Guion de prácticas 4

Ejercicios con Punteros

DECSAI

Metodología de la Programación

Curso 2021/2022

Relación de ejercicios

En este guion se pondrán en práctica los conceptos de punteros. Puede implementar varios ejercicios en un único fichero, crear módulos, funciones de test, etc.

1) Indique que ocurre en las siguientes situaciones

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

int main (){
  int a = 5, *p;

  a = *p * a;
  if (a == *p)
    cout << "a es igual a *p" << endl;
  else
    cout << "a es diferente a *p" << endl;
}
```

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

int main (){
  int a = 5, *p;

  *p = *p * a;
  if (a == *p)
    cout << "a es igual a *p" << endl;
  else
    cout << "a es diferente a *p" << endl;
}
```

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

int main (){
  int a = 5, *p = &a;

  *p = *p * a;
  if (a == *p)
    cout << "a es igual a *p" << endl;
  else
    cout << "a es diferente a *p" << endl;
}
```

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

int main (){
  int a = 5, *p = &a, **p2 = &p;

  **p2 = *p + (**p2 / a);
  *p = a+1;
  a = **p2 / 2;
  cout << "a es igual a: " << a << endl;
}
```

```
int *v1;
int * const v2;
const int * v3;
const int * const v4;
```

```
v1 = v2;
v1 = v3;
v1 = v4;
```

```
v2 = v1;
v2 = v3;
v2 = v4;
```

```
v3 = v1;
v3 = v2;
v3 = v4;
```

```
v4 = v1;
v4 = v2;
v4 = v3;
```

```
*v1=*v2;
*v1=*v3;
*v1=*v4;
```

```
*v2=*v1;
*v2=*v3;
*v2=*v4;
```

```
*v3=*v1;
*v3=*v2;
*v3=*v4;
```

```
*v4=*v1;
*v4=*v2;
*v4=*v3;
```

2) Implemente una función que reciba un array de números enteros junto con su longitud y que devuelva un puntero al elemento mayor. No utilice el operador [], trabaje con aritmética de punteros. Considere la siguiente declaración:

```
const int MAX
int vector[MAX];
int usados; // 0 <= usados < MAX
```

Muestre como usaría la función para mostrar en la salida estándar:

1. El elemento mayor del vector.
2. El elemento mayor de la primera mitad.
3. El elemento mayor de la segunda mitad.

3) Implemente una función que reciba una cadena de caracteres tipo C y la invierta (sin usar el operador []).

4) Describa la salida del código mostrado en la Fig. 2. Escriba, compile y ejecute el código para comprobar si lo ha entendido correctamente.

5) Considere un array `v` de números enteros de tamaño `n`. Se desea dividir el array en dos secciones: la primera contendrá a todos los elementos menores o iguales al primero (`v[0]`) y la otra, los mayores. Para ello, se propone el siguiente algoritmo:

1. Colocamos un puntero al principio del array y lo adelantamos mientras el elemento apuntado sea menor o igual que el primero (`v[0]`).
2. Colocamos un puntero al final del array y lo atrasamos mientras el elemento apuntado sea mayor que el primero (`v[0]`).
3. Si los punteros no se han cruzado, es que se han encontrado dos elementos mal colocados. Se intercambian los valores apuntados y se vuelve a empezar.

El algoritmo termina cuando los dos punteros se crucen, habiendo quedado todos los elementos ordenados según el criterio inicial.

Implemente un programa que genere un array aleatorio `v` de tamaño `N = 20` con valores entre [-10, 10]. Posteriormente asigne `v[0] = 0`, muestre el array y reorganícelo usando el algoritmo antes descrito. Muestre el array para comprobar el resultado. A continuación puede ver varios ejemplos donde se indica el array de entrada (E) y la correspondiente salida (S).

E: 0, -5, -9, 1, 7, 6, 8, -1, -8, 7 S: 0, -5, -9, -8, -1, 6, 8, 7, 1, 7	E: 0, 7, 8, -1, 7, 5, 8, -10, 5, 4 S: 0, -10, -1, 8, 7, 5, 8, 7, 5, 4
E: 0, 0, 7, 3, -2, -1, 5, -1, -2, -7 S: 0, 0, -7, -2, -2, -1, -1, 5, 3, 7	E: 0, -1, 6, -4, 3, -10, 7, -9, 9, 3 S: 0, -1, -9, -4, -10, 3, 7, 6, 9, 3

El siguiente ejemplo muestra los intercambios realizados

```
0, 7, -1, 0, 8, 4, -7, -9, 0, -3,

Intercambio: 7 con -3
0, -3, -1, 0, 8, 4, -7, -9, 0, 7,

Intercambio: 8 con 0
0, -3, -1, 0, 0, 4, -7, -9, 8, 7,

Intercambio: 4 con -9
0, -3, -1, 0, 0, -9, -7, 4, 8, 7,
```

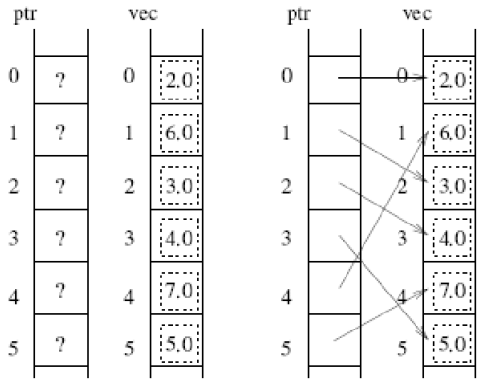

Figura 1: Estado de los arrays antes y después de la ordenación

6) Escriba una función que reciba como entrada un array de números enteros junto con su longitud y que devuelva un array de punteros a los elementos del array de entrada de forma que los elementos apuntados estén ordenados (véase la Figura 1). La ordenación debe hacerse con el método de la burbuja.

Note que el array de punteros debe ser un parámetro de la función. Una vez escrita la función, considere las siguientes declaraciones:

```
const int MAX = 12;
int vec[MAX];
int *ptr[MAX];
```

y escriba un trozo de código que, haciendo uso de la función, permita:

1. Ordenando punteros, mostrar los elementos del vector, ordenados.
2. ¿Sería posible ordenar solamente la segunda mitad del array? ¿Cómo lo haría?

```
#include <iostream>
#include <cstdlib>

using namespace std;

void mostrar(int *v, int k){
  for(int i = 0; i < k; i++)
     cout << v[i] << ", ";
  cout << endl;
}

void rellenar(int *v, int k, int min, int max){
  for(int i = 0; i < k; i++)
    v[i] = (random()% (max - min)) + min;
}

int main()
{ const int N = 10;
  int datos[N];

  // para rellenar el array con valores al azar entre [min, max]
  // utiliza las dos instrucciones siguientes

  //srand(time(0));
  //rellenar(datos, N, min, max);

  int signo = 1;
  for(int i = 0; i < N; i++){
    datos[i] = i * signo;
    signo = signo * -1;
   }

  mostrar(datos, N);

  int *p = &datos[2];
  mostrar(p, 5);

  p = &datos[4];
  mostrar(p, 5);

  int *p1, *p2;
  p1 = &datos[3];
  p2 = &datos[8];
  cout << "Salida 1: " << p2 - p1 << endl;

  p1 = &datos[0];
  cout << "Salida 2: " << p2 - p1 << endl;

  p1 = datos;
  cout << "Salida 3: " << p2 - p1 << endl;

  p1 = &datos[5];
  cout << "Salida 4: " << *p2 << " - " << *p1
       << " = " << *p2 - *p1 << endl;
}
```

Figura 2: Describa la salida del código