**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 № 6**

**з дисципліни “ Основи програмування ”**

**тема “Формат даних XML”**

|  |  |  |
| --- | --- | --- |
| **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 користуватись Qt Creator і створювати консольні Qt проекти.

Використати наслідування класів, віртуальні функції та абстрактний клас.

Опанувати формат даних XML та навчитись його зчитувати і формувати.

Використати поліморфізм для заміни способу роботи з файловим сховищем даних.

**Постановка завдання**

Кроки виконання:

Qt проект:

Обов'язково додати у .gitignore в корені репозиторія додаткові рядки ігнорування для Qt.

Встановити Qt i Qt Creator і відкрити Qt Creator.

Створити через Qt Creator у директорії labs репозиторія новий консольний Qt проект із назвою lab6.

Налаштування створеного Qt проекту будуть розміщені у файлі lab6.pro. Відкрити файл проекту і задати версію C++17.

Скопіювати у створений проект ваше рішення лабораторної роботи №5 (файли з кодом і директорію data, без CMakeLists.txt).

(UPD-1). Перенести CSV файли з директорії data у data/csv. Змінити шлях до сховища даних використовуючи назву проекту, наприклад ../lab6/data/csv.

Це пояснюється тим, що Qt за замовчуванням збирає executable файл програми у директорії, сусідній директорії з проектом.

Тепер проект має збиратись, а зібрана програма запускатись і працювати аналогічно початковому рішенню.

Абстрактний клас і наслідування.

Перейменувати модуль storage на csv\_storage, а клас Storage на CsvStorage (лише у модулі).

Створити новий абстрактний клас Storage і публічно наслідувати CsvStorage від нього.

Змінити спосіб створення сховища з використанням upcasting (див. Додаток A).

Сховище даних XML:

(UPD-1) Створити у проекті директорію data/xml та додати у неї два XML файли (з розширенням .xml).

Використати будь-який онлайн сервіс для перетворення з CSV в XML і заповнити створені файли даними основних та додаткових сутностей (приклад див. Додаток B).

(UPD-2) Скопіювати модуль csv\_storage і назвати його xml\_storage з класом XmlStorage, що публічно наслідується від абстрактного класу Storage.

Видалити з xml\_storage все, що стосується CSV.

Реалізувати XmlStorage з використанням Qt XML класів:

Додати у файл налаштувань проекту (.pro) QT += xml як перший рядок .

Підключити у модулі xml\_storage заголовочний файл #include <QtXml>

(UPD-1) При створенні об'єкта сховища замінити його тип на XmlStorage, а шлях до файлу на ../lab6/data/xml (див. Додаток C).

**Тексти коду програм**

|  |
| --- |
| **main.cpp** |
| #include "csv.h"  #include "csv\_storage.h"  #include "xml\_storage.h"  #include "storage.h"  #include "cui.h"  using namespace std;  int main()  {  XmlStorage xml\_storage("../lab6/data/xml");  Storage \* storagex = &xml\_storage;  storagex->load();  // CsvStorage csv\_storage("../lab6/data/csv");  // Storage \* storage = &csv\_storage;  // storage->load();  Cui cui(storagex);  cui.show();  return 0;  } |

|  |
| --- |
| **storage.h** |
| #pragma once  #include <iostream>  #include <fstream>  #include <vector>  #include <sstream>  #include "optional.h"  #include "dino.h"  #include "period.h"  using namespace std;  class Storage  {  public:  virtual bool load() = 0;  virtual bool save() = 0;  // dinosaurs  virtual vector<Dino> getAllDinos(void) = 0;  virtual optional<Dino> getDinoById(int Dino\_id) = 0;  virtual bool updateDino(const Dino &dvno) = 0;  virtual bool removeDino(int Dino\_id) = 0;  virtual int insertDino(const Dino &dvno) = 0;  // periods  virtual vector<Per> getAllPers(void) = 0;  virtual optional<Per> getPerById(int Per\_id) = 0;  virtual bool updatePer(const Per &pr) = 0;  virtual bool removePer(int Per\_id) = 0;  virtual int insertPer(const Per &pr) = 0;  }; |

|  |
| --- |
| **storage.h** |
| #pragma once  #include <iostream>  #include <fstream>  #include <vector>  #include <sstream>  #include "optional.h"  #include "dino.h"  #include "period.h"  using namespace std;  class Storage  {  public:  virtual bool load() = 0;  virtual bool save() = 0;  // dinosaurs  virtual vector<Dino> getAllDinos(void) = 0;  virtual optional<Dino> getDinoById(int Dino\_id) = 0;  virtual bool updateDino(const Dino &dvno) = 0;  virtual bool removeDino(int Dino\_id) = 0;  virtual int insertDino(const Dino &dvno) = 0;  // periods  virtual vector<Per> getAllPers(void) = 0;  virtual optional<Per> getPerById(int Per\_id) = 0;  virtual bool updatePer(const Per &pr) = 0;  virtual bool removePer(int Per\_id) = 0;  virtual int insertPer(const Per &pr) = 0;  }; |

|  |
| --- |
| **csv\_storage.h** |
| #pragma once  #include <vector>  #include <string>  #include "dino.h"  #include "period.h"  #include "csv.h"  #include "optional.h"  #include "storage.h"  using std::string;  using std::vector;  class CsvStorage : public 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:  CsvStorage(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);  }; |

|  |
| --- |
| **csv\_storage.cpp** |
| #include "csv\_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 CsvStorage::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 CsvStorage::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> CsvStorage::getAllDinos()  {  return this->dinos\_;  }  optional<Dino> CsvStorage::getDinoById(int Dino\_id)  {  for(Dino & c : this->dinos\_)  {  if(c.id == Dino\_id)  {  return c;  }  }  return nullopt;  }  bool CsvStorage::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 CsvStorage::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 CsvStorage::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 CsvStorage::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> CsvStorage::getAllPers()  {  return this->pers\_;  }  optional<Per> CsvStorage::getPerById(int id)  {  for(Per & p : this->pers\_)  {  if(p.id == id)  {  return p;  }  }  return nullopt;  }  bool CsvStorage::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 CsvStorage::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 CsvStorage::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 CsvStorage::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;  } |

|  |
| --- |
| **xml\_storage.h** |
| #pragma once  #include <QtCore>  #include <QtXml>  #include <QString>  #include <QFile>  #include <QtXml>  #include <QDebug>  #include <QTextStream>  #include <string>  #include <vector>  #include "dino.h"  #include "period.h"  #include "optional.h"  #include "storage.h"  using std::string;  using std::vector;  class XmlStorage : public Storage  {  const string dir\_name\_;  vector<Dino> dinos\_;  vector<Per> pers\_;  int getNewDinoId();  int getNewPerId();  public:  XmlStorage(const string & dir\_name) : dir\_name\_(dir\_name) { }  bool load();  bool save();  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);  bool loadfromstr(string filename, int strtype);  } |

|  |
| --- |
| **xml\_storage.cpp** |
| #include "xml\_storage.h"  using namespace std;  Dino domElementToDino(QDomElement & element)  {  Dino d;  d.id = element.attributeNode("id").value().toInt();  d.name = element.attributeNode("name").value().toStdString();  d.per = element.attributeNode("period").value().toStdString();  d.fam = element.attributeNode("family").value().toStdString();  d.disc = element.attributeNode("discovery").value().toInt();  d.mass = element.attributeNode("mass").value().toFloat();  return d;  }  Per domElementToPer(QDomElement & element)  {  Per p;  p.id = element.attributeNode("id").value().toInt();  p.name = element.attributeNode("name").value().toStdString();  p.time = element.attributeNode("time").value().toFloat();  return p;  }  bool errorcheck(QString & text, QDomDocument & doc)  {  QString errorMessage;  int errorLine;  int errorCol;  bool is\_parsed = doc.setContent(text, &errorMessage, &errorLine, &errorCol);  if(!is\_parsed)  {  qDebug() << "Error parsing xml: " << errorMessage;  qDebug() << "In line: " << errorLine;  qDebug() << "In colu: " << errorCol;  return false;  }  return true;  }  bool XmlStorage::loadfromstr(string filename, int strtype)  {  QString s\_filename = QString::fromStdString(filename);  QFile s\_file(s\_filename);  bool is\_opened = s\_file.open(QFile::ReadOnly);  if (!is\_opened)  {  cout << "file wasnt found" << endl;  return false;  }  QTextStream ts(&s\_file);  QString text = ts.readAll();  QDomDocument doc;  if(!errorcheck(text, doc))  {  return false;  }  QDomElement root = doc.documentElement();  for(int i = 0; i < root.childNodes().size(); i++)  {  QDomNode node = root.childNodes().at(i);  if(node.isElement())  {  QDomElement element = node.toElement();  if(strtype == 1)  {  Dino dino = domElementToDino(element);  this->dinos\_.push\_back(dino);  }  if(strtype == 2)  {  Per per = domElementToPer(element);  this->pers\_.push\_back(per);  }  }  }  s\_file.close();  return true;  }  bool XmlStorage::load()  {  string filename = this->dir\_name\_ + "/data.xml";  if(!loadfromstr(filename, 1))  {  return false;  }  string filename2 = this->dir\_name\_ + "/data2.xml";  if(!loadfromstr(filename2, 2))  {  return false;  }  return true;  }  QDomElement dinoToDomEl(QDomDocument & doc, Dino & dino)  {  QDomElement dino\_el = doc.createElement("dino");  dino\_el.setAttribute("id", dino.id);  dino\_el.setAttribute("name", dino.name.c\_str());  dino\_el.setAttribute("period", dino.per.c\_str());  dino\_el.setAttribute("family", dino.fam.c\_str());  dino\_el.setAttribute("discovery", dino.disc);  dino\_el.setAttribute("mass", dino.mass);  return dino\_el;  }  QDomElement perToDomEl(QDomDocument & doc, Per & per)  {  QDomElement per\_el = doc.createElement("period");  per\_el.setAttribute("id", per.id);  per\_el.setAttribute("name", per.name.c\_str());  per\_el.setAttribute("time", per.time);  return per\_el;  }  bool tofile(QDomDocument & doc, string & filename)  {  QString xml\_text = doc.toString(4);  QString s\_filename = QString::fromStdString(filename);  QFile file(s\_filename);  if(!file.open(QFile::WriteOnly))  {  qDebug() << "Couldn't open file to write : " << s\_filename;  return false;  }  QTextStream ts(&file);  ts << xml\_text;  file.close();  return true;  }  bool XmlStorage::save()  {  QDomDocument doc;  QDomElement root = doc.createElement("dinos");  for(Dino & dino : this->dinos\_)  {  QDomElement dino\_el = dinoToDomEl(doc, dino);  root.appendChild(dino\_el);  }  doc.appendChild(root);  string filename = this->dir\_name\_ + "/data.xml";  tofile(doc, filename);  QDomDocument doc2;  QDomElement root2 = doc2.createElement("periods");  for(Per & per : this->pers\_)  {  QDomElement per\_el = perToDomEl(doc2, per);  root2.appendChild(per\_el);  }  doc2.appendChild(root2);  string filename2 = this->dir\_name\_ + "/data2.xml";  tofile(doc2, filename2);  return true;  }  int XmlStorage::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;  }  bool XmlStorage::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;  }  vector<Dino> XmlStorage::getAllDinos()  {  return this->dinos\_;  }  optional<Dino> XmlStorage::getDinoById(int Dino\_id)  {  for(Dino & c : this->dinos\_)  {  if(c.id == Dino\_id)  {  return c;  }  }  return nullopt;  }  int XmlStorage::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 XmlStorage::updateDino(const Dino &dvno)  {  int Dino\_id = dvno.id;  for(Dino & c : this->dinos\_)  {  if(c.id == Dino\_id)  {  c.name = dvno.name;  c.per = dvno.per;  c.fam = dvno.fam;  c.disc = dvno.disc;  c.mass = dvno.mass;  return true;  }  }  return false;  }  vector<Per> XmlStorage::getAllPers()  {  return this->pers\_;  }  optional<Per> XmlStorage::getPerById(int id)  {  for(Per & p : this->pers\_)  {  if(p.id == id)  {  return p;  }  }  return nullopt;  }  bool XmlStorage::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 XmlStorage::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 XmlStorage::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 XmlStorage::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.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;  } |

|  |
| --- |
| **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** |
| 1,Plotosaurus,Cretaceous,Mosasauridae,1941,2400.000000  2,Coelophysis,Triassic,Coelophysidae,1887,20.000000  3,Excison,Modern,Destroid,2012,75.000000  5,Diplodocus,Jurassic,Diplodocidae,1877,13000.000000  6,KJSawka,Modern,Destroid,2014,75.000000  7,Kek,Sus,Lel,228,1337.000000 |

|  |
| --- |
| **data2.csv** |
| 6,Stone,30000.000000  9,Iron,2500.000000  42,Golden,100.000000  43,Platinum,67854.000000 |

|  |
| --- |
| **data.xml** |
| <dinos>  <dino discovery="1941" period="Cretaceous" name="Plotosaurus" mass="2400" id="1" family="Mosasauridae"/>  <dino discovery="2012" period="Modern" name="Excision" mass="75" id="3" family="Destroid"/>  <dino discovery="1877" period="Jurassic" name="Diplodocus" mass="13000" id="4" family="Diplodocidae"/>  <dino discovery="1800" period="1800" name="Diego" mass="75.5" id="5" family="Brando"/>  </dinos> |

|  |
| --- |
| **data2.xml** |
| <periods>  <period time="30000" name="Stone" id="6"/>  <period time="2500" name="Iron" id="9"/>  <period time="1000" name="Golden" id="42"/>  <period time="9000" name="Platinum" id="43"/>  <period time="678995.6875" name="new" id="44"/>  </periods> |

**Робота коду**

Програма будується у QT.

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.

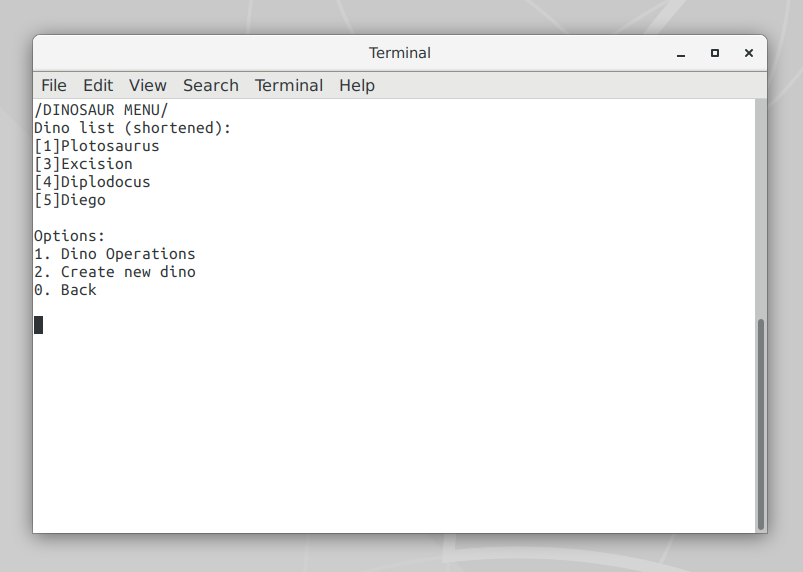
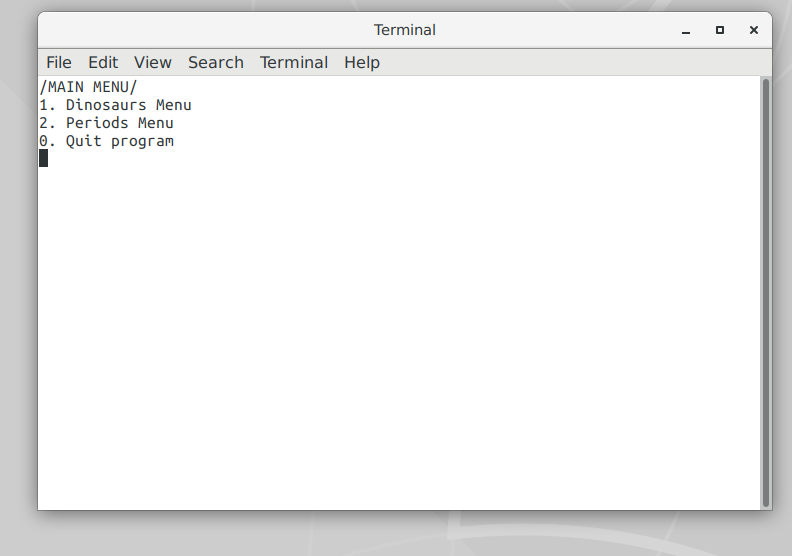
Якщо було вказане значення після -n, то зі списку структур видаляються структури, у яких числове значення менше за введене.

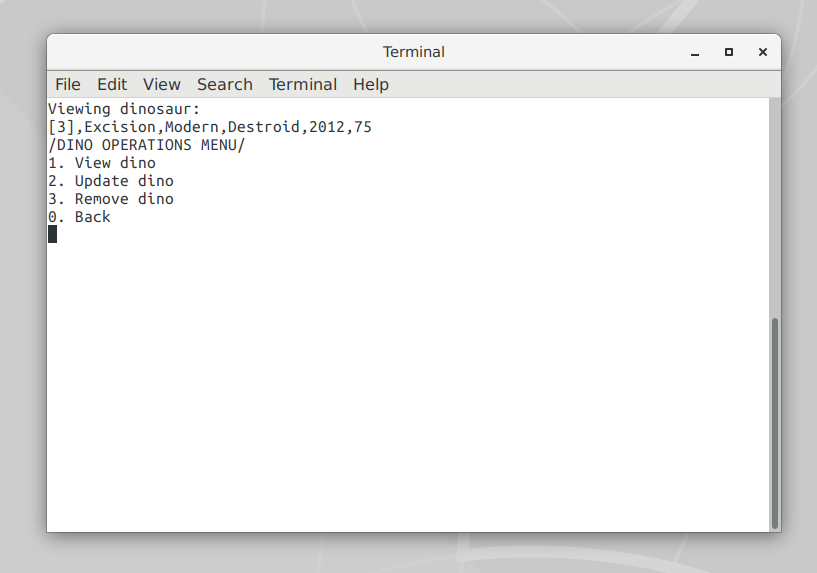
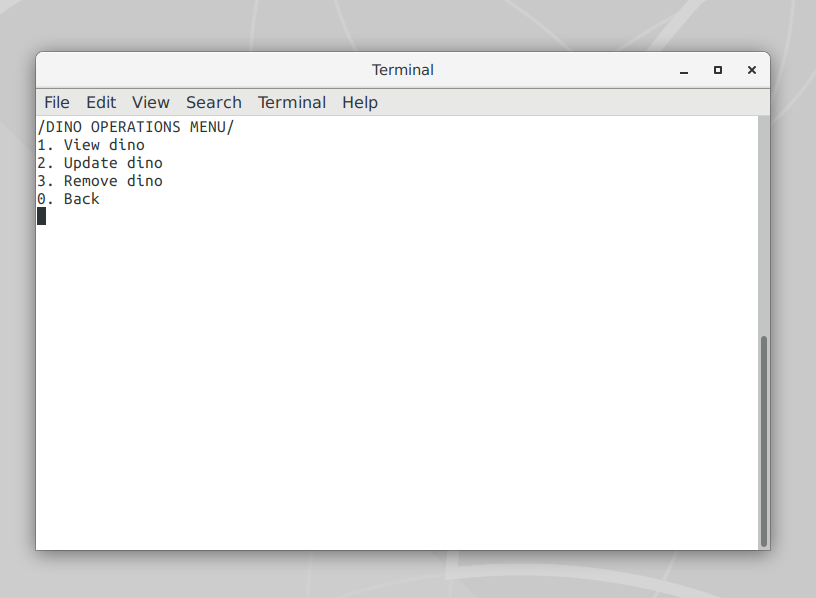
Потім створюється новий список списків, який заповнюється із списку структур і виводиться або у консоль, або у файл виводу, який був вказаний у вводі.

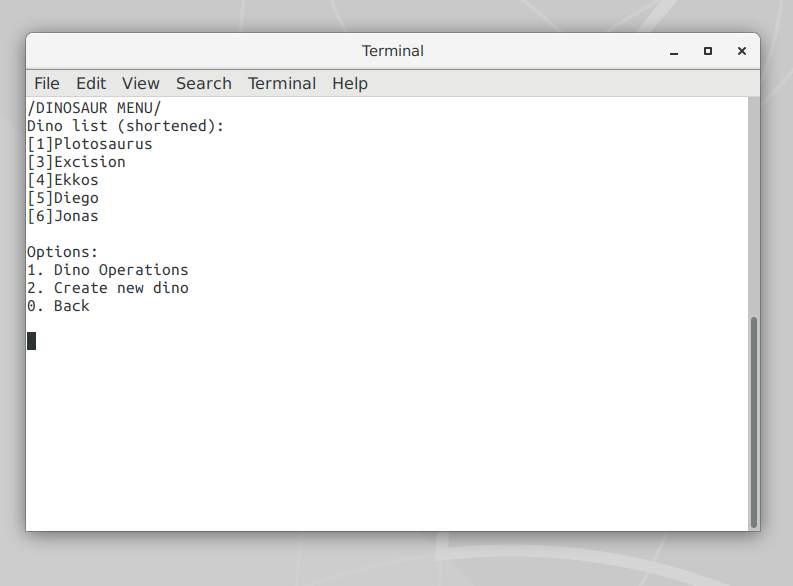
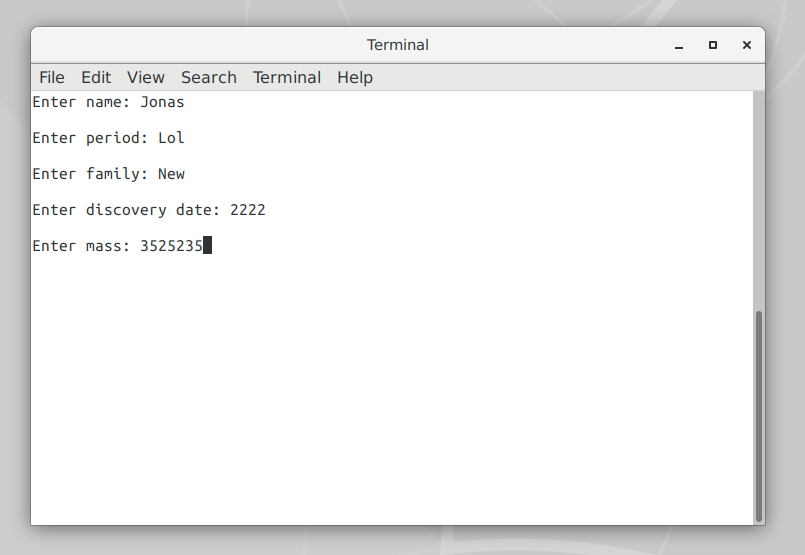
Списки очищуються і програма завершує роботу.

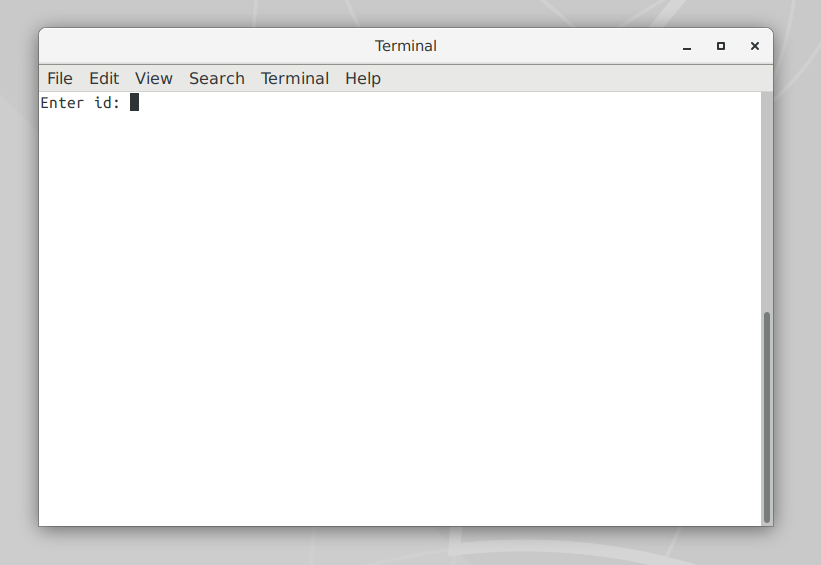
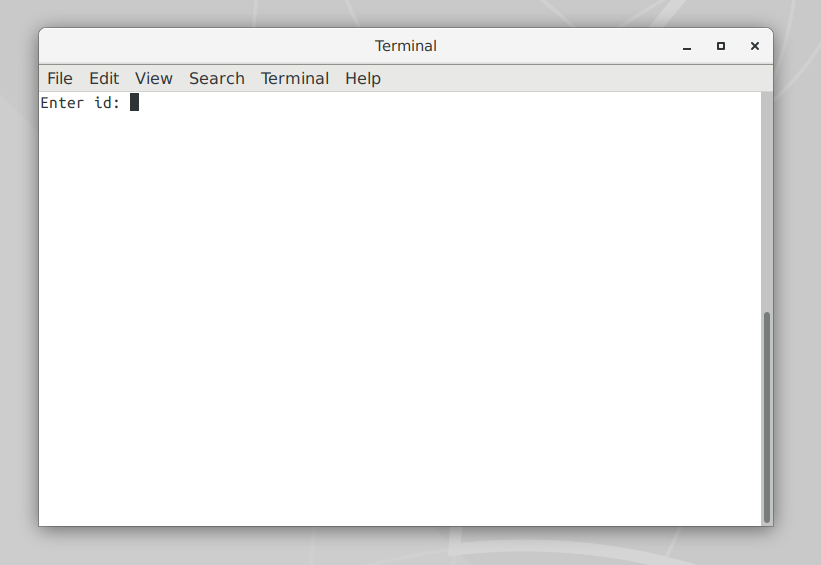
**Приклади результатів**

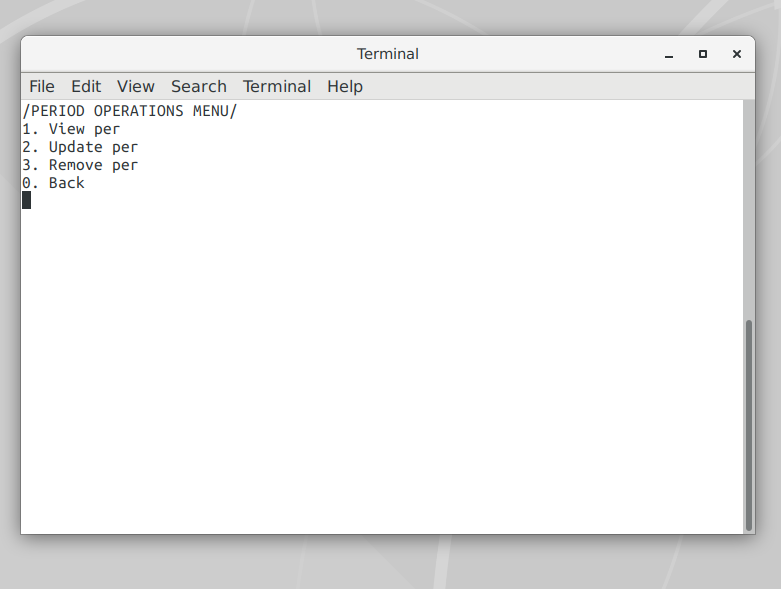
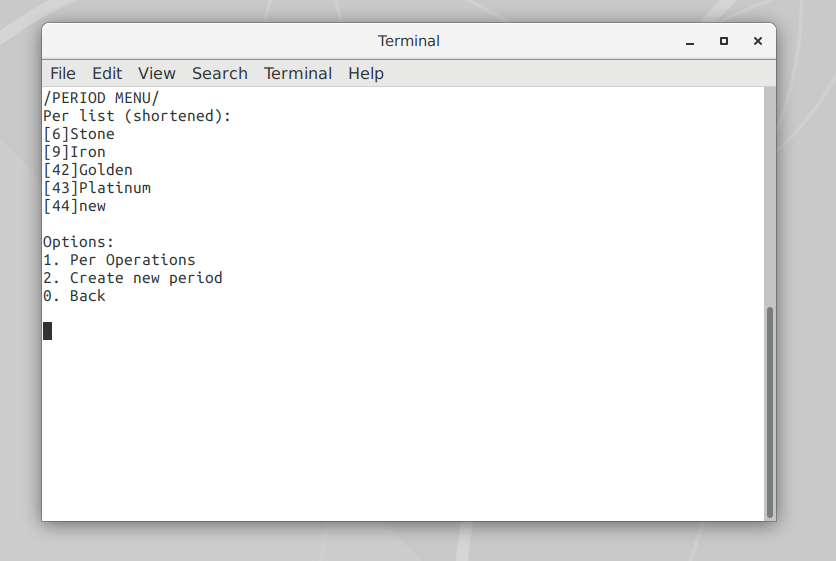
Результат у консолі:











**Висновки**

Робота допомогла розібратися в роботі XML.