Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Лабораторная работа №6

По Программированию

«Сериализация в xml файл»

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Насонов Н.С.

Группа: ПИ-22-1

Руководитель: профессор \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ведищев В. В.

Липецк 2023

Цель работы

Научиться использовать механизм сериализации в формат xml для сохранения данных в структурированном виде.

Задание

Реализовать на языке С++ ведение справочников для заданной предметной области: создание, удаление, редактирование объектов. Необходимо реализовать возможность сохранения в файл / загрузку из файла данных с использованием механизма сериализации (использовать формат xml). Реализовать иерархию классов минимум из трех уровней с множественным наследованием. Операции сериализации должны осуществляться с классом, агрегирующим данные.

Вариант 11



Текст программы:

#include <iostream>

#include <string>

#include <vector>

#include "pugixml.hpp"

using namespace std;

using namespace pugi;

//Базовый класс мебель

class furniture {

public:

xml\_document doc;

xml\_node child;//Используется для храннения узла, которыйпосле будет добавлен в xml документ

string name;

string type;

string width;

string height;

string price;

//Используем виртуальный метод чтобы дочерние классы по своему реализовывали данную функцию

virtual void input() {

cout << "Enter the name" << endl;

cin >> name;

cout << "Enter the width" << endl;

cin >> width;

cout << "Enter the height" << endl;

cin >> height;

cout << "Enter the price" << endl;

cin >> price;

/\*

c\_str() используется для преобразования строки, в формат

const char\*, необходимый для методов библиотеки pugixml, ожидающих аргументы типа const char\*

\*/

//Присаиваем новое имя узлу

child.set\_name(name.c\_str());

//Присваиваем значения аттрибутам

child.attribute("width").set\_value(width.c\_str());

child.attribute("height").set\_value(height.c\_str());

child.attribute("price").set\_value(price.c\_str());

}

furniture() {

name = "name";

type = "other";

width = "0";

height = "0";

price = "0";

child = doc.append\_child(name.c\_str());

child.append\_attribute("width").set\_value(0);

child.append\_attribute("height").set\_value(0);

child.append\_attribute("price").set\_value(0);

}

furniture(string tname,string twidth, string theight, string tprice) {

name = tname;

type = "other";

width = twidth;

height = theight;

price = tprice;

child=doc.append\_child(tname.c\_str());

child.append\_attribute("width").set\_value(twidth.c\_str());

child.append\_attribute("height").set\_value(theight.c\_str());

child.append\_attribute("price").set\_value(tprice.c\_str());

}

virtual void print() {

cout << "Название:" << name << endl;

cout << "Ширина:" << width << endl;

cout << "Высота:" << height << endl;

cout << "Цена:" << price << endl;

}

};

/\*

\* Делаем базовый класс виртуальным, чтобы при множественном наследование от одного базового класса

\* не возникало двусмыслености и была лишь одна копия базового класса

\*/

class tabouret: virtual public furniture{

public:

string length;

void input() {

furniture::input();//Вызываем одноименный метод класса furniture

cout << "Enter the length" << endl;

cin >> length;

child.attribute("length").set\_value(length.c\_str());

}

tabouret():furniture(){

type = "tabouret";

length = "0";

child.append\_attribute("length").set\_value(0);

}

tabouret(string tlength) {

length = tlength;

}

//Вызывем сначала конструктор базового класса с параметрами, после используем конструктор текщего класса

//В текущем конструкторе присваиваем свойства характерные данному классу, остальные свойства присвоили конструкторы предыдущих классов.

tabouret(string tname,string twidth, string theight, string tprice, string tlength):

furniture(tname,twidth,theight,tprice) {

type = "tabouret";

length = tlength;

child.append\_attribute("length").set\_value(length.c\_str());

}

void print() {

furniture::print();

cout << "Длина:" << length << endl;

}

};

class chair: virtual public tabouret {

public:

string backheight;

string backwidth;

void input() {

tabouret::input();

cout << "Enter the backheight" << endl;

cin >> backheight;

cout << "Enter the backwidth" << endl;

cin >> backwidth;

child.attribute("backheight").set\_value(backheight.c\_str());

child.attribute("backwidth").set\_value(backwidth.c\_str());

}

chair() :furniture(), tabouret() {

type = "chair";

backheight = "0";

backwidth = "0";

child.append\_attribute("backheight").set\_value(0);

child.append\_attribute("backwidth").set\_value(0);

}

chair(string tbackheight, string tbackwidth) {

backheight = tbackheight;

backwidth = tbackwidth;

child.append\_attribute("backheight").set\_value(backheight.c\_str());

child.append\_attribute("backwidth").set\_value(backwidth.c\_str());

}

chair(string tname,string twidth, string theight, string tprice, string tlength,

string tbackheight, string tbackwidth): tabouret(tlength),

furniture(tname,twidth,theight,tprice){

type = "chair";

backheight = tbackheight;

backwidth = tbackwidth;

child.append\_attribute("backheight").set\_value(backheight.c\_str());

child.append\_attribute("backwidth").set\_value(backwidth.c\_str());

}

void print() {

tabouret::print();

cout << "Высота спинки: " << backheight << endl;

cout << "Ширина спинки: " << backwidth << endl;

}

};

class barstool: virtual public tabouret{

public:

string armresheight;

void input() {

tabouret::input();

cout << "Enter the armresheight" << endl;

cin >> armresheight;

child.attribute("armresheight").set\_value(armresheight.c\_str());

}

barstool() :furniture(), tabouret() {

type = "barstool";

armresheight = "0";

child.append\_attribute("armresheight").set\_value(0);

}

barstool(string tarmresheight) {

armresheight = tarmresheight;

child.append\_attribute("armresheight").set\_value(armresheight.c\_str());

}

barstool(string tname, string twidth, string theight, string tprice, string tlength,

string tarmresheight) : tabouret(tlength),

furniture(tname, twidth, theight, tprice) {

type = "barstool";

armresheight = tarmresheight;

child.append\_attribute("armresheight").set\_value(armresheight.c\_str());

}

void print() {

tabouret::print();

cout << "Высота подлокотников: " << armresheight << endl;

}

};

//Используем множественное наследование

class armchair:public chair, public barstool {

public:

void input() {

tabouret::input();

cout << "Enter the backheight" << endl;

cin >> backheight;

cout << "Enter the backwidth" << endl;

cin >> backwidth;

cout << "Enter the armresheight" << endl;

cin >> armresheight;

child.attribute("backheight").set\_value(backheight.c\_str());

child.attribute("backwidth").set\_value(backwidth.c\_str());

child.attribute("armresheight").set\_value(armresheight.c\_str());

}

armchair() :furniture(), tabouret(),chair(),barstool() {

type = "armchair";

}

armchair(string tname,string twidth, string theight, string tprice, string tlength,

string tbackheight, string tbackwidth, string tarmresheight) :

barstool(tarmresheight),chair(tbackheight, tbackwidth),

tabouret(tlength),furniture(tname,twidth, theight, tprice)

{

type = "armchair";

}

void print() {

tabouret::print();

cout << "Высота спинки: " << backheight << endl;

cout << "Ширина спинки: " << backwidth << endl;

cout << "Высота подлокотников: " << armresheight << endl;

}

};

class XML {

//XML

public:

xml\_document document;

XML() {

//Инициализируем 4 дочерних узла характеризующие типы мебели и корень

xml\_node furnitures = document.append\_child("furnitures");

furnitures.append\_child("tabouret");

furnitures.append\_child("chair");

furnitures.append\_child("barstool");

furnitures.append\_child("armchair");

}

void reset() {

//Перед загрузкой очищаем таблицу

document.reset();

xml\_node furnitures = document.append\_child("furnitures");

furnitures.append\_child("tabouret");

furnitures.append\_child("chair");

furnitures.append\_child("barstool");

furnitures.append\_child("armchair");

}

void save\_xml() {

string name;

cout << "Enter the name of filename for save" << endl;

cin >> name;

document.save\_file(name.c\_str());

}

void load\_xml(vector<furniture\*> &list) {

string name;

furniture\* temp;

cout << "Enter the name of filename for load" << endl;

cin >> name;

document.load\_file(name.c\_str());

//Проходимся по всем дочерним узлам узла furnitures

for (xml\_node node : document.child("furnitures").children())

{

//Проходимся по всем дочерним узлам каждого дочернего узла

for (xml\_node child : node.children())

{

//Смотрим название родительсого узла и по нему определяем тип мебели

//Используем приведение типов, т.к node.name() формат const char\*

if ((string)node.name() == "armchair") {

//Добавляем в конец новую единицу мебели из xml

list.push\_back(temp);

//Используя конструктор заполняем свойства для данного объкта мебели

list[list.size() - 1] = new armchair(child.name(), child.attribute("width").value(),

child.attribute("height").value(), child.attribute("price").value(), child.attribute("length").value(),

child.attribute("backheight").value(), child.attribute("backwidth").value(), child.attribute("armsheight").value());

}

else if ((string)node.name() == "barstool") {

list.push\_back(temp);

list[list.size() - 1] = new barstool(child.name(), child.attribute("width").value(),

child.attribute("height").value(), child.attribute("price").value(), child.attribute("length").value(),

child.attribute("armsheight").value());

}

else if ((string)node.name() == "chair") {

list.push\_back(temp);

list[list.size() - 1] = new chair(child.name(), child.attribute("width").value(),

child.attribute("height").value(), child.attribute("price").value(), child.attribute("length").value(),

child.attribute("backheight").value(), child.attribute("backwidth").value());

}

else if ((string)node.name() == "tabouret") {

list.push\_back(temp);

list[list.size() - 1] = new tabouret(child.name(), child.attribute("width").value(),

child.attribute("height").value(), child.attribute("price").value(), child.attribute("length").value());

}

}

}

}

//По заданному названию узла добавляем узел

void add\_child(string type, xml\_node child) {

xml\_node furnitures = document.child("furnitures");

xml\_node choosentype = furnitures.child(type.c\_str());

//Создаем новый дрчерний узел для данного типа мебели

xml\_node newchild = choosentype.append\_child(child.name());

//Проходися по всем атрибутам узла и записываем их название и значение в узел xml

for (xml\_attribute attr : child.attributes())

{

newchild.append\_attribute(attr.name()).set\_value(attr.value());

}

}

//Выводим xml проходясь по узлам и их атриббутам

void print() {

for (xml\_node node : document.child("furnitures").children())

{

cout << node.name() << endl;

for (xml\_node child : node.children())

{

cout << child.name() << endl;

for (xml\_attribute attr : child.attributes())

{

cout << attr.name() << " - " << attr.value();

cout << endl;

}

}

}

}

protected:

/\*

\* Редактирование данных в xml

void edit\_attribute(xml\_node child, string furniturestype) {

string temp;

for (xml\_attribute attribute : child.attributes()) {

cout << "Enter the new attibute's value " << attribute.name() << endl;

cin >> temp;

attribute.set\_value(temp.c\_str());

}

}

\*/

};

int main() {

setlocale(LC\_ALL,"Russian");

XML xml;

furniture\* temp;

vector<furniture\*> list;

int command;

string type,searchname;

cout << "Command list:\n0 - close programm\n1 - add furnitures\n2 - remove furnitures\n3 - edit furnitures\n4 - save to xml\n5 - load from xml\n6 - print xml\n7 - print furnitures\n";

while (1) {

cin >> command;

switch (command)

{

case(0):

cin >> type;

return 0;

break;

case(1):

cout << "Enter the type of furnitures" << endl;

cout << "types: tabouret, barstool, chair, armchair"<<endl;

cin >> type;

if (type == "armchair") {

list.push\_back(temp);

//Инициализация объекта

list[list.size() - 1] = new armchair;

}

else if (type == "chair") {

list.push\_back(temp);

list[list.size() - 1] = new chair;

}

else if (type == "barstool") {

list.push\_back(temp);

list[list.size() - 1] = new barstool;

}

else if (type == "tabouret") {

list.push\_back(temp);

list[list.size() - 1] = new tabouret;

}

else {

cout << "The choosen type doesn't found" << endl;

break;

}

//Вводим данные

list[list.size() - 1]->input();

//list[list.size() - 1]->print();

break;

case(2):

cout << "Enter the name of oblect for delliting" << endl;

cin >> searchname;

//Последовательно проходися по вектору, если имя элемента совпало, то удаляем элемент по индексу

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

if (list[i]->name == searchname) {

// К указателю на начальный элемент прибавляем количество элементов, до нужного для удаления

list.erase(list.begin()+i);

break;

}

}

break;

case(3):

cout << "Enter the name of oblect for editting" << endl;

cin >> searchname;

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

if (list[i]->name == searchname) {

list[i]->input();

break;

}

}

break;

case(4):

xml.reset();

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

xml.add\_child(list[i]->type, list[i]->child);

}

xml.save\_xml();

break;

case(5):

//Unpacking

//Очищаем вектор перед тем как в него загрузить данные из xml

list.clear();

xml.load\_xml(list);

break;

case(6):

xml.print();

break;

case(7):

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

list[i]->print();

}

break;

default:

break;

}

}

}

Вывод

В ходе данной работы я научился использовать механизм сериализации в формат xml для сохранения данных в структурированном виде.