INSTITUTO POLITECNICO NACIONAL

UPIITA

Programación Avanzada

Práctica 1 Clases y Objetos

Objetivo

Utilizar el paradigma de orientación a objetos para programar una aplicación que permita modelar polinomios como elementos de problemas del área científica básica correspondientes a la formación de estudiantes de ingeniería.

# Clases

Considere las clases

## Clase Rac:

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\usepackage{amsmath,amssymb,amsfonts,latexsym}
\def\siguion{
\sigma-
}
\pagestyle{empty}
\begin{document}
%{\large
\begin{verbatim}
/** Rac.h - Una clase para modelar n\'umeros racionales
*/
#include <iostream>  /* cout, ostream*/
using namespace std; 
#ifndef Rac_h
#define Rac_h
    /** 
     *  Clase para representar n\'umeros racionales
     *  @author Lamberto Maza Casas
     *  @version 2018.02.12
     */
struct Rac {
    int calc_mcd();
    void set_mcd(int MCD);
    Rac& operator+(Rac& RacObj);
    Rac& operator-(Rac& RacObj);
    Rac& operator*(Rac& RacObj);
    Rac& operator/(Rac& RacObj);
    Rac& operator+(int intVal);
    friend Rac& operator+(int intVal,Rac& RacObj);    
    friend ostream& operator<<(ostream&  ostreamOut, Rac& RacObj);    
    Rac(const Rac&);
    Rac(int intNum, int intDen);
    Rac();
    Rac(int){ }
    void simplificar();    
    std::string string_show();
    int n;  /* numerador */
    int d;  /* denominador*/
    int mcd; /* m\'aximo com\'un divisor*/
};//end struct Rac
#endif // Rac_h
\end{verbatim}
%}
\end{document}

## Clase Polinomio:

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\usepackage{amsmath,amssymb,amsfonts,latexsym}
\def\siguion{
\sigma-
}
\pagestyle{empty}
\begin{document}
%{\large
\begin{verbatim}
/** Polinomio.h - Una clase para representar polinomios.
*/
#include <iostream>  /* cout, ostream*/
using namespace std;
#include "Rac.h"
#ifndef Polinomio_h
#define Polinomio_h
#define maximo(a,b) ((a>b)?a:b)
struct Polinomio {
    Polinomio& operator+(Polinomio& PolObj);
    Polinomio& operator-(Polinomio& PolObj);
    Polinomio& operator*(Polinomio& PolObj);
//    FcnDTrans& operator/(Polinomio& PolObj);
    friend ostream& operator<<(ostream& ostreamObj, Polinomio& PolObj);
    std::string string_show();
    Polinomio(int intGrado, Rac * RacPt);
    Polinomio(){ }
    Polinomio(int){ }
    int grado;
    Rac * A;
};//end struct Polinomio
#endif // Polinomio_h
\end{verbatim}
%}
\end{document}

## Clase Directory\_entry:

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\usepackage{amsmath,amssymb,amsfonts,latexsym}
\def\siguion{
\sigma-
}
\pagestyle{empty}
\begin{document}
%{\large
\begin{verbatim}
/** Directory_entry.h
*/
#ifndef DIRECTORY_ENTRY
#define DIRECTORY_ENTRY
#include <string>
class Directory_entry {
public:
 std::string var_name;
 std::string coeffs;
 
 Directory_entry();
 Directory_entry(std::string var_name,std::string coeffs);
};
#endif /*DIRECTORY_ENTRY*/
\end{verbatim}
%}
\end{document}

## Clase Pol\_Directory:

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\usepackage{amsmath,amssymb,amsfonts,latexsym}
\def\siguion{
\sigma-
}
\pagestyle{empty}
\begin{document}
%{\large
\begin{verbatim}
/** Pol_Directory.h 
Basado en Listado 1.9 Array_Based_PD.h, p\'ag 114 del libro:
    Estructura de datos con C++, Objetos, abstracciones y dise\~no, 
 de Elliot B. Koffman/Paul A. T. Wolfgang, 
 Editorial Mc Graw Hill Interamericana, 2008.
 ISBN-10:970-10-6578-6
 ISBN-13:978-970-10-6578-5  
*/
#ifndef POL_DIRECTORY
#define POL_DIRECTORY
#include <string>
/**Archivo de especificaci\'on para el directorio 
   de polinomios basado en un conjunto. */
#include "Directory_entry.h"
class Pol_Directory {
 public:
 /** Construir un directorio de polinomios vacio. */
 Pol_Directory();
 
 /** Destruir el directorio de polinomios cunado ya no se necesita. */
 ~Pol_Directory();
 
 /** Cargar el o los archivos de datos que contienen el directorio, 
     establecer la conexi\'on con la fuente de datos.
  @param source_name El nombre del archivo (fuente de datos) con 
                     las entradas del directorio de polinomios. */
 void load_data(const std::string& source_name);
 
 /** Buscar una entrada.
     @param name El nombre de la variable a buscar
     @return Los coefiecientes asociados con la variable o una 
             cadena vacia si la variable no se encuentra en el directorio.
 */
 std::string lookup_entry(const std::string& var_name) const;
 /** Modificar el contenido asociado con el 
         nombre de la variable var_name por el 
         de coeffs, o agregar una nueva entrada 
         con el nombre de variable var_name y 
         los coeficientes coeffs.
    @param var_name El nombre de la variable
    @param coeffs Los nuevos coeficientes
    @return Los coeficientes antiguos o una cadena vacia si esta 
            es una nueva entrada
 */
 std::string add_or_change_entry(const std::string& var_name,
const std::string& coeffs); 
\end{verbatim}
% /** Eliminar la entrada con un nombre 
%         espec\'ifico del directorio.
%     @param var_name El nombre de la variable 
%         a remover
%     @return El nombre de la variable o una 
%         cadena vacia si es que no se encuentra 
%        en el directorio.
% */
% std::string remove_entry(
%                  const std::string& var_name);
 
% /** Escribir el contenido del directorio en el archivo de datos. */
% void save(); 
% private:
% // Funciones privadas
% /** Buscar en el directorio el nombre de la variable var_name.
%     @param var_name El nombre de variable a ser encontrado
%  @return El indice de la entrada que 
%          contiene el nombre de la variable, o 
%          el tama\~no del directorio en caso 
%          de que el nombre de la variable no 
%          se haya encontrado. 
% */
% int find(const std::string& var_name) const;
% /** Agregar una nueva entrada con un nombre espec\'ifico y sus coeficientes 
%     al arreglo de entradas del directorio de polinomios.
%     @param var_name El nombre de la variable a ser agregada
%     @param coeffs Los coefcientes de la variable a ser agregada
% */
% void add(const std::string& var_name,const std::string& coeffs);
% 
% /** Eliminar la entrada de un \'indice espec\'ifico.
%     @param index El \'indice de la entrada a ser eliminada
% */
% void remove_entry(int index);
% 
% /** Crear un nuevo conjunto de las entradas del directorio 
%     con el doble de la capacidad del actual.
% */
% void reallocate();
% 
% /** El n\'umero de entradas en el directorio. */
% int size;
% /** La capacidad real del directorio. */
% int capacity;
% /** Apuntador del arreglo que contiene los datos del directorio. */
% Directory_entry *the_directory;
% /** El nombre del archivo de datos que contiene los datos del directorio. */
% std::string source_name;
% /** Bandera booleana para indicar si se modific\'o el directorio. */
% bool modified;
%};//end class Pol_Directory
%#endif /*POL_DIRECTORY*/
%\end{verbatim}
%}
\end{document}

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\usepackage{amsmath,amssymb,amsfonts,latexsym}
\def\siguion{
\sigma-
}
\pagestyle{empty}
\begin{document}
%{\large
\begin{verbatim}
/** Eliminar la entrada con un nombre 
    espec\'ifico del directorio.
    @param var_name El nombre de la variable 
    a remover 
    @return El nombre de la variable o una 
    cadena vacia si es que no se encuentra 
    en el directorio.
*/
 std::string remove_entry(
                  const std::string& var_name);
 
/** Escribir el contenido del directorio en 
    el archivo de datos. */
 void save(); 
 private:
// Funciones privadas
/** Buscar en el directorio el nombre de la 
    variable var_name.
    @param var_name El nombre de variable a 
    ser encontrada
    @return El indice de la entrada que 
    contiene el nombre de la variable, o 
    el tama\~no del directorio en caso de que 
    el nombre de la variable no se haya 
    encontrado. 
 */
 int find(const std::string& var_name) const;
/** Agregar una nueva entrada con un nombre 
    espec\'ifico y sus coeficientes 
    al arreglo de entradas del directorio de 
    polinomios.
    @param var_name El nombre de la variable 
    a ser agregada
    @param coeffs Los coefcientes de la 
    variable a ser agregada
*/
 void add(const std::string& var_name,
          const std::string& coeffs);
 
/** Eliminar la entrada de un \'indice 
    espec\'ifico.
    @param index El \'indice de la entrada 
    a ser eliminada
*/
 void remove_entry(int index);

\end{verbatim}
%}
\end{document}

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\usepackage{amsmath,amssymb,amsfonts,latexsym}
\def\siguion{
\sigma-
}
\pagestyle{empty}
\begin{document}
%{\large
\begin{verbatim} 
/** Crear un nuevo conjunto de las 
    entradas del directorio con el doble 
    de la capacidad del actual.
*/
 void reallocate();
 
 /** El n\'umero de entradas en el directorio.
       */
 int size;
 /** La capacidad real del directorio. */
 int capacity;
 /** Apuntador del arreglo que contiene 
         los datos del directorio. */
 Directory_entry *the_directory;
 /** El nombre del archivo de datos que 
         contiene los datos del directorio. */
 std::string source_name;
 /** Bandera booleana para indicar si se 
         modific\'o el directorio. */
 bool modified;
};//end class Pol_Directory
#endif /*POL_DIRECTORY*/
\end{verbatim}
%}
\end{document}

## Clase String\_Tokenizer:

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\usepackage{amsmath,amssymb,amsfonts,latexsym}
\def\siguion{
\sigma-
}
\pagestyle{empty}
\begin{document}
%{\large
\begin{verbatim}
/** String_Tokenizer.h
*/
#ifndef STRING_TOKENIZER_H
#define STRING_TOKENIZER_H
#include <string>
/** La clase String_Tokenizer divide una cadena en una secuencia 
    de subcadenas, denominadas tokens, y separadas por delimitadores
*/
class String_Tokenizer{
  public:
/** Construccion de un String_Tokenizer
    @param source La cadena a ser dividida en tokens
    @param delim La cadena que contiene los delimitadores. Si se omite
                 este parametro, se asume un caracter de espacio.
  */
  String_Tokenizer(std::string source, std::string delim=" "):
    the_source(source), the_delim(delim), start(0), end(0) {
    find_next();
  }
  String_Tokenizer():the_source(""), the_delim(" "), start(0), end(0) {
   find_next();
  }
\end{verbatim}
%}
\end{document}

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\usepackage{amsmath,amssymb,amsfonts,latexsym}
\def\siguion{
\sigma-
}
\pagestyle{empty}
\begin{document}
%{\large
\begin{verbatim}
/** Determinar si existen mas tokens
    @return true si existen mas tokens
  */
  bool has_more_tokens();
/** Recuperar el siguiente token
    @return El siguiente token. Si no hay mas tokens, regresara un string 
            vacio
  */
  std::string next_token();

  private:
  /** Ubicar start y end de manera que start sea el indice del comienzo del 
      siguiente token y end sea la terminacion. 
    */
    void find_next();
  /** La cadena a ser dividida en tokens */
    std::string the_source;
  /** La cadena de delimitadores */
    std::string the_delim;
  /** El indice del inicio del siguiente token */
    size_t start;
  /** El indice de la terminacion del siguiente token */
    size_t end;
};//end class String_Tokenizer
    
#endif /* STRING_TOKENIZER_H */

\end{verbatim}
%}
\end{document}

Las implementaciones de las cinco clases anteriores se muestran a continuación.

## Archivo Rac.cpp

#include <cmath>

#include "Rac.h"

/\*\*

\* Clase para representar n\'umeros racionales

\* @author Lamberto Maza Casas

\* @version 2018.02.12

\*/

int Rac::calc\_mcd()

{

int M,N,tmp,MCD;

/\*si numerador!=0 y denominador!=0, usar Algoritmo de Euclides\*/

if((n!=0)&&(d!=0)){

if(n>d){

M=abs(n);N=abs(d);

}else{

M=abs(d);N=abs(n);

}/\* ALGORITMO DE EUCLIDES (300 a.C.) \*/

while((tmp=M%N)!=0){ /\* M = QN + tmp \*/

M=N;

N=tmp;

}/\*cuando este while termina, en N se tiene el mcd\*/

MCD=N;

}else{/\*si no, hacer mcd=1\*/

MCD=1;/\*Para evitar problemas cuando n=0 o d=0\*/

}

return MCD;

}

void Rac::set\_mcd(int MCD)

{

mcd=MCD;

}

Rac& Rac::operator+(Rac& RacObj)

{

Rac RacR;

Rac& RacResult=RacR;

RacResult.n=n\*RacObj.d+d\*RacObj.n;

RacResult.d=d\*RacObj.d;

RacResult.set\_mcd(RacResult.calc\_mcd());

RacResult.simplificar();

// cout<<RacResult.n<<"/"<<RacResult.d<<endl;

return RacResult;

}

Rac& Rac::operator-(Rac& RacObj)

{

Rac RacR;

Rac& RacResult=RacR;

RacResult.n=n\*RacObj.d-d\*RacObj.n;

RacResult.d=d\*RacObj.d;

RacResult.set\_mcd(calc\_mcd());

RacResult.simplificar();

return RacResult;

}

Rac& Rac::operator\*(Rac& RacObj)

{

Rac RacR;

Rac& RacResult=RacR;

RacResult.n=n\*RacObj.n;

RacResult.d=d\*RacObj.d;

RacResult.set\_mcd(calc\_mcd());

RacResult.simplificar();

return RacResult;

}

Rac& Rac::operator/(Rac& RacObj)

{

Rac RacR;

Rac& RacResult=RacR;

RacResult.n=n\*RacObj.d;

RacResult.d=d\*RacObj.n;

RacResult.set\_mcd(calc\_mcd());

RacResult.simplificar();

return RacResult;

}

ostream& operator<<(ostream& ostreamOut, Rac& RacObj)

{

ostreamOut<<"\\frac{"<<RacObj.n<<"}{"\

<<RacObj.d<<"}"<<endl;

return ostreamOut;

}

Rac::Rac(int intNum, int intDen):n(intNum),d(intDen)

{

set\_mcd(calc\_mcd());

}

Rac::Rac():mcd(1) /\*para evitar divisi\'on entre cero si se llama a simplificar\*/

{

}

void Rac::simplificar()

{

n=n/mcd;

d=d/mcd;

}

Rac::Rac(const Rac& R):n(R.n),d(R.d)

{

mcd=calc\_mcd();

}

Rac& Rac::operator+(int intVal)

{

Rac op(intVal,1),R;

R=\*this+op;

Rac& r=R;

return r;

}

Rac& operator+(int intVal,Rac& RacObj)

{

Rac op(intVal,1),R;

R=op+RacObj;

Rac& r=R;

return r;

}

std::string Rac::string\_show(){

char str[128];

sprintf(str,"%d/%d",n,d);

string Result=string(str);

return Result;

}

## Archivo Polinomio.cpp

#include "Polinomio.h"

/\*\* Operador referencia a

\* Una referencia es un nombre alternativo (un sin\'onimo) para

\* un objeto. Su utilizaci\'on la podemos observar en el dise\~no

\* de clases (por ejemplo, en el constructor de copia), en el valor

\* retornado por una funci\'on para permitir que dicha funci\'on sea

\* utilizada a ambos lados del operador de asignaci\'on (la funci\'on

\* devuelve una referencia), o para permitir que los argumentosen la

\* llamada puedan cambiar (paso de par\'ametros por referencia). La

\* forma general de expresar una referencia es:

\* tipo& referencia=variable

\*/

Polinomio& Polinomio::operator+(Polinomio& PolObj)

{

int grad=maximo(grado,PolObj.grado);

Polinomio ResP(grad,new Rac[grad+1]);

Polinomio& ResPol=ResP;

if(grado==grad){/\*si el obj implicito es el de mayor grado\*/

for(int i=0;i<grado+1;i++){

(ResPol.A+i)->n=(A+i)->n;

(ResPol.A+i)->d=(A+i)->d;

}

}else{/\*si el segundo operando es el de mayor grado\*/

for(int i=0;i<PolObj.grado+1;i++){

(ResPol.A+i)->n=(PolObj.A+i)->n;

(ResPol.A+i)->d=(PolObj.A+i)->d;

}

}

if(grado<=grad){/\*si el objeto implicito es el polinomio de menor grado\*/

//FIXMEEEEEEE!!!!!! FIXED: 2018.03.13.15.21

// i\in\{0,1,2,...,grado\} y queremos que los coeficientes del objeto

// impl\'icito se le sumen a los coeficientes del resultado correspondientes

// a los \'indices \{grad - grado +0,grad - grado +1...,grad - 2,grad - 1,grad\}

for(int i=0;i<grado+1;i++){

// \*(ResPol.A+grad-grado+i)=\*(ResPol.A+grad-grado+i)+\*(PolObj.A+i);

\*(ResPol.A+grad-grado+i)=\*(ResPol.A+grad-grado+i)+\*(A+i);

}

}else{/\*si el segundo operando es el de menor grado\*/

for(int i=0;i<ResPol.grado+1;i++){

// \*(ResPol.A+i)=\*(ResPol.A+i)+\*(A+i);

\*(ResPol.A+grad-PolObj.grado+i)=\*(ResPol.A+grad-PolObj.grado+i)+\*(PolObj.A+i);

}

}

return ResPol;

}

Polinomio& Polinomio::operator-(Polinomio& PolObj)

{

int grad=maximo(grado,PolObj.grado);

Polinomio ResP(grad,new Rac[grad+1]);

Polinomio& ResPol=ResP;

if(grado==grad){/\*si el obj implicito es el de mayor grado\*/

for(int i=0;i<grado+1;i++){

(ResPol.A+i)->n=(A+i)->n;

(ResPol.A+i)->d=(A+i)->d;

}

}else{/\*si el segundo operando es el de mayor grado\*/

for(int i=0;i<PolObj.grado+1;i++){

(ResPol.A+i)->n=(PolObj.A+i)->n;

(ResPol.A+i)->d=(PolObj.A+i)->d;

}

}

if(grado<=grad){/\*si el objeto implicito es el polinomio de menor grado\*/

for(int i=0;i<grado+1;i++){

\*(ResPol.A+i)=\*(ResPol.A+i)-\*(PolObj.A+i);

}

}else{/\*si el segundo operando es el de menor grado\*/

for(int i=0;i<ResPol.grado+1;i++){

\*(ResPol.A+i)=\*(ResPol.A+i)-\*(A+i);

}

}

return ResPol;

}

Polinomio& Polinomio::operator\*(Polinomio& PolObj)

{

int grad=grado+PolObj.grado;

Polinomio ResP(grad,new Rac[grad+1]);

Polinomio& ResPol=ResP; /\* ResultPolynomial \*/

Rac factor1,factor2,factor1porfactor2,sumaparcial;

for(int i=0;i<ResPol.grado+1;i++){

\*(ResPol.A+i)=Rac(0,1); /\* All coeffs of ResPol must be initialized to \frac{0}{1} \*/

}

for(int k=0;k<grad+1;k++){

for(int i=0;i<grado+1;i++){

for(int j=0;j<PolObj.grado+1;j++){

if((grado-i+PolObj.grado-j)==grad-k){

factor1=\*(A+i);

factor2=\*(new Rac((PolObj.A+j)->n,(PolObj.A+j)->d));

factor1porfactor2=factor1\*factor2;

sumaparcial=\*(ResPol.A+k)+factor1porfactor2;

\*(ResPol.A+k)=sumaparcial;

}

}

}

}

return ResPol;

}

//FcnDTrans& Polinomio::operator/(Polinomio& PolObj)

//{

//}

ostream& operator<<(ostream& ostreamOut, Polinomio& PolObj)

{

Rac \*rPt;

int intV;

ostreamOut<<"\\begin{array}{";

for(int j=0;j<PolObj.grado+1;j++){

ostreamOut<<"c";

}

ostreamOut<<"}"<<endl;

for(int i=0;i<PolObj.grado+1;i++){

rPt=(PolObj.A+i);

intV=rPt->n;

if((PolObj.A+i)->d==1){

if(i){

ostreamOut<<"&"<<intV<<endl;

}else{

ostreamOut<<intV<<endl;

}

}else{

if(i){

ostreamOut<<"&\\frac{"<<intV<<"}{"<<(PolObj.A+i)->d<<"}"<<endl;

}else{

ostreamOut<<"\\frac{"<<intV<<"}{"<<(PolObj.A+i)->d<<"}"<<endl;

}

}

}

ostreamOut<<"\\end{array}"<<endl;

return ostreamOut;

}

Polinomio::Polinomio(int intGrado, Rac \* RacPt):grado(intGrado),A(RacPt)

{ }

std::string Polinomio::string\_show(){

string Result="";

for(int i=0;i<grado;i++){

Result+=A[i].string\_show()+":";

}

Result+=A[grado].string\_show();

return Result;

}

## Achivo Directory\_entry.cpp

/\*\* Directory\_entry.cpp

\*/

#include "Directory\_entry.h"

Directory\_entry::Directory\_entry()

{ //Deliberadamente vacio

}

Directory\_entry::Directory\_entry(std::string var\_name,std::string coeffs){

this->var\_name=var\_name;

this->coeffs=coeffs;

}

## Archivo Pol\_Directory.cpp

/\*\* Pol\_Directory.cpp

\*/

#include <iostream>

#include <fstream> /\*ifstream,ofstream\*/

#include "Pol\_Directory.h"

using namespace std;

Pol\_Directory::Pol\_Directory():size(0),capacity(1){

the\_directory=new Directory\_entry[1];

}

Pol\_Directory::~Pol\_Directory(){

delete[] the\_directory;

}

/\*\* Funci\'on para cargar el archivo de datos.

pre: El almacenamiento del directorio se ha creado y se encuentra vacio.

Si existe el archivo consistir\'a de pares nombre\_de\_variable-coeficientes

en l\'ineas adyacentes.

post: Los datos a partir del archivo se cargan en el directorio.

@param source\_name El nombre del archivo de datos

\*/

void Pol\_Directory::load\_data(const std::string& source\_name){

// Recordar el nombre del archivo fuente para utilizarlo al salvar.

this->source\_name=source\_name;

// Crear un flujo de entrada que provenga de este archivo

ifstream in(source\_name.c\_str());

if(in){ // si el flujo existe

string name;

string coeffs;

while(getline(in,name)){

if(getline(in,coeffs)){

// cout<<"name="<<name<<" coeffs="<<coeffs<<endl;

add(name,coeffs);

}

}

in.close();

}

}

/\*\* Modificar el contenido asociado con el nombre de la variable

var\_name por el de coeffs, o agregar una nueva entrada con el

nombre de variable var\_name y los coeficientes coeffs.

@param var\_name El nombre de la variable

@param coeffs Los nuevos coeficientes

@return Los coeficientes antiguos o una cadena vacia si esta

es una nueva entrada

\*/

std::string Pol\_Directory::add\_or\_change\_entry(const std::string& var\_name,const std::string& coeffs)

{

string old\_coeffs="";

int index=find(var\_name);

if(index!=-1){

old\_coeffs=the\_directory[index].coeffs;

the\_directory[index].coeffs=coeffs;

}else{

add(var\_name,coeffs);

}

modified=true;

return old\_coeffs;

}

/\*\* Adicionar un nuevo par nombre-coeffs al directorio.

@param name El nombre de la variable a seragregada

@param coeffs Los coeficientes de la variable a ser agregada

\*/

void Pol\_Directory::add(const std::string& var\_name,const std::string& coeffs){

if(size==capacity){// if no room, reallocate

reallocate();

}

// Agregar una nueva entrada e incrementar size.

the\_directory[size]=Directory\_entry(var\_name,coeffs);

size++;

}

/\*\* Crear un nuevo arreglo de las entradas del directorio

con el doble de capacidad del actual.

\*/

void Pol\_Directory::reallocate(){

// Duplicar la capacidad

capacity\*=2;

// Crear un nuevo arreglo del directorio

Directory\_entry \*new\_directory=new Directory\_entry[capacity];

// Copiar el antiguo directorio al nuevo

for(int i=0;i<size;i++){

new\_directory[i]=the\_directory[i];

}

//Devolver la memoria ocupada por el antiguo directorio

delete[] the\_directory;

// Poner the\_directory a apuntar al nuevo directorio

the\_directory=new\_directory;

}

std::string Pol\_Directory::lookup\_entry(const std::string& var\_name) const {

int index=find(var\_name);

if(index!=-1){

return the\_directory[index].coeffs;

}else{

return "";

}

}

int Pol\_Directory::find(const std::string& var\_name) const {

for(int i=0;i<size;i++){

if(the\_directory[i].var\_name==var\_name){

return i;

}

}

return -1;

}

/\*\* Funci\'on para guardar el directorio.

pre: el directorio ha sido cargado con datos

post: contiene el directorio vuelto a escribir en el archivo

bajo la forma par var\_name-coeffs en l\'ineas adyacentes.

modified se vuelve a colocar en false.

\*/

void Pol\_Directory::save(){

if(modified){// if not modified do nothing

ofstream out(source\_name.c\_str());

for(int i=0;i<size;i++){

out<<the\_directory[i].var\_name<<endl;

out<<the\_directory[i].coeffs<<endl;

}

}

}

## Archivo String\_Tokenizer.cpp

/\*\* String\_Tokenizer.cpp

\*/

#include "String\_Tokenizer.h"

using std::string;

/\*\* Ubicar start y end de modo que start sea el indice del comienzo

del siguiente token y end sea la terminacion.

\*/

void String\_Tokenizer::find\_next(){

//Encuentra el primer caracter que no es un delimitador.

start=the\_source.find\_first\_not\_of(the\_delim, end);

//Encuentra el siguiente delimitador

end=the\_source.find\_first\_of(the\_delim, start);

}

/\*\* Determina si existen mas tokens

@return true si hay mas tokens

\*/

bool String\_Tokenizer::has\_more\_tokens(){

return start!=string::npos;

}

/\*\* Recupera el siguiente token

@return El siguiente token. Si no hay mas tokens regresa una

cadena vacia.

\*/

string String\_Tokenizer::next\_token(){

//Asegura la existencia de un siguiente token

if(!has\_more\_tokens())

return "";

//Guarda el siguiente token

string token=the\_source.substr(start, end-start);

//Encuentra el siguiente token

find\_next();

//Regresa el siguiente token

return token;

}

Ahora los archivos con funciones main

## Archivo comando0.cpp

#include <iostream>

#include <vector>

using namespace std;

#include <stdio.h>

#include <stdlib.h>

#include "Pol\_Directory.h"

#include "String\_Tokenizer.h"

#include "Polinomio.h"

#define PRINTING\_DEBUG

//#define COMANDO\_0

//Prototipos de funciones

void proccess\_operation(Pol\_Directory&,string&,unsigned char,string&);

void do\_suma(Pol\_Directory&,string&,string&);

void do\_save(Pol\_Directory&);

#ifdef COMANDO\_0

int main(int argc,char \*argv[]){

// cout<<"Dummy HOLA MUNDO desde "<<argv[0]<<"!!!"<<endl;

// cout<<"Dummy HOLA MUNDO!!!"<<endl;

if(argc<2){

cout<<"Faltan argumentos!"<<endl;

return 1;

}else{

for(int i=0;i<argc-1;i++){

cout<<"argv["<<i<<"]: "<<argv[i]<<endl;

}

Pol\_Directory the\_directory;

string source\_name(argv[1]);

the\_directory.load\_data(source\_name);

string Operand1(argv[2]);

string Operador(argv[3]);

string Operand2(argv[4]);

unsigned char \*tmpPt=(unsigned char\*)malloc(sizeof(unsigned char));

// \*tmpPt=\*((unsigned char\*)(&(\*argv[3])));

\*tmpPt=\*((unsigned char\*)(argv[3]));

#ifdef PRINTING\_DEBUG

cout<<"\*tmpPt="<<\*tmpPt<<endl;

#endif

proccess\_operation(the\_directory,Operand1,\*tmpPt,Operand2);

}

return 0;

}

#endif /\*COMANDO\_0\*/

## Archivo Test\_Shell.cpp

#include <iostream>

#include <vector>

using namespace std;

#include <stdio.h>

#include <windows.h> /\*STARTUPINFO, PROCESS\_INFORMATION\*/

#include "Pol\_Directory.h"

#include "String\_Tokenizer.h"

#include "Polinomio.h"

#define MAXSIZE 1000

int last\_backslash(TCHAR\*);

void get\_substr(TCHAR\* cmdL,int intCantidad,char charBuf[MAXSIZE]);

#define TEST\_SHELL

#ifdef TEST\_SHELL

int main(){

TCHAR\* cmdline;

STARTUPINFO startup\_info;

PROCESS\_INFORMATION process\_info;

char charBuffer[MAXSIZE];

char exepath[MAXSIZE];

char str[MAXSIZE];

cmdline = GetCommandLine(); /\*obtiene la orden o comando equivalente de CLI\*/

/\*CLI: Command Line Interface\*/

cout<<cmdline<<endl;

int intCantidad=last\_backslash(cmdline);

printf("intCantidad=%i\n",intCantidad);

get\_substr(cmdline,intCantidad,charBuffer);

printf("%s\n",charBuffer);

char workspace[]="workspace.txt";

sprintf(exepath,"%s%s",charBuffer,"comando0.exe");

char dummy[]="dummy";

sprintf(str,"%s ./%s %s %s %s %s","comando0.exe",

workspace,"var1","+","var2",dummy);

printf("IMPRIMIR COMANDO A EJECUTAR PASANDO ARGUMENTOS:\n");

printf("%s\n",str);

fill\_n( (char\*)&process\_info, sizeof( PROCESS\_INFORMATION ), 0 );

fill\_n( (char\*)&startup\_info, sizeof( STARTUPINFO ), 0 );

startup\_info.cb = sizeof( STARTUPINFO );

if (!CreateProcess(

exepath,

str,

NULL,

NULL,

TRUE,

NORMAL\_PRIORITY\_CLASS,

NULL,

NULL,

&startup\_info,

&process\_info

)){

return 2;

}

CloseHandle( process\_info.hThread );

// wait for the child program to terminate

WaitForSingleObject( process\_info.hProcess, INFINITE );

//Leer el resultado

string source\_name=string(workspace);

Pol\_Directory the\_directory;

the\_directory.load\_data(source\_name);

string coeffs0=the\_directory.lookup\_entry("ans");

String\_Tokenizer tokenizer0(coeffs0,":");

vector<string> arr0;

while(tokenizer0.has\_more\_tokens()){

arr0.push\_back(tokenizer0.next\_token());

}//end while()

int num,den; /\*numerador,denominador\*/

Rac \*RacPt0=new Rac[arr0.size()];

string stringIntNum,stringIntDen;

String\_Tokenizer tokenizerForRac;

for(int i=0;i<arr0.size();i++){

tokenizerForRac=String\_Tokenizer(arr0[i],"/");

stringIntNum=tokenizerForRac.next\_token();

num=atoi(stringIntNum.c\_str());

stringIntDen=tokenizerForRac.next\_token();

den=atoi(stringIntDen.c\_str());

\*(RacPt0+i)=Rac(num,den);

}

Polinomio PolResult0(arr0.size()-1,RacPt0);

cout<<PolResult0<<endl;

cout<<PolResult0.string\_show()<<endl;

return 0;

}//end main()

#endif

/\*\*

\* Devuelve la cantidad de car\'acteres que hay que usar de cmdL

\* para incluir hasta el \'ultimo backslash (es decir, desde el

\* car\'acter 0 hasta el car\'acter que devuelve esta funci\'on)

\*/

int last\_backslash(TCHAR\* cmdL){

int r=0,i=0,intI=0;

while('\0'!=cmdL[i]){

if('\\'==cmdL[i]){

r++;

}

i++;

}/\*Now r is the amount of backslashes\*/

i=0;

while((intI<r)&&('\0'!=cmdL[i])){

if('\\'==cmdL[i]){

intI++;

}

i++;

}

return i;

}

void get\_substr(TCHAR\* cmdL,int intCantidad,char charBuf[MAXSIZE]){

int i=0;

while(i<intCantidad){

charBuf[i]=cmdL[i];

i++;

}

charBuf[i]='\0';

}