УНИВЕРСИТЕТ ИТМО

Факультет среднего профессионального образования

Специальность 09.02.07«Информационные системы и программирование»

ОТЧЁТ

О ЛАБОРАТОРНОЙ РАБОТЕ №8

по теме: Разработка программ с использованием структурных паттернов.

по дисциплине: Разработка программных модулей

Выполнил:  
студент группы Y2437  
Князев А. А.

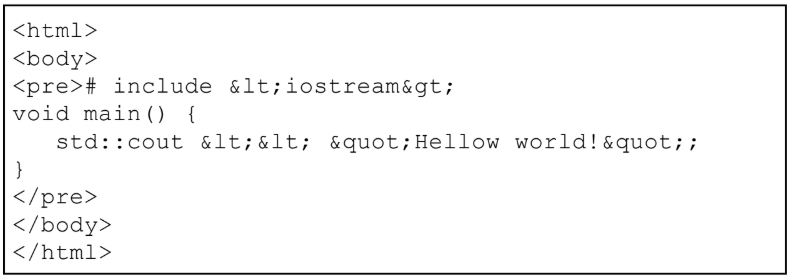
Проверил:  
Ржевский В.А.  
Дата: «\_\_» \_\_\_\_\_\_20\_\_г.  
Оценка:\_\_\_\_\_\_\_\_\_

Санкт-Петербург 2020

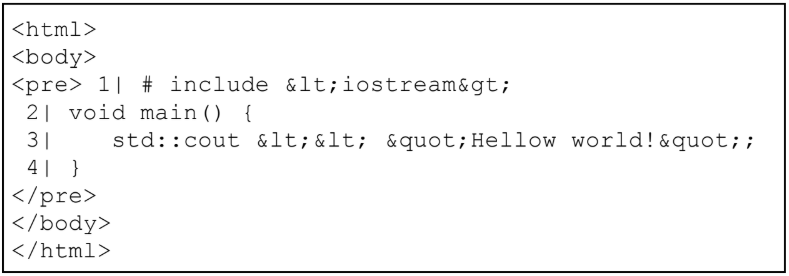
ЗАДАНИЕ

Decorator. Используя паттерн декоратор, реализовать набор классов,позволяющий разнообразным образом переводить C++ код в HTML страницы: с подсветкой ключевых слов или токенов, расстановкой номеров строк и пр.

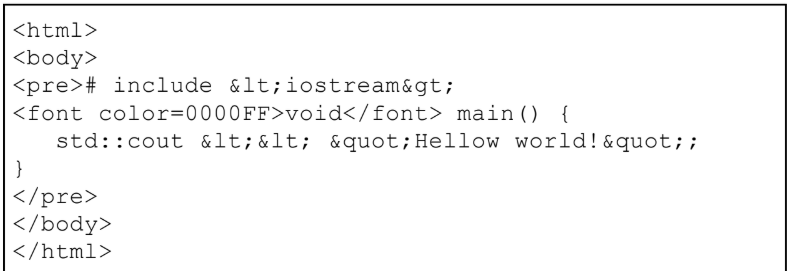
HTMLDecorator – простое преобразование исходного кода в HTML файл. Для этого достаточно добавить открывающие и закрывающие тэги в начало и конец файла, а так же заменить спецсимволы: одинарные и двойные\ кавычки, знаки больше меньше и знак амперсанда.



LineDecorator – простое добавление номеров строк в код. При обработке текста необходимо учесть ситуации, когда строк в коде более 10, 100, 1000, и т.п.



СPPDecorator – примитивная подсветка ключевых слов C++. Чтобы задать цвет текста в html можно использовать тег font с атрибутом color <font color=(цвет в hex представлении)>. Цвет описывается в RGB представлении – по байту на компонент. То есть 000000 – черный, FF0000 – красный, 00FF00 – зеленый, 0000FF – синий.



КОД ПРОГРАММЫ

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class Text {

int textSize = 0;

string\* text = new string[textSize];

friend ostream& operator<<(ostream& out, Text& text);

friend class LineDecorator;

friend class HTMLDecorator;

friend class СPPDecorator;

public:

void SetSize(int size) {

this->textSize = size;

}

Text()

{

}

Text(string\* text)

{

this->text = text;

}

int GetSize() {

return this->textSize;

}

string\* GetText() {

return this->text;

}

void deleteText() {

delete[] this->text;

}

void addStr(string str) {

string\* text\_copy = new string[this->textSize];

for (int i = 0; i < this->textSize; i++)

{

text\_copy[i] = this->text[i];

}

delete[] text;

text = new string[textSize + 1];

for (int i = 0; i < textSize; i++)

{

this->text[i] = text\_copy[i];

}

text[textSize] = str + "<br>";

textSize++;

delete[] text\_copy;

}

void operator=(Text& text) {

delete[] this->text;

string\* t2 = text.GetText();

this->text = new string[text.GetSize()];

for (int i = 0; i < text.GetSize(); i++)

{

this->text[i] = t2[i];

}

}

};

ostream& operator<<(ostream& out, Text& text) {

for (int i = 0; i < text.textSize; i++)

{

out << text.text[i] << endl;

}

return out;

}

ofstream& operator<<(ofstream& out, Text& text) {

for (int i = 0; i < text.GetSize(); i++)

{

out << text.GetText()[i];

}

return out;

}

class Component {

public:

virtual Text read() = 0;

};

class ConcreteComponent : public Component {

protected:

string pathIn = "C:\\Users\\Home\\Desktop\\input.txt";

string pathOut = "C:\\Users\\Home\\Desktop\\out.txt";

public:

Text read() override {

ifstream ifs;

Text text;

ifs.open(pathIn);

if (ifs.is\_open())

{

string str;

while (!ifs.eof()) {

getline(ifs, str);

text.addStr(str);

}

}

else {

cout << "Ошибка открытия файла!" << endl;

}

return text;

}

};

class Decorator : public Component {

protected:

Component\* component\_;

string pathIn = "C:Users\\Home\\Desktop\\input.txt";

public:

Decorator()

{

}

Decorator(Component\* component) : component\_(component) {

}

Text read() override {

return this->component\_->read();

}

};

class HTMLDecorator : public Decorator {

public:

HTMLDecorator(Component\* component) : Decorator(component) {

}

Text read() override {

Text text = Decorator::read();

for (int i = 0; i < text.textSize; i++)

{

while (true) {

if (text.text[i].find("&") == -1)

{

break;

}

text.text[i].replace(text.text[i].find("&"), 1, "&gt;");

}

}

for (int i = 0; i < text.textSize; i++)

{

while (text.text[i].find(">") != -1 || text.text[i].find("<") != -1) {

if (text.text[i].find("<") != -1)

text.text[i].replace(text.text[i].find("<"), 1, "&lt;");

if (text.text[i].find(">") != -1)

text.text[i].replace(text.text[i].find(">"), 1, "&gt;");

}

}

for (int i = 0; i < text.textSize; i++)

{

while (text.text[i].find("&lt;br&gt;") != -1) {

text.text[i].replace(text.text[i].find("&lt;br&gt;"), 10, "<br>");

}

}

text.textSize += 2;

string\* text\_copy = new string[text.textSize];

for (int i = 0, j = 1; i < text.textSize-2; j++, i++)

{

text\_copy[j] = text.text[i];

}

text\_copy[0] = "<html style=\"background-color: #172638; color: #fff; \">\n<body>\n";

text\_copy[text.textSize - 1] = "</html>\n</body>\n";

delete[] text.text;

text.text = new string[text.textSize];

for (int i = 0; i < text.textSize; i++)

{

text.text[i] = text\_copy[i];

}

delete[] text\_copy;

Text t(text);

return t;

}

};

class LineDecorator : public Decorator {

public:

LineDecorator(Component\* component) : Decorator(component) {

}

Text read() override {

Text text = Decorator::read();

for (int i = 1; i < text.textSize - 1; i++)

{

int size = text.text[i].size();

if (size) {

string str = "<div style=\"color: #7b797d; display: inline-block; border-right: 2px solid #fff; width: 20px; padding-right: 10px; margin-right: 10px;\">" + to\_string(i) + "</div>" + text.text[i];

text.text[i].replace(0, size, str);

}

}

return text;

}

};

class СPPDecorator : public Decorator {

public:

СPPDecorator()

{

}

СPPDecorator(Component\* component) : Decorator(component) {

}

Text read() override {

Text text = Decorator::read();

const int dataTypesSize = 5;

const int comandsSize = 8;

const int clasesSize = 6;

string dataTypes[dataTypesSize]{ "void ", "int ", "double ", "string ", "ifstream " },

comands[comandsSize]{ "if ", "else ", "while ", "for ", "return ", "#include", "using", "namespace" },

clases[clasesSize]{ "class ", "public:", "protected:", "private:", "virtual ", "const " };

for (int i = 0; i < dataTypesSize; i++)

{

for (int j = 0; j < text.textSize; j++)

{

auto position = text.text[j].find(dataTypes[i], i);

string tempStr = "<span style=\"color: #c00fee\">" + dataTypes[i] + "</span> ";

if (position != string::npos)

{

text.text[j].replace(position, dataTypes[i].size(), tempStr);

}

}

}

for (int i = 0; i < comandsSize; i++)

{

for (int j = 0; j < text.textSize; j++)

{

auto position = text.text[j].find(comands[i], i);

string tempStr = "<span style=\"color: #869b9b\">" + comands[i] + "</span> ";

if (position != string::npos)

{

text.text[j].replace(position, comands[i].size(), tempStr);

}

}

}

for (int i = 0; i < clasesSize; i++)

{

for (int j = 0; j < text.textSize; j++)

{

auto position = text.text[j].find(clases[i], i);

string tempStr = "<span style=\"color: #5696b6\">" + clases[i] + "</span> ";

if (position != string::npos)

{

text.text[j].replace(position, clases[i].size(), tempStr);

}

}

}

return text;

}

};

void ClientCode(Component\* component) {

ofstream ofs;

string pathOut = "C:\\Users\\Home\\Desktop\\out.html";

ofs.open(pathOut, ofstream::trunc);

if (!ofs.is\_open()) {

cout << "Ошибка открытия файла (при записи)" << endl;

}

Text t = component->read();

ofs << t;

cout << "Файл успешно конвертирован" << endl;

}

int main() {

setlocale(LC\_ALL, "ru");

Component\* simple = new ConcreteComponent;

Component\* decorator1 = new HTMLDecorator(simple);

Component\* decorator2 = new LineDecorator(decorator1);

Component\* decorator3 = new СPPDecorator(decorator2);

ClientCode(decorator3);

delete simple;

delete decorator1;

delete decorator2;

delete decorator3;

return 0;

}

РЕЗУЛЬТАТЫ ПРОГРАММЫ

*Рисунок 1 – результат работы программы*

