

Notas para programar C++

Daniel Alconchel Vázquez

Metodología de la Programación II

Anotaciones para programar mejor

- La conversión de cadena de caracteres ('10') a string es automática
- Podemos convertir un string a un puntero constante con la función de la siguiente forma:

```
std::string l;
const char *s = l.c_str();
```
- cout, cin son objetos, por lo que podemos crear otros objetos como:

```
ostream &os           istream &is
os << " — ";         is >> — ;
```

ostream → flujo salida
istream → flujo entrada

Pasar parámetros como argumentos del main:

Nota: args[0] = llamada al programa

llamada al prog	Indicador parámetro	valor parámetro	Indicador parámetro	valor parámetro
-----------------	---------------------	-----------------	---------------------	-----------------

- 1) comprobar que el nº de parámetros es correcto
- 2) comprobar los indicadores de parámetros para ver que son correctos.
- 3) Procesar los datos.

- La función atoi() pasa de cadena → int

```
#include <cstdlib>
atoi(args[1])
```

- Resumen inserción de datos

```
#include <iostream>
ifstream entrada1, entrada2, entrada3(100%);
ofstream salida;
entrada1.open (" ruta absoluta del archivo ")
if (entrada1) Mira si se ha abierto el archivo
    entrada1 >> d Equivale a cin
entrada1.close();
el se fuerza el cierre del programa
cerr << " Error no se han abierto los datos "
```

- Datos compuestos pasan por const & . Los datos simples dependen si las funciones son constantes o no.
- Podemos actualizar un string como


```
string s = " "; word
s += word + "-"
```

Resumen memoria dinámica y punteros

Sintaxis puntero:

$$<\text{tipo}> \ * <\text{identificador}>$$

Cuando se declara un puntero se reserva memoria para albergar la dirección de memoria de un dato, no el dato en sí.

Operadores:

- & <var> devuelve la dirección de la variable <var>
Se suele usar para asignar valores a datos de tipo puntero



- * <puntero> devuelve el valor del objeto apuntado por puntero

Inicialización:

- Se puede inicializar con la dirección de una variable
 - Asignarle la dirección nula
- $$\text{int } * \text{puntero} = \text{NULL}$$
- Asignarse otro puntero (solo si es del mismo tipo)

Operadores relacionales

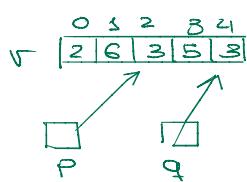
- Los operadores relacionales <, >, <= , >= , != , == son aplicables a punteros. El valor del puntero (dirección de memoria) se comporta como un número entero

$p_1 == p_2$ // Comprueban si apuntan a la misma dirección de memoria
 $*p_1 == *p_2$ // Comprobar si lo almacenado en esas direcciones son iguales

En el caso $<$, $>$, $<=$, $>=$ si los usamos con los valores del puntero (direcciones de memoria) permiten conocer la posición de un objeto respecto a otro en la memoria.

$$p = \&r[2]$$

$$q = \&r[4]$$



$$\begin{array}{ll} p == q & \text{false} \\ p != q & \text{true} \\ +p == +q & \text{true} \\ p > q & \text{false} \\ p < q & \text{true} \end{array}$$

$$\begin{array}{ll} p > q & \text{false} \\ p < q & \text{true} \\ p >= q & \text{true} \\ p <= q & \text{true} \\ p >= q & \text{false} \end{array}$$

Operadores aritméticos:

- Al sumar o restar un número N al valor del puntero, éste se incrementa o decrementa en determinado número de posiciones, en función del tipo de dato apuntado

Punteros y vectores:

- Un vector es un puntero constante, luego se puede usar igual, menos combinar la dirección de memoria a la que apunta

Punteros y cadenas:

- Las cadenas de caracteres son punteros constantes de tipo char

Notación de corchetes:

- * Se copia el contenido literal en el array
 - * Es posible modificar caracteres de la cadena
- ```
char cod1[5] = "Hola" // copia literal "Hola" en cod1
cod1[2] = 'b' // cod1 contiene ahora "Hoba"
```

Notación de punteros:

- \* Copia la dirección de memoria de la constante literal en el puntero
  - \* No es posible modificar caracteres de la cadena.
- ```
const char *cod2 = "Hola" // se asigna al puntero
cod2[2] = 'b' // Error
```

Punteros a objetos struct o class

Un puntero puede apuntar a un objeto de estructura o clase.

```
class Persona {
private:
    int edad;
    double estatura;
public:
    int getEdad() const;
    double getEstatura() const;
```

```
void setEdad(int n);
void setEstatura(double n);
};

Persona *ope, *ptr;
ope.setEdad(27); ptr.setEstatura(1.98);
ptr = &ope;
cout << *ptr.getEdad() << endl;
```

Si p es un puntero a un objeto struct o class podemos acceder a sus datos miembros de dos formas:

- * (p). miembro
- * p-> miembro

Puntero a punteros

Un puntero a puntero es un puntero que contiene la dirección de memoria de otro puntero.

```
int a = 5;  
int * p;  
int ** q;  
p = &a;  
q = &p;
```

Array de punteros

Podemos hacer un vector donde cada posición sea un puntero.

Puntero y const :

```
const int * p // El dato apuntado es constante  
int const * p // El puntero apunta a una dirección const.  
const int const * p // Ambas
```

Puntero a funciones:

Contiene la dirección de memoria de una función, o sea la dirección donde comienza el código que realiza la tarea de la función apuntada. Con estos punteros podemos hacer las siguientes operaciones:

- * Usarlos como parámetro a una función <tipo> (*nombre)(param)
- * Ser devueltos con una función con return (*nombre)(param) Para llamar
- * Crear arrays de puntero a funciones
- * Asignarlos a otras variables puntero a función
- * Usarlos para llamar a la función apuntada

Metodología de la memoria dinámica:

- 1) Reservar memoria
- 2) Utilizar memoria reservada.
- 3) Liberar memoria reservada

Operador new:

Reserva una zona de memoria en el Heap del sistema adecuado para almacenar un dato del tipo tipo (sizeof(tipo) bytes) devolviendo la dirección de memoria donde empieza la zona reservada.

$$\langle \text{tipo} \rangle * p$$
$$p = \text{new} \langle \text{tipo} \rangle$$

Si new no puede reservar espacio se provoca una excepción y el programa termina.

Operador delete:

Liberar la memoria del Heap que previamente se había reservado y que se encuentra referenciada por un puntero.

delete puntero.

Creación de array dinámicos:

$\langle \text{tipo} \rangle * p$

$p = \text{new} \langle \text{tipo} \rangle [\text{num}]$

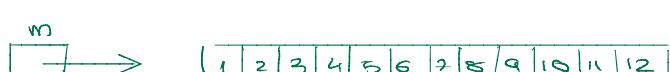
cuando acabemos de usarlo delete [] puntero

```
int * copia_vector (const int r[], int n) {
    int * copia = new int[n]
    for (int i=0; i<n; i++)
        copia[i] = r[i]
    return copia
}
```

Creación de matrices dinámicas:

- Usando un array 1D:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12



Creación de la matriz:

```
int * m;
int nfil, ncol;
m = new int [nfil*ncol];
```

Acceso a elemento f,c

int a;

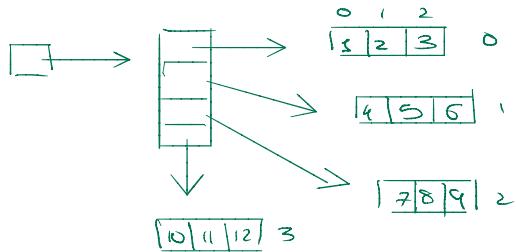
$a = m[f * ncol + c]$

Liberación:

delete [] m

Usando array 1D en punteros a arrays 1D:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12



Creación de la matriz:

```
int **m;  
int nfil, ncol;  
m = new int *[nfil];  
for (int i=0; i<nfil, ++i)  
    m[i] = new int [ncol];
```

Acceso al elemento m_{ij} :

```
int a;  
a=m[i][j];
```

Liberación:

```
for (int i=0; i<nfil; ++i)  
    delete [] m[i];  
delete [] m;
```

Lista de celdas enlazadas: