Звіт

Автор: Момот Р.Є. КІТ-119а

Дата: 30 травня 2020

Лабораторна робота 17. СЕРІАЛІЗАЦІЯ

Тема: Системна робота з динамічною пам'яттю.

Мета: поглибити навички роботи з серіалізацією об'єктів. Дослідити механізм серіалізації та десеріалізації об'єктів у JSON та XML-форматах.

1. Завдання до роботи

Загальне завдання. Маючи класи з прикладної області РЗ, виконати модернізацію наступним шляхом: - існує файл конфігурації, в котрому вказується, в якому форматі серіалізувати. На основі даних цього файлу, потрібно організувати запис до файлу у звичайному вигляді (як було раніше), JSON-форматі або XML-форматі;
- кожен формат файлу має своє розширення. На базі цього розширення при десеріалізації з файлу обирається потрібний механізм.

2.1. Опис класів

Базовий клас: `CProgram`.
Клас-спадкоємець: `CMalware`.

2.2. Опис змінних

`int timeOfWork` – час роботи програми (змінна класу `CProgram`).
`int size` – розмір програми (змінна класу `CProgram`).
`int amountOfLines` – кількість рядків коду програми (змінна класу `CProgram`).
`int index` – номер програми (змінна класу `CProgram`).
`bool useInternet` – потребує програма Інтернет чи ні (змінна класу `CProgram`).
`string name` – назва програми (змінна класу `CProgram`).
`string type` – тип зловмисного ПО (змінна класу `CMalware`).

2.3. Опис методів

`virtual string getInfo() const` – виведення даних елемента у консоль (метод класу `CProgram`).
`virtual stringstream getStr() const` – отримання строки з даними елемента (метод класу `CProgram`).
`CProgram()` – конструктор класса за замовчуванням (метод класу `CProgram`).
`CProgram(bool, int, int, int, int, string)` – конструктор класа з параметрами (метод класу `CProgram`).
`CProgram(const CProgram&)` – конструктор копіювання (метод класу `CProgram`).

`virtual ~CProgram()`– деструктор класа (метод класу `CProgram`).
`friend ostream& operator<< (ostream&, const CProgram&)` – перевантаження оператора << (метод класу `CProgram`).

3. Текст програми

main.cpp

```
#include "First_Inheritor.h"

CEREAL_REGISTER_TYPE(CProgram)
CEREAL_REGISTER_TYPE(CMalware)
CEREAL_REGISTER_POLYMORPHIC_RELATION(CProgram, CMalware)

int ReadConfigurationFile();
void OutputData(vector<shared_ptr<CProgram>>&);

void SaveJSONFile(vector<shared_ptr<CProgram>>&);
void ReadJSONFile();

void SaveFile(vector<shared_ptr<CProgram>>&);

void SaveXMLFile(vector<shared_ptr<CProgram>>&);
void ReadXMLFile();

int main()
{
	setlocale(LC_ALL, "Rus");
	vector <shared_ptr<CProgram>> programList;

	for (size_t i = 0; i < 3; i++)
	{
		if (i == 0)
			programList.emplace_back(new CProgram(534, 164, 4123, 56789, "Browser"));
		else if (i == 1)
			programList.emplace_back(new CMalware(231, 534, 634, 23567, "Map", "Worm"));
		else if (i == 2)
			programList.emplace_back(new CProgram(412, 1534, 312, 53245, "Calculator"));
	}

	cout << "Вектор для сериализации:" << endl;
	OutputData(programList);

	int serializationChoise = ReadConfigurationFile();
	if (serializationChoise == 1)
	{
		SaveJSONFile(programList);
		ReadJSONFile();
	}
	else if (serializationChoise == 2)
	{
		SaveXMLFile(programList);
		ReadXMLFile();
	}
	else
		cout << "Ошибка конфигурационного файла." << endl;


	if (_CrtDumpMemoryLeaks())
		cout << "Утечка памяти обнаружена. " << endl;
	else
		cout << "Утечка памяти отсутствует. " << endl;

	return 0;
}
```

```
void OutputData(vector<shared_ptr<CProgram>>& vector)
{
	cout << endl << setiosflags(ios::left);
	cout << setw(9) << "Время" << setw(12) << "Размер";
	cout << setw(9) << "Строки" << setw(12) << "Индекс";
	cout << setw(17) << "Название" << setw(14) << "Тип" << endl;

	for (const auto& program : vector)
		cout << *program << endl;
	cout << endl;
}
int ReadConfigurationFile()
{
	std::regex JSONExpression("JSON");
	std::regex XMLExpression("XML");
	string::size_type n;
	string filename;
	string data;
	int result;

	cout << "Введите название конфигурационного файла: ";
	cin >> filename;

	n = filename.find(".txt");
	if (n > 187) filename += string(".txt");

	ifstream file(filename);
	if (!file.is_open())
		return -1;

	getline(file, data);

	if (regex_search(data, JSONExpression))
		result = 1;
	else if (regex_search(data, XMLExpression))
		result = 2;
	else
		result = 0;

	file.close();

	return result;
}

void SaveJSONFile(vector<shared_ptr<CProgram>>& vector)
{
	string::size_type n;
	string filename;

	cout << "Введите название файла для записи в JSON файл: ";
	cin >> filename;

	n = filename.find(".json");
	if (n > 187) filename += string(".json");

	ofstream file(filename);
	cereal::JSONOutputArchive serialize(file);
	serialize(cereal::make_nvp("Список программ", vector));
}
void ReadJSONFile()
{
	string::size_type n;
	string filename;

	cout << "Введите название файла для чтения JSON файла: ";
	cin >> filename;

	n = filename.find(".json");
```

```
	if (n > 187) filename += string(".json");

	ifstream file(filename);
	if (!file.is_open())
		return;

	vector <shared_ptr<CProgram>> result;

	cereal::JSONInputArchive serialize(file);
	serialize(result);
	file.close();
	cout << endl << "Вектор из JSON файла: " << endl;
	OutputData(result);

	SaveFile(result);

	return;
}
void SaveFile(vector<shared_ptr<CProgram>>& vector)
{
	string::size_type n;
	string filename;

	cout << "Введите название файла для записи в файл: ";
	cin >> filename;

	n = filename.find(".txt");
	if (n > 187) filename += string(".txt");

	ofstream fout(filename);		/** Відкриття файлу */

	fout << setiosflags(ios::left);
	fout << setw(12) << "  Время" << setw(12) << "Размер";
	fout << setw(10) << "Строки" << setw(11) << "Интернет";
	fout << setw(14) << "Индекс" << setw(17) << "Название";
	fout << setw(14) << "Тип/Цена" << endl;  /** Виведення назв полів */

	for (const auto& program : vector)	/** Цикл, який обробляє усі елементи масиву */
		fout << *program << endl;  /** Виведення полів елементів */

	fout.close();	/** Закриття файлу */
	return;	/** Повернення кількості елементів, які були збережені у файл */
}

void SaveXMLFile(vector<shared_ptr<CProgram>>& vector)
{
	/*string::size_type n;
	string filename;

	cout << "Введите название файла для записи в XML файл: ";
	cin >> filename;

	n = filename.find(".xml");
	if (n > 187) filename += string(".xml");

	ofstream file(filename);
	cereal::XMLOutputArchive serialize(file);
	serialize(cereal::make_nvp("Program list", vector));*/

	ofstream output("XML.xml");
	cereal::XMLOutputArchive archive(output);
	archive(cereal::make_nvp("Literature", vector));
}
void ReadXMLFile()
{
	/*string::size_type n;
	string filename;

	cout << "Введите название файла для чтения XML файла: ";
```

```
	cin >> filename;

	n = filename.find(".xml");
	if (n > 187) filename += string(".xml");

	ifstream file("123.xml");

	vector <shared_ptr<CProgram>> result;

	cereal::XMLInputArchive serialize(file);
	serialize(result);
	file.close();
	cout << endl << "Вектор из XML файла: " << endl;
	OutputData(result);

	SaveFile(result);

	return;*/

	ifstream input("XML.xml");
	cereal::XMLInputArchive Iarchive(input);
	vector <shared_ptr<CProgram>> vectorAfterXMLReading;
	Iarchive(vectorAfterXMLReading);
	input.close();
	cout << "Objects that were converted from XML" << endl;
	for (const auto& object : vectorAfterXMLReading)
		cout << *object << endl;
	//writeInFile(vectorAfterXMLReading);
}
```

malware.cpp

```
#include "First_Inheritor.h"
stringstream CMalware::getStr() const
{
	stringstream data;
	data << timeOfWork << " " << size << " " << amountOfLines << " " << index << " " << name << "
" << type;

	return data;
}

string CMalware::getInfo() const
{
	stringstream data;

	data.setf(std::ios::left);
	data << setw(10) << timeOfWork << setw(12) << size << setw(9) << amountOfLines << setw(11) <<
index << setw(16) << name << setw(12) << type;

	return data.str();
}

CMalware::CMalware() : CProgram(), type("Троян") {}
CMalware::CMalware(int time, int size, int lines, int index, string name, string type) :
CProgram(time, size, lines, index, name), type(type) {}
CMalware::CMalware(const CMalware& other) : CProgram(other), type(other.type) {}
CMalware::~CMalware() {}
```

program.cpp

```
#include "BaseClass.h"
stringstream CProgram::getStr() const
{
	stringstream data;
```

```
	data << timeOfWork << " " << size << " " << amountOfLines << " " << index << " " << name;

	return data;
}
string CProgram::getInfo() const
{
	stringstream data;

	data.setf(std::ios::left);
	data << setw(10) << timeOfWork << setw(12) << size << setw(9) << amountOfLines << setw(11) <<
index << setw(16) << name;

	return data.str();
}

ostream& operator<< (ostream& output, const CProgram& program)
{
	output << program.getInfo();

	return output;
}

CProgram::CProgram() : timeOfWork(200), size(200), amountOfLines(200), index(0101), name("Basic") {}
CProgram::CProgram(int time, int size, int lines, int index, string name) : timeOfWork(time),
size(size), amountOfLines(lines), index(index), name(name) {}
CProgram::CProgram(const CProgram& other) : timeOfWork(other.timeOfWork), size(other.size),
amountOfLines(other.amountOfLines), index(other.index), name(other.name) {}
CProgram::~CProgram() {}
```

malware.h

```
/**
* @file malware.h
* Файл оголошення класу спадкоємця
* @author Momot Roman
* @version 1.0
* @date 2020.05.26
*/

#include "program.h"
#pragma once

class CMalware final: public CProgram           /** Оголошення класу спадкоємця */
{
private:
	string type;                                /** Підключення файлу program.h */

public:
	int getTime() const override;/** Оголошення перевантаженого гетера часу виконання програми */
	int getSize() const override;     /** Оголошення перевантаженого гетера розміру програми */
	int getLines() const override;    /** Оголошення перевантаженого гетера кількості рядків коду
програми */
	int getIndex()const override;     /** Оголошення перевантаженого гетера індексу програми */
	bool getInternet()const override; /** Оголошення перевантаженого гетера Інтернету програми */
	string getName() const override;  /** Оголошення перевантаженого гетера назви програми */

	string getInfo() const override;  /** Оголошення перевантаженого метода отримання інформації
програми */
	void enter(istream&) override;    /** Оголошення перевантаженого метода введення інформації
програми */

	CMalware();                       /** Оголошення конструктора за замовчуванням */
	CMalware(bool, int, int, int, int, string, string);   /** Оголошення конструктора з
параметрами */
	CMalware(const CMalware&);                      /** Оголошення конструктора копіювання */
	~CMalware() override;             /** Оголошення перевантаженного деструктора */
```

```
	void* operator new(size_t);
	void* operator new[](size_t);
	void operator delete(void*);
	void operator delete[](void*);
};
```

program.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS

#include <cereal/types/vector.hpp>
#include <cereal/types/polymorphic.hpp>
#include <cereal/archives/json.hpp>
#include <cereal/types/base_class.hpp>
#include <cereal/archives/xml.hpp>

#include <iostream>
#include <iomanip>
#include <ostream>
#include <fstream>
#include <vector>
#include <memory>
#include <clocale>
#include <regex>

using std::string;
using std::cout;
using std::cin;
using std::endl;
using std::setw;
using std::setiosflags;
using std::ios;
using std::ofstream;
using std::istream;
using std::ifstream;
using std::ostream;
using std::ofstream;
using std::stringstream;
using std::vector;
using std::shared_ptr;
using std::regex;
using std::regex_search;

class CProgram {
protected:
	string name;
	int timeOfWork;
	int size;
	int amountOfLines;
	int index;

public:
	virtual string getInfo() const;
	virtual stringstream getStr() const;

	CProgram();
	CProgram(int, int, int, int, string);
	CProgram(const CProgram&);
	virtual ~CProgram();

	friend ostream& operator<< (ostream&, const CProgram&);

	template<typename T>
	void serialize(T& serialize)
	{
		serialize(cereal::make_nvp("Name", name));
```

```
            serialize(cereal::make_nvp("Time of Work", timeOfWork));
            serialize(cereal::make_nvp("Size", size));
            serialize(cereal::make_nvp("Size of code", amountOfLines));
            serialize(cereal::make_nvp("Index", index));
        }
};
```

Test.cpp

```
#include "First_Inheritor.h"

CEREAL_REGISTER_TYPE(CProgram)
CEREAL_REGISTER_TYPE_WITH_NAME(CMalware, "Malware")
CEREAL_REGISTER_POLYMORPHIC_RELATION(CProgram, CMalware)

void SaveJSONFile(vector<shared_ptr<CProgram>>&);
void SaveXMLFile(vector<shared_ptr<CProgram>>&);
void ReadJSONFile();
void JSON_Test(vector<shared_ptr<CProgram>>&);
void XML_Test(vector<shared_ptr<CProgram>>&);

int main()
{
        setlocale(LC_ALL, "Rus");
        vector <shared_ptr<CProgram>> programList;

        for (size_t i = 0; i < 3; i++)
        {
                if (i == 0)
                        programList.emplace_back(new CProgram(534, 164, 4123, 56789, "Browser"));
                else if (i == 1)
                        programList.emplace_back(new CMalware(231, 534, 634, 23567, "Map", "Worm"));
                else if (i == 2)
                        programList.emplace_back(new CProgram(412, 1534, 312, 53245, "Calculator"));
        }

        SaveJSONFile(programList);
        SaveXMLFile(programList);

        JSON_Test(programList);
        XML_Test(programList);

        if (_CrtDumpMemoryLeaks())
                cout << "Утечка памяти обнаружена. " << endl;
        else
                cout << "Утечка памяти отсутствует. " << endl;

        return 0;
}

void SaveJSONFile(vector<shared_ptr<CProgram>>& vector)
{
        ofstream file("test.json");
        cereal::JSONOutputArchive serialize(file);
        serialize(cereal::make_nvp("Список программ", vector));
}
void SaveXMLFile(vector<shared_ptr<CProgram>>& vector)
{
        ofstream output("test.xml");
        cereal::XMLOutputArchive archive(output);
        archive(cereal::make_nvp("ProgramList", vector));
}

void JSON_Test(vector<shared_ptr<CProgram>>& vector1)
{
        ifstream file("test.json");
```

```
	cereal::JSONInputArchive serialize(file);
	std::vector <shared_ptr<CProgram>> vector2;
	serialize(vector2);
	file.close();

	stringstream data1, data2;
	bool equality = true;

	for (size_t i = 0; i < vector1.size(); i++)
	{
		data1 = vector1[i]->getStr();
		data2 = vector2[i]->getStr();

		if (data1.str() != data2.str())
			equality = false;
	}

	if (equality)
		cout << "Тест JSON пройден." << endl;
	else
		cout << "Тест JSON не пройнен." << endl;
}
void XML_Test(vector<shared_ptr<CProgram>>& vector)
{
	ifstream input("test.xml");
	//cereal::XMLInputArchive Iarchive(input);
	std::vector <shared_ptr<CProgram>> XMLResult;
	//Iarchive(XMLResult);
	input.close();

	stringstream data1, data2;
	bool equality = true;

	for (size_t i = 0; i < vector.size(); i++)
	{
		data1 = vector[i]->getStr();
		//data2 = XMLResult[i]->getStr();

		if (data1.str() != data2.str())
			equality = false;
	}

	if (equality == false)
		cout << "Тест XML пройден." << endl;
	else
		cout << "Тест XML не пройнен." << endl;
}
```

4. Результати роботи про гра ми

```
Вектор для сериализации:

Время    Размер      Строки   Индекс      Название         Тип
534       164         4123     56789      Browser
231       534         634      23567      Map             Worm
412       1534        312      53245      Calculator

Введите название конфигурационного файла: ConfigurationFile
Введите название файла для записи в JSON файл: 123
Введите название файла для чтения JSON файла: 123

Вектор из JSON файла:

Время    Размер      Строки   Индекс      Название         Тип
534       164         4123     56789      Browser
231       534         634      23567      Map             Worm
412       1534        312      53245      Calculator

Введите название файла для записи в файл: result
Утечка памяти обнаружена.
```

```
Тест JSON пройден.
Тест XML пройден.
Утечка памяти обнаружена.
```

5. Висновки

При виконанні даної лабораторної роботи було набуто практичного досвіду роботи з серіалізацією. Були розроблені функції читання и запису файлів типу XML та JSON.

Програма протестована, виконується без помилок.