МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждениевысшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**ОТЧЁТ по Лабораторной работе №1**

**по дисциплине «ПРОГРАММИРОВАНИЕ»**

**«Структуры данных, паттерны и типизация»**

*5 - вариант*

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АБс-222  Студент: Гатауллин Д.Р. | Преподаватель: Медведев М.А. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**Цели и задачи работы**: Проверка навыков разработки у студентов.

**Задание к работе**:

В соответствии с вашей предметной областью реализуйте следующие задачи (5 вар – Безопасность автомобильного транспорта):

1. Блок linked lists: каждая задача оценивается в 3 балла.

a. Напишите функцию, которая создает пустой связанный список.

b. Напишите функцию, которая добавляет новый узел со значением в начало связанного списка.

c. Напишите функцию, которая добавляет новый узел со значением в конец связанного списка.

d. Напишите тест, который проверяет наличие и удаляет первый узел из связанного списка.

e. Напишите тест, которая проверяет наличие и удаляет все узлы со значением value из связанного списка.

Примечание: обратите внимание, что в каждой функции передается указатель на голову списка по ссылке (Node\*&), чтобы изменения, внесенные внутри функций, отразились на вызывающей стороне.

2. Блок stack. Каждая задача оценивается в 5 баллов

a. Реализуйте обратную польскую запись: стек можно использовать для преобразования математического выражения из инфиксной нотации в постфиксную (обратную польскую) запись, а затем для вычисления результата.

b. Реализуйте управление вызовами функций: стек можно использовать для хранения информации о вызове функций, включая локальные переменные, адреса возврата и другие важные данные. Это позволяет программе возвращаться к предыдущим точкам выполнения и продолжать работу после завершения вызова функции.

c. Реализуйте обработку и отмена операций: стек можно использовать для реализации функциональности "отмена" (undo) и "повтор" (redo). Каждая операция сохраняется в стеке, и при отмене операции она извлекается из стека и отменяется, а при повторе операции она снова применяется.

d. Реализуйте обход деревьев и графов: стек можно использовать для реализации алгоритмов обхода деревьев и графов в глубину (Depth-First Search, DFS). При обходе каждый узел добавляется в стек, а затем извлекается, чтобы обойти его соседние узлы.

e. Реализуйте проверку сбалансированности скобок: стек можно использовать для проверки сбалансированности скобок в строке. При обнаружении открывающейся скобки она добавляется в стек, а при обнаружении закрывающейся скобки она

сравнивается с вершиной стека. Если скобки согласованы, они удаляются из стека, иначе возвращается ошибка.

3. Блок очередей. Каждая задача оценивается в 5 баллов

a. Реализуйте систему обработки заявок: очередь можно использовать для моделирования системы обработки заявок, где каждая заявка добавляется в конец очереди и обрабатывается по мере своей очереди.

b. Реализуйте планирование задач: очередь можно использовать для планирования и выполнения задач в определенном порядке. Новые задачи добавляются в конец очереди, а текущая выполняется и удаляется из очереди.

c. Реализуйте буферизацию данных: очередь может использоваться в качестве буфера для временного хранения данных, пока они не будут обработаны или переданы другому процессу.

d. Реализуйте обработку событий: очередь можно использовать для обработки событий в системе. События добавляются в очередь и обрабатываются в порядке их поступления.

e. Реализуйте координацию работы множества потоков: очередь может использоваться для координации работы множества потоков, где каждый поток добавляет задачи в очередь, а другие потоки извлекают задачи и выполняют их. Это позволяет эффективно распределить работу между потоками и избежать состояния гонки.

Примечание: пункт 3.е дает +10 баллов к оценке работы.

4. Блок hash. Каждая задача оценивается в 5 баллов

a. Реализуйте ускорение поиска: хэш-таблицы могут использоваться для эффективного поиска элементов по ключу. Они позволяют получить доступ к элементу за константное время (O(1)), что делает их полезными для реализации словарей, кэшей и других структур данных, где необходим быстрый доступ к данным.

b. Реализуйте устранение дубликатов: хэш-таблицы могут использоваться для удаления дубликатов из набора данных. При добавлении элементов в хэш-таблицу дубликаты будут автоматически игнорироваться, так как они будут иметь одинаковый хэш-код.

c. Реализуйте кэширование: хэш-таблицы могут использоваться для реализации кэшей, где результаты предыдущих вычислений или запросов хранятся в памяти для быстрого доступа. Ключи могут быть параметрами запроса или входными данными, а значения — результатами операций или вычислений.

d. Реализуйте подсчет частоты элементов: хэш-таблицы могут использоваться для подсчета частоты встречаемости элементов в наборе данных. Ключи могут быть элементами, а значения — количеством их появлений. Это полезно, например, для нахождения наиболее популярных элементов или удаления редко встречающихся элементов.

e. Реализуйте ассоциативный массив содержащий набор базовых операций: хэш-таблицы могут использоваться для реализации ассоциативных массивов, где каждому ключу соответствует значение. Это позволяет эффективно выполнять операции вставки, удаления и поиска элементов по ключу.

5. Блок рекурсия: дает 15 баллов

a. Реализуйте задачу комбинаторики согласно предметной области: рекурсия может использоваться для решения задач комбинаторики, таких как генерация всех возможных перестановок, сочетаний или размещений элементов. Рекурсивная тфункция может вызывать себя для генерации всех комбинаций с уменьшением размера задачи на каждой итерации. Смотреть принцип «Разделяй и властвуй».

6. Блок инкапсуляция: каждая задача оценивается в 4 балла

a. Создание шаблона класса "Счет": инкапсуляция позволяет скрыть детали реализации и предоставить публичные методы для взаимодействия с объектом "Счет", такие как пополнение, списание и получение баланса.

b. Реализация шаблона класса "Обьект": инкапсуляция позволяет скрыть внутреннюю структуру и состояние объекта, предоставляя публичные методы для доступа к информации объекта (не менее 5).

c. Создание класса "Пользователь": инкапсуляция позволяет скрыть данные о пользователе, такие как логин, пароль и другие конфиденциальные сведения, предоставляя публичные методы для аутентификации, изменения пароля и работы с профилем.

d. Реализация класса "Файловый менеджер": инкапсуляция позволяет скрыть внутренние детали работы с файлами и папками, предоставляя публичные методы для операций, таких как создание, удаление, переименование файлов и папок, а также чтение и запись данных в файлы.

7. Блок наследования: оценивается в 4 балла

a. Разработайте класс и его подклассы согласно предметной области: наследование позволяет создать базовый класс, от которого можно создать подклассы. Каждый подкласс может иметь свои специфические методы и свойства, а также наследовать общие методы и свойства от базового класса.

8. Блок полиморфизма: оценивается в 4 балла

a. Разработайте механизм позволяющий применять общие методы для различных классов объектов согласно предметной области.

9. Блок типизации: оценивается в 3 балла

a. Создать словарь, где ключами являются строки, а значениями - списки чисел. Использовать неявную типизацию данных для определения типов ключей и значений словаря.

b. Написать функцию, которая принимает на вход два аргумента: число и строку. Функция должна возвращать результат, полученный путем повторения строки указанное количество раз. Использовать неявную типизацию данных для определения типов аргументов и результата функции.

**Текст программы, результаты выполнения**

Git - <https://github.com/SpongebobGatik/pr1_2024>

1

Main.cpp

#include "list.h"

#include "gtest/gtest.h"

int main(int argc, char\*\* argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

List.h

#ifndef LIST\_H

#define LIST\_H

#include <iostream>

#include "gtest/gtest.h"

using namespace std;

struct OneNode {

int value;

OneNode\* next;

};

struct OneList {

OneNode\* head;

OneNode\* tail;

OneList() : head(NULL), tail(NULL) {};

~OneList() {

while (head != NULL) {

OneNode\* temp = head;

head = head->next;

delete temp;

}

}

void PrintList();

};

OneList\* CreateOneList();

OneList\* AddStart(OneList& list, int value);

OneList\* AddEnd(OneList& list, int value);

#endif

List.cpp

#include "list.h"

void OneList::PrintList() {

OneNode\* item = head;

while (item != NULL) {

cout << item->value << endl;

item = item->next;

}

}

OneList\* CreateOneList() {

OneList list;

return &list;

}

OneList\* AddStart(OneList& list, int value) {

auto NewNode = new OneNode{ value };

if (list.head == NULL) {

list.head = NewNode;

list.tail = NewNode;

return &list;

}

NewNode->next = list.head;

list.head = NewNode;

return &list;

}

OneList\* AddEnd(OneList& list, int value) {

auto NewNode = new OneNode{ value };

if (list.head == NULL) {

list.head = NewNode;

list.tail = NewNode;

return &list;

}

list.tail->next = NewNode;

list.tail = NewNode;

return &list;

}

OneList\* RemoveFirst(OneList& list) {

if (list.head == NULL) {

return &list;

}

OneNode\* temp = list.head;

list.head = list.head->next;

delete temp;

return &list;

}

OneList\* RemoveValue(OneList& list, int value) {

OneNode\* current = list.head;

OneNode\* prev = NULL;

while (current != NULL) {

if (current->value == value) {

if (prev == NULL) {

list.head = current->next;

}

else {

prev->next = current->next;

}

OneNode\* temp = current;

current = current->next;

delete temp;

}

else {

prev = current;

current = current->next;

}

}

return &list;

}

TEST(TestOneList, TestRemoveFirstNode) {

OneList list = \*CreateOneList();

list = \*AddStart(list, 1);

list = \*AddStart(list, 2);

list = \*AddStart(list, 3);

list = \*AddEnd(list, 4);

EXPECT\_EQ(list.head->value, 3);

list = \*RemoveFirst(list);

EXPECT\_EQ(list.head->value, 2);

}

TEST(TestOneList, TestRemoveValue) {

OneList list = \*CreateOneList();

list = \*AddStart(list, 1);

list = \*AddStart(list, 2);

list = \*AddStart(list, 3);

list = \*AddStart(list, 2);

list = \*AddStart(list, 2);

list = \*AddStart(list, 2);

list = \*AddEnd(list, 4);

list = \*RemoveValue(list, 2);

OneNode\* item = list.head;

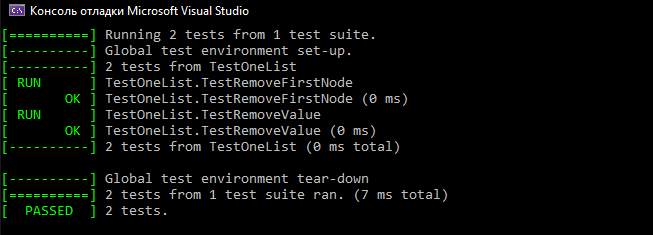
while (item != NULL) {

EXPECT\_NE(item->value, 2);

item = item->next;

}

}



2

a

#include <stack>

#include <string>

#include <sstream>

#include <iostream>

using namespace std;

int evalRPN(string tokens) {

istringstream iss(tokens);

string token;

stack<int> stack;

while (getline(iss, token, ' ')) {

if (token == "+" || token == "-" || token == "\*" || token == "/") {

int op2 = stack.top();

stack.pop();

int op1 = stack.top();

stack.pop();

if (token == "+") stack.push(op1 + op2);

else if (token == "-") stack.push(op1 - op2);

else if (token == "\*") stack.push(op1 \* op2);

else stack.push(op1 / op2);

}

else {

stack.push(stoi(token));

}

}

return stack.top();

}

int main() {

string rpn = "2 3 + 4 \*";

cout << evalRPN(rpn) << endl; // Output: 20

return 0;

}



b

#include <stack>

#include <iostream>

using namespace std;

void funcA();

void funcB();

stack<void (\*)()> callStack;

int main() {

callStack.push(funcA);

callStack.push(funcB);

while (!callStack.empty()) {

callStack.top()();

callStack.pop();

}

return 0;

}

void funcA() {

cout << "Function A" << endl;

}

void funcB() {

cout << "Function B" << endl;

}



c

#include <stack>

#include <iostream>

using namespace std;

struct Operation {

string action;

int value;

};

stack<Operation> operations;

void perform(const Operation& op) {

cout << "Performing operation: " << op.action << " with value: " << op.value << endl;

operations.push(op);

}

void undo() {

if (!operations.empty()) {

Operation op = operations.top();

operations.pop();

cout << "Undoing operation: " << op.action << " with value: " << op.value << endl;

}

}

int main() {

perform({ "add", 5 });

perform({ "subtract", 3 });

undo();

return 0;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

d

#include <stack>

#include <vector>

#include <iostream>

using namespace std;

struct Node {

int value;

vector<Node\*> children;

};

void depthFirstSearch(Node\* root) {

stack<Node\*> stack;

stack.push(root);

while (!stack.empty()) {

Node\* node = stack.top();

stack.pop();

cout << "Узел: " << node->value << endl;

for (Node\* child : node->children) {

stack.push(child);

}

}

}

int main() {

setlocale(LC\_ALL, "Rus");

// Создание узлов

Node node1{ 1, {} };

Node node2{ 2, {} };

Node node3{ 3, {} };

Node node4{ 4, {} };

Node node5{ 5, {} };

// Создание связей между узлами

node1.children = { &node2, &node3 };

node2.children = { &node4, &node5 };

// Вызов функции обхода дерева в глубину

depthFirstSearch(&node1);

return 0;

}

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

e

#include <stack>

#include <string>

#include <iostream>

using namespace std;

bool isBalanced(const string& expression) {

stack<char> stack;

for (char c : expression) {

if (c == '(' || c == '[' || c == '{') {

stack.push(c);

}

else if (c == ')' || c == ']' || c == '}') {

if (stack.empty()) return false;

char top = stack.top();

stack.pop();

if ((c == ')' && top != '(') || (c == ']' && top != '[') || (c == '}' && top != '{')) {

return false;

}

}

}

return stack.empty();

}

int main() {

string expression = "{[()]}";

string expression2 = "{[()]{}";

cout << (isBalanced(expression) ? "Balanced" : "Not balanced((((") << endl;

cout << (isBalanced(expression2) ? "Balanced" : "Not balanced((((") << endl;

return 0;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

3

a

#include <iostream>

#include <queue>

#include <string>

using namespace std;

struct Car {

string number;

string model;

};

int main() {

setlocale(LC\_ALL, "Rus");

queue<Car> cars;

// Добавляем заявки в очередь

cars.push({ "A123BC", "Toyota" });

cars.push({ "B456DE", "BMW" });

cars.push({ "C789FG", "Mercedes" });

while (!cars.empty()) {

Car car = cars.front();

cout << "Обрабатывается автомобиль " << car.model << " с номером " << car.number << endl;

// Удаляем заявку из очереди после обработки

cars.pop();

}

return 0;

}

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

b

#include <iostream>

#include <queue>

#include <string>

using namespace std;

struct Task {

string carNumber;

string taskDescription;

};

int main() {

setlocale(LC\_ALL, "Rus");

queue<Task> tasks;

// Добавляем задачи в очередь

tasks.push({ "A123BC", "Проверить тормозную систему" });

tasks.push({ "B456DE", "Заменить масло в двигателе" });

tasks.push({ "C789FG", "Проверить состояние шин" });

while (!tasks.empty()) {

Task task = tasks.front();

cout << "Выполняется задача для автомобиля " << task.carNumber << ": " << task.taskDescription << endl;

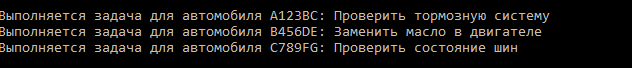
// Удаляем задачу из очереди после выполнения

tasks.pop();

}

return 0;

}



c

#include <iostream>

#include <queue>

using namespace std;

// Структура данных для хранения информации о безопасности автомобильного транспорта

struct CarSafetyData {

string carModel;

int safetyRating;

};

int main() {

// Создаем очередь для хранения данных о безопасности автомобильного транспорта

queue<CarSafetyData> buffer;

// Добавляем данные в буфер

buffer.push({ "Model S", 5 });

buffer.push({ "Model 3", 5 });

// Обрабатываем данные из буфера

while (!buffer.empty()) {

CarSafetyData data = buffer.front();

buffer.pop();

cout << "Car Model: " << data.carModel << ", Safety Rating: " << data.safetyRating << endl;

}

return 0;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

d

#include <iostream>

#include <queue>

#include <string>

using namespace std;

struct Event {

string type;

string description;

};

int main() {

setlocale(LC\_ALL, "Rus");

queue<Event> events;

// Добавляем события в очередь

events.push({ "Accident", "Авария на улице Мира" });

events.push({ "TrafficJam", "Пробка на проспекте Ленина" });

events.push({ "RoadWork", "Ремонт дороги на улице Советской" });

// Обрабатываем события в порядке их поступления

while (!events.empty()) {

Event currentEvent = events.front();

events.pop();

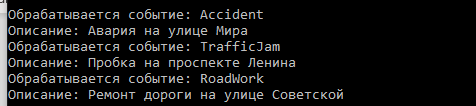
cout << "Обрабатывается событие: " << currentEvent.type << endl;

cout << "Описание: " << currentEvent.description << endl;

}

return 0;

}



e

#include <iostream>

#include <queue>

#include <thread>

#include <mutex>

#include <condition\_variable>

std::queue<int> tasks;

std::mutex mtx;

std::condition\_variable cv;

bool finished = false;

void producer(int id) {

for (int i = 0; i < 10; ++i) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(100));

std::unique\_lock<std::mutex> lock(mtx);

std::cout << "Producer " << id << " добавил задачу " << i << std::endl;

tasks.push(i);

cv.notify\_all();

}

}

void consumer(int id) {

while (true) {

std::unique\_lock<std::mutex> lock(mtx);

cv.wait(lock, [] { return !tasks.empty() || finished; });

if (finished && tasks.empty()) {

return;

}

int value = tasks.front();

tasks.pop();

std::cout << "Consumer " << id << " выполнил задачу " << value << std::endl;

lock.unlock();

cv.notify\_all();

}

}

int main() {

setlocale(LC\_ALL, "Rus");

std::thread producers[2];

std::thread consumers[2];

for (int i = 0; i < 2; ++i) {

producers[i] = std::thread(producer, i + 1);

consumers[i] = std::thread(consumer, i + 1);

}

for (int i = 0; i < 2; ++i) {

producers[i].join();

}

finished = true;

cv.notify\_all();

for (int i = 0; i < 2; ++i) {

consumers[i].join();

}

return 0;

}

Изображение выглядит как текст, снимок экрана

Автоматически созданное описаниеИзображение выглядит как снимок экрана, текст, шаблон, монохромный

Автоматически созданное описание

4

#include <string>

#include <vector>

#include <iostream>

using namespace std;

class NodeHashTable {

public:

string key;

pair<string, int> element;

NodeHashTable\* next;

NodeHashTable\* prev;

NodeHashTable() : key(""), element(make\_pair("", 0)), next(nullptr), prev(nullptr) {}

NodeHashTable(const string& key, const string& element) : key(key), element(make\_pair(element, 1)), next(nullptr), prev(nullptr) {}

};

class HashTable {

public:

vector<NodeHashTable\*> nodes;

int count;

HashTable() : nodes(vector<NodeHashTable\*>(100000, nullptr)), count(0) {}

void printHashTable();

void freeHashTable();

void HSET(const string& key, const string& value);

void HDEL(const string& key);

pair<string, int> HGET(const string& key);

};

// Функция для вычисления хеша

int calculateHashT(const string& element) {

int hash = 0;

for (char c : element) {

hash = 31 \* hash + c;

}

return abs(hash) % 100000;

}

// Функция для добавления элемента в хеш-таблицу

void HashTable::HSET(const string& key, const string& value) {

int hash = calculateHashT(key);

NodeHashTable\* newNode = new NodeHashTable(key, value);

NodeHashTable\* current = nodes[hash];

while (current != nullptr) {

if (current->key == key) {

current->element.second++; // Увеличиваем частоту

delete newNode;

return;

}

if (current->next == nullptr) {

break;

}

current = current->next;

}

if (current == nullptr) {

nodes[hash] = newNode;

}

else {

current->next = newNode;

newNode->prev = current;

}

count++;

}

// Функция для получения элемента из хеш-таблицы

pair<string, int> HashTable::HGET(const string& key) {

int hash = calculateHashT(key);

NodeHashTable\* current = nodes[hash]; // Получаем узел по хеш-ключу

while (current != nullptr) { // Перебираем узлы в цепочке

if (current->key == key) { // Если ключ совпадает

return current->element;

}

current = current->next; // Переходим к следующему узлу

}

return { "", 0 };

}

// Функция для удаления элемента из хеш-таблицы

void HashTable::HDEL(const string& key) {

int hash = calculateHashT(key);

NodeHashTable\* current = nodes[hash]; // Получаем узел по хеш-ключу

NodeHashTable\* nodeToRemove = nullptr;

while (current != nullptr) { // Перебираем узлы в цепочке

if (current->key == key) { // Если ключ совпадает

nodeToRemove = current; // Устанавливаем узел для удаления

break;

}

current = current->next; // Переходим к следующему узлу

}

if (nodeToRemove != nullptr) {

if (nodeToRemove->prev != nullptr) { // Если у узла есть предыдущий узел

nodeToRemove->prev->next = nodeToRemove->next; // Удаляем узел из цепочки

}

else {

nodes[hash] = nodeToRemove->next; // Устанавливаем следующий узел как начало цепочки

}

if (nodeToRemove->next != nullptr) { // Если у узла есть следующий узел

nodeToRemove->next->prev = nodeToRemove->prev; // Устанавливаем предыдущий узел для следующего узла

}

// Освобождаем память удаляемого узла

delete nodeToRemove;

count--;

}

else {

cout << "Ключ не найден.\n";

}

}

// Функция для освобождения памяти хеш-таблицы

void HashTable::freeHashTable() {

for (int i = 0; i < 100000; i++) {

NodeHashTable\* current = nodes[i];

while (current != nullptr) {

NodeHashTable\* temp = current;

current = current->next;

delete temp;

}

nodes[i] = nullptr; // Установка указателя в nullptr после удаления узлов

}

}

// Функция для вывода хеш-таблицы

void HashTable::printHashTable() {

cout << "Хеш-таблица:\n";

for (int i = 0; i < 100000; i++) {

NodeHashTable\* current = nodes[i];

if (current != nullptr) {

cout << "Индекс " << i << ": ";

while (current != nullptr) {

cout << "(" << current->key << ", " << current->element.first << ", " << current->element.second << ") ";

current = current->next;

}

cout << "\n";

}

}

}

int main() {

setlocale(LC\_ALL, "Rus");

HashTable hashTable;

// Добавление элементов

hashTable.HSET("ключ1", "значение1");

hashTable.HSET("ключ2", "значение2");

hashTable.HSET("ключ3", "значение3");

// Повторное добавление элемента с тем же ключом увеличивает частоту

hashTable.HSET("ключ1", "значение1");

// Вывод хеш-таблицы

hashTable.printHashTable();

// Получение элемента по ключу

pair<string, int> value = hashTable.HGET("ключ1");

if (!value.first.empty()) {

cout << "Значение для 'ключ1': " << value.first << "\n";

}

else {

cout << "'ключ1' не найден.\n";

}

// Удаление элемента по ключу

hashTable.HDEL("ключ2");

// Вывод хеш-таблицы после удаления элемента

hashTable.printHashTable();

// Освобождение памяти хеш-таблицы

hashTable.freeHashTable();

return 0;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

5

#include <iostream>

#include <vector>

#include <string>

using namespace std;

// Все возможные буквы в номере машины

const vector<char> letters = { 'A', 'B', 'C', 'E', 'H', 'K', 'M', 'O', 'P', 'T', 'X', 'Y' };

// Рекурсивная функция для генерации номеров

void generateNumbers(string number, int letterCount, int digitCount) {

if (letterCount == 0 && digitCount == 0) {

cout << number << endl;

return;

}

if (letterCount > 0) {

for (char letter : letters) {

generateNumbers(number + letter, letterCount - 1, digitCount);

}

}

if (digitCount > 0) {

for (int digit = 0; digit <= 9; ++digit) {

generateNumbers(number + to\_string(digit), letterCount, digitCount - 1);

}

}

}

int main() {

// Начинаем с пустого номера, 3 буквы и 3 цифры

generateNumbers("", 3, 3);

return 0;

}

Изображение выглядит как текст, снимок экрана, черный, дизайн

Автоматически созданное описание

6

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

template <typename T>

class Account {

private:

T balance;

public:

Account(T initial\_balance) : balance(initial\_balance) {}

void deposit(T amount) { balance += amount; }

void withdraw(T amount) { if (balance >= amount) balance -= amount; }

T get\_balance() const { return balance; }

};

template <typename T>

class Object {

private:

T data[5];

public:

Object(T d[5]) { for (int i = 0; i < 5; i++) data[i] = d[i]; }

T get\_data(int index) const { return data[index]; }

void set\_data(int index, T value) { data[index] = value; }

int size() const { return 5; }

};

class User {

private:

string login;

string password;

public:

User(string l, string p) : login(l), password(p) {}

bool authenticate(string l, string p) { return l == login && p == password; }

void change\_password(string new\_password) { password = new\_password; }

string get\_profile() const { return "User Profile: " + login; }

};

class FileManager {

public:

void create\_file(const string& filename) {

ofstream file(filename);

file.close();

}

void delete\_file(const string& filename) {

remove(filename.c\_str());

}

void rename\_file(const string& old\_filename, const string& new\_filename) {

rename(old\_filename.c\_str(), new\_filename.c\_str());

}

void write\_to\_file(const string& filename, const string& content) {

ofstream file(filename);

file << content;

file.close();

}

string read\_from\_file(const string& filename) {

ifstream file(filename);

string content((istreambuf\_iterator<char>(file)), istreambuf\_iterator<char>());

return content;

}

};

int main() {

// Создание объекта класса "Счет"

Account<double> myAccount(100.0);

myAccount.deposit(50.0);

cout << "Balance: " << myAccount.get\_balance() << endl;

// Создание объекта класса "Объект"

int arr[5] = { 1, 2, 3, 4, 5 };

Object<int> myObject(arr);

cout << "Object data at index 2: " << myObject.get\_data(2) << endl;

// Создание объекта класса "Пользователь"

User myUser("username", "password");

if (myUser.authenticate("username", "password")) {

cout << "Authentication successful!" << endl;

}

// Создание объекта класса "Файловый менеджер"

FileManager myFileManager;

myFileManager.create\_file("test.txt");

myFileManager.write\_to\_file("test.txt", "Hello, world!");

cout << "File content: " << myFileManager.read\_from\_file("test.txt") << endl;

return 0;

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

7

#include <iostream>

#include <string>

// Базовый класс "Безопасность автомобильного транспорта"

class RoadSafety {

protected:

std::string name; // название меры безопасности

public:

RoadSafety(const std::string& n) : name(n) {}

virtual void displayInfo() {

std::cout << "Мера безопасности: " << name << std::endl;

}

};

// Подкласс "Пассивная безопасность" (наследуется от RoadSafety)

class PassiveSafety : public RoadSafety {

private:

int airbagsCount; // количество подушек безопасности

public:

PassiveSafety(const std::string& n, int airbags) : RoadSafety(n), airbagsCount(airbags) {}

void displayInfo() override {

RoadSafety::displayInfo();

std::cout << "Количество подушек безопасности: " << airbagsCount << std::endl;

}

};

// Подкласс "Активная безопасность" (наследуется от RoadSafety)

class ActiveSafety : public RoadSafety {

private:

bool hasABS; // наличие системы ABS

bool hasESP; // наличие системы ESP

public:

ActiveSafety(const std::string& n, bool abs, bool esp) : RoadSafety(n), hasABS(abs), hasESP(esp) {}

void displayInfo() override {

RoadSafety::displayInfo();

std::cout << "Наличие системы ABS: " << (hasABS ? "Да" : "Нет") << std::endl;

std::cout << "Наличие системы ESP: " << (hasESP ? "Да" : "Нет") << std::endl;

}

};

// Подкласс "Профилактическая безопасность" (наследуется от RoadSafety)

class PreventiveSafety : public RoadSafety {

private:

int maintenanceInterval; // интервал обслуживания в месяцах

public:

PreventiveSafety(const std::string& n, int interval) : RoadSafety(n), maintenanceInterval(interval) {}

void displayInfo() override {

RoadSafety::displayInfo();

std::cout << "Интервал обслуживания: " << maintenanceInterval << " месяцев" << std::endl;

}

};

int main() {

setlocale(LC\_ALL, "Rus");

PassiveSafety ps("Пассивная безопасность", 6);

ActiveSafety as("Активная безопасность", true, true);

PreventiveSafety pfs("Профилактическая безопасность", 12);

ps.displayInfo();

std::cout << std::endl;

as.displayInfo();

std::cout << std::endl;

pfs.displayInfo();

return 0;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

8

#include <iostream>

// Базовый класс

class VehicleSafety {

public:

virtual void safetyFeature() = 0; // Чистая виртуальная функция

};

// Производный класс: CarSafety

class CarSafety : public VehicleSafety {

public:

void safetyFeature() override {

std::cout << "Автомобиль: ABS, подушки безопасности, система стабилизации и что-то ещё" << std::endl;

}

};

// Производный класс: BusSafety

class BusSafety : public VehicleSafety {

public:

void safetyFeature() override {

std::cout << "Автобус: аварийные выходы, огнетушители, система стабилизации и что-то ещё" << std::endl;

}

};

int main() {

setlocale(LC\_ALL, "Rus");

CarSafety car;

BusSafety bus;

VehicleSafety\* vehicle1 = &car;

VehicleSafety\* vehicle2 = &bus;

vehicle1->safetyFeature();

vehicle2->safetyFeature();

return 0;

}

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

9

#include <iostream>

#include <map>

#include <vector>

#include <string>

// a. Создание словаря с ключами-строками и значениями-списками чисел

auto createDictionary() {

std::map<std::string, std::vector<int>> dictionary;

dictionary["speed\_limits"] = { 30, 50, 60, 70, 90, 110, 130 }; // скоростные ограничения

dictionary["seatbelt\_fines"] = { 100, 200, 500, 1000 }; // штрафы за не использование ремня безопасности

return dictionary;

}

// b. Функция, которая повторяет строку указанное количество раз

auto repeatString(int count, const std::string& str) {

std::string result;

for (int i = 0; i < count; ++i) {

result += str;

}

return result;

}

int main() {

auto dictionary = createDictionary();

for (const auto& pair : dictionary) {

std::cout << pair.first << ": ";

for (int num : pair.second) {

std::cout << num << " ";

}

std::cout << std::endl;

}

std::cout << repeatString(4, "AVE ") << "AVTF" << std::endl;

return 0;

}

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

**Вывод**: задания выполнены и, по идее, соответствуют всем требованиям.