tecnun
CAMPUS TECNOLÓGICO DE LA UNIVERSIDAD DE NAVARRA
NAFARROAKO UNIBERTSITATEKO CAMPUS TEKNOLOGIKOA
Escuela Superior de Ingenieros · Ingeniarien Goi Mailako Eskola

Prácticas de C++

Practica Nº 2

Informática II
Fundamentos de Programación

Prof. Dr. Paul Bustamante

ÍNDICE

1. Introducción.

Como ya lo hemos venido haciendo, el primer ejercicio de esta práctica debe realizarlo solo, con el fin de que pueda ganar más experiencia en la programación en C++.

En algunos ejercicios se dará el código, lo que no significa que sólo tenga que escribirlo en el ordenador, debe tratar de realizarlos por su cuenta.

Recuerde borrar los ficheros que estén en los subdirectorios ***debug*** y ***release*** del proyecto, para que libere espacio en su disco ***G***\.

1.1 Ejercicio 1: Reserva dinámica de memoria para vectores.

En este ejercicio vamos a hacer uso de la reserva dinámica de memoria para almacenar un número determinado de valores (obtenidos de forma aleatoria, entre 0 y 100) y ordenarlos de mayor a menor.

El código para obtener los números y para la reserva dinámica de memoria se lo doy, el resto debe tratar de implementarlo solo.

Debe crear el proyecto ***Ejer1*** y el fichero ***ordena.cpp*** para introducir el código.

```
// fichero ordena.cpp
// ordena usando memoria dinamica
#include <iostream.h>
#include <stdlib.h>        //para rand()
void main(void)
{
    int Num;          //Numero de datos
    int *datos;       //puntero a int

    cout << "Cuantos numeros desea generar:";
    cin >> Num;
    //asignacion dinamica de memoria
    datos = new int[Num];
    if (datos == NULL) cout << "Error";
    //Llenar el vector
    for (int i=0;i<Num;i++){
        datos[i] = rand() * 100 / RAND_MAX;
    }
    //ordena los datos -> Hacer aquí el algoritmo de ordenacion
    . . .
    . . .
    //imprime los datos ordenados
    for (i=0;i<Num;i++) cout << i << ":" << datos[i] <<endl;
    //liberar memoria
    delete [] datos;
}
```

1.2 Ejercicio 2: Introducción a la estadística.

En este ejercicio va a tener una pequeña introducción a la estadística (realmente ya la ha tenido y quizá no lo ha notado).

El ejercicio consiste en pedir una serie de números al usuario y hallar el máximo, el mínimo y la media aritmética de ellos. Debe crear una variable puntero tipo ***float*** y pedir al usuario que introduzca el número de datos, luego debe introducir todos los datos a calcular.

Recuerde que debe reservar memoria de forma dinámica para almacenar el vector de datos.

Debe crear el proyecto ***Ejer2*** y el fichero ***estadistica.cpp*** para escribir el código en él.

La salida del programa debe ser algo así (es un ejemplo, no tienen que coincidir):

```
Numero de datos: 10
Máximo:  25
Minimo: 4
Media Aritmética: 14.6
Los datos Introducidos son:
..//Mostrar los datos en la consola
```

1.3 Ejercicio 3: Matrices dinámicas.

Ha llegado el momento de empezar a trabajar con la reserva de memoria dinámica para matrices.

Los pasos para hacer la *reserva dinámica de memoria* para matrices son:

a. Crear el array de punteros del tipo de datos: ***float **datos;***

b. Reservar memoria para el array de punteros: ***datos = new float*[fil];***

c. Hacer un bucle para reservar memoria para ***col*** columnas de cada fila:

```
for (int i=0;i<fil;i++) datos[i] = new float[col];
```

d. Ya tenemos creada la matriz, podemos trabajar con ella con los índices, por medio de los corchetes [], de la forma datos[i][j].

e. Finalmente, debemos hacer otro bucle para liberar la memoria de cada fila y luego debemos liberar la memoria asignada al array de punteros, en ese estricto orden

En el siguiente ejemplo verá cómo asignar memoria para una matriz de ***fil$_x$col***:

```
// fichero matriz.cpp
// crear matrices usando memoria dinamica
#include <iostream.h>
#include <string.h>        //para strlen
#include <stdlib.h>        //para atoi()
void main(void)
{
   int fil;  //numero de filas
   int col;  //numero de columnas
   float **datos;
   cout << "Num. Filas:";  cin >> fil;
   cout << "Num. Columnas:";  cin >> col;
   datos = new float*[fil];   //vector filas
   //reserva memoria para las columnas de cada fila
   for (int i=0;i<fil;i++)
      datos[i] = new float[col];

   //ya puede usar la matriz
   for (int f=0;f<fil;f++)
```

```
            for (int c=0;c<col;c++) datos[f][c] = (float)rand()/RAND_MAX;

        //imprime los valores
        for ( f=0;f<fil;f++){
            for (int c=0;c<col;c++) {
                cout << f << "," << c << ":"<<datos[f][c]<< " ";
            }
            cout <<endl;
        }

        //liberar memoria
        for (f=0;f<fil;f++) delete [] datos[f]; //libera filas
        delete [] datos;        //libera vector
}
```

Debe crear un proyecto ***Ejer3*** y el fichero ***matriz.cpp*** para escribir el código.

1.4 Ejercicio 4: Producto de matriz por matriz.

Basándose en el ejercicio anterior, vamos a realizar un programa que permita hacer el producto de matrices con asignación dinámica de memoria.

Debe crear un proyecto llamado ***Ejer4*** que contenga un programa que multiplique dos matrices **MatA** y **MatB** de cualquier tamaño; llámalo ***matxmat2.cpp***.

Pasos a seguir:

1- Debe crear tres punteros a punteros: ***`float **mA, **mB, **mC`***.

2- Pedir por teclado las dimensiones de la matriz **MatA** (filas y columnas).

3- Pedir por teclado sólo las columnas de la matriz **MatB**, ya que las filas tienen que ser igual a las columnas de **MatA** para la multiplicación.

4- Realizar la reserva dinámica de memoria de todas las matrices, usando para ello el operador ***new.***

5- Pedir los datos de las dos matrices **MatA** y **MatB**.

6- Realizar la multiplicación, como en el ejercicio de la práctica anterior, en sendos bucles ***for*** anidados, y almacenar el resultado en la matriz C.

7- Finalmente debe sacar por consola los datos de la matriz resultante:

8- Liberar la memoria asignada de todas las matrices, utilizando el operador ***delete***.

Recuerde la fórmula para multiplicar dos matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \qquad i = 1...n, j = 1,...,n$$

1.5 Ejercicio 5: Ordenando Nombres

Este quinto ejercicio consiste en crear un programa capaz de leer desde el teclado un número determinado de nombres, almacenarlos en un "***vector de punteros a caracteres***" (en otras palabras, una matriz de caracteres) y ordenarlos, por medio de la función ***strcmp***. Se recuerda que cada letra (***char***) es "tratada como si fuera un número" (una casilla) y por lo tanto almacenar un nombre se asemeja a almacenar una fila de una matriz. Se necesita por lo tanto un doble puntero (***char **nombres***) para poder almacenar un conjunto de palabras. En esta "matriz de letras" el número de columnas puede variar para cada fila (es ***strlen(nombre)+1***). Utilizando una única llamada a la función ***strcpy()*** es posible rellenar una fila completa de dicha ***matriz de caracteres***. Para utilizar dicha función es necesario incluir el fichero ***string.h***.

Crea un proyecto llamado ***Ejer5*** y el fichero ***nombres.cpp***, donde va a escribir el siguiente código y debe completar lo que falta:

```
// fichero nombres.cpp
#include <iostream.h>
#include <string.h>
#include <stdio.h>      //para gets()
void main(void)
{
     char tmp[120];
     char** pnombres;
     int Num;

    cout << "Cuantos Nombres desea ordenar:";
    cin >> Num;
    cin.ignore();

     pnombres = new char*[Num]; //Espacio para Num nombres
    //pedir datos
    for (int i=0;i<Num;i++) {
       cout << "Escriba el nombre " << i+1 << ":" << flush;
       cin.getline(tmp,120);

       // +1 porque hay que guardar '\0'
       pnombres[i] = new char[strlen(tmp)+1];
       // debe copiar las letras de tmp a la matriz pnombres
       . . .
     }
    //algoritmo de ordenación
    . . .
    //sacar datos por consola
    . . .
    //liberar memoria de la matriz de caracteres
    . . .
}
```

1.6 Ejercicio 6: Ordenando Nombres v1.1

Este sexto y último ejercicio consiste en hacer una modificación al programa anterior para que pida cómo desea ordenar los nombres, si de forma ascendente o descendente, y luego que pida los nombres a ordenar.