**МIНIСТЕРСТВО  ОСВIТИ І НАУКИ  УКРАЇНИ**

**НАЦІОНАЛЬНИЙ   ТЕХНІЧНИЙ УНІВЕРСИТЕТ   УКРАЇНИ**

**“КИЇВСЬКИЙ  ПОЛІТЕХНІЧНИЙ  ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

**Лабораторна робота № 5**

**з дисципліни “ Основи програмування ”**

**тема “C++ STL класи”**

|  |  |  |
| --- | --- | --- |
| **Виконав**  **студент I курсу**  **групи КП-82**  **Мельничук Олексій Геннадійович**  **варіант №15** |  | **Перевірив**  **“\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.**  **викладач**  **Гадиняк Руслан Анатолійович**  **(прізвище, ім’я, по батькові)** |

**Київ 2019**

**Мета роботи**

Навчитись використовувати стандартні шаблонні контейнери C++ STL.

**Постановка завдання**

Визначення та позначки:

MainEntity - позначка основної сутності з лабораторної роботи №2, в коді має називатись як англійська версія назви типу даних, наприклад, Student.

ExtraEntity - позначка додаткової сутності, тип сутності отримати з цього завдання, в коді має називатись як англійська версія назви типу даних, наприклад, Course.

CUI - абревіатура від Console User Interface - Консольний Інтерфейс Користувача.

Кроки виконання:

Скопіювати у проект даного завдання ваше рішення лабораторної роботи №2.

Виконати міграцію коду з С на С++:

НЕ переписувати АТД на власні класи, замість АТД List використати стандартний клас контейнер std::vector<T> із заголовку <vector>.

Структура сутності за варіантом має залишитися структурою. Також вона повинна обов'язково містити поле цілочисельного ідентифікатора id.

Замінити всі використання const char \* та масивів символів, що містять рядки на std::string там, де це має сенс.

Переписати модуль csv на С++ (інтерфейс модуля). Приклад використання цього модуля.

Сховище даних (версія №1):

Створити у корені проекту директорію data і перенести в неї CSV файл з основними сутностями.

Переназвати CSV файл так, щоби він був множиною назви сутності англійською мовою (приклад, students.csv).

Створити модуль storage, що міститиме першу версію класу Storage

Консольний інтерфейс користувача (версія №1).

Реалізувати консольний інтерфейс користувача (CUI) за допомогою модуля cui із класом Cui (див. Додаток А), що дозволятиме користувачу:

Переглянути список всіх сутностей зі сховища (можна показувати лише коротку інформацію, наприклад, ідентифікатор + назву)

Обрати сутність за її ідентифікатором (підменю роботи з сутністю)

Переглянути детальну інформацію про обрану сутність

Змінити поле обраної сутності (окрім ідентифікатора)

Видалити обрану сутність

Створити, заповнити поля (окрім ідентифікатора) і додати нову сутність в сховище (при додаванні у сховище отримати від нього ідентифікатор нової сутності).

Переписати основну логіку роботи програми:

Видалити обробку аргументів командного рядка.

Створити екземпляр сховища для доступу до CSV файлів з диреторії data.

Створити екземпляр CUI для взаємодії з користувачем і зв'язати його зі створеним сховищем даних (див. Додаток А).

Додаткова сутність ExtraEntity:

Тип даних. Створити у окремому модулі структуру даних для ExtraEntity з декількома полями (різного типу), серед яких обов'язково має бути поле id - ідентифікатор сутності: унікальне ціле число.

Дані. Створити у директорії data новий CSV файл (приклад, courses.csv), у якому перечислити дані всіх екземплярів цього додаткового типу даних (приклад, Course), де значення стовпця id довільні (але унікальні).

Сховище. Реалізувати другу версію класу Storage, що включає поля і методи для роботи з об'єктами додаткового типу даних.

CUI. Додати до CUI головне меню, у якому користувач може зробити вибір з яким типом сутності він хоче працювати (основна чи додаткова). Додати підменю для роботи з додатковими сутностями аналогічно існуючим.

**Тексти коду програм**

|  |
| --- |
| **main.cpp** |
| #include "csv.h"  #include "storage.h"  #include "cui.h"  using namespace std;  int main(int argc, char \*argv[])  {  Storage storage("./data");  storage.load();  Cui cui(&storage);  cui.show();  return 0;  } |

|  |
| --- |
| **storage.h** |
| #pragma once  #include <vector>  #include <string>  #include "dino.h"  #include "period.h"  #include "csv.h"  #include "optional.h"  using std::string;  using std::vector;  class Storage  {  const string dir\_name\_;  vector<Dino> dinos\_;  vector<Per> pers\_;  static Dino rowToDino(const CsvRow &row);  static CsvRow DinoToRow(const Dino &dino);  static Per rowToPer(const CsvRow &row);  static CsvRow PerToRow(const Per &pr);  int getNewDinoId();  int getNewPerId();  public:  Storage(const string & dir\_name) : dir\_name\_(dir\_name) { }  bool load();  bool save();  // Dinos  vector<Dino> getAllDinos();  optional<Dino> getDinoById(int Dino\_id);  bool updateDino(const Dino &dvno);  bool removeDino(int Dino\_id);  int insertDino(const Dino &dvno);  vector<Per> getAllPers();  optional<Per> getPerById(int Per\_id);  bool updatePer(const Per &pr);  bool removePer(int Per\_id);  int insertPer(const Per &pr);  }; |

|  |
| --- |
| **storage.cpp** |
| #include "storage.h"  #include <fstream>  #include <iostream>  using namespace std;  string fillStringFromFile(string fromw)  {  string csv;  string line;  ifstream myfile(fromw);  string::iterator it = line.end();  if (myfile.is\_open())  {  while (getline(myfile, line))  {  line+="\n";  csv += line;  }  myfile.close();  }  else  {  cerr << "File was not found" << endl;  abort();  }  it = csv.end();  it--;  \*it = '\0';  return csv;  }  void writeToFile(string from, string tofile)  {  ofstream myfile(tofile);  if(myfile.is\_open())  {  myfile << from;  myfile.close();  }  else  {  cout << "Cannot open file\n"  << endl;  abort();  }  }  bool Storage::load()  {  string dinos\_filename = this->dir\_name\_ + "/data.csv";  string dinos\_csv = fillStringFromFile(dinos\_filename);  CsvTable dinos\_table = Csv::createTableFromString(dinos\_csv);  for(CsvRow & row : dinos\_table)  {  Dino newone;  newone.id = stoi(row[0]);  newone.name = row[1];  newone.per = row[2];  newone.fam = row[3];  newone.disc = stoi(row[4]);  newone.mass = stod(row[5]);  this->dinos\_.push\_back(newone);  }  string pers\_filename = this->dir\_name\_ + "/data2.csv";  string pers\_csv = fillStringFromFile(pers\_filename);  CsvTable pers\_table = Csv::createTableFromString(pers\_csv);  for(CsvRow & row2 : pers\_table)  {  Per newone;  newone.id = stoi(row2[0]);  newone.name = row2[1];  newone.time = stod(row2[2]);  this->pers\_.push\_back(newone);  }  return true;  }  bool Storage::save()  {  string dinos\_filename = this->dir\_name\_ + "/data.csv";  CsvTable newt;  for(Dino & a : this->dinos\_)  {  CsvRow row;  row.push\_back(to\_string(a.id));  row.push\_back(a.name);  row.push\_back(a.per);  row.push\_back(a.fam);  row.push\_back(to\_string(a.disc));  row.push\_back(to\_string(a.mass));  newt.push\_back(row);  }  string csv\_txt = Csv::createStringFromTable(newt);    writeToFile(csv\_txt, dinos\_filename);  string pers\_filename = this->dir\_name\_ + "/data2.csv";  CsvTable newt2;  for(Per & p : this->pers\_)  {  CsvRow row;  row.push\_back(to\_string(p.id));  row.push\_back(p.name);  row.push\_back(to\_string(p.time));  newt2.push\_back(row);  }  string csv\_txt2 = Csv::createStringFromTable(newt2);    writeToFile(csv\_txt2, pers\_filename);  return true;  }  vector<Dino> Storage::getAllDinos()  {  return this->dinos\_;  }  optional<Dino> Storage::getDinoById(int Dino\_id)  {  for(Dino & c : this->dinos\_)  {  if(c.id == Dino\_id)  {  return c;  }  }  return nullopt;  }  bool Storage::removeDino(int Dino\_id)  {  int index = -1;  for(int i = 0; i < this->dinos\_.size(); i++)  {  if(this->dinos\_[i].id == Dino\_id)  {  index = i;  break;  }  }  if(index >= 0)  {  this->dinos\_.erase(this->dinos\_.begin() + index);  return true;  }  return false;  }  int Storage::getNewDinoId()  {  int max\_id = 0;  for(Dino & d : this->dinos\_)  {  if(d.id > max\_id)  {  max\_id = d.id;  }  }  int new\_id = max\_id + 1;  return new\_id;  }  int Storage::insertDino(const Dino &dvno)  {  int new\_id = this->getNewDinoId();  Dino copy = dvno;  copy.id = new\_id;  this->dinos\_.push\_back(copy);  return new\_id;  }  bool Storage::updateDino(const Dino &dvno)  {  int Dino\_id = dvno.id;  for(Dino & c : this->dinos\_)  {  if(c.id == Dino\_id)  {  c.disc = dvno.disc;  c.fam = dvno.fam;  c.mass = dvno.mass;  c.name = dvno.name;  c.per = dvno.per;  return true;  }  }  return false;  }  vector<Per> Storage::getAllPers()  {  return this->pers\_;  }  optional<Per> Storage::getPerById(int id)  {  for(Per & p : this->pers\_)  {  if(p.id == id)  {  return p;  }  }  return nullopt;  }  bool Storage::removePer(int id)  {  int index = -1;  for(int i = 0; i < this->pers\_.size(); i++)  {  if(this->pers\_[i].id == id)  {  index = i;  break;  }  }  if(index >= 0)  {  this->pers\_.erase(this->pers\_.begin() + index);  return true;  }  return false;  }  int Storage::getNewPerId()  {  int max\_id = 0;  for(Per & p : this->pers\_)  {  if(p.id > max\_id)  {  max\_id = p.id;  }  }  int new\_id = max\_id + 1;  return new\_id;  }  int Storage::insertPer(const Per &pr)  {  int new\_id = this->getNewPerId();  Per copy = pr;  copy.id = new\_id;  this->pers\_.push\_back(copy);  return new\_id;  }  bool Storage::updatePer(const Per &pr)  {  int Per\_id = pr.id;  for(Per & p : this->pers\_)  {  if(p.id == Per\_id)  {  p.name = pr.name;  p.time = pr.time;  return true;  }  }  return false;  } |

|  |
| --- |
| **csv.h** |
| #pragma once  #include <string>  #include <cassert>  #include <cstdio>  #include <cstdlib>  #include <cstring>  #include <iostream>  #include <vector>  #include <fstream>  using std::vector;  using std::string;  using namespace std;  using CsvRow = vector<string>; // type name alias  using CsvTable = vector<CsvRow>; // type name alias  namespace Csv  {  // parse csv, create & fill string table  CsvTable createTableFromString(const string &csv\_str);  // return a string filled with csv from string table  string createStringFromTable(const CsvTable &csv\_table);  } // namespace Csv |

|  |
| --- |
| **csv.cpp** |
| #include "csv.h"  CsvTable Csv::createTableFromString(const string & csv\_str)  {  int bracount = 0;  string buf;  const char \* p = csv\_str.c\_str();  CsvRow row;  CsvTable table;  while (1)  {  if (bracount == 0)  {  if (\*p == '\"')  {  bracount = 1;  }  else  {  if (\*p == '\n' || \*p == '\0')  {  row.push\_back(buf);  table.push\_back(row);  row.clear();  buf.clear();  }  else if(\*p == ',')  {  row.push\_back(buf);  buf.clear();  }  else  {  buf += \*p;  }  }  }  else  {  if (\*p == '\"')  {  if (\*(p + 1) == '\"')  {  buf += '\"';  p += 1;  }  else  {  row.push\_back(buf);  buf.clear();  if (\*(p + 1) == ',')  {  p += 1;  bracount = 0;  }  else if(\*(p + 1) == '\n' || \*(p + 1) == '\0')  {  table.push\_back(row);  row.clear();  p += 1;  bracount = 0;  }  else  {  std::cerr << "CSV format error: invalid char" << std::endl;  abort();  }  }  }  else  {  buf += \*p;  }  }  if (\*p == '\0')  {  break;  }  p += 1;  }  return table;  }  string Csv::createStringFromTable(const CsvTable &csv\_table)  {  bool escape = false;  string buf;  string::size\_type n = 0;  std::string bracount;  CsvRow row;  for (int i = 0; i < csv\_table.size(); i++)  {  row = csv\_table.at(i);  for (int j = 0; j < row.size(); j++)  {  n = 0;  bracount.clear();  std::string value = row.at(j);  if((value.find(',') != string::npos || value.find('\n') != string::npos) && value.find('\"') == string::npos)  {  buf += "\"";  buf += value;  buf += "\"";  }  else if(value.find('\"') != string::npos)  {  buf += "\"";  bracount = value;  while(n != string::npos)  {  if(n != 0)  {  bracount.insert(n, "\"");  n += 2;  }  n = value.find('\"', n);  }  buf += bracount;  buf += "\"";  }  else  {  buf += value;  }    if (j != csv\_table[i].size() - 1)  {  buf += ",";  }  }  if (i != csv\_table.size() - 1)  {  buf += "\n";  }  }  return buf;  }  // CsvTable Csv::createTableFromString(const string & csv\_str)  // {  // CsvTable table;  // const char \*csvString = csv\_str.c\_str();  // char buf[100];  // for (int i = 0; i < 100; i++)  // {  // buf[i] = '\0';  // }  // //BUFFER TRACKER  // int bufX = 0;  // bool state = true;  // string nonconst = csv\_str;  // string::iterator p = nonconst.begin();  // CsvRow row;  // int commacount = 0;  // while (1)  // {  // if (state == true)  // {  // if (\*p == '"')  // {  // buf[bufX] = \*p;  // bufX++;  // buf[bufX] = \*p;  // bufX++;  // }  // else  // {  // if(\*p == ' ' || \*p == ',' || \*p == '\n')  // {  // if (\*p == ' ')  // {  // for (int i = strlen(buf); i > 0; i--)  // {  // buf[i] = buf[i - 1];  // }  // buf[0] = '"';  // bufX++;  // buf[bufX] = \*p;  // bufX++;  // state = false;  // }  // else if (\*p == ',')  // {  // if(buf[0] == '\0')  // {  // buf[0] = '"';  // buf[1] = '"';  // buf[2] = '\0';  // row.push\_back(buf);  // for (int i = 0; i < bufX + 1; i++)  // {  // buf[i] = '\0';  // }  // commacount++;  // bufX = 0;  // p++;  // continue;  // }  // if (\*(p + 1) == ' ' || \*(p + 1) == '\n')  // {  // for (int i = strlen(buf); i > 0; i--)  // {  // buf[i] = buf[i - 1];  // }  // buf[0] = '"';  // bufX++;  // buf[bufX] = \*p;  // bufX++;  // state = false;  // }  // else if (\*(p+1) != ' ' || \*(p+1) != '\n')  // {  // buf[bufX] = '\0';  // row.push\_back(buf);  // for (int i = 0; i < bufX + 1; i++)  // {  // buf[i] = '\0';  // }  // commacount++;  // bufX = 0;  // }  // }  // else if(\*p == '\n')  // {  // if (commacount == 5)  // {  // buf[bufX] = '\0';  // row.push\_back(buf);  // for (int i = 0; i < bufX + 1; i++)  // {  // buf[i] = '\0';  // }  // table.push\_back(row);  // row.erase(row.begin(), row.end());  // commacount = 0;  // bufX = 0;  // p++;  // continue;  // }  // else  // {  // state = false;  // for (int i = strlen(buf); i > 0; i--)  // {  // buf[i] = buf[i - 1];  // }  // buf[0] = '"';  // bufX++;  // buf[bufX] = \*p;  // bufX++;  // }  // }  // }  // else  // {  // if (\*p == '\0')  // {  // buf[bufX] = '\0';  // row.push\_back(buf);  // for (int i = 0; i < bufX + 1; i++)  // {  // buf[i] = '\0';  // }  // table.push\_back(row);  // row.erase(row.begin(),row.end());    // bufX = 0;  // }  // else  // {  // buf[bufX] = \*p;  // bufX += 1;  // }  // }  // }  // }  // else if(state == false) // open brackets reading  // {  // if (\*p == '"')  // {  // buf[bufX] = \*p;  // bufX++;  // buf[bufX] = \*p;  // bufX++;  // }  // else  // {  // if(\*p == ' ' || \*p == ',' || \*p == '\n')  // {  // if (\*p == ' ')  // {  // buf[bufX] = \*p;  // bufX++;  // }  // else if (\*p == ',')  // {  // if (\*(p + 1) == ' ' || \*(p + 1) == '\n')  // {  // buf[bufX] = \*p;  // bufX++;  // }  // else  // {  // buf[bufX] = '"';  // bufX++;  // buf[bufX] = '\0';  // state = true;  // row.push\_back(buf);  // for (int i = 0; i < bufX + 1; i++)  // {  // buf[i] = '\0';  // }  // commacount++;  // //  // bufX = 0;  // }  // }  // else if (\*p == '\n')  // {  // if (commacount == 5)  // {  // state = true;  // buf[bufX] = '"';  // bufX++;  // buf[bufX] = '\0';  // row.push\_back(buf);  // for (int i = 0; i < bufX + 1; i++)  // {  // buf[i] = '\0';  // }  // table.push\_back(row);  // row.erase(row.begin(),row.end());  // commacount = 0;  // bufX = 0;  // }  // else  // {  // buf[bufX] = \*p;  // bufX++;  // }  // }  // }  // else  // {  // if(\*p == '\0')  // {  // buf[bufX] = '"';  // bufX++;  // buf[bufX] = '\0';  // row.push\_back(buf);  // for (int i = 0; i < bufX + 1; i++)  // {  // buf[i] = '\0';  // }    // table.push\_back(row);  // row.erase(row.begin(),row.end());  // bufX = 0;  // }  // else  // {  // buf[bufX] = \*p;  // bufX += 1;  // }  // }  // }  // }  // if (\*p == '\0')  // {  // break;  // }  // p += 1;  // }  // return table;  // }  // string Csv::createStringFromTable(const CsvTable &csv\_table)  // {  // string to;  // string::iterator it = to.end();  // for(int i = 0; i < csv\_table.size(); i++)  // {  // for(int j = 0; j < csv\_table[i].size(); j++)  // {  // to+=csv\_table[i][j];  // if(j == csv\_table[i].size() - 1)  // {  // to+="\n";  // }  // else to+=',';  // }  // }  // // it = to.end();  // // it--;  // // \*it = '\0';  // return to;  // } |

|  |
| --- |
| **dino.h** |
| #pragma once  #include <string>  using namespace std;  struct Dino  {  int id;  string name;  string per;  string fam;  int disc;  double mass;  }; |

|  |
| --- |
| **period.h** |
| #pragma once  #include <string>  using namespace std;  struct Per  {  int id;  string name;  double time;  }; |

|  |
| --- |
| **data.csv** |
| Plotosaurus,Cretaceous,Mosasauridae,1941,2400  Coelophysis,Triassic,Coelophysidae,1887,20  Echinodon,Cretaceous,Heterodontosauridae,1861,7  Saltasaurus,Cretaceous,Saltasauridae,1977,7000  Diplodocus,Jurassic,Diplodocidae,1877,13000 |

|  |
| --- |
| **CMakeLists.txt** |
| cmake\_minimum\_required(VERSION 3.6.1)  file(GLOB SOURCES "\*.c")  add\_executable(exec\_project ${SOURCES}) |

**Робота коду**

Програма будується з допомогою Сmake.

Програма сканує ввід і з’ясовує наявність потрібних параметрів.

Створюється список структур типу “динозавр”.

Якщо не було вказано файл вводу або він не був знайдений, то список заповнюється захаркоджено.

Якщо файл таблиці був знайдений, то створюється список списків, який заповнюється значеннями з таблиці, а потім список структур заповнюється з цього списку списків, який потім очищується.

Якщо було вказане значення після -n, то зі списку структур видаляються структури, у яких числове значення менше за введене.

Потім створюється новий список списків, який заповнюється із списку структур і виводиться або у консоль, або у файл виводу, який був вказаний у вводі.

Списки очищуються і програма завершує роботу.

**Приклади результатів**

Результат у консолі:

|  |
| --- |
| /MAIN MENU/  1. Dinosaurs Menu  2. Periods Menu  0. Quit program  /DINOSAUR MENU/  Dino list (shortened):  [1]Plotosaurus  [2]Coelophysis  [3]Excison  [5]Diplodocus  [6]KJSawka  Options:  1. Dino Operations  2. Create new dino  0. Back    Enter name: gffh  Enter period: bfhbf  Enter family: dnbgfb  Enter discovery date: 2132  Enter mass: 53242  /DINOSAUR MENU/  Dino list (shortened):  [1]Plotosaurus  [2]Coelophysis  [3]Excison  [5]Diplodocus  [6]KJSawka  [7]gffh  Options:  1. Dino Operations  2. Create new dino  0. Back |

**Висновки**

Робота допомогла розібратися в роботі стандартних контейнерів, розумінні шаблонів класів та їх використанні.