Guion de prácticas 3

Funciones y Arrays

DECSAI

Metodología de la Programación

Curso 2018/2019

Introducción

En este guion se pondrán en práctica los conceptos asociados al uso de arrays, matrices y su paso como parámetros a funciones.

Módulo para la gestión de arrays de enteros

Dado un array v de enteros, de tamaño n, implemente las siguientes funciones (utilice exactamente los nombres indicados):

- `obtieneMaxMin`: devuelve (por referencia) las posiciones donde se encuentran los valores máximo y mínimo del array (en ese orden). Si estos valores se repiten, basta con devolver una de las posiciones donde ocurren. La llamada sería `obtieneMaxMin(v, n, pos_max, pos_min)`, siendo `pos_max, pos_min` variables de tipo `int`.
- `promedioAjustado`: calcula el promedio de los valores del array, sin tener en cuenta ni el valor máximo ni el mínimo (si estuvieran repetidos, solo se descartan una vez).
- `masFrecuente`: devuelve el valor más frecuente de v y su frecuencia. Si hubiera más de uno, devuelve cualquiera de ellos. La llamada sería `masFrecuente(v, n, valor, frec)`, siendo `valor, frec` variables de tipo `int`.
- `estaOrdenado`: devuelve `true` si v está ordenado de forma creciente. La llamada sería `estaOrdenado(v, n)`.
- `todosDistintos`: comprueba si todos los elementos de v son diferentes. En otras palabras, devuelve `true` si no hay valores repetidos. La llamada sería `todosDistintos(v, n)`.
- `mezclaOrdenada`: recibe dos arrays ordenados de forma creciente v_1, v_2 con tamaños n_1, n_2 y los mezcla para devolver un array ordenado v_3 (y su tamaño n_3) que contendrá los valores de v_1 y v_2. NO debe copiar v_1 y v_2 en v_3 y luego ordenar este último. Puede ver a continuación algunos ejemplos:

v1	v2	v3
{2, 4, 6}	{1, 3, 5}	{1, 2, 3, 4, 5, 6}
{4, 7, 10}	{1, 5}	{1, 4, 5, 7, 10}
{9}	{4, 7}	{4, 7, 9}

La llamada sería `mezclaOrdenada(v1, n1, v2, n2, v3, n3)`.

Tareas

A partir de una carpeta `practica3`, cree las carpetas `src, include, bin, obj`. Las funciones previas se incorporarán a un módulo llamado `GestionArrays`. En la parte pública del módulo defina `const int MAX = 20`. Puede implementar todas las funciones adicionales que crea oportunas. Decida si debe hacerlas disponibles en la parte pública.

Utilice el fichero `testArrays.cpp`, disponible en PRADO, para probar el módulo. Puede observar que se realizan una serie de pruebas para comprobar su correcto funcionamiento. Puede encontrar una breve descripción de las mismas al final de este guión.

Registro de Temperaturas

Suponga que dispone de una matriz M de tamaño 12×31 para almacenar las temperaturas diarias. Cada fila representa un mes (la fila 0 corresponde a enero, mientras que la 11 a diciembre) y las columnas, días del mes (para simplificar, supondremos que todos los meses tienen 31 días y que las temperaturas se representan con valores enteros).

Se pretende desarrollar un módulo `Calculo` para procesar dicha matriz y obtener algunas estadísticas básicas sobre las temperaturas anuales. Estas funciones (que en todos los casos recibirán, al menos, la matriz como parámetro) son:

- `rellenarMatriz`: rellena la matriz con valores aleatorios entre $[min, max]$ (valores pasados como parámetros). Utilice el código de la Fig. 2.
- `mediaMensualAjustada`: devuelve el promedio de las temperaturas de un mes dado. Para el cálculo del promedio no se tendrán en cuenta los valores máximo y mínimo de temperaturas.
- `resumenMaximasMinimas`: devuelve dos vectores (de tamaño 12) con las temperaturas máxima y mínimas de cada mes, respectivamente
- `secuenciaDiasCalidos`: recibe una temperatura de referencia t y un número de días d y devuelve un array con los indices de los meses donde ocurrió que en d días seguidos la temperatura fue mayor que t. Implemente primero una función que calcule si en un array de enteros, existen d valores consecutivos mayores a cierto umbral t
- `mostrarMatriz`: muestra por pantalla el contenido de la matriz.

Cuando sea posible, utilice las funciones disponibles en el módulo `GestionArrays`, recordando que cada fila de la matriz se puede tratar como un array.

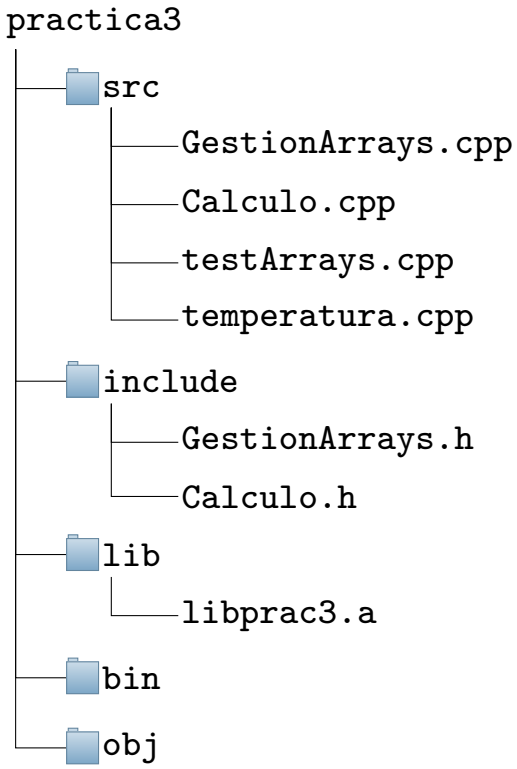

Figura 1: Estructura de carpetas para la entrega. El profesor utilizará un fichero `Makefile` basado en esta estructura para compilar y luego ejecutar sus programas.

Tareas

Partiendo de la estructura de directorios ya disponible del punto anterior, se pide implementar las funciones anteriores en un módulo `Calculo` (que hará uso de `GestionArrays`).

A partir de ambos módulos, construya una biblioteca `libprac3.a` y luego escriba un fichero `temperatura.cpp` que llame a las funciones que procesan la matriz. Plantee un esquema similar al fichero utilizado para probar el módulo `GestionArrays` (si necesita escribir valores específicos en la matriz, hágalo.).

Al finalizar la práctica, debe disponer de la estructura de directorios y ficheros que se muestra en la Fig. 1.

NOTA: esta práctica requerirá una entrega y autoevaluación. Próximamente se publicarán las instrucciones.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

const int FIL = 10;
const int COL = 5;
const int MIN = 0, MAX=10, NUM_VALORES = MAX-MIN+1;

int main(){
int M[FIL][COL];
int util_f, util_c;
int i,j;

time_t t;
srand ((int) time(&t));

// relleno la matriz con valores aleatorios
for(i = 0; i < FIL; i++)
   for(j = 0; j < COL; j++)
       M[i][j] = (rand() % NUM_VALORES) + MIN;

util_f = FIL; util_c = COL;

// muestro la matriz
for(i = 0; i < util_f; i++){
   cout << endl;
   for(j = 0; j < util_c; j++)
       cout << M[i][j] << "\t";
   }
}
```

Figura 2: Código para rellenar una matriz con números enteros aleatorios entre MIN y MAX.

Descripción del programa de pruebas

Utilice el fichero `mainParte1.cpp`, disponible en PRADO, para probar el módulo `GestionArrays`. Puede observar que se realizan una serie de pruebas para comprobar su correcto funcionamiento. Cada prueba se implementa como una función.

En todos los casos el planteamiento es similar. Se definen los datos de entrada `X` para la función `F`. Sabiendo que la salida esperada debe ser `Y`, se comprueba si `F(X) == Y`. Esta comprobación se realiza mediante una "aserción"[1].

[1]Según la R.A.E, aserción es 1) Acción y efecto de afirmar o dar por cierto algo, 2) Proposición en que se afirma o da por cierto algo.

Así, para probar la función `estaOrdenado` se propone el siguiente código:

```
void testOrden(){
  int n = MAX/2;
  int v[n];

  // caso 1
  for(int i = 0; i < n; i++)
    v[i] = i + 1;

  assert(estaOrdenado(v,n));

  // caso 2
  v[1] = 6;
  assert(!estaOrdenado(v,n));
}
```

En este ejemplo, el funcionamiento es el siguiente. En primer lugar se crea un array `v` de `n` elementos con valores `v` = {1,2,3,...,n}.

Luego aparece una función `assert` (disponible en el módulo `assert.h`) que comprueba si el parámetro es verdadero. En realidad `assert` (aserción, en español) es una macro de depuración que implementa una aserción (`test`) usada para comprobar suposiciones en el programa. Esta macro se expande como un bloque `if`, en el que se comprueba la condición `test` y, dependiendo de si es o no verdadera, puede abortar el programa.

Si `test` se evalúa como cero (es decir, si es falsa) entonces se aborta el programa mediante la función `abort()` y se imprime un mensaje en `stderr` en el que se incluyen la condición `test`, el nombre del archivo fuente y el número de línea en la que se llamó a `assert()`.

Por tanto, por la forma en que se ha construido el array `v` esperamos que la llamada `estaOrdenado(v,n)` devuelva `true`. Luego, con `v[1] = 6` forzamos la inclusión de un elemento repetido. Por tanto, esperamos que la llamada a `estaOrdenado(v,n)` devuelva `false`, y por tanto, llamamos a `assert` de la manera indicada.

Si nuestro programa llega al final, entonces todas las aserciones se han verificado y podemos asegurar que nuestro código ha pasado estas pruebas.