# PROGRAMACIÓN

# HOJA GUÍA

# PRÁCTICA: Punteros

## Problemas planteados

1. Sin ejecutarlo, ¿qué mostraría el siguiente código?

int x = 5, y = 12, z;

int \*p1, \*p2, \*p3;

p1 = &x;

p2 = &y;

z = \*p1 \* \*p2;

p3 = &z;

(\*p3)++;

p1 = p3;

cout << \*p1 << " " << \*p2<< " "<< \*p3;

Respuesta:

61 12 61

1. Dada una lista de enteros desordenada, queremos ordenarla. Escribe los respectivos subprogramas de ordenación empleando cualquiera de los algoritmos vistos en clase para:
2. Una lista implementada como array de datos dinámicos.
3. Una lista implementada como array dinámico.

Respuesta:

a)

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

const int Max = 100;

typedef struct{

int \*elementos[Max];

int contador;

}tLista;

void cargar(tLista& lista, bool& ok);

void Mostrar(const tLista& lista);

void ordenarBurbuja(tLista& lista);

void destruir(tLista &lista);

int main()

{

tLista lista;

bool ok;

cargar(lista, ok);

if (!ok) {

cout << "Error: no hay archivo o demasiados datos"

<< endl;

}

else {

    Mostrar(lista);

    ordenarBurbuja(lista);

destruir(lista);

}

}

void ordenarBurbuja(tLista& lista) {

  int i = 0;

  bool inter = true;

  int \*temp;

  if (lista.contador == 0) {

    cout << endl << "ERROR! No existen productos en el registro" << endl << endl;

  }

  else {

    while ((i < lista.contador - 1) && inter) {

      inter = false;

      for (int j = lista.contador - 1; j > i; j--) {

        if (lista.elementos[j] < lista.elementos[j - 1]) {

          temp = lista.elementos[j];

          lista.elementos[j] = lista.elementos[j - 1];

          lista.elementos[j - 1] = temp;

          inter = true;

        }

      }

      if (inter) {

        i++;

      }

    }

    cout << endl << ">> Se ha ordenado el registro! " << endl << endl;

    Mostrar(lista);

  }

}

void cargar(tLista& lista, bool& ok) {

ifstream archivo;

int dato;

bool abierto = true, overflow = false;

lista.contador = 0;

archivo.open("lista.txt");

if (!archivo.is\_open()) {

abierto = false;

}

else {

archivo >> dato;

while ((dato >= 0) && !overflow) {

if (lista.contador == Max) {

overflow = true;

}

else {

lista.elementos[lista.contador] = new int(dato);

lista.contador++;

archivo >> dato;

}

}

archivo.close();

}

ok = abierto && !overflow;

}

void Mostrar(const tLista& lista)

{

  cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

  cout << endl << setw(45) << "L I S T A" << endl << endl;

  cout << " LISTA DE NUMEROS ENTEROS: ";

  for (int i = 0; i < lista.contador; i++)

{

    cout << "[" << \*lista.elementos[i] << "]";

  }

  cout << endl << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

  cout << endl;

}

void destruir(tLista &lista)

{

for(int i = 0; i < lista.contador; i++){

delete lista.elementos[i];

}

lista.contador = 0;

}

b)

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

const int Max = 100;

typedef struct{

int \*elementos;

int contador;

}tLista;

void cargar(tLista& lista, bool& ok);

void Mostrar(const tLista& lista);

void ordenarBurbuja(tLista& lista);

void destruir(tLista &lista);

int main()

{

tLista lista;

bool ok;

cargar(lista, ok);

if (!ok) {

cout << "Error: no hay archivo o demasiados datos"

<< endl;

}

else {

    Mostrar(lista);

    ordenarBurbuja(lista);

destruir(lista);

}

}

void ordenarBurbuja(tLista& lista) {

  int i = 0;

  bool inter = true;

  int temp;

  if (lista.contador == 0) {

    cout << endl << "ERROR! No existen productos en el registro" << endl << endl;

  }

  else {

    while ((i < lista.contador - 1) && inter) {

      inter = false;

      for (int j = lista.contador - 1; j > i; j--) {

        if (lista.elementos[j] < lista.elementos[j - 1]) {

          temp = lista.elementos[j];

          lista.elementos[j] = lista.elementos[j - 1];

          lista.elementos[j - 1] = temp;

          inter = true;

        }

      }

      if (inter) {

        i++;

      }

    }

    cout << endl << ">> Se ha ordenado el registro! " << endl << endl;

    Mostrar(lista);

  }

}

void cargar(tLista& lista, bool& ok) {

ifstream archivo;

int dato;

bool abierto = true, overflow = false;

lista.contador = 0;

archivo.open("lista.txt");

if (!archivo.is\_open()) {

abierto = false;

}

else {

lista.elementos = new int[Max];

archivo >> dato;

while ((dato >= 0) && !overflow) {

if (lista.contador == Max) {

overflow = true; // ¡Demasiados!

}

else {

lista.elementos[lista.contador] = dato;

lista.contador++;

archivo >> dato;

}

}

archivo.close();

}

ok = abierto && !overflow;

}

void Mostrar(const tLista& lista)

{

  cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

  cout << endl << setw(45) << "L I S T A" << endl << endl;

  cout << " LISTA DE NUMEROS ENTEROS: ";

  for (int i = 0; i < lista.contador; i++)

{

    cout << "[" << lista.elementos[i] << "]";

  }

  cout << endl << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

  cout << endl;

}

void destruir(tLista &lista)

{

delete [] lista.elementos;

lista.contador =0;

}

1. Los siguientes fragmentos de código emplean memoria dinámica, pero su funcionamiento no es evidente. Indica, para cada fragmento, cuál es el resultado de la ejecución y si el código tiene algún problema o defecto. Indica también en qué zona de la memoria se guarda cada uno de los datos.

Imagen que contiene texto

Descripción generada con confianza muy alta

Respuesta:

1. 32

100

Montón (heap): \*p

1. 42

¡Error!:

delete q - Dato dinámico destruido.

Montón (heap):

\*p, \*q

1. ¡Error!:

qq = pp - Asigna una dirección de memoria a un dato entero.

q = \*qq - qq no es un puntero.

p = q - Asigna un dato entero a un puntero que recibe una dirección de memoria.

cout << \*\*qq acceso de memoria inexistente.

Monton (heap):

\*p, \*\*pp

Pila (Stack): q, qq

1. ¿Cómo declararías un puntero constante p para apuntar a una constante entera? (Repasa las diapositivas 31-32 de la presentación del tema, y el ejercicio de repl.it 4.27 Punteros y constantes).

Respuesta:

Puntero Constante apuntando a una constante entera

Ejemplo:

const int i = 3;

const int \*const p = &i;

cout << \*p;

Ni el puntero ni la constante pueden ser modificados.

1. Como podemos tener punteros que apunten a cualquier tipo de datos, también podemos tener punteros que apunten a punteros:

int x = 5;

int \*p = &x; // Puntero a entero

int \*\*pp = &p; // Puntero a puntero a entero

Para acceder a x a través de p escribimos \*p.

Para acceder a x a través de pp escribimos \*\*pp, o \*(\*pp).

Con \*pp accedemos a p, el otro puntero.

Indica qué es lo que muestra el siguiente código:

* 1. int x = 5, y = 8;
  2. int \*px = &x, \*py = &y, \*p;
  3. int \*\*ppx = &px, \*\*ppy = &py, \*\*pp;
  4. p = px; px = py;
  5. py = p;
  6. pp = ppx;
  7. ppx = ppy;
  8. ppy = pp;
  9. cout << \*\*ppx << "  " << \*\*ppy;

Respuesta:

5 8

También, dibuja los distintos datos y cómo van apuntando los punteros a los otros datos a medida que se ejecutan las líneas de instrucciones.

Respuesta:

5

px x

8

py y

5

ppx px x

8

ppy py y

5

p px x

8

px py y

5

py p px x

8

pp ppx px py y

5

ppx ppy py p px x

8

ppy pp ppx px py y

## Informe

* **En este mismo documento se deberá resolver los problemas y subir este archivo al aula virtual, en la respectiva tarea en formato de Word.**