Manejo de Memoria en C++

Matías L. Marenchino

28 de junio de 2012

Ascentio Technologies S.A.

Tabla de Contenidos I

Outline

Definición

Un puntero es una variable que almacena una dirección de memoria.

Qué es una dirección de memoria

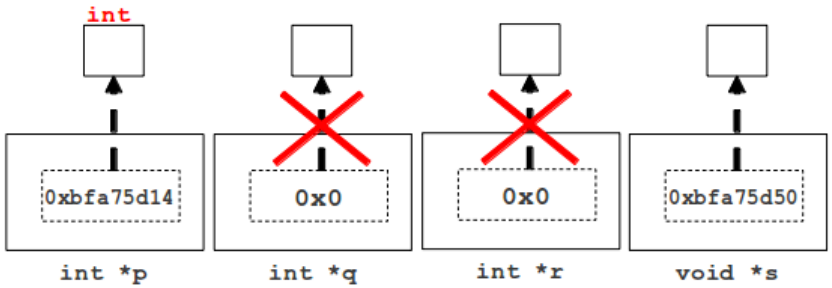
- Las variables son ubicadas en una ubicación única dentro de la memoria de la computadora. Esta única ubicación tiene asociada una única dirección, la dirección de memoria.
- En general, las variables contienen valores como el número 5 o la palabra “hola” y éstos valores son almacenados en una ubicación específica en la memoria.
- Pero los punteros son diferentes porque almacenan direcciones de memoria (y tienen la habilidad de apuntar -puntero- mediante el uso de tal dirección).

Outline

*

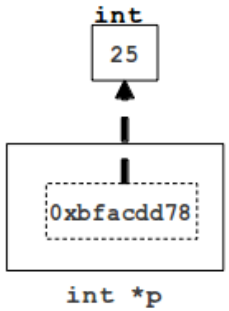
Permite definir un puntero:

```
int *p;
int *q = 0;
int *r = NULL;
void *s;
```



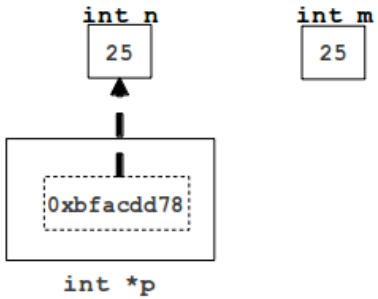
Reference operator &

```
int n = 25;
int *p = &n;
```



Dereference operator *

```
int n = 25;
int *p = &n;
int m = *p;
```

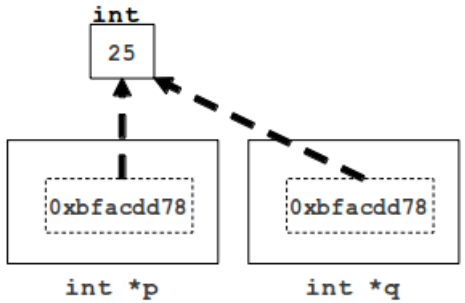


Identidad

p == &*p == *&p

Copia de punteros

```
int n = 25;
int *p = &n;
int *q = p;
```



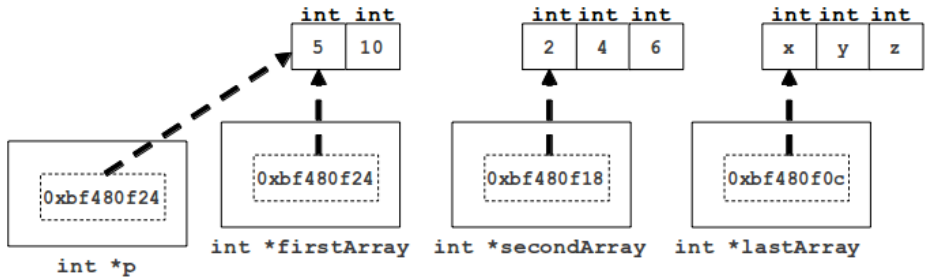
Outline

Definición

Un arreglo en C++ es un conjunto de datos que se alamacenan en memoria de manera contigua con el mismo nombre.

Ejemplo

```
int firstarray [] = {5, 10};
int secondarray [3] = {2, 4, 6};
int lastarray [3];
int *p = firstarray;
```



Qué pasa si?

```
lastarray = secondarray;
```

Outline

Acceso a elementos

El acceso a un elemento de un arreglo se hace de la siguiente manera:

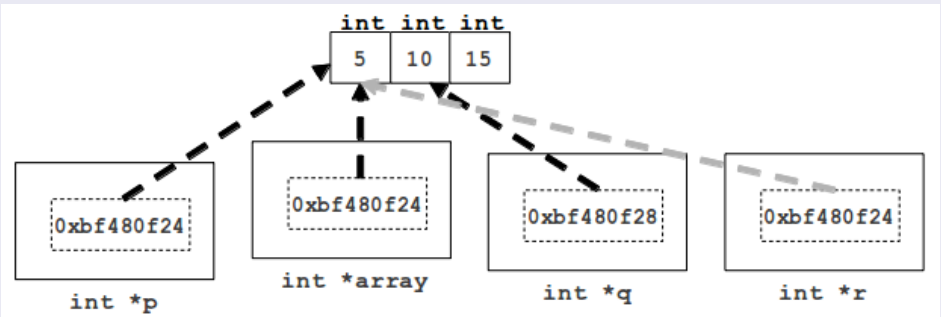
```
name [index];
```

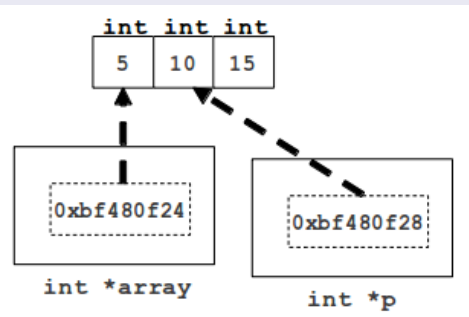
El tipo de tal expresión será el mismo al cual apunta la variable “name”.

Aritmética sobre punteros

```
int array [3] = {5, 10, 15};
int *p = array;
int *q = p + 1;
int *r = q - 1;
```

```
p++;
```





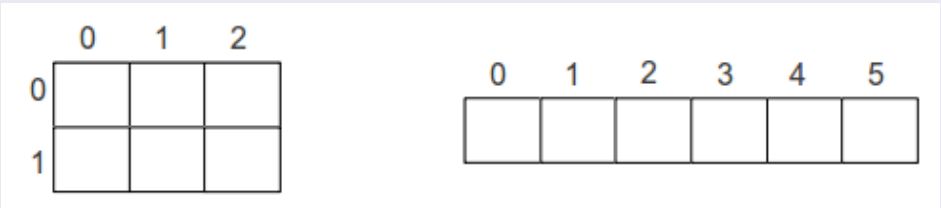
Outline

Qué son

Pueden verse como arreglos de arreglos. Por ejemplo, en el caso de 2 dimensiones, son Matrices.

Ejemplo

```
int matrix [3][2];
```



Outline

Puntero a constante

```
const int *p = ...;
```

Puntero constante

```
int * const p;
```

Puntero constante a constante

```
const int * const p = ...;
```

Outline

Ejemplos:

```
int i = 10;
int *p = &i;
int array [3] = {1,2,3};
float secondArray [5];
int *q = array;
```

Tamaños

sizeof(i) = 4
sizeof(p) = 4
sizeof(array) = 12
sizeof(secondArray) = 40
sizeof(q) = 4

Outline

Definición

Una referencia es un nombre alternativo para un objeto. Es como un puntero constante que se de-referencia automáticamente. Es una especie de alias o "alter ego" del objeto al que se refiere.

&

```
int i = 10;
int &r = i;
```

Todo aquello que le haga a *r*, se verá reflejado en *i* (y viceversa).

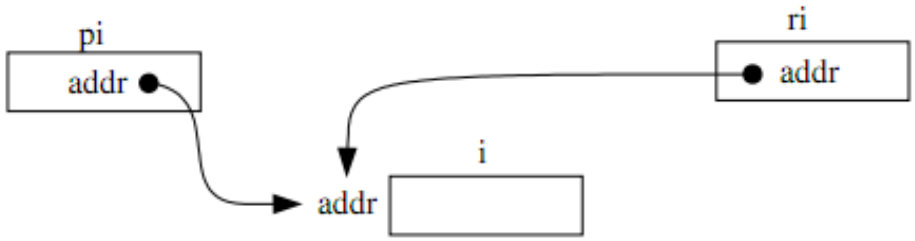
Outline

Punteros

```
int i = 4;
int* pi = &i;
```

Referencias

```
int i = 4;
int& ri = i;
```



Acceso

```
cout << *pi << endl;
```

```
cout << ri << endl;
```

Manejo de memoria

```
pi++;
```

```
//imposible, la referencia es constante
```

Ejemplos en la práctica

```
int i = 10;
int &r = i;
r = 20;                 // i = 20, j = UND, r = 20
int j = 34;
r = j;                  // i = 34, j = 34,  r = 34
j++;                    // i = 34, j = 35,  r = 34
```

==========

```
int *p = new int(3);        // *p = 3, r = UND
int &r = *p;                // *p = 3, r = 3
r++;                        // *p = 4, r = 4
p++;                        // SOAB
...
```

==========

```
void swap(int a, int b)
{
  int tmp = a;
  a = b;
  b = tmp;
}
...
int main (void)
{
  int x = 3, y = 7;
  swap(x, y);
  // x = 3 & y = 7
  ...
```

```
void swap(int *a, int *b)
{
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
...
int main (void)
{
  int x = 3, y = 7;
  swap(&x, &y);
  // x = 7 & y = 3
  ...
```

```
void swap(int &a, int &b)
{
  int tmp = a;
  a = b;
  b = tmp;
}
...
int main (void)
{
  int x = 3, y = 7;
  swap(x, y);
  // x = 7 & y = 3
  ...
```

Outline

new y delete

Los operadores new y delete constituyen la manera para hacer asignación dinámica de memoria y lirerarla. Son una evolución de los operadores malloc y free del lenguaje C.

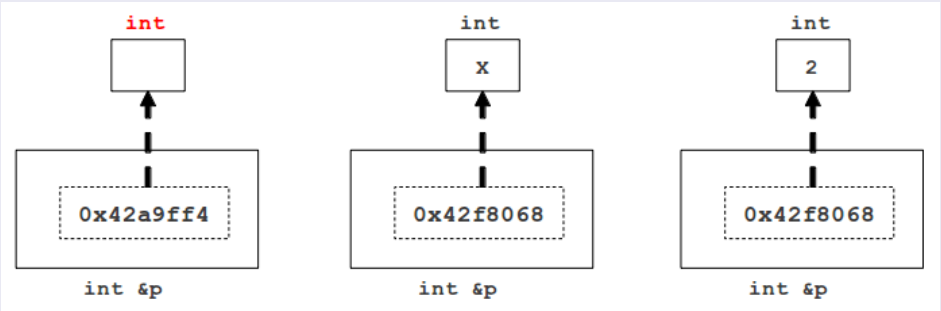
Uso

Se usan de a pares: new y delete o new [] y delete [].
Cada vez que se solicita memoria mediante una llamada a new, se debe matener la referencia a esa porción de memoria para después liberarla con el operador delete.

Outline

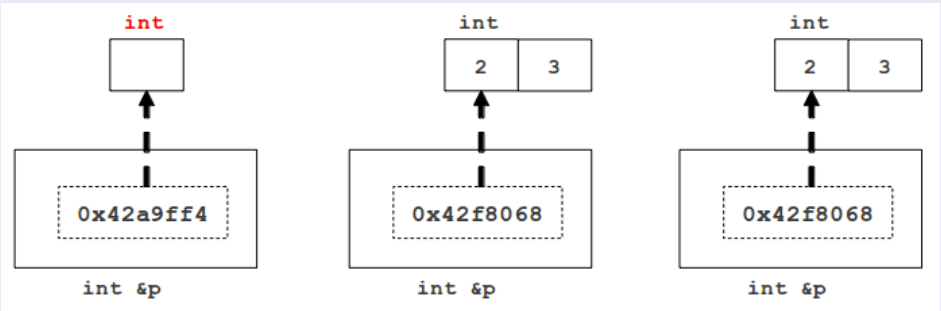
new y delete

```
int *p;
p = new int;
*p = 2;
...;
delete p;
```



new [] y delete []

```
int *p;
p = new int [2];
p[0] = 2;
p[1] = 3;
...;
delete [] p;
```



FIN

En papel:

- http://vision.fe.uni-lj.si/classes/SDV-vaje/vaje/labsheet03.pdf (no hacer los ej. 14 y 15, está también disponible en ./labsheet03.pdf)
- Resolver los ejercicios del libro de Stroustrup sección 5:
  - 3
  - 4
  - 5; A qué se debe?
  - 6

Librería de vectores

Implementar una librería para el manejo de colas “queue”, según los siguientes requerimientos:

- Implementar una clase vector;
- la clase vector tendrá un atributo (por lo pronto, del tipo) “double * array”;
- posibilidad de construir un objeto a partir de un “double * array”;
- se tienen que poder sumar vectores;
- cálculo de producto escalar entre vectores (dot product);
- disponer de métodos getter y setter para el arreglo (esto puede servir para ser usado en otras librerías);
- implementar la clase cola basada en un vector;
- la cola tiene métodos enqueue, dequeue y un getter del vector o arreglo (o lo que convenga)