Informe de Laboratorio 02

Tema: Punteros en C++

Nota

Estudiante	Escuela	Asignatura
Reyser Julio Zapata Butron rzapata@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Tecnología de Objetos Semestre: VI Código: 1703240

Laboratorio	Tema	Duración
02	Punteros en C++	02 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - B	26 septiembre 2024	03 octubre 2024

1. Repositorio de Github

- Repositorio de Github donde se encuentra el actual laboratorio
  `https://github.com/ReyserLynnn/tec-obj-lab-c-24b/tree/main/laboratorio02/src`
- Repositorio de Github donde se encuentran los laboratorios del curso
  `https://github.com/ReyserLynnn/tec-obj-lab-c-24b.git`

2. Ejercicios

En los siguientes ejercicios, se presentará código, captura de ejecución y una explicación en general por cada ejercicio.

2.1. Ejercicio 1

- Implementar una calculadora con 3 clases en el lenguaje c++, donde la primera analizará la operación matemática (suma, resta....), la segunda administrará las operaciones matemáticas (el núcleo de la calculadora), y la tercera procesara la operación ingresada. El programa recibirá de entrada una cadena de texto con la operación a realizar (“10+37”) (“45+14-42”) (“1+2+3+4+5+6”). Como máximo el programa recibe 6 números a operar.

```cpp
#include <iostream>
#include <string>

class Operacion
{
public:
  static char *analizarOperaciones(const std::string &operacion, int &countOperadores)
  {
    char *operadores = new char[5];
    countOperadores = 0;

    for (char c : operacion)
    {
      if (c == '+' || c == '-' || c == '*' || c == '/')
      {
        operadores[countOperadores++] = c;
      }
    }
    return operadores;
  }
};

class NucleoCalculadora
{
public:
  static int realizarOperacion(int a, int b, char operador)
  {
    switch (operador)
    {
    case '+':
      return a + b;
    case '-':
      return a - b;
    case '*':
      return a * b;
    case '/':
      return (b != 0) ? a / b : 0;
    default:
      return 0;
    }
  };
};

class Procesador
{
private:
  int *numeros;
  char *operadores;
  int countNumeros;
  int countOperadores;

public:
  Procesador(const std::string &operacion) : countNumeros(0), countOperadores(0)
  {
    numeros = new int[6];
    operadores = Operacion::analizarOperaciones(operacion, countOperadores);
    analizarOperacion(operacion);
  }

  void analizarOperacion(const std::string &operacion)
  {
    int num = 0;
    bool isNum = false;

```

```
65      for (int i = 0; i < operacion.size(); i++)
66      {
67        char c = operacion[i];
68
69        if (isdigit(c))
70        {
71          num = num * 10 + (c - '0');
72          isNum = true;
73        }
74        else if (c == '+' || c == '-' || c == '*' || c == '/')
75        {
76          if (isNum)
77          {
78            if (countNumeros < 6)
79            {
80
81              numeros[countNumeros++] = num;
82              num = 0;
83            }
84            isNum = false;
85          }
86        }
87      }
88
89      if (isNum && countNumeros < 6)
90      {
91        numeros[countNumeros++] = num;
92      }
93    }
94
95    int procesar()
96    {
97      int resultado = numeros[0];
98
99      for (int i = 0; i < countOperadores; i++)
100     {
101       resultado = NucleoCalculadora::realizarOperacion(resultado, numeros[i + 1], operadores[i]);
102     }
103
104     return resultado;
105   }
106
107   ~Procesador()
108   {
109     delete[] numeros;
110     delete[] operadores;
111   }
112 };
113
114 int main()
115 {
116
117   std::cout << "\n[+] Ingrese la operacion a realizar: ";
118
119   std::string operacion;
120   std::cin >> operacion;
121
122   Procesador procesador(operacion);
123   std::cout << "\n[+] Resultado: " << procesador.procesar() << std::endl;
124
125   return 0;
126 }
```

Listing 1: ejercicio1.cpp

Ejecución del ejercicio

```
> .\ejercicio1.exe
[+] Ingrese la operacion a realizar: 10+37
[+] Resultado: 47
> .\ejercicio1.exe
[+] Ingrese la operacion a realizar: 45+14-42
[+] Resultado: 17
> .\ejercicio1.exe
[+] Ingrese la operacion a realizar: 1+2+3+4+5+6
[+] Resultado: 21
> .\ejercicio1.exe
[+] Ingrese la operacion a realizar: 45/5*3
[+] Resultado: 27
```

Análisis del código:

En el anterior código, presenta una calculadora básica para realizar operaciones matemáticas con una cadena de entrada que contiene hasta seis números y operadores matemáticos (`+`, `-`, `*`, `/`). La clase `Operacion` define el método `analizarOperaciones`, que recibe una cadena de texto, analiza los operadores y los almacena dinámicamente en un arreglo. La clase `NucleoCalculadora` contiene el método `realizarOperacion`, el cual evalúa la operación matemática entre dos números dependiendo del operador. La clase `Procesador` se encarga de extraer los números de la cadena, almacenarlos en un arreglo dinámico y procesar las operaciones de manera secuencial usando el método `realizarOperacion`. Se utiliza manejo manual de memoria con `new` y `delete` para los arreglos de números y operadores, garantizando que no haya fugas de memoria mediante el destructor de la clase `Procesador`.

2.2. Ejercicio 2

- Implementar con punteros una lista doblemente enlazada, utilizar clases o struct.

```cpp
#include <iostream>

using namespace std;

struct Nodo
{
  int valor;
  Nodo *siguiente;
  Nodo *anterior;
```

```
10
11    Nodo(int val) : valor(val), siguiente(nullptr), anterior(nullptr) {}
12  };
13
14  class ListaDobleEnlazada
15  {
16  private:
17    Nodo *cabeza;
18    Nodo *cola;
19
20  public:
21    ListaDobleEnlazada() : cabeza(nullptr), cola(nullptr) {}
22
23    void insertarFinal(int valor)
24    {
25      Nodo *nuevoNodo = new Nodo(valor);
26      if (cabeza == nullptr)
27      {
28        cabeza = cola = nuevoNodo;
29      }
30      else
31      {
32        cola->siguiente = nuevoNodo;
33        nuevoNodo->anterior = cola;
34        cola = nuevoNodo;
35      }
36    }
37
38    void insertarInicio(int valor)
39    {
40      Nodo *nuevoNodo = new Nodo(valor);
41      if (cabeza == nullptr)
42      {
43        cabeza = cola = nuevoNodo;
44      }
45      else
46      {
47        nuevoNodo->siguiente = cabeza;
48        cabeza->anterior = nuevoNodo;
49        cabeza = nuevoNodo;
50      }
51    }
52
53    void eliminarFinal()
54    {
55      if (cola == nullptr)
56      {
57        cout << "\n[-] La lista esta vacia, no se puede eliminar." << endl;
58        return;
59      }
60
61      if (cabeza == cola)
62      {
63        delete cabeza;
64        cabeza = cola = nullptr;
65      }
66      else
67      {
68        Nodo *nodoAEliminar = cola;
69        cola = cola->anterior;
70        cola->siguiente = nullptr;
71        delete nodoAEliminar;
72      }
73    }
74
```

```
75    void eliminarInicio()
76    {
77      if (cabeza == nullptr)
78      {
79        cout << "\n[-] La lista esta vacia, no se puede eliminar." << endl;
80        return;
81      }
82
83      if (cabeza == cola)
84      {
85        delete cabeza;
86        cabeza = cola = nullptr;
87      }
88      else
89      {
90        Nodo *nodoAEliminar = cabeza;
91        cabeza = cabeza->siguiente;
92        cabeza->anterior = nullptr;
93        delete nodoAEliminar;
94      }
95    }
96
97    void mostrarDesdeInicio() const
98    {
99      Nodo *actual = cabeza;
100     while (actual != nullptr)
101     {
102       cout << actual->valor << " ";
103       actual = actual->siguiente;
104     }
105     cout << endl;
106   }
107
108   void mostrarDesdeFinal() const
109   {
110     Nodo *actual = cola;
111     while (actual != nullptr)
112     {
113       cout << actual->valor << " ";
114       actual = actual->anterior;
115     }
116     cout << endl;
117   }
118
119   ~ListaDobleEnlazada()
120   {
121     while (cabeza != nullptr)
122     {
123       Nodo *temp = cabeza;
124       cabeza = cabeza->siguiente;
125       delete temp;
126     }
127   }
128 };
129
130 int main()
131 {
132   ListaDobleEnlazada lista;
133
134   lista.insertarFinal(10);
135   lista.insertarFinal(20);
136   lista.insertarInicio(5);
137   lista.insertarInicio(1);
138
139   cout << "\n[+] Lista desde el inicio: ";
```

```
140     lista.mostrarDesdeInicio();
141
142     cout << "[+] Lista desde el final: ";
143     lista.mostrarDesdeFinal();
144
145     lista.eliminarInicio();
146     cout << "[+] Lista despues de eliminar el inicio: ";
147     lista.mostrarDesdeInicio();
148
149     lista.eliminarFinal();
150     cout << "[+] Lista despues de eliminar el final: ";
151     lista.mostrarDesdeInicio();
152
153     printf("\n");
154     return 0;
155 }
```

Listing 2: ejercicio2.cpp

Ejecución del ejercicio

Se realizaron las siguientes insercciones:

- lista.insertarFinal(10);
- lista.insertarFinal(20);
- lista.insertarInicio(5);
- lista.insertarInicio(1);

```
> .\ejercicio2.exe

[+] Lista desde el inicio: 1 5 10 20
[+] Lista desde el final: 20 10 5 1
[+] Lista despues de eliminar el inicio: 5 10 20
[+] Lista despues de eliminar el final: 5 10
```

Análisis del código:

En el anterior código, se implementó una lista doblemente enlazada mediante la estructura `Nodo` y la clase `ListaDobleEnlazada`. Cada `Nodo` contiene un valor entero y punteros a los nodos adyacentes (`siguiente` y `anterior`). La clase `ListaDobleEnlazada` maneja dos punteros principales: `cabeza` y `cola`, que apuntan al inicio y final de la lista, respectivamente. Los métodos `insertarFinal` e `insertarInicio` permiten agregar nodos al final o al inicio de la lista. Las operaciones `eliminarFinal` y `eliminarInicio` eliminan los nodos correspondientes, ajustando los punteros de manera adecuada. La lista puede ser recorrida en ambas direcciones con los métodos `mostrarDesdeInicio` y `mostrarDesdeFinal`. Finalmente, el destructor libera la memoria de todos los nodos para evitar fugas.

3. Cuestionario

1. **¿Qué es la memoria dinámica y para que nos sirve?**

   La memoria dinámica es un tipo de memoria que se asigna y libera en tiempo de ejecución, en lugar de hacerlo en tiempo de compilación. Esto nos permite reservar espacio en el heap según sea necesario, especialmente útil cuando no sabemos de antemano cuánta memoria necesitaremos, como en el caso de estructuras de datos que pueden cambiar de tamaño (listas, árboles, etc.). Sirve para manejar grandes cantidades de datos de manera flexible y eficiente.

2. **¿Cuáles son las principales recomendaciones para el uso de la memoria dinámica?**

   Es importante asegurarse de liberar la memoria asignada dinámicamente una vez que ya no se necesite, utilizando la palabra clave `delete` o `delete[]` en C++. Esto evita fugas de memoria, que pueden causar que un programa consuma más memoria de la necesaria. También se recomienda inicializar punteros a `nullptr` después de liberarlos para evitar el acceso accidental a direcciones de memoria inválidas.

3. **¿Qué sentencias son usadas en C++ para el manejo de memoria dinámica?. ¿Cómo se utilizan?**

   En C++, se usa la sentencia `new` para asignar memoria dinámica y `delete` para liberarla. Por ejemplo, `int *p = new int;` asigna un entero en el heap y devuelve un puntero a esa memoria. Para liberar esa memoria, se usa `delete p;`. Si es un arreglo dinámico, se utiliza `delete[] p;`. Es importante usar esto para evitar fugas de memoria.