



Universidad Nacional Abierta
Vicerrectorado Académico
Área de Ingeniería
Carrera Ingeniería de Sistemas

Trabajo práctico sustitutivo (Corrección)

Asignatura: Computación II Código: 324

Fecha de devolución: A más tardar el 13/03/2021 (Sin prórroga)

Nombre del Estudiante: Glemand A Pineda C

Cédula de Identidad: 9347512

Centro Local / Unidad de Apoyo: Táchira

Correo electrónico: glempineda@gmail.com

Teléfono celular: 04169115964

Carrera: Ing Sistemas 326

Número de originales: 1

Lapso: 2021-1

Resultados de Corrección

	Objetivos			
	1	2	3	4
Logrado: 1				
No logrado: 0				

M: 1, U: 1, O: 1 C/D: 1/1

1. Implemente en Lenguaje C++ una función que permita obtener la parte izquierda de una cadena de caracteres definida por el número de caracteres que la forman.

Para la implementación de este programa se va a seguir la siguiente estrategia de algoritmo:

1. Se introduce el texto en el arreglo.
2. Se determina el tamaño del texto en el arreglo.
3. Se imprime el arreglo de carácter basado en su tamaño.

COMENTARIO DE CORRECCION: Efectivamente estoy de acuerdo con que los array utilizan memoria que no se va a utilizar ya que es necesario definir el tamaño previamente. Corregí en main la definicion del arreglo con un array builtin (asi lo llaman) lo que no logre con el tiempo en insertar la cadena antes y después definir el arreglo basado en la cadena.

A continuación se anexa el código en lenguaje C++.

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

//Determina el tamaño de la cadena
int getLength(char* str){
    int s=0;
    for(int i=0;str[i];i++){
        s++;
    }
    return s;
}
```

```

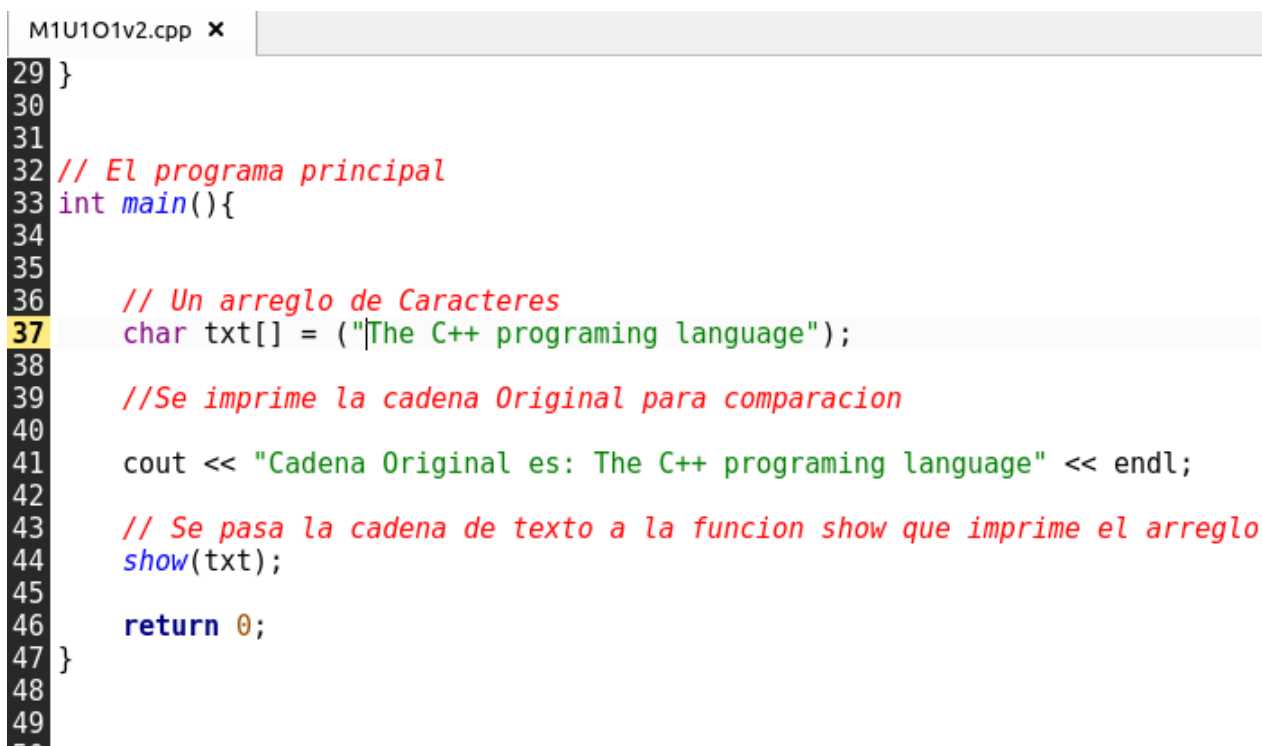
//Imprime la cadena
void show(char* str){

    //se calcula el tamaño de la cadena de caracteres dentro del arreglo.
    int n=getLength(str);

    for(int k=0;k<=(n-1);k++){
        cout<<str[k]<<"-";
    }

    cout<<endl;
}

```



The screenshot shows a code editor window titled 'M1U1O1v2.cpp'. The code is as follows:

```

29 }
30
31
32 // El programa principal
33 int main(){
34
35
36     // Un arreglo de Caracteres
37     char txt[] = ("The C++ programing language");
38
39     //Se imprime la cadena Original para comparacion
40
41     cout << "Cadena Original es: The C++ programing language" << endl;
42
43     // Se pasa la cadena de texto a la funcion show que imprime el arreglo
44     show(txt);
45
46     return 0;
47 }
48
49
50

```

Les anexo el programa el código y fue compilado en Linux G++ cpu atom.

M1U1O1.cpp

M1U1O1.b

La salida del programa dentro de main ya introduzco una cadena de texto y se imprime solo la cadena de texto y no los 100 campos del array, para eso agregue los símbolos de -.

```
gp@gp-pc:~/programs$  
gp@gp-pc:~/programs$  
gp@gp-pc:~/programs$  
gp@gp-pc:~/programs$  
gp@gp-pc:~/programs$ ./M1U101.b  
Cadena Original es: The C++ programing language  
T-h-e- -C-+-+ -p-r-o-g-r-a-m-i-n-g- -l-a-n-g-u-a-g-e-  
gp@gp-pc:~/programs$  
gp@gp-pc:~/programs$  
gp@gp-pc:~/programs$ █
```

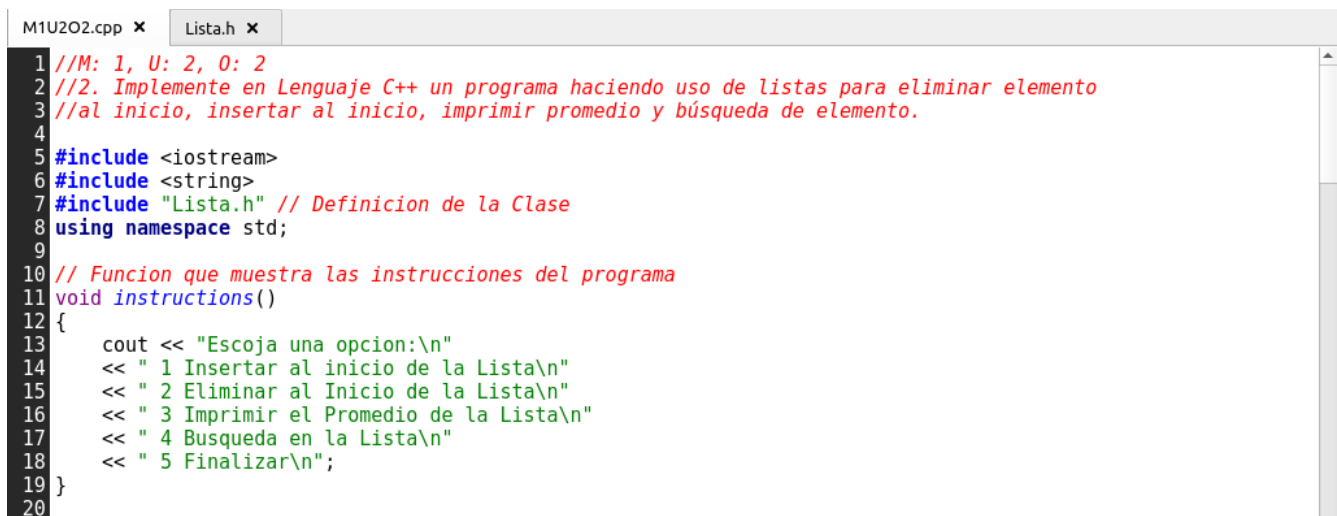
M: 1, U: 2, O: 2 C/D: 1/1

2. Implemente en Lenguaje C++ un programa haciendo uso de listas para eliminar elemento al inicio, insertar al inicio, imprimir promedio y búsqueda de elemento.

Para la implementación del programa se utilizaron dos archivos :

1. el M1U2O2.cpp que contiene el menú de programa donde se muestran las opciones disponibles para el usuario y el llamado de la clase.
2. El Lista.h archivo de cabecera donde están definidos los procesos de la clase como Insertar Nodo, eliminar Nodo , promedio de Lista y búsqueda de Valor en la Lista.

COMENTARIO DE CORRECCIÓN: Profesor efectivamente coloque un chequeo para validar la selección del menú , pero no he logrado la manera de robustecer la entrada de datos de la pila para que sea solo enteros. Cuando introduzco letras falla y tengo entendido que deberían convertirse a valores enteros.



```
1 //M: 1, U: 2, O: 2
2 //2. Implemente en Lenguaje C++ un programa haciendo uso de listas para eliminar elemento
3 //al inicio, insertar al inicio, imprimir promedio y búsqueda de elemento.
4
5 #include <iostream>
6 #include <string>
7 #include "Lista.h" // Definicion de la Clase
8 using namespace std;
9
10 // Funcion que muestra las instrucciones del programa
11 void instructions()
12 {
13     cout << "Escoja una opcion:\n"
14     << " 1 Insertar al inicio de la Lista\n"
15     << " 2 Eliminar al Inicio de la Lista\n"
16     << " 3 Imprimir el Promedio de la Lista\n"
17     << " 4 Busqueda en la Lista\n"
18     << " 5 Finalizar\n";
19 }
20
```

En el Archivo M1U2O2.cpp se crea una función de Instrucciones del menú. Después se crea la función de prueba de la lista. Donde esta un switch con los casos donde se valida la selección del usuario.

```
/home/gp/Documents/2021-1/324/TPS324/M1U2O2.cpp LibreOffice Writer
Archivo Editar Opciones Buscar Ayuda
M1U2O2.cpp x
21
22 // Funcion para Probar la Lista
23 void testList( List &listObject , const string &typeName )
24 {
25     int choice=0; // Almacena escogencia
26     int value=0; // Valor de objeto de la Lista
27
28     cout << "Prueba de la Lista de Valores " << typeName << "\n";
29
30
31     do // Realiza el Menu de Opciones
32     {
33         bool val;
34
35         //Validacion de la entrada del usuario
36         do {
37
38             instructions(); // Se muestra el menu
39             cout << "? ";
40             cin >> choice;|
41
42             if ((choice < 1) || (choice > 5))
43                 val=false;
44             else
```

Buscar...

Codificación: UTF-8 Sintaxis: cpp Líneas: 114 Caracteres Seleccionados: 0 Palabras: C

```
/home/gp/Documents/2021-1/324/TPS324/M1U2O2.cpp LibreOffice Writer
Archivo Editar Opciones Buscar Ayuda
M1U2O2.cpp x
44     else
45         val=true;
46
47     }while (val == false);
48
49
50     switch ( choice )
51     {
52     case 1:
53         cout << "Introducir " << typeName << ": ";
54         cin >> value;
55         listObject.insertAtFront( value );
56         listObject.print();
57         break;
58     case 2:
59         if ( listObject.removeFromFront( value ) )
60             cout << value << " Removido de la Lista\n";
61         listObject.print();
62
63         break;
64     case 3:
65         cout << "El promedio de los Valores es: ";
66         cout << listObject.printMedia()<<endl;
67         listObject.print();
68
69     }
70
71     } while ( choice < 5 );
72
73     cout << "Fin del programa de Prueba\n\n";
74
75 }
```

```
/home/gp/Documents/2021-1/324/TPS324/M1U2O2.cpp LibreOffice Writer
Archivo Editar Opciones Buscar Ayuda
M1U2O2.cpp x
64     case 3:
65         cout << "El promedio de los Valores es: ";
66         cout << listObject.printMedia()<<endl;
67         listObject.print();
68
69         break;
70
71     case 4:
72         cout << "Introducir Valor Buscado: ";
73         cin >> value;
74         listObject.finding( value );
75         listObject.print();
76         break;
77
78     } // fin switch
79
80     } while ( choice < 5 );
81
82     cout << "Fin del programa de Prueba\n\n";
83
84 }
85
86 //Llamado del programa principal
87
```

Codificación: UTF-8 Sintaxis: cpp Líneas: 114 Caracteres Seleccionados: 0 Palabras: 0

Y finalmente el programa Main

/home/gp/programs/M1U2O2.cppnd Pineda.odt - LibreOffice Writer

Archivo Editar Opciones Buscar Ayuda

M1U2O2.cpp x Lista.h x

```
71 }
72 }
73
74
75 //Llamado del programa principal
76 int main()
77 {
78     //Llamado de la funcion de prueba de la Lista y Creacion de la Lista
79     List integerList;
80     testList( integerList, "entero" );
81 }
82
83
84
85
86
87
88
89
90
91
92
93
94
```

Buscar...

Codificación: UTF-8 Sintaxis: cpp Líneas: 102 Caracteres Seleccionados: 0 Palabras: C

En el archivo lista.h se disponen de los siguientes procesos y clases. Los procesos tienen la función de Insertar Nodo, Eliminar Nodo, imprimir el promedio y la búsqueda de un valor. A continuación nombraremos los procesos dentro del archivo de cabecera.

- Default Constructor
- Destructor
- void insertAtFront(const int &value) > Realiza la Inserción del Nuevo Nodo
- bool removeFromFront(int &value) > Realiza el borrado del Nuevo Nodo
- bool isEmpty() const > Verifica que este vacía la lista.
- void print() const > Imprime la Lista
- double printMedia() > Imprime el promedio de Valores de la lista
- void finding(const int &value) > Realiza la búsqueda de los valores y muestra su posición de memoria.

```
//Lista.h

#ifndef LISTA_H
#define LISTA_H

#include <iostream>

class ListNode
{
public:
    explicit ListNode( const int &info ) // constructor
    : data( info ), nextPtr( nullptr )
    {
        // Cuerpo Vacio
    }

    int getData() const // retorna la data
    {
        return data;
    }

    int data; // datos
    ListNode *nextPtr; // proximo nodo
};

class List
{
public:
```

```

// default constructor
List()
: firstPtr( nullptr ), lastPtr( nullptr )
{
    // Vacio
}

// destructor
~List()
{
    if ( !isEmpty() ) //La lista no esta Vacia
    {
        std::cout << "Destruyendo los Nodos ...\n";

        ListNode *currentPtr = firstPtr;
        ListNode *tempPtr = nullptr;

        while ( currentPtr != nullptr ) // Borra los nodos restantes
        {
            tempPtr = currentPtr;
            std::cout << tempPtr->data << " Borrado\n";
            currentPtr = currentPtr->nextPtr;
            delete tempPtr;
        }

        std::cout << "Todos los nodos Fueron destruidos\n\n";
    }

// Inserta el Nodo al Inicio
void insertAtFront( const int &value )
{
    ListNode *newPtr = getNewNode( value );

    if ( isEmpty() ) // Si la lista esta vacia
        firstPtr = lastPtr = newPtr; // Solo hay un nodo
    else // La lista no esta Vacia
    {
        newPtr->nextPtr = firstPtr; // Apunta el nuevo nodo al 1er Antiguo
        firstPtr = newPtr; // Asigna el nuevo nodo como primero
    }
}

// Borra el nodo de la lista
bool removeFromFront( int &value )
{

```

```

        if ( isEmpty() ) // Lista es Vacía
            return false;
        else
        {
            ListNode *tempPtr = firstPtr; // Item a borrar
            if ( firstPtr == lastPtr )
                firstPtr = lastPtr = nullptr; // No hay mas Nodos
            else
            {
                firstPtr = firstPtr->nextPtr; // Apunta al segundo nodo
                value = tempPtr->data; // Recupero data a ser removida
                delete tempPtr; // Elimina el nodo
                return true; // delete successful
            }
        }
    }
}

```

```

// Lista Vacía?
bool isEmpty() const
{
    return firstPtr == nullptr;
}

```

```

// Muestra el contenido de la Lista
void print() const
{
    if ( isEmpty() ) // Lista Vacía?
    {
        std::cout << "The list is empty\n\n";
        return;
    }

    ListNode *currentPtr = firstPtr;
    std::cout << "La Lista es: ";

    while ( currentPtr != nullptr ) // Obtiene la Data
    {
        std::cout << currentPtr->data << ' ';
        currentPtr = currentPtr->nextPtr;
    }

    std::cout << "\n\n";
}

```

```

// Imprime la media de la pila
double printMedia( )
{

```

```

        if ( isEmpty() ) // Lista Vacia
        {
            std::cout << "La lista esta vacia\n\n";
        }

        ListNode *currentPtr = firstPtr;
        int n=0 , suma=0;

        while ( currentPtr != nullptr ) // Obtiene la data
        {
            suma+= currentPtr->data ;
            n+=1;
            currentPtr = currentPtr->nextPtr;
        }

        return (double) suma/n;
    }

    // Busqueda de un Nodo
    void finding( const int &value )
    {
        if ( isEmpty() ) // Si la lista esta vacia
            std::cout << "La lista esta vacia\n\n";
        else // La lista no esta Vacia
        {
            ListNode *currentPtr = firstPtr;
            int valorBuscado;

            while ( currentPtr != nullptr ) // Obtiene la data
            {
                valorBuscado= currentPtr->data;

                if (value == valorBuscado)
                    std::cout << "El Valor " << value << " Esta ubicado en "
<< currentPtr << std::endl;

                currentPtr = currentPtr->nextPtr;
            }
        }
    }

private:
    ListNode *firstPtr; // Apuntador al primer nodo

```

```
        ListNode *lastPtr; // Apuntador al ultimo nodo

        // Funcion para agregar un nodo nuevo
        ListNode *getNewNode( const int &value )
        {
            return new ListNode( value );
        }
    };

#endif
```

Les anexo el programa el código y fue compilado en Linux G++ cpu atom.

La salida del programa tenemos

```
gp@gp-pc:~/programs$ ./M1U2O2.b
Prueba de la Lista de Valores entero
Escoja una opcion:
 1 Insertar al inicio de la Lista
 2 Eliminar al Inicio de la Lista
 3 Imprimir el Promedio de la Lista
 4 Busqueda en la Lista
 5 Finalizar
? 1
Introducir entero: 34
La Lista es: 34

? 1
Introducir entero: 56
La Lista es: 56 34

? 1
Introducir entero: 89
La Lista es: 89 56 34

? 3
El promedio de los Valores es: 59.6667
La Lista es: 89 56 34

? 2
89 Removido de la Lista
La Lista es: 56 34

? 4
Introducir Valor Buscado: 34
```

El Valor 34 Esta ubicado en 0x5622b0f046d0

La Lista es: 56 34

? 5

Fin del programa de Prueba

Destruyendo los Nodos ...

56 Borrado

34 Borrado

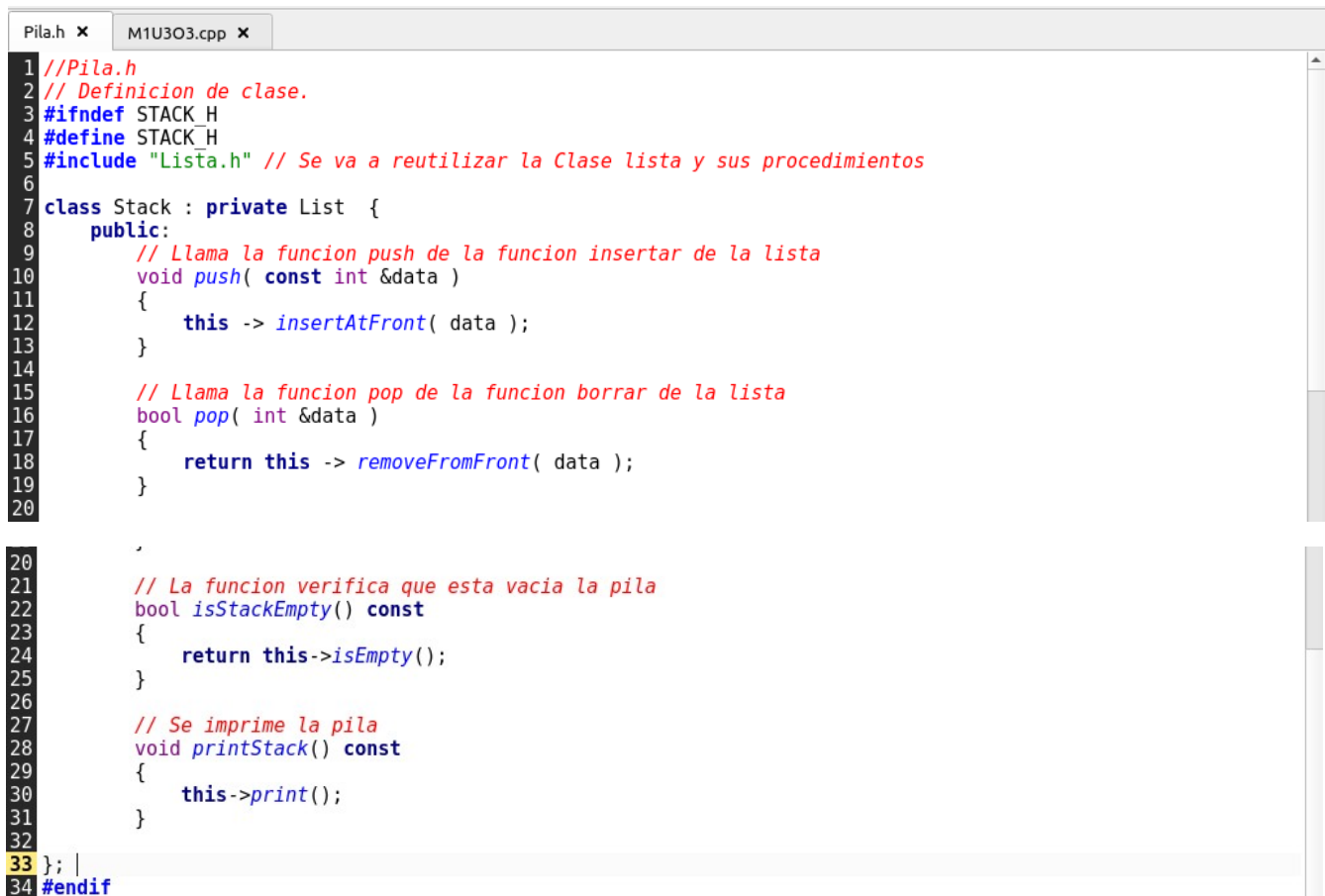
Todos los nodos Fueron destruidos

M: 2, U: 3, O: 3 C/D: 1/1

3. Realice en Lenguaje C++ un programa para la implementación de las secuencias con dos pilas.

Para este programa vamos a reutilizar los procesos del programa anterior lista.h en donde utilizamos las funciones de insertar, borrar, imprimir y verificar vacío.

En el archivo pila.h se crea las funciones push , pop , isEmpty y printStack.



```
1 //Pila.h
2 // Definicion de clase.
3 #ifndef STACK_H
4 #define STACK_H
5 #include "Lista.h" // Se va a reutilizar la Clase lista y sus procedimientos
6
7 class Stack : private List {
8     public:
9         // Llama la funcion push de la funcion insertar de la lista
10        void push( const int &data )
11        {
12            this -> insertAtFront( data );
13        }
14
15        // Llama la funcion pop de la funcion borrar de la lista
16        bool pop( int &data )
17        {
18            return this -> removeFromFront( data );
19        }
20
21        // La funcion verifica que esta vacia la pila
22        bool isEmpty() const
23        {
24            return this->isEmpty();
25        }
26
27        // Se imprime la pila
28        void printStack() const
29        {
30            this->print();
31        }
32    };
33 #endif
```

Push: Inserta data en el tope de la pila.

Pop: Elimina data en el tope de la pila.

isEmpty: Verifica que la pila este vacía.

PrintStack: Imprime los datos de la Pila.

En el Archivo de prueba M1U3O3.cpp creamos dos pilas y las procedemos a llenar con datos y después se vacía para verificar el uso de la secuencia en las dos pilas

```
//M: 2, U: 3, O: 3
//3. Realice en Lenguaje C++ un programa para la implementación
//de las secuencias con dos pilas.

#include <iostream>
#include "Pila.h" // Definicion de la clase Pila
using namespace std;

int main()
{
    Stack intStack; // Crea la primera pila
    cout << "Se procesa la Pila de Entero" << endl;

    // Se introduce datos a la Pila
    for ( int i = 0; i < 3; ++i )
    {
        intStack.push( i );
        intStack.printStack();
    }

    int popInteger; // Se almacena el valor sacado de la pila

    // Se saca los enteros de la pila
    while ( !intStack.isStackEmpty() )
    {
        intStack.pop( popInteger );
        cout << popInteger << " Sacado de la Pila" << endl;
        intStack.printStack();
    }

    Stack int2Stack; // Se crea la segunda pila de enteros
    cout << "*****" << endl;
    cout << "Se procesa la Segunda Pila de Entero" << endl;

    // Se introduce datos a la Pila
    for ( int i = 30; i < 37; ++i )
    {
        int2Stack.push( i );
        int2Stack.printStack();
    }

    int popInteger2; // Se almacena el valor sacado de la pila
```



```

// Se saca los enteros de la pila
while ( !int2Stack.isEmpty() )
{
    int2Stack.pop( popInteger2 );
    cout << popInteger2 << " Sacado de la Pila" << endl;
    int2Stack.printStack();
}
}

```

Les anexo el programa el código y fue compilado en Linux G++ cpu atom. Se anexa la salida del programa donde se puede ver el llenado y vaciado de la pila.

```

gp@gp-pc:~/programs$ ./M1U3O3.b
Se procesa la Pila de Entero
La Lista es: 0

La Lista es: 1 0

La Lista es: 2 1 0

2 Sacado de la Pila
La Lista es: 1 0

1 Sacado de la Pila
La Lista es: 0

0 Sacado de la Pila
The list is empty

*****
Se procesa la Segunda Pila de Entero
La Lista es: 30

La Lista es: 31 30

La Lista es: 32 31 30

La Lista es: 33 32 31 30

La Lista es: 34 33 32 31 30

La Lista es: 35 34 33 32 31 30

La Lista es: 36 35 34 33 32 31 30

36 Sacado de la Pila

```

La Lista es: 35 34 33 32 31 30

35 Sacado de la Pila

La Lista es: 34 33 32 31 30

34 Sacado de la Pila

La Lista es: 33 32 31 30

33 Sacado de la Pila

La Lista es: 32 31 30

32 Sacado de la Pila

La Lista es: 31 30

31 Sacado de la Pila

La Lista es: 30

30 Sacado de la Pila

The list is empty

Todos los nodos Fueron destruidos

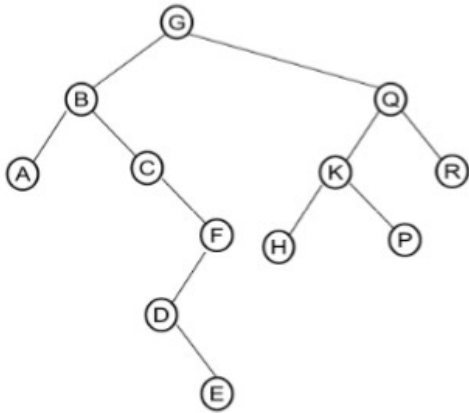
Todos los nodos Fueron destruidos

gp@gp-pc:~/programs\$

M: 2, U: 4, O: 4 C/D: 1/1

4. Dado el siguiente árbol binario, implemente un programa en Lenguaje C++ que encuentre:

- a) La cantidad de nodos del árbol binario: `int ab_numNodos(ArbolBinario *pab)`
- b) El número de hojas del árbol binario: `int ab_numHojas(ArbolBinario *pab)`
- c) La profundidad del árbol binario: `int ab_profundidad(ArbolBinario *pab)`



Para la realización de este programa se utilizaron dos archivos el M1U4O4.cpp y arbol.h. El archivo M1U4O4.cpp contiene el main, la definición de la clase , el llenado del arbol con el insertNode.

El `charTree.preOrderTraversal()` lo que realiza es la impresion del arbol para verificar que este correctamente cargado. Se puede observar las relaciones de memoria.

El `charTree.numNodos()` llama a la cuenta de todos los nodos del arbol.

El `charTree.numHojas()` llama a la cuenta de las hojas independientes

el `charTree.profundidad()` cuenta la profundidad de las ramas.

NOTA DE CORRECCIÓN:

En este programa el recorrido de hace nuevamente no identifico ninguno en particular, a diferencia de los grafos que para SPF si necesito saber el inicio y el final. Solo utilizo el `ptr != nullptr` para saber si esta vacio o para saber su valor. Como es Binario solo se sigue dos lados izquierdo o derecho.

```

// Creando el arbol binario
#include <iostream>
#include <iomanip>
#include "arbol.h" // Definicion de la Clase arbol
using namespace std;

int main()
{
    Tree< char > charTree; // Crea el arbol de caracteres
    cout << "Se introducen los valores en el arbol:\n";

    // insert char to charTree
    char intValue = 'G';
    charTree.insertNode( intValue );
    intValue = 'B';
    charTree.insertNode( intValue );
    intValue = 'A';
    charTree.insertNode( intValue );
    intValue = 'C';
    charTree.insertNode( intValue );
    intValue = 'F';
    charTree.insertNode( intValue );
    intValue = 'D';
    charTree.insertNode( intValue );
    intValue = 'E';
    charTree.insertNode( intValue );
    intValue = 'Q';
    charTree.insertNode( intValue );
    intValue = 'K';
    charTree.insertNode( intValue );
    intValue = 'H';
    charTree.insertNode( intValue );
    intValue = 'P';
    charTree.insertNode( intValue );
    intValue = 'R';
    charTree.insertNode( intValue );

    cout << "\nChequeo del Arbol\n";
    charTree.preOrderTraversal();

    cout << "\nCuenta de los Nodos\n";
    charTree.numNodos();

    cout << "\nCuenta de las Hojas\n";
    charTree.numHojas();
}

```

```

        cout << "\nCuenta de la Profundidad\n";
        charTree.profundidad();

        cout << endl;
    }

```

Se anexa las clase árbol

```

#include <iostream>

// Definicion del Arbol Nodo
template< typename NODETYPE >
class TreeNode
{
    public:
        // constructor
        TreeNode( const NODETYPE &d )
        : leftPtr( nullptr ), // apuntador izquierdo
          data( d ), // data
          rightPtr( nullptr ) // apuntador derecho
        {
            // vacio
        }

        TreeNode< NODETYPE > *leftPtr; // apuntador izquierdo
        NODETYPE data;
        TreeNode< NODETYPE > *rightPtr; // apuntador derecho
};

// Definicion de la clase arbol
template< typename NODETYPE > class Tree
{
    public:
        // constructor
        Tree()
        : rootPtr( nullptr ) { /* empty body */ }

        // Inserta nodos en el arbol
        void insertNode( const NODETYPE &value )
        {
            insertNodeHelper( &rootPtr , value );
        }

```

```

}

// Realiza el chequeo de la data introducida en el arbol
void preOrderTraversal() const
{
    preOrderHelper( rootPtr );
}

// cuenta el numero de nodos
void numNodos() const
{
    std::cout << numNodosHelper( rootPtr );
}

// cuenta el numero de hojas
void numHojas() const
{
    std::cout << (numNodosHelper( rootPtr )-numHojasHelper( rootPtr ));
}

// cuenta la profundidad del arbol
void profundidad() const
{
    std::cout << profundidadHelper( rootPtr );
}

```

private:

```

TreeNode< NODETYPE > *rootPtr;

// Utilizada para recibir un apuntador y modificar la data
void insertNodeHelper(
TreeNode< NODETYPE > **ptr , const NODETYPE &value )
{
    // Si esta vacio crea un nuevo nodo
    if ( *ptr == nullptr )
        *ptr = new TreeNode< NODETYPE >( value );
    else // si no esta vacio
    {
        // Si la data es menor se inserta
        if ( value < ( *ptr )->data )
            insertNodeHelper( &( ( *ptr )->leftPtr ), value );
        else
        {
            //Si la data es mayor se inserta
            if ( value > ( *ptr )->data )
                insertNodeHelper( &( ( *ptr )->rightPtr ), value );
            else // se ignoran los duplicados

```

```

        std::cout << value << " duplicated" << std::endl;
    }
}

// Se imprime el arbol
void preOrderHelper ( TreeNode< NODETYPE > *ptr ) const
{
    if ( ptr != nullptr )
    {
        std::cout << ptr->data << ' ';
        std::cout << ptr << ' ';
        std::cout << ptr->leftPtr << ' ';
        std::cout << ptr->rightPtr << ' ' << std::endl;
        preOrderHelper( ptr->leftPtr ); // atraviesa la izquierda
        preOrderHelper( ptr->rightPtr ); // atraviesa la derecha
    }
}

// cuenta todos los nodos del arbol a medida que pasa por ellos
int numNodosHelper ( TreeNode< NODETYPE > *ptr ) const
{
    int count=0;

    if ( ptr != nullptr )
    {
        count=1;

        count = count + numNodosHelper( ptr->leftPtr );
        count = count + numNodosHelper( ptr->rightPtr );
    }

    return count;
}

// cuenta las hojas del arbol. En este caso fallo la logica de == me toco hacerla con !=
//y restarla del total
int numHojasHelper ( TreeNode< NODETYPE > *ptr ) const
{
    int count=0;

    if ( ptr != nullptr )
    {
        if ( ((ptr->leftPtr) != nullptr)||((ptr->rightPtr) != nullptr)){

```

```

        count=1;
        count = count + numHojasHelper( ptr->leftPtr ); // traverse left subtree
        count = count + numHojasHelper( ptr->rightPtr );// traverse right subtree
    }

} // end if

return count;

}

```

```

// cuenta la profundidad del lado derecho y el izquierdo
int profundidadHelper ( TreeNode< NODETYPE > *ptr ) const
{
    int countl=0;
    int countr=0;

    if ( ptr != nullptr )
    {
        countl=1;
        countr=1;

        countl = countl + profundidadHelper( ptr->leftPtr );

        countr = countr + profundidadHelper( ptr->rightPtr );

    } // end if

    if (countl > countr)
        return countl;
    else return countr;

}

```

```

}; // end class Tree

```


La salida del programa después de compilado es el siguiente:

```
gp@gp-pc: ~/programs$ g++ M1U404.cpp arbol.h -o M1U404.b
gp@gp-pc: ~/programs$ ./M1U404.b
Enter char values of three:

Preorder traversal
G 0x55afe6cca2c0 0x55afe6cca2e0 0x55afe6cca3a0
B 0x55afe6cca2e0 0x55afe6cca300 0x55afe6cca320
A 0x55afe6cca300 0 0
C 0x55afe6cca320 0 0x55afe6cca340
F 0x55afe6cca340 0x55afe6cca360 0
D 0x55afe6cca360 0 0x55afe6cca380
E 0x55afe6cca380 0 0
Q 0x55afe6cca3a0 0x55afe6cca3c0 0x55afe6cca420
K 0x55afe6cca3c0 0x55afe6cca3e0 0x55afe6cca400
H 0x55afe6cca3e0 0 0
P 0x55afe6cca400 0 0
R 0x55afe6cca420 0 0

Cuenta de los Nodos
12
Cuenta de las Hojas
5
Cuenta de laProfundidad
6
gp@gp-pc: ~/programs$
```

Conclusiones

En la realización de esta actividad hemos aprendido a trabajar nuevas estructuras de datos mas complejas como los punteros , las clases, arboles y listas. Que nos permiten manejar otras situaciones mas abstractas del mundo real y de esta manera programar soluciones a diversos problemas.

Referencias

- Algorithms C++. Yang Hu. Verejava. 2020.
- C++ How to Program. Paul Deitel. 9 Edition. Pearson Editorial. 2014
- C Data ++ Plus Structures. Nell Dale. Third Edition. Jones and Bartlett Publishers. 2003
- Data Structures and Algorithms in C++. .Second Edition. Michael T. Goodrich. John Wiley & Sons, Inc. 2001.
- Effective C++ Digital Collection. Scoot Meyers. Addison-Wesley. 2012.
- Fourth Edition Data Structures and Algorithm Analysis in C++. Mark Allen Weiss. Pearson. 2014.
- Objects, Abstraction, Data Structures And Design Using C ++. Elliot B. Koffman. John Wiley & Sons, Inc . 2012.
- Programación en C,C++, Java y UML. Luis Joyanes Aguilar. 2da Edicion. Editorial McGrawHill. 2014