Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

ЗВІТ

про виконання комп’ютерного практикуму № 7

з дисципліни

“Об’єктно-орієнтоване програмування”

Тема: STD И ДРУГИЕ ВОЗМОЖНОСТИ С++

|  |  |  |
| --- | --- | --- |
| Прийняв: |  | Виконав: |
| Головченко М.М. |  | студент 2-го курсу  гр. ІП-52 ФІОТ  Набоков Е.М |

Київ 2016

ЗМІСТ

[1 ПОСТАНОВКА ЗАДАЧІPAGEREF \_Toc462750603 \h3](#_Toc462750603)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІPAGEREF \_Toc462750604 \h4](#_Toc462750604)

[ЗМІСТ 2](#_Toc469438121)

[1 ПОСТАНОВКА ЗАДАЧІ 3](#_Toc469438122)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 4](#_Toc469438123)

[3 АЛГОРИТМ 5](#_Toc469438124)

[3.1 Медогляд 5](#_Toc469438125)

[3.2 Результати дня 5](#_Toc469438126)

[4 ДІАГРАМА КЛАСІВ 6](#_Toc469438127)

[5 ВИХІДНИЙ КОД 7](#_Toc469438128)

[6 ВІДЕОКОПІЯ РЕЗУЛЬТАТУ 21](#_Toc469438129)

[ВИСНОВОК 22](#_Toc469438130)

[3 АЛГОРИТМPAGEREF \_Toc462750605 \h5](#_Toc462750605)

[4 ДІАГРАММА КЛАСІВPAGEREF \_Toc462750606 \h8](#_Toc462750606)

[5 ВИХІДНИЙ КОДPAGEREF \_Toc462750607 \h9](#_Toc462750607)

[6 ВІДЕОКОПІЯ РЕЗУЛЬТАТУPAGEREF \_Toc462750608 \h20](#_Toc462750608)

[7 ВИСНОВОКPAGEREF \_Toc462750609 \h21](#_Toc462750609)

1 ПОСТАНОВКА ЗАДАЧІ

Согласно варианту создать набор связанных классов для хранения информации о предметной области. Предусмотреть в них необходимые атрибуты, конструкторы, методы и т.д. Разработать вариативный шаблонный класс согласно варианту, который будет хранить в виде коллекции конкретный набор данных для предметной области (данные унаследовать от ранее созданного набора классов). Создать динамический массив таких коллекций и выполнить для него задание согласно варианту.

1. Детский садик. Для вычисления платы за пребывание детей в детском саду утром воспитателями производится учёт поступающих детей. Собранные сведения о пришедших детях передаются заведующей садиком; на основании этих сведений формируется квитанция об оплате (за месяц), которая передаётся родителям через воспитателей. После оплаты квитанция приносится воспитателям. Если в течении 10 дней квитанция не оплачивается, то ребёнок в детский садик не принимается. Сведения об отсутствующих детях передаются медицинской сестре. Те дети, которые отсутствовали более трёх дней, принимаются в садик только при наличии медицинской справки о здоровье. Сформировать коллекцию данных с информацией о посещаемости детьми детского сада. Задание. Отобразить информацию о детях, которым необходимо принести медицинскую справку.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Шаблон - это предписание для создания класса, в котором один или несколько типов либо значений параметризованные.

В программировании шаблон с переменным числом аргументов называется вариативным.

На рис. 4.1 вказана діаграма класів програми.

3 АЛГОРИТМ

3.1 Отчет детей

1. Початок

2. Пов’язуємо кожну дитину та її батька та створюємо документ для кожної дитини

3.Для кожного документу

3.1. Якщо відвіданість менша за 3 дні, то

3.2. Повідомити лікаря про це як хвору та зазначити, що повинна бути присутя медична довідка.

4. Надаємо документи до керівника садка

5. Керівник повинен вирахувати і надати платежі вчителю, що у свою чергу до батьків

6. Батьки сплачують. У разі, якщо вони не сплачуються протягом 10 днів за дитсадок, то не приймаєтся до нього.

7.Кінець

3.2 Результати дня

1.Початок

2.Отримати результати медогляду на поточний день

3.Для кожної документу

3.1. Якщо дитина була більше трьох разів та батьки оплатили, то її зараховують до садка

5.Кінець

4 ДІАГРАМА КЛАСІВ

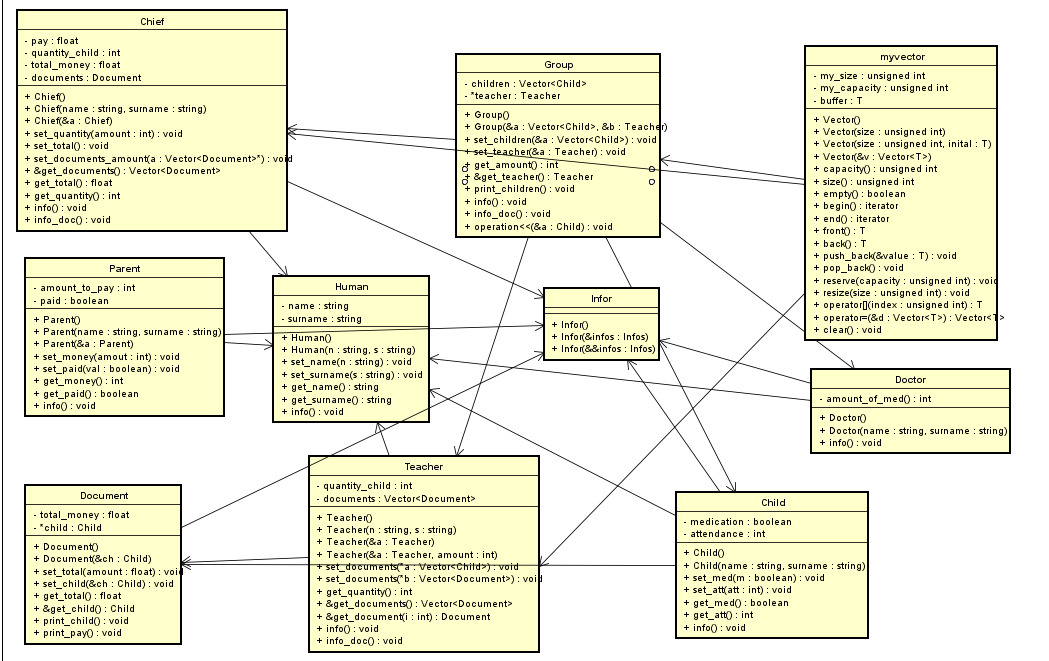


Рисунок 4.1 – Діаграма класів програми

5 ВИХІДНИЙ КОД

#include "helper.h"

int main() {

system("color F0");

author();

cout << endl << endl;

string days;

const int SIZE = 10;

Chief chief("Iryna", "Pavlivla");

Teacher teacher("Alex", "Pavlov");

Vector<Child> children(SIZE);

Vector<Parent> parents(SIZE);

Group IP\_52;

string choice;

main\_logic(parents, children, teacher, IP\_52, chief, days);

while (true) {

menu();

cout << "Your choice: ";

enter\_choice(choice, 0, 5);

work\_logic(stoi(choice), parents, children, teacher, IP\_52, chief);

system("pause");

}

}

#pragma once

#include <ctime>

#include <iostream>

#include "Chief.h"

#include "Group.h"

using namespace std;

const string names[] = { "Brad", "Josh", "Gregory", "Mark", "Marta", "Paul", "Anna", "May", "Hannah", "Tristony", "Nataly", "Chris", "Jacky" };

const string surnames[] = { "Green", "Brown", "Finn", "RD", "Rockors", "Wert", "Alma", "Cred", "Essehn", "Gos", "Dre", "Parker", "Chan" };

void author();

void menu();

void enter\_choice(string& choice, int since, int till);

void work\_logic(int choice, Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief);

void display\_everything(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief);

void display\_medication(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief);

void display\_document(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief);

void display\_unaccepted(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief);

void fill\_random(Vector<Child>&, Vector<Parent>&, int days);

int get\_random(int since, int till);

void main\_logic(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief, string &days);

template<class ...Infos>

class Infor : public Infos...

{

public:

Infor() {};

Infor(Infos&...infos) : Infos(infos)... {};

Infor(Infos&&... infos) : Infos(infos)... {};

};

#include "helper.h"

void author() {

cout << "OOP" << endl;

cout << "Lab - 6 Templates" << endl;

cout << "Eduard Nabokov" << endl;

}

void menu() {

cout << "[1] -> Display every child." << endl;

cout << "[2] -> Display who's required to bring medical certification." << endl;

cout << "[3] -> Display document of each child." << endl;

cout << "[4] -> Display who was unaccepted." << endl;

cout << "[5] -> Exit" << endl;

}

void enter\_choice(string &choice, int since, int till) {

cin >> choice;

}

void work\_logic(int choice, Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief) {

switch (choice) {

case 1: display\_everything(parents, children, teacher, IP\_52, chief); break;

case 2: display\_medication(parents, children, teacher, IP\_52, chief); break;

case 3: display\_document(parents, children, teacher, IP\_52, chief); break;

case 4: display\_unaccepted(parents, children, teacher, IP\_52, chief); break;

case 5: exit(1);

}

}

void display\_everything(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief) {

cout << "EVERYTHING" << endl;

printf("%5s %-10s %-10s %5s %9s %3s", "N", "Name", "Surname", "Attend", "Med cert", "Pay\n");

int i = 0;

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

{

Infor<Child, Document, Parent> common(children[i], teacher.get\_document(i), parents[i]);

printf("%5d %-10s %-10s %-7d %-9d %-3d",

i + 1,

common.get\_child().get\_name().c\_str(),

common.get\_child().get\_surname().c\_str(),

common.get\_att(),

common.get\_med(),

common.get\_money()

);

cout << endl;

}

}

void display\_medication(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief) {

cout << "Children that are required to bring med-cert" << endl;

printf("%5s %-10s %-10s %6s %3s", "N", "Name", "Surname", "Attend", "Med cert\n");

int i = 0;

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

{

Infor<Child> common(children[i]);

if (common.get\_med())

{

printf("%5d %-10s %-10s %-6d %-9d",

i + 1,

common.get\_name().c\_str(),

common.get\_surname().c\_str(),

common.get\_att(),

common.get\_med()

);

cout << endl;

}

}

}

void display\_document(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief) {

cout << "All documents of every child" << endl;

printf("%5s %-10s %-10s %6s %3s", "N", "Name", "Surname", "Attend", "Pay\n");

int i = 0;

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

{

Infor<Document> common(teacher.get\_document(i));

printf("%5d %-10s %-10s %-6d %-9d",

i + 1,

common.get\_child().get\_name().c\_str(),

common.get\_child().get\_surname().c\_str(),

common.get\_child().get\_att(),

common.get\_total()

);

cout << endl;

}

}

void display\_unaccepted(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief) {

cout << "All children who wasn't accepted." << endl;

printf("%5s %-10s %-7s %6s %3s", "N", "Name", "Surname", "Pay", "Paid\n");

int i = 0;

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

{

Infor<Parent> common(parents[i]);

if (!common.get\_money())

{

common.set\_paid(0);

printf("%5d %-10s %-10s %-6d %-3d ",

i + 1,

common.get\_name().c\_str(),

common.get\_surname().c\_str(),

common.get\_money(),

common.get\_paid()

);

cout << endl;

}

}

}

void fill\_random(Vector<Child> &children, Vector<Parent> &parents, int days) {

cout << "Children: came to kindergarten." << endl;

int random;

int attendance;

bool med;

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

{

random = get\_random(0, names->size());

children[i].set\_name(names[random]);

random = get\_random(0, names->size());

parents[i].set\_name(names[random]);

random = get\_random(0, surnames->size());

children[i].set\_surname(surnames[random]);

parents[i].set\_surname(surnames[random]);

attendance = get\_random(0, days);

children[i].set\_att(attendance);

if (attendance <= 3)

children[i].set\_med(true);

med = get\_random(0, 1);

parents[i].set\_paid(med);

}

}

int get\_random(int since, int till) {

return rand() % till + since;

}

void main\_logic(Vector<Parent>& parents, Vector<Child> &children, Teacher &teacher, Group &IP\_52, Chief &chief, string &days) {

srand(time(0));

cout << "Please, enter the number of days (no more than 30): ";

enter\_choice(days, 0, 30);

fill\_random(children, parents, 15);

IP\_52.set\_children(children);

IP\_52.set\_teacher(teacher);

IP\_52.get\_teacher().set\_documents(&children);

chief.set\_documents\_amount(&teacher.get\_documents());

teacher.set\_documents\_full(&chief.get\_documents());

cout << "Parents: got document from teacher." << endl;

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

parents[i].set\_money(teacher.get\_document(i).get\_total());

}

cout << endl;

}

#pragma once

#include <iostream>

#include <string>

#include "myvector.h"

using namespace std;

class Human

{

string name = "Not given";

string surname = "Not given";

public:

Human();

Human(string n, string s);

void set\_name(string n);

void set\_surname(string s);

string get\_name();

string get\_surname();

virtual void info();

~Human();

};

#include "Human.h"

Human::Human()

{

}

Human::Human(string n, string s) {

name = n;

surname = s;

}

void Human::set\_name(string n) {

name = n;

}

void Human::set\_surname(string s) {

surname = s;

}

string Human::get\_name() {

return name;

}

string Human::get\_surname() {

return surname;

}

void Human::info() {

cout << "BaseClass info" << endl;

}

Human::~Human()

{

}

#pragma once

#include "Human.h"

#include "Document.h"

class Chief :

public Human

{

const float pay = 20.5;

int quantity\_child = 0;

float total\_money = 0;

Vector<Document> documents;

public:

Chief();

Chief(string name, string surname);

Chief(Chief &a);

void set\_quantity(int amount);

void set\_total();

void set\_documents\_amount(Vector<Document> \*);

Vector<Document> &get\_documents();

float get\_total();

int get\_quantity();

void info();

void info\_doc();

~Chief();

};

#include "Chief.h"

Chief::Chief()

{

cout << "Chief: the person was created." << endl;

}

Chief::Chief(string n, string s) : Human(n, s) {

cout << "Chief: the person was created." << endl;

};

Chief::Chief(Chief &a) {

this->total\_money = a.total\_money;

this->documents = a.documents;

this->quantity\_child = a.quantity\_child;

}

void Chief::set\_quantity(int amount) {

quantity\_child = amount;

}

void Chief::set\_total() {

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

total\_money += documents[i].get\_total();

}

}

void Chief::set\_documents\_amount(Vector<Document> \*doc) {

cout << "Chief: got an amount of documents." << endl;

documents.resize(doc->size());

for (int i = 0; i < doc->size(); i++) {

documents[i].set\_total(pay \* (\*doc)[i].get\_child().get\_att());

documents[i].set\_child((\*doc)[i].get\_child());

}

set\_total();

}

float Chief::get\_total() {

return total\_money;

}

int Chief::get\_quantity() {

return quantity\_child;

}

Vector<Document> &Chief::get\_documents() {

return documents;

}

void Chief::info() {

cout << get\_name() << " " << get\_surname() << endl;

}

void Chief::info\_doc() {

cout << "Total amount of money: " << get\_total() << endl << endl;

printf("%5s %-10s %-10s %6s", "N", "Name", "Surname", "Pay\n");

int i = 0;

for (auto &value : documents)

{

//cout << value.get\_child().get\_name() << endl;

i++;

printf("%5d %-10s %-10s %6.3f", i, value.get\_child().get\_name().c\_str(), value.get\_child().get\_surname().c\_str(), value.get\_total());

cout << endl;

}

}

Chief::~Chief()

{

}

#pragma once

#include "Human.h"

class Child :

public Human

{

bool medication = false;

int attendance = 0;

public:

Child();

Child(string name, string surname);

void set\_med(bool m);

void set\_att(int att);

bool get\_med();

int get\_att();

void info();

~Child();

};

#include "Child.h"

Child::Child()

{

}

Child::Child(string n, string s) : Human(n, s) {};

void Child::set\_med(bool med) {

medication = med;

}

void Child::set\_att(int att) {

attendance = att;

}

bool Child::get\_med() {

return medication;

}

int Child::get\_att() {

return attendance;

}

void Child::info() {

cout << "Child: " << get\_name() << " " << get\_surname() << " " << get\_med() << " " << get\_att() << endl;

}

Child::~Child()

{

}

#pragma once

#include "Child.h"

#include "Parent.h"

class Document

{

float total\_money = 0;

Child \*child;

public:

Document();

Document(Child &ch);

void set\_total(float amount);

void set\_child(Child&);

float get\_total();

Child &get\_child();

void print\_child();

void print\_pay();

~Document();

};

#include "Document.h"

Document::Document(){}

Document::Document(Child &ch) {

child = &ch;

}

void Document::set\_total(float amount) {

total\_money = amount;

}

void Document::set\_child(Child &ch) {

child = &ch;

}

float Document::get\_total() {

return total\_money;

}

Child &Document::get\_child() {

return \*child;

}

void Document::print\_child() {

this->get\_child().info();

}

void Document::print\_pay() {

cout << "Pay: " << get\_total() << endl;

}

Document::~Document()

{

}

#pragma once

#include "Child.h"

#include "Teacher.h"

//#include <vector>

class Group

{

Vector<Child> children;

Teacher \*teacher;

public:

Group();

Group(Vector<Child>&, Teacher &);

void set\_children(Vector<Child>&);

void set\_teacher(Teacher &);

int get\_amount();

Teacher &get\_teacher();

void print\_children();

void info();

void info\_doc();

void operator<<(Child &);

//friend class Teacher;

~Group();

};

#include "Group.h"

Group::Group()

{

}

Group::Group(Vector<Child>& ch, Teacher &t) {

children = ch;

teacher = &t;

}

void Group::set\_children(Vector<Child>& ch) {

children = ch;

}

void Group::set\_teacher(Teacher& t) {

teacher = &t;

}

int Group::get\_amount() {

return children.size();

}

Teacher &Group::get\_teacher() {

return \*teacher;

}

void Group::print\_children() {

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

children[i].info();

}

}

void Group::info() {

cout << "Teacher: ";

teacher->info();

for (auto &value : children)

value.info();

}

void Group::info\_doc() {

teacher->info\_doc();

}

void Group::operator<<(Child &t) {

this->children.push\_back(t);

}

Group::~Group(){

}

#pragma once

template <class T>

class Vector

{

unsigned int my\_size;

unsigned int my\_capacity;

T \* buffer;

public:

typedef T \* iterator;

Vector();

Vector(unsigned int size);

Vector(unsigned int size, const T & initial);

Vector(const Vector<T> & v);

~Vector();

unsigned int capacity() const;

unsigned int size() const;

bool empty() const;

iterator begin();

iterator end();

T & front();

T & back();

void push\_back(const T & value);

void pop\_back();

void reserve(unsigned int capacity);

void resize(unsigned int size);

T & operator[](unsigned int index);

Vector<T> & operator=(const Vector<T> &);

void clear();

};

template<class T>

Vector<T>::Vector()

{

my\_capacity = 0;

my\_size = 0;

buffer = 0;

}

template<class T>

Vector<T>::Vector(const Vector<T> & v)

{

my\_size = v.my\_size;

my\_capacity = v.my\_capacity;

buffer = new T[my\_size];

for (unsigned int i = 0; i < my\_size; i++)

buffer[i] = v.buffer[i];

}

template<class T>

Vector<T>::Vector(unsigned int size)

{

my\_capacity = size;

my\_size = size;

buffer = new T[size];

}

template<class T>

Vector<T>::Vector(unsigned int size, const T & initial)

{

my\_size = size;

my\_capacity = size;

buffer = new T[size];

for (unsigned int i = 0; i < size; i++)

buffer[i] = initial;

//T();

}

template<class T>

Vector<T> & Vector<T>::operator = (const Vector<T> & v)

{

delete[] buffer;

my\_size = v.my\_size;

my\_capacity = v.my\_capacity;

buffer = new T[my\_size];

for (unsigned int i = 0; i < my\_size; i++)

buffer[i] = v.buffer[i];

return \*this;

}

template<class T>

typename Vector<T>::iterator Vector<T>::begin()

{

return buffer;

}

template<class T>

typename Vector<T>::iterator Vector<T>::end()

{

return buffer + size();

}

template<class T>

T& Vector<T>::front()

{

return buffer[0];

}

template<class T>

T& Vector<T>::back()

{

return buffer[my\_size - 1];

}

template<class T>

void Vector<T>::push\_back(const T & v)

{

if (my\_size >= my\_capacity)

reserve(my\_capacity + 5);

buffer[my\_size++] = v;

}

template<class T>

void Vector<T>::pop\_back()

{

my\_size--;

}

template<class T>

void Vector<T>::reserve(unsigned int capacity)

{

if (buffer == 0)

{

my\_size = 0;

my\_capacity = 0;

}

T \* Newbuffer = new T[capacity];

//assert(Newbuffer);

unsigned int l\_Size = capacity < my\_size ? capacity : my\_size;

//copy (buffer, buffer + l\_Size, Newbuffer);

for (unsigned int i = 0; i < l\_Size; i++)

Newbuffer[i] = buffer[i];

my\_capacity = capacity;

delete[] buffer;

buffer = Newbuffer;

}

template<class T>

unsigned int Vector<T>::size()const//

{

return my\_size;

}

template<class T>

void Vector<T>::resize(unsigned int size)

{

reserve(size);

my\_size = size;

}

template<class T>

T& Vector<T>::operator[](unsigned int index)

{

return buffer[index];

}

template<class T>

unsigned int Vector<T>::capacity()const

{

return my\_capacity;

}

template<class T>

Vector<T>::~Vector()

{

delete[] buffer;

}

template <class T>

void Vector<T>::clear()

{

my\_capacity = 0;

my\_size = 0;

buffer = 0;

}

#pragma once

#include "Human.h"

class Parent :

public Human

{

int amount\_to\_pay = 0;

bool paid = false;

public:

Parent();

Parent(string name, string surname);

Parent(Parent&);

void set\_money(int amount);

void set\_paid(bool val);

int get\_money();

bool get\_paid();

void info();

~Parent();

};

#include "Parent.h"

Parent::Parent()

{}

Parent::Parent(string n, string s) : Human(n, s) {

cout << "Parent: the person was created." << endl;

};

Parent::Parent(Parent& p) {

this->set\_name(p.get\_name());

this->set\_surname(p.get\_surname());

this->set\_money(p.get\_money());

}

void Parent::set\_money(int amount) {

amount\_to\_pay = amount;

}

void Parent::set\_paid(bool val) {

paid = val;

}

int Parent::get\_money() {

return amount\_to\_pay;

}

bool Parent::get\_paid() {

return paid;

}

void Parent::info() {

cout << get\_name() << " " << get\_surname() << " " << get\_money() << " " << get\_paid() << endl;

}

Parent::~Parent()

{

}

#pragma once

#include "Human.h"

#include "Document.h"

class Teacher :

public Human

{

int quantity\_child = 0;

Vector<Document> documents;

public:

Teacher();

Teacher(string n, string s);

Teacher(Teacher &);

Teacher(Teacher &, int amount);

void set\_documents(Vector<Child> \*);

void set\_documents\_full(Vector<Document> \*);

int get\_quantity();

Vector<Document> &get\_documents();

Document &get\_document(int i);

void info();

void info\_doc();

~Teacher();

};

#include "Teacher.h"

Teacher::Teacher()

{

cout << "Teacher: the person was created." << endl;

}

Teacher::Teacher(string n, string s) : Human(n, s) {

cout << "Teacher: the person was created." << endl;

};

Teacher::Teacher(Teacher &t) : Human(t.get\_name(), t.get\_surname()) {

cout << "Teacher: the person was created." << endl;

};

Teacher::Teacher(Teacher &t, int amount) : Human(t.get\_name(), t.get\_surname()), quantity\_child(amount) {

cout << "Teacher: the person was created." << endl;

};

void Teacher::set\_documents(Vector<Child> \*ch) {

cout << "Teacher: document created." << endl;

for (int i = 0; i < ch->size(); i++) {

documents.push\_back(Document((\*ch)[i]));

}

}

void Teacher::set\_documents\_full(Vector<Document> \*doc) {

cout << "Teacher: got updated documents." << endl;

documents = \*doc;

}

Vector<Document> &Teacher::get\_documents() {

return documents;

}

Document &Teacher::get\_document(int i) {

return documents[i];

}

int Teacher::get\_quantity(){

return documents.size();

}

void Teacher::info() {

cout << get\_name() << " " << get\_surname() << endl;

}

void Teacher::info\_doc() {

cout << get\_quantity() << endl;

for (auto &value : documents)

value.print\_child();

}

Teacher::~Teacher()

{

}

6 ВІДЕОКОПІЯ РЕЗУЛЬТАТУ

На рисунку 6.1 показано результат виконання програми.

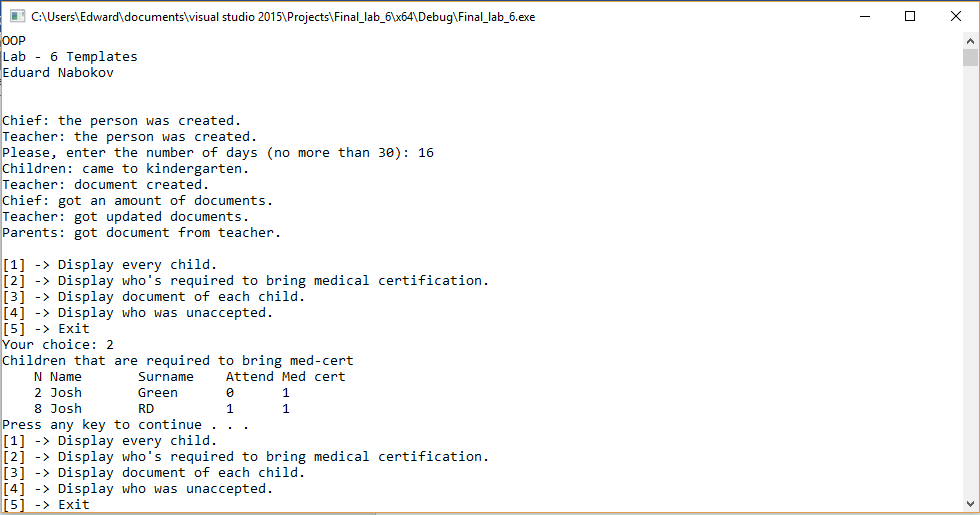


Рисунок 6.1 – Відеокопія результату виконання програми

ВИСНОВОК

У рамках цієї лабораторної роботи я використав стандартні шаблони класів та функцій С++ для вирішення задачі про роботу дитсадка, у тому числі – варіативних шаблонів, що дозволяють використовувати довільну кількість параметрів та формувати колекції на основі базових або користувацьких типів. Шаблони дозволяють використовувати один і той же код для різних типів даних, що зменшує необхідну кількість коду та робить код більш універсальним.