Reporte de Laboratorio 5
Colas y Pilas en C++

Dunia Barahona - B40806

1 de noviembre de 2016

Índice

---

1. Introducción

Este laboratorio tuvo como objetivo comprender el funcionamiento de las listas como una estructura de datos abstracta, tanto con punteros como con arreglos, e implementarlas en C++ haciendo uso de clases emplantilladas.

1.1. Objetivos

- Construir la clase LCP (lista con punteros) de manera emplantillada.
- Construir clases Casino, Mesa y Portero.
- Usar objetos de tipo LCP, ya sea como pilas o colas, como atributos de las clases creadas.

2. Código

2.1. Clase emplantillada: nodo

nodo.h

```
#ifndef NODO_H
#define NODO_H

#include "string"
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <string>
#include <stdlib.h>

using namespace std;

template <typename DT>  // DT (datatype): tipo de dato, es el que se reemplaza

class nodo { // Lista enlazada
public:

DT dato;  // Contenido
nodo* sn; // Puntero: siguiente nodo

nodo() { // Constructor por defecto
this->dato= -1;
this->sn= NULL;
}

nodo(DT dato) { // Constructor por parametro
this->dato= dato;
this->sn= NULL;
}

~nodo() { // Destructor
}

void print()    {  // Imprime un nodo
if (this->sn==NULL) {
cout<< this->dato <<endl;
}
else {
cout<< this->dato << "-> ";
}
}
```

```
void erase() { // Se elimina ese nodo
delete this->sn;
delete this;
}
};
#endif /* NODO_H */
```

2.2. Clase emplantillada: LCP

LCP.h

```
#ifndef LCP_H
#define LCP_H
#include "nodo.h"

template <typename DT>

class LCP {      //Lista con punteros
public:

int cn; // Cantidad de nodos
nodo<DT>* hn; // head: puntero de tipo nodo emplantillado

LCP() { // Constructor por defecto
cn= 0;
this->hn= NULL;
}

~LCP() { // Destructor
}

void add_head(DT dato) { // Inserta dato al inicio de la lista

nodo<DT>* nuevo_nodo= new nodo<DT> (dato);
nodo<DT>* temp= hn;

if (hn==NULL) {
hn = nuevo_nodo;
}
else {
nuevo_nodo->sn = hn;
hn = nuevo_nodo;
}
cn++;
}

void add_tail(DT dato) { // Inserta dato al final de la lista
```

```
nodo<DT>* nuevo_nodo= new nodo<DT> (dato);

if (hn==NULL) {
hn = nuevo_nodo;
}
else {
nodo<DT>* temp= hn;
while (temp->sn!=NULL) {
temp= temp->sn;
}
temp->sn= nuevo_nodo;
}
cn++;
}

nodo<DT>* find_e(DT esp, int opc) { // Encuentra un elemento en especifico
// Si opc es 0 devuelve nodo anterior al nodo con el elemento especifico
// Si opc es 1 devuelve nodo que con el elemento especÃfico

nodo<DT>* temp= hn;
nodo<DT>* n_ant;
nodo<DT>* n_esp;
int cont= 0;

if (hn->dato==esp)
{    n_ant=NULL;
n_esp= hn;
}
else {
while (temp!=NULL) {
if (temp->sn!=NULL) {
n_ant= temp;
n_esp= temp->sn;
}
if (n_esp->dato==esp) {
temp= NULL;
}
else {
temp= temp->sn;
cont++;
}
}
if (cont==cn) {
n_ant=NULL;
n_esp=NULL;

if (esp!='o') {
```

```
cout<<esp<<" no esta en la lista"<<endl;
}
}
}

if (opc==0) {
return n_ant;
}
else {
return n_esp;
}
delete temp;
delete n_ant;
delete n_esp;
}

void rm_head() { // Elimina primer elemento de la lista

if (hn==NULL) {
cout<<"Lista vacia"<<endl;
}
else {
hn= hn->sn;
}
cn--;
}

void rm_tail() { // Elimina ultimo elemento de la lista

if (cn==0) {
cout<<"Lista vacia"<<endl;
}

else {
nodo<DT>* tempPRE;
nodo<DT>* temp= hn;
while (temp->sn!=NULL) {
tempPRE= temp;
temp= temp->sn;
}
tempPRE->sn= NULL;
delete temp;
cn--;
}
}

void rm_e(DT esp) { // Elimina un elemento en especifico
```

```
nodo<DT>* n_esp= this->find_e(esp, 1);

if (n_esp!=NULL) {
nodo<DT>* temp= this->find_e(esp, 0);
if (temp==NULL) {
this->rm_head();
}
else {
temp->sn= n_esp->sn;
delete n_esp;
cn--;
}
}
}

void clear() { //Limpiar lista
nodo<DT>* temp= hn;
while (temp!=NULL) {
if (temp->sn==NULL) {
hn= temp;
hn= NULL;
cn--;
}
else {
hn= temp;
delete hn;
cn--;
}
temp= temp->sn;
}
}

void print() { // Imprime elementos de la lista
nodo<DT>* temp= hn;
if (this->cn==0){
cout<<"Lista vacia"<<endl;
}
else {
while (temp!=NULL) {
if (temp->sn==NULL) {
cout<< temp->dato;
}
else {
cout<< temp->dato << "-> ";
}
temp= temp->sn;
}
cout<<endl;
```

```
}
}

void find_rm(DT dato) {
// Encuentra todos los elementos iguales al dato dado y lo elimina de la lista
nodo<DT>* temp= hn;
while (temp->sn!=NULL) {
if (temp->dato == dato)
{ rm_e(dato);
}
temp= temp->sn;
}
if (temp->dato == dato && temp->sn==NULL) {
rm_tail();
}
}

};
#endif /* LCP_H */
```

2.3. Clase Portero

Portero.h

```
#ifndef PORTERO_H
#define PORTERO_H

#include "LCP.h"

class Portero {
public:

int cE;
int cT;
double cD;
int ciclos;

LCP<char>* sala_E; //Sala de espera de ejecutivos: COLA
LCP<char>* sala_T; //Sala de espera de trabajadores: COLA
LCP<char>* sala_D; //Sala de espera de desempleados: COLA

Portero();
~Portero();

void ubicar(char** clientes);
char llamar_jugador();
};
#endif /* PORTERO_H */
```

Portero.cpp

```
#include "Portero.h"

Portero::Portero() {
this->cE= 0;
this->cT= 0;
this->cD= 0.0;
this->ciclos= 0;

this->sala_E= new LCP<char>();
this->sala_T= new LCP<char>();
this->sala_D= new LCP<char>();

}
Portero::~Portero() {
}

void Portero::ubicar(char** clientes) { //Ubica clientes en salas respectivas
int i=0;
while (clientes[1][i]!='\0')
{
if (clientes[1][i]=='E')
{
sala_E->add_tail(clientes[1][i]);
}
else if (clientes[1][i]=='T')
{
sala_T->add_tail(clientes[1][i]);
}
else if (clientes[1][i]=='D')
{
sala_D->add_tail(clientes[1][i]);
}
i++;
}
sala_E->print();
sala_T->print();
sala_D->print();
}
//
char Portero::llamar_jugador() {
// Devuelve jugador que sera enviado a la mesa,
// Devuelve 'n' si no salio ningun jugador
// Devuelve 'o' si ya no hay mas clientes en salas de espera
int clientes= sala_E->cn + sala_T->cn + sala_D->cn;
char jugador;
if (clientes>0)
```

```
{
if (cE<2) {
if (sala_E->hn==NULL) {
cE++;
jugador='n';
}
else {
jugador='E';
sala_E->rm_head();
cE++;
}
}
else {
if (cT<1) {
if (sala_T->hn==NULL) {
cT++;
jugador='n';
}
else {
jugador='T';
sala_T->rm_head();
cT++;
}
}
else {
if (sala_D->hn==NULL) {
cD++;
jugador='n';
}
else {
cD+=0.5;
if (cD==1) {
jugador='D';
sala_D->rm_head();
}
else {
jugador='n';
}
}
}
}
ciclos++;
if (ciclos==4 && cE==2 && cT==1) {
cE=0;
cT=0;
ciclos=0;
if (cD==1) {
cD=0;
```

```
}
}
}
else
{
cout<<"No hay mas clientes"<<endl;
jugador='o';
}
return jugador;
}
```

2.4. Clase Mesa

Mesa.h

```
#ifndef MESA_H
#define MESA_H

#include "LCP.h"
#include "math.h"

class Mesa {
public:

LCP<int>* maso; // Maso de cartas
LCP<int>* cartas; // Cartas de cada jugador
LCP<char>* jugadores; // Jugadores de la mesa
int campos; // Campos de la mesa
string ganador;
string perdedor;

Mesa();
~Mesa();

int esta_llena();
void barajar();
int dar_carta();
void repartir(LCP<int>* cartas, int k);
int turno();
void partida();
void print_jugada(LCP<char>* jugadores, LCP<int>* cartas);
void print_maso();
};
#endif /* MESA_H */
```

Mesa.cpp

```
#include "Mesa.h"
```

```
Mesa::Mesa() {
this->maso= new LCP<int>();                    //pila
this->cartas= new LCP<int>();          //pila
this->jugadores= new LCP<char>();      //cola
this->campos= 3;
this->perdedor="";              //mayor a 21
this->ganador ="";              //igual a 21 o cerca de 21
}
Mesa::~Mesa() {
}

int Mesa::esta_llena() { // Determina si la mesa esta llena o no
if ((3-jugadores->cn) > 0)      {
return 0;
}
else {
return 1;
}
}

void Mesa::barajar() { // Baraja el maso. Llena de manera aleatoria la cola maso
if (maso->hn!=NULL) {
maso->clear();
}
for (int i = 0; i < 13; i++) {
maso->add_head(rand() %10 + 1);
}
}

int Mesa::dar_carta() { // Entrega una carta del maso
int c= maso->hn->dato;
maso->rm_head();
return c;
}

void Mesa::repartir(LCP<int>* cartas, int k) { // Reparte cartas
// Si no tienen cartas se entregan dos, si ya tienen cartas se entrega solo una
int carta=0;
barajar();
if (cartas->hn==NULL)   {
for (int j = 0; j < k; j++) {
for (int i = 0; i < 2; i++) {
carta+= dar_carta();
}
cartas->add_tail(carta);
carta=0;
}
```

```
}
else {
nodo<int>* temp= cartas->hn;
int ct=0;
if (cartas->cn < k) {
for (int i = cartas->cn; i < k; i++) {
cartas->add_tail(0);
}
}
for (int i = 0; i < k; i++) {
carta= temp->dato + dar_carta();
cartas->add_tail(carta);
cartas->rm_head();
temp= temp->sn;
}
}
}

int Mesa::turno() { // En cada turno se reparten cartas y se verifica si algun juga
int se_acaba= 0;        //no
LCP<char>* ganadores= new LCP<char>();  //temporal
LCP<char>* perdedores= new LCP<char>(); //temporal
repartir(cartas, jugadores->cn);
cout<<endl;
print_jugada(jugadores, cartas);

nodo<int>* tempC= cartas->hn;           //crea nodo temporal y guarda en Ã©l al noc
nodo<char>* tempJ= jugadores->hn;       //crea nodo temporal y guarda en Ã©l al noc

while (tempC!=NULL) {                           //recorre las cartas que tiene cada
if (tempC->dato>18 && tempC->dato<22)   // Si la carta del jugador esta entre 19 y
{
ganadores->add_tail(tempJ->dato);
ganador+=tempJ->dato;
ganador+=' ';
jugadores->rm_e(tempJ->dato);
se_acaba= 1;
}
if (tempC->dato>21)             // Si la suma de las cartas del jugador es mayor a
{
perdedores->add_tail(tempJ->dato);
perdedor+=tempJ->dato;
perdedor+=' ';
jugadores->rm_e(tempJ->dato);
se_acaba= 1;
}
tempC= tempC->sn;
```

```
tempJ= tempJ->sn;
}
////
if (jugadores->cn == 0)
{
if (ganadores->cn >0)
{
tempJ= ganadores->hn;    //jugadores = ganadores
while (tempJ!=NULL) {
jugadores->add_tail(tempJ->dato);
tempJ= tempJ->sn;
}
ganadores->clear();
}
}
else     // si jugadores tiene algo
{
if (ganadores->cn >0)
{
tempJ= jugadores->hn;    //perdedores = jugadores
while (tempJ!=NULL) {
perdedor+=tempJ->dato;
perdedor+=' ';
tempJ= tempJ->sn;
}
jugadores->clear();

tempJ= ganadores->hn;    //jugadores = ganadores
while (tempJ!=NULL) {
jugadores->add_tail(tempJ->dato);
tempJ= tempJ->sn;
}
ganadores->clear();
}
if (ganadores->cn == 0 && perdedores->cn>0)
{
tempJ= jugadores->hn;    //ganadores = jugadores
while (tempJ!=NULL) {
ganador+=tempJ->dato;
ganador+=' ';
tempJ= tempJ->sn;
}
}
}
if (ganador!="")
{
cout<<endl<<"Gano:\t"<<ganador<<endl;
}
```

```
if (perdedor!="")
{
cout<<"Salio:\t"<<perdedor<<endl;
}
cout<<"Jugadores: ";
jugadores->print();
campos= 3 - jugadores->cn;
cout<<"Campos:\t"<<campos<<endl;
ganadores->clear();
perdedores->clear();

return se_acaba;
}

void Mesa::partida() { // La partida inicia con la mesa llena y se acaba hasta que
cout<<" ///////////////// Inicia partida"<<endl;
int se_acaba= 0;
while (se_acaba== 0)
{
se_acaba= turno();
}
if (se_acaba==1)
{
cartas->clear();
ganador="";
perdedor="";
cout<<endl;
}
}

void Mesa::print_jugada(LCP<char>* jugadores, LCP<int>* cartas) { // Imprime jugado
nodo<char>* jt= jugadores->hn;
nodo<int>* ct= cartas->hn;

if (jugadores->cn==0 || cartas->cn==0){
cout<<"No hay jugadores"<<endl;
}
else {
while (jt!=NULL) {
cout<<"Juega "<<jt->dato<<" con "<<ct->dato<<endl;
jt= jt->sn;
ct= ct->sn;
}
}
}

void Mesa::print_maso() { // Imprime el maso de cartas
nodo<int>* temp= maso->hn;
```

```
if (maso->cn==0){
cout<<"Pila vacÃa"<<endl;
}
else {
while (temp!=NULL) {
cout<< "carta: "<<temp->dato <<endl;
temp= temp->sn;
}
}
cout<<endl;
}
```

2.5. Clase Casino

Casino.h

```
#ifndef CASINO_H
#define CASINO_H

#include "Portero.h"
#include "Mesa.h"

class Casino {
public:

Portero* P;
Mesa* mesa_1;
Mesa* mesa_2;
Mesa* mesa_3;

Casino();
~Casino();

void jugar();

};
#endif /* CASINO_H */
```

Casino.cpp

```
#include "Casino.h"

Casino::Casino() {
this->P= new Portero();
this->mesa_1= new Mesa();
this->mesa_2= new Mesa();
this->mesa_3= new Mesa();
}
```

```
Casino::~Casino() {
}

void Casino::jugar() {
// Las mesas juegan hasta que se hayan ido todos los clientes
char j='i';
int ell;
while (j!='o')
{
ell= mesa_1->esta_llena();
if (ell==0)     {
cout<<"Mesa 1 /////////////// Llama jugadores"<<endl;
cout<<"Jugadores antes:   ";
mesa_1->jugadores->print();

while (ell==0)
{
j= P->llamar_jugador();
if (j!='n')
{
mesa_1->jugadores->add_tail(j);
}
ell= mesa_1->esta_llena();
}
mesa_1->jugadores->find_rm('o');
cout<<"Jugadores despues: ";
mesa_1->jugadores->print();
cout<<endl;
}
else
{
cout<<"Mesa 1";
mesa_1->partida();
}

ell= mesa_2->esta_llena();
if (ell==0)     //si hay campos disponibles
{
cout<<"Mesa 2 /////////////// Llama jugadores"<<endl;
cout<<"Jugadores antes:   ";
mesa_2->jugadores->print();

while (ell==0)
{
j= P->llamar_jugador();
if (j!='n')
{
```

```
mesa_2->jugadores->add_tail(j);
}
ell= mesa_2->esta_llena();
}
mesa_2->jugadores->find_rm('o');
cout<<"Jugadores despues: ";
mesa_2->jugadores->print();
cout<<endl;
}
else
{
cout<<"Mesa 2";
mesa_2->partida();
}
ell= mesa_3->esta_llena();
if (ell==0)      //si hay campos disponibles
{
cout<<"Mesa 3 /////////////// Llama jugadores"<<endl;
cout<<"Jugadores antes:    ";
mesa_3->jugadores->print();

while (ell==0)
{
j= P->llamar_jugador();
if (j!='n')
{
mesa_3->jugadores->add_tail(j);
}
ell= mesa_3->esta_llena();
}
mesa_3->jugadores->find_rm('o');
cout<<"Jugadores despues: ";
mesa_3->jugadores->print();
cout<<endl;
}
else
{
cout<<"Mesa 3";
mesa_3->partida();
}
}
// cuando ya no hay mas clientes en sala de espera
int cont= 100;
while (cont>0)
{
if (mesa_1->jugadores->cn>0)
{
cout<<"Mesa 1";
```

```
mesa_1->partida();
}
if (mesa_2->jugadores->cn>0)
{
cout<<"Mesa 2";
mesa_2->partida();
}
if (mesa_3->jugadores->cn>0)
{
cout<<"Mesa 3";
mesa_3->partida();
}

cont= mesa_1->jugadores->cn + mesa_2->jugadores->cn + mesa_3->jugadores->cn;

}
}
```

3. Conclusiones

1. Encontrar un dato en específico tarda más en una lista con punteros que con arreglos.
2. Las listas tienen muchas funcionalidades.
3. Emplantillar clases permite usar cualquier tipo de dato.