Seminario de solución de problemas de Estructuras de Datos II

Clave: I5889

NRC: 59623

Calendario: 2016B

Sección: D10

Actividad 05: CSV Importación, Modificación y Eliminación

Alumno: Navarro Presas Moisés Alejandro

Código del Alumno: 215861509

Cli.hpp

#ifndef CLI\_HPP

#define CLI\_HPP

#include <iostream>

#include <cstring>

#include "email.hpp"

using namespace std;

#ifdef \_WIN32

#define CLEAR "cls"

#else

#define CLEAR "clear"

#endif

#define OPT\_EXIT 0

#define OPT\_WRITE 1

#define OPT\_READ 2

#define OPT\_MODIFY 3

#define OPT\_DELETE 4

#define OPT\_EXPORT\_CSV 5

#define OPT\_IMPORT\_CSV 6

#define OPT\_MODIFY\_CSV 7

#define OPT\_DELETE\_CSV 8

#define OPT\_ERROR\_FILE -1

#define INVALID\_OPTION "Opcion invalida\n"

#define MSG\_DONE "Listo\n"

#define MSG\_NOT\_FOUND "No encontrado\n"

#define MSG\_ERROR\_FILE "Error en el archivo\n"

#define MSG\_PAUSE "Presione entrar para continuar . . ."

void display\_menu()

{

cout << OPT\_WRITE << ") Escribir" << endl;

cout << OPT\_READ << ") Leer" << endl;

cout << OPT\_MODIFY << ") Modificar" << endl;

cout << OPT\_DELETE << ") Eliminar" << endl;

cout << OPT\_EXPORT\_CSV << ") Exportar" << endl;

cout << OPT\_IMPORT\_CSV << ") Importar" << endl;

cout << OPT\_MODIFY\_CSV << ") Modificar (CSV)" << endl;

cout << OPT\_DELETE\_CSV << ") Eliminar (CSV)" << endl;

cout << OPT\_EXIT << ") Salir" << endl;

}

int get\_int(string msg = ">", int def = -1) {

int i;

cout << msg;

if(!(cin >> i)) {

cin.clear();

cin.ignore();

i = def;

}

cin.ignore();

return i;

}

bool get\_bool(string msg = ">")

{

char c;

cout << msg << "(y/n)";

cin >> c;

return c == 'y';

}

string get\_string(string msg = ">")

{

string s;

cout << msg;

getline(cin, s);

return s;

}

string get\_text(string msg = ">")

{

char c;

string s;

cout << msg;

while (cin.get(c) && c != '~')

{

s.append(1, c);

}

s.append(1, '\0');

cin.clear();

return s;

}

void clear\_screen() {

system(CLEAR);

}

void msg(string m) {

cout << m;

}

void pause() {

cout << MSG\_PAUSE;

cin.ignore();

}

string timet\_to\_date(time\_t time)

{

char date\_c[11];

if (!strftime(date\_c, sizeof(date\_c), "%Y-%m-%d", localtime(&time)))

{

strcpy(date\_c, "1970-01-01");

}

string s(date\_c, date\_c+11);

return s;

}

string timet\_to\_time(time\_t time)

{

char time\_c[6];

if (!strftime(time\_c, sizeof(time\_c), "%H:%M", localtime(&time)))

{

strcpy(time\_c, "00:00");

}

string s(time\_c, time\_c+6);

return s;

}

/\* ---------------------------------------------------------------------------\*/

void fill(Email\* email)

{

email->set\_id(get\_int("ID>"));

email->set\_from(get\_string("Remitente>").c\_str());

email->set\_to(get\_string("Destinatario>").c\_str());

email->set\_cc(get\_string("CC>").c\_str());

email->set\_bcc(get\_string("BCC>").c\_str());

email->set\_subject(get\_string("Asunto>").c\_str());

email->set\_content(get\_text("Contenido>").c\_str());

}

void display(Email\* email)

{

if (strcmp(email->get\_from(), "") != 0)

{

cout << "ID: " << email->get\_id() << endl;

cout << "Fecha: " << timet\_to\_date(email->get\_time()) << endl;

cout << "Hora: " << timet\_to\_time(email->get\_time()) << endl;

cout << "Remitente: " << email->get\_from() << endl;

cout << "Destinatario: " << email->get\_to() << endl;

cout << "CC: " << email->get\_cc() << endl;

cout << "BCC: " << email->get\_bcc() << endl;

cout << "Asunto: " << email->get\_subject() << endl;

cout << "Contenido: " << email->get\_content() << endl;

}

else

{

msg(MSG\_NOT\_FOUND);

}

}

#endif

Csv\_manager.hpp

#ifndef CSV\_MANAGER\_HPP

#define CSV\_MANAGER\_HPP

#include <fstream>

#include <vector>

#include <cstdio>

template <typename T>

class CSV\_Manager

{

private:

std::fstream file;

std::string to\_standard(std::string field);

std::vector<std::string> from\_standard(std::string row);

std::string path;

bool open();

void close();

std::string next\_row();

T\* next\_record(std::string row = "x");

std::string to\_row(T\* data);

public:

CSV\_Manager(const char\* path);

~CSV\_Manager();

bool is\_open();

void append(T\* data);

template <typename F>

void for\_each(F function);

template <typename V, typename F>

T\* find (V value, F function, size\_t\* num\_row = nullptr);

void write\_in(T\* data, const size\_t num\_row);

};

template <typename T>

CSV\_Manager<T>::CSV\_Manager(const char\* path)

{

this->path = path;

}

template <typename T>

bool CSV\_Manager<T>::open()

{

file.open(path, std::ios::in | std::ios::out);

if (!file.is\_open())

{

std::ofstream aux(path);

aux.close();

file.open(path, std::ios::in | std::ios::out);

file.seekg(0, std::ios::beg);

}

return file.is\_open();

}

template <typename T>

CSV\_Manager<T>::~CSV\_Manager()

{

if (file.is\_open())

{

file.close();

}

}

template <typename T>

bool CSV\_Manager<T>::is\_open()

{

return file.is\_open();

}

template <typename T>

std::string CSV\_Manager<T>::to\_row(T\* data)

{

std::string record;

auto fields = data->get\_fields();

size\_t num\_fields = fields.size();

for (auto field : fields)

{

record += to\_standard(field);

if (--num\_fields)

{

record += ",";

}

}

return record + '\n';

}

template <typename T>

void CSV\_Manager<T>::append(T\* data)

{

open();

if (!file.is\_open()) return;

file.seekp(0, std::ios::end);

file << to\_row(data);

file.close();

}

template <typename T>

std::string CSV\_Manager<T>::to\_standard(std::string field)

{

bool quotes = false;

std::string standard;

for (auto c : field)

{

if (c == '\"' || c == ',' || c == '\n')

{

quotes = true;

if (c == '\"')

{

standard += "\"";

}

}

standard += c;

}

if (quotes) return "\"" + standard + "\"";

return field;

}

template <typename T>

std::vector<std::string> CSV\_Manager<T>::from\_standard(std::string row)

{

std::vector<std::string> fields;

bool quotes = false;

bool last\_is\_quote = false;

std::string aux;

for (auto c : row)

{

if ((c == ',' || c == '\n' || c == EOF) && !quotes)

{

fields.push\_back(aux);

aux = "";

}

else

{

if (c == '"')

{

if(last\_is\_quote && !quotes)

{

aux += c;

}

last\_is\_quote = true;

quotes = !quotes;

}

else

{

last\_is\_quote = false;

aux += c;

}

}

}

return fields;

}

template <typename T>

std::string CSV\_Manager<T>::next\_row()

{

if (!file.is\_open()) return "";

std::string row;

bool quotes = false;

char c;

file.clear();

while(true)

{

file.read(&c, 1);

if (file.eof()) return "";

if (c == EOF || (!quotes && c == '\n')) break;

if (c == '"')

{

quotes = !quotes;

}

row += c;

}

return row + '\n';

}

template <typename T>

T\* CSV\_Manager<T>::next\_record(std::string row)

{

T\* record;

if (row == "x")

row = next\_row();

if (file.eof()) return nullptr;

record = new T;

record->set\_fields(from\_standard(row));

return record;

}

template <typename T>

template <typename F>

void CSV\_Manager<T>::for\_each(F function)

{

open();

T\* aux;

while (true)

{

aux = next\_record();

if (!aux) break;

function(aux);

delete aux;

}

file.close();

}

template <typename T>

template <typename V, typename F>

T\* CSV\_Manager<T>::find(V value, F compare, size\_t\* num\_row)

{

open();

T\* aux;

while (true)

{

aux = next\_record();

if (!aux) break;

if(compare(\*aux, value) == 0)

{

file.close();

return aux;

}

delete aux;

if (num\_row != nullptr)

{

++\*num\_row;

}

}

file.close();

return nullptr;

}

template <typename T>

void CSV\_Manager<T>::write\_in(T \*data, const size\_t num\_row)

{

open();

if (!file.is\_open()) return;

std::ofstream aux\_file("~" + path);

if (!aux\_file.is\_open()) return;

size\_t i = 0;

std::string row;

while (true)

{

row = next\_row();

if (file.eof()) break;

if (i == num\_row)

{

if (data != nullptr)

{

aux\_file << to\_row(data);

}

}

else

{

aux\_file << row;

}

++i;

}

aux\_file.close();

file.close();

remove(path.c\_str());

rename(("~" + path).c\_str(), path.c\_str());

}

#endif

Email.hpp

#ifndef EMAIL\_HPP

#define EMAIL\_HPP

#include <ctime>

#include <cstring>

#include <vector>

#include <string>

class Email

{

private:

int id;

time\_t time;

char from[256];

char to[256];

char cc[256];

char bcc[256];

char subject[78];

char content[512];

public:

void set\_id(const int id);

int get\_id();

void set\_time(const time\_t time);

time\_t get\_time();

void set\_from(const char \*from);

char\* get\_from();

void set\_to(const char \*to);

char\* get\_to();

void set\_cc(const char \*cc);

char\* get\_cc();

void set\_bcc(const char \*bcc);

char\* get\_bcc();

void set\_subject(const char \*subject);

char\* get\_subject();

void set\_content(const char \*content);

char\* get\_content();

static int cmp\_from(char \*from, Email\* a);

bool empty();

std::vector<std::string> get\_fields();

void set\_fields(std::vector<std::string> fields);

bool operator==(const Email& other) const;

bool operator!=(const Email& other) const;

static bool equal(const Email& a, const Email& b);

static bool diff(const Email& a, const Email& b);

static bool compare\_id(Email& a, const int id);

Email();

};

Email::Email()

{

id = 0;

from[0] = '\x0';

to[0] = '\x0';

cc[0] = '\x0';

bcc[0] = '\x0';

subject[0] = '\x0';

content[0] = '\x0';

}

void Email::set\_id(const int id)

{

this->id = id;

}

int Email::get\_id()

{

return id;

}

void Email::set\_time(const time\_t time)

{

this->time = time;

}

time\_t Email::get\_time()

{

return time;

}

void Email::set\_from(const char\* from)

{

strcpy(this->from, from);

}

char\* Email::get\_from()

{

return from;

}

void Email::set\_to(const char\* to)

{

strcpy(this->to, to);

}

char\* Email::get\_to()

{

return to;

}

void Email::set\_cc(const char\* cc)

{

strcpy(this->cc, cc);

}

char\* Email::get\_cc()

{

return cc;

}

void Email::set\_bcc(const char\* bcc)

{

strcpy(this->bcc, bcc);

}

char\* Email::get\_bcc()

{

return bcc;

}

void Email::set\_subject(const char\* subject)

{

strcpy(this->subject, subject);

}

char\* Email::get\_subject()

{

return subject;

}

void Email::set\_content(const char\* content)

{

strcpy(this->content, content);

}

char\* Email::get\_content()

{

return content;

}

int Email::cmp\_from(char\* from, Email\* b)

{

return strcmp(from, b->get\_from());

}

std::vector<std::string> Email::get\_fields()

{

std::vector<std::string> fields;

std::string aux;

char date\_c[11];

char time\_c[6];

fields.push\_back(std::to\_string(id));

strftime(date\_c, sizeof(date\_c), "%Y-%m-%d", localtime(&time));

aux = date\_c;

fields.push\_back(aux);

strftime(time\_c, sizeof(time\_c), "%H:%M", localtime(&time));

aux = time\_c;

fields.push\_back(aux);

aux = from;

fields.push\_back(aux);

aux = to;

fields.push\_back(aux);

aux = cc;

fields.push\_back(aux);

aux = bcc;

fields.push\_back(aux);

aux = subject;

fields.push\_back(aux);

aux = content;

fields.push\_back(aux);

return fields;

}

void Email::set\_fields(std::vector<std::string> fields)

{

std::string date;

std::string time\_c;

id = atoi(fields[0].c\_str());

date = fields[1];

date += ' ' + fields[2];

struct tm tm;

strptime(date.c\_str(), "%Y-%m-%d %H:%M", &tm);

time = mktime(&tm);

strcpy(from,fields[3].c\_str());

strcpy(to,fields[4].c\_str());

strcpy(cc,fields[5].c\_str());

strcpy(bcc,fields[6].c\_str());

strcpy(subject,fields[7].c\_str());

strcpy(content,fields[8].c\_str());

}

bool Email::operator==(const Email& other) const

{

return id == other.id && time == other.time && strcmp(from, other.from) == 0 &&

strcmp(to, other.to) == 0 && strcmp(cc, other.cc) == 0 &&

strcmp(bcc, other.bcc) == 0 && strcmp(subject, other.subject) == 0 &&

strcmp(content, other.content) == 0;

}

bool Email::operator!=(const Email& other) const

{

return !operator==(other);

}

bool Email::equal(const Email& a, const Email& b)

{

return a == b;

}

bool Email::diff(const Email& a, const Email& b)

{

return a != b;

}

bool Email::empty()

{

return id == 0;

}

bool Email::compare\_id(Email& a, const int id)

{

return id - a.get\_id();

}

#endif

Fixed\_file\_manager.hpp

#ifndef FIXED\_FILE\_MANAGER\_HPP

#define FIXED\_FILE\_MANAGER\_HPP

#include <fstream>

template <typename T>

class FixedFileManager

{

private:

std::fstream file;

std::string path;

bool open();

public:

FixedFileManager(const char\* path);

~FixedFileManager();

bool is\_open();

bool write(const T\* data, const size\_t pos);

bool append(T\* data);

T\* read(const size\_t pos);

template <typename V, typename F>

size\_t find(V value, F cmp);

template <typename F>

void for\_each(F function);

template <typename F, typename C, typename O>

void for\_each(F function, C compare, O object);

void clean();

};

template <typename T>

FixedFileManager<T>::FixedFileManager(const char\* path)

{

this->path = path;

}

template <typename T>

bool FixedFileManager<T>::open()

{

file.open(path, std::ios::in | std::ios::out | std::ios::binary);

if (!file.is\_open())

{

std::ofstream aux(path, std::ios::binary);

aux.close();

file.open(path, std::ios::in | std::ios::out | std::ios::binary);

}

return file.is\_open();

}

template <typename T>

FixedFileManager<T>::~FixedFileManager()

{

if (file.is\_open())

{

file.close();

}

}

template <typename T>

bool FixedFileManager<T>::is\_open()

{

return file.is\_open();

}

template <typename T>

bool FixedFileManager<T>::write(const T\* data, const size\_t pos)

{

open();

if (file.is\_open())

{

file.seekp(pos \* sizeof(T));

file.write((char\*)data, sizeof(T));

file.close();

return true;

}

file.close();

return false;

}

template <typename T>

bool FixedFileManager<T>::append(T\* data)

{

open();

if (file.is\_open())

{

size\_t pos = data->get\_id();

file.seekp(pos \* sizeof(T));

file.write((char\*)data, sizeof(T));

file.close();

return true;

}

file.close();

return false;

}

template <typename T>

T\* FixedFileManager<T>::read(const size\_t pos)

{

open();

if (file.is\_open())

{

T\* data = new T;

file.seekg(pos \* sizeof(T), std::ios::beg);

file.read((char\*)data, sizeof(T));

file.close();

return data;

}

file.close();

return nullptr;

}

template <typename T>

template <typename V, typename F>

size\_t FixedFileManager<T>::find(V value, F cmp)

{

open();

if (file.is\_open())

{

T\* aux = new T;

file.seekg(0, std::ios::beg);

while (true)

{

file.read((char\*)aux, sizeof(T));

if (file.eof()) break;

if (cmp(value, aux) == 0)

{

file.close();

return ((long)file.tellg() - sizeof(T)) / sizeof(T);

}

}

file.clear();

}

file.close();

return (size\_t)-1;

}

template <typename T>

template <typename F>

void FixedFileManager<T>::for\_each(F function)

{

open();

if (!file.is\_open()) return;

T\* aux = new T;

file.seekg(0, std::ios::beg);

while (true)

{

file.read((char\*)aux, sizeof(T));

if (!aux->empty())

{

if (file.eof()) break;

function(aux);

}

}

file.clear();

file.close();

}

template <typename T>

template <typename F, typename C, typename O>

void FixedFileManager<T>::for\_each(F function, C compare, O object)

{

open();

if (!file.is\_open()) return;

T\* aux = new T;

file.seekg(0, std::ios::beg);

while (true)

{

file.read((char\*)aux, sizeof(T));

if(compare(\*aux, object) == 0)

{

if (file.eof()) break;

function(aux);

}

}

file.clear();

file.close();

}

template <typename T>

void FixedFileManager<T>::clean()

{

if(open())

{

file.close();

remove(path.c\_str());

}

}

#endif

Main.cpp

#include <functional>

#include "cli.hpp"

#include "email.hpp"

#include "fixed\_file\_manager.hpp"

#include "csv\_manager.hpp"

using namespace std;

#define FIXED\_PATH "email.db"

#define CSV\_PATH "backup.csv"

FixedFileManager<Email> fixed\_file(FIXED\_PATH);

CSV\_Manager<Email> csv\_file(CSV\_PATH);

auto append\_to\_csv = std::bind(&CSV\_Manager<Email>::append, &csv\_file, std::placeholders::\_1);

auto write\_to\_db = std::bind(&FixedFileManager<Email>::append, &fixed\_file, std::placeholders::\_1);

size\_t search\_by()

{

msg("1) ID\n");

msg("2) Remitente\n");

switch (get\_int())

{

case 1:

{

return get\_int("ID>");

}

case 2:

{

char from[256];

size\_t pos;

strcpy(from, get\_string("Remitente>").c\_str());

pos = fixed\_file.find(from, Email::cmp\_from);

return pos;

}

}

return -1;

}

int main()

{

short opt;

do

{

clear\_screen();

display\_menu();

opt = get\_int();

switch (opt)

{

case OPT\_EXIT:

{

break;

}

case OPT\_WRITE:

{

Email \*email = new Email;

fill(email);

email->set\_time(time(nullptr));

fixed\_file.write(email, email->get\_id());

delete email;

msg(MSG\_DONE);

break;

}

case OPT\_READ:

{

Email\* email;

size\_t pos = search\_by();

if (pos != (size\_t)-1 && (email = fixed\_file.read(pos)) && email != nullptr)

{

display(email);

delete email;

}

else

{

msg(MSG\_NOT\_FOUND);

}

break;

}

case OPT\_MODIFY:

{

Email\* email;

size\_t pos = search\_by();

if (pos != (size\_t)-1 && (email = fixed\_file.read(pos)) && email != nullptr)

{

display(email);

msg("1) Fecha y Hora\n");

msg("2) Remitente\n");

msg("3) Destinatario\n");

msg("4) CC\n");

msg("5) BCC\n");

msg("6) Asunto\n");

msg("7) Contenido\n");

switch (get\_int())

{

case 1:

{

struct tm tm;

strptime(get\_string("Y-M-D H:M>").c\_str(), "%Y-%m-%d %H:%M", &tm);

time\_t time = mktime(&tm);

email->set\_time(time);

break;

}

case 2:

{

email->set\_from(get\_string("Remitente>").c\_str());

break;

}

case 3:

{

email->set\_to(get\_string("Destinatario>").c\_str());

break;

}

case 4:

{

email->set\_cc(get\_string("CC>").c\_str());

break;

}

case 5:

{

email->set\_bcc(get\_string("BCC>").c\_str());

break;

}

case 6:

{

email->set\_subject(get\_string("Asunto>").c\_str());

break;

}

case 7:

{

email->set\_content(get\_text("Contenido>").c\_str());

break;

}

default:

msg(INVALID\_OPTION);

}

fixed\_file.write(email, email->get\_id());

delete email;

}

else

{

msg(MSG\_NOT\_FOUND);

}

break;

}

case OPT\_DELETE:

{

Email\* email;

size\_t pos = search\_by();

if (pos != (size\_t)-1 && (email = fixed\_file.read(pos)) && email != nullptr)

{

display(email);

if(get\_bool("Seguro?"))

{

Email empty;

fixed\_file.write(&empty, email->get\_id());

}

delete email;

}

else

{

msg(MSG\_NOT\_FOUND);

}

break;

}

case OPT\_EXPORT\_CSV:

{

fixed\_file.for\_each(append\_to\_csv);

break;

}

case OPT\_IMPORT\_CSV:

{

if (get\_bool("Seguro? esto borrara el archivo anterior"))

{

fixed\_file.clean();

csv\_file.for\_each(write\_to\_db);

}

}

case OPT\_MODIFY\_CSV:

{

Email\* email;

size\_t\* num\_row = new size\_t;

\*num\_row = 0;

email = csv\_file.find(get\_int("ID>"), Email::compare\_id, num\_row);

if (email != nullptr)

{

display(email);

msg("1) Fecha y Hora\n");

msg("2) Remitente\n");

msg("3) Destinatario\n");

msg("4) CC\n");

msg("5) BCC\n");

msg("6) Asunto\n");

msg("7) Contenido\n");

switch (get\_int())

{

case 1:

{

struct tm tm;

strptime(get\_string("Y-M-D H:M>").c\_str(), "%Y-%m-%d %H:%M", &tm);

time\_t time = mktime(&tm);

email->set\_time(time);

break;

}

case 2:

{

email->set\_from(get\_string("Remitente>").c\_str());

break;

}

case 3:

{

email->set\_to(get\_string("Destinatario>").c\_str());

break;

}

case 4:

{

email->set\_cc(get\_string("CC>").c\_str());

break;

}

case 5:

{

email->set\_bcc(get\_string("BCC>").c\_str());

break;

}

case 6:

{

email->set\_subject(get\_string("Asunto>").c\_str());

break;

}

case 7:

{

email->set\_content(get\_text("Contenido>").c\_str());

break;

}

default:

msg(INVALID\_OPTION);

}

csv\_file.write\_in(email, \*num\_row);

delete num\_row;

delete email;

}

else

{

msg(MSG\_NOT\_FOUND);

}

break;

}

case OPT\_DELETE\_CSV:

{

Email\* email;

size\_t\* num\_row = new size\_t;

\*num\_row = 0;

email = csv\_file.find(get\_int("ID>"), Email::compare\_id, num\_row);

if (email != nullptr)

{

display(email);

if(get\_bool("Seguro?"))

{

csv\_file.write\_in(nullptr, \*num\_row);

delete num\_row;

delete email;

}

}

else

{

msg(MSG\_NOT\_FOUND);

}

break;

}

case OPT\_ERROR\_FILE:

{

msg(MSG\_ERROR\_FILE);

break;

}

default:

{

msg(INVALID\_OPTION);

break;

}

}

pause();

} while(opt != OPT\_EXIT);

return 0;

}





