Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Инженерно-экономический факультет

Кафедра экономической информатики

Лабораторная работа № 2

«ПРАКТИЧЕСКИЕ ПРИЕМЫ

ИСПОЛЬЗОВАНИЯ ШАБЛОНОВ ТИПОВ И

ИЕРАРХИИ КЛАССОВ»

Вариант 2

Выполнил: Студент гр.272303

Вдовенко Н. Д.

Проверил: Старший преподаватель кафедры ЭИ

Салапура М. Н.

Минск 2023

Цель работы: научиться использовать шаблоны функций и классов при работе со smart-указателями и транзакциями.

Задачи: написать код к приложенным заданиям, пользуясь приобретенными знаниями.

**Краткие теоретические сведения**

Шаблоны в C++ позволяют реализовать параметрический полиморфизм, что позволяет многократно использовать код, корректно по отношению к различным типам данных.

Механизм шаблонов позволяет унифицировать алгоритмы для различных типов данных, избегая необходимости писать различные функции для каждого типа данных.

Обобщенные функции перегружают сами себя, определяя общий набор операций для данных различных типов. Тип данных передается функции как параметр.

Обобщенные (шаблонные) функции в C++: Обобщенные функции позволяют создавать функции, которые могут работать с разными типами данных, используя шаблоны. Это позволяет программистам создавать универсальные функции без необходимости переписывания кода для каждого типа данных.

Перегрузка шаблонов функций: Перегрузка шаблонов функций позволяет предоставить разные версии функций с разными параметрами, что дополняет возможности обобщенных функций.

Обобщенные классы позволяют создавать классы, которые могут работать с разными типами данных. Это увеличивает гибкость кода, позволяя создавать универсальные структуры данных.

Создание экземпляров обобщенных классов с указанием конкретных типов данных: для создания конкретных объектов обобщенных классов, нужно указать тип данных при объявлении объекта. Это позволяет создавать экземпляры классов для разных типов данных.

Для управления динамической памятью в C++ используют операторы new, которые резервируют и инициализируют объекты в динамической памяти, и delete, которые удаляют объекты и освобождают выделенную память.

Интеллектуальные указатели — это особые указатели, которые автоматически удаляют объект, на который указывают, когда объект больше не нужен. Они помогают предотвратить утечку памяти и висячие ссылки.

Интеллектуальные указатели обычно реализуют семантику владения ресурсом, что означает, что они управляют временем жизни объекта и автоматически его уничтожают при необходимости.

Интеллектуальные указатели могут использоваться для обеспечения механизма транзакций, где объекты содержат указатели на текущее и предыдущее состояния и функции для начала, фиксации, отмены и повторения транзакций, обеспечивая целостность данных и независимость от других транзакций.

**Контрольные вопросы**

1. Особенности использования динамической памяти в C++:

- Для выделения и освобождения динамической памяти используются операторы `new` и `delete`.

- Динамически выделенная память не освобождается автоматически и может привести к утечкам памяти.

- Необходимо аккуратно управлять динамической памятью, чтобы избежать висячих указателей и двойных удалений.

2. Smart-указатели используются для автоматического управления динамической памятью, предотвращения утечек и висячих указателей. Они представляют собой специальные классы, которые оборачивают указатели и автоматически освобождают ресурсы при выходе из области видимости.

3. Библиотека C++ предоставляет три основных вида интеллектуальных указателей: `std::shared\_ptr`, `std::unique\_ptr` и `std::weak\_ptr`.

4. В собственном классе "умного указателя" должны быть реализованы методы для управления ресурсом, такие как конструктор, деструктор, операторы копирования и присваивания. Пример методов может включать инициализацию и освобождение ресурсов.

5. Указатель `std::shared\_ptr` позволяет нескольким указателям указывать на один и тот же объект. Пример создания и инициализации: `std::shared\_ptr<int> sharedPtr = std::make\_shared<int>(42);`

6. Указатель `std::unique\_ptr` позволяет только одному указателю указывать на объект. Пример создания и инициализации: `std::unique\_ptr<int> uniquePtr(new int(42));`

7. Указатель `std::weak\_ptr` является второстепенной ссылкой на объект, управляемый указателем `std::shared\_ptr`. Пример создания и инициализации: `std::shared\_ptr<int> sharedPtr = std::make\_shared<int>(42); std::weak\_ptr<int> weakPtr = sharedPtr;`

8. Утечка памяти в программировании - это ситуация, когда выделенная память не была освобождена, и программа больше не имеет доступа к ней. Пример утечки памяти: неосвобожденная динамически выделенная память, например, забытый указатель `new` без соответствующего `delete`.

9. Висячий указатель - это указатель, который ссылается на уже удаленный объект или на недопустимый объект. Пример висячего указателя: указатель на объект, который был удален оператором `delete`.

10. Идиома программирования RAII (Resource Acquisition Is Initialization) предполагает, что получение ресурса, такого как динамически выделенная память или файл, должно происходить в конструкторе объекта, а освобождение ресурса - в его деструкторе.

11. Транзакция в программировании - это группа операций, представляющая собой логически неделимую единицу работы с данными. Транзакции используются для обеспечения целостности данных и предотвращения конфликтов при параллельном доступе к данным.

12. Операции при реализации механизма транзакций включают в себя начало транзакции, фиксацию транзакции, отмену транзакции и повтор транзакции. Пример реализации одной из операций, например, начала транзакции, может включать в себя сохранение текущего состояния данных и начало записи изменений.

**Индивидуальное задание**

. Разработать набор классов (минимум 5) по теме «делопроизводство в учреждении образования». Корректно реализовать связи между классами. Использовать smart-указатели для создания программы учета делопроизводства в учреждении образования. Имеется несколько объектов документации и некоторое количество сотрудников учреждения образования, которые работают с документацией. Документ считается сформированным, когда с ним не работает ни один сотрудник. В разработанном наборе классов должен быть хотя бы один шаблонный класс. Все классы должны иметь методы получения и установки значений полей. Реализовать механизм транзакций, который позволит сотрудникам откатывать изменения, внесенные в документы. Использовать конструктор с параметрами, конструктор без параметров, конструктор копирования, деструктор.

**Листинг кода**

#include "classes.h"

#include "libs.h"

#include "InputChecks.h"

//Employee

int Employee::count = 0;

Employee::Employee() : name("") {

++count;

employeeID = count;

}

Employee::Employee(const std::string& name, int employeeID) : name(name), employeeID(employeeID) {

}

Employee::Employee(const Employee& other) : name(other.name), employeeID(other.employeeID) {

}

Employee::~Employee() {

}

string Employee::getName() const {

return name;

}

void Employee::setName(const string& name) {

this->name = name;

}

int Employee::getEmployeeID() const {

return employeeID;

}

void Employee::runMenu() {

DocumentManagement documentManager;

int choice;

while (true) {

cout << "\nMenu:" << endl;

cout << "1. Create document" << endl;

cout << "2. Edit document" << endl;

cout << "3. View documents" << endl;

cout << "4. Delete document" << endl;

cout << "5. Exit" << endl;

cout << "Choose an action: ";

choice = GetIntNumber();

switch (choice) {

case 1:

{

string title, content;

cout << "Enter the document title: ";

cin.ignore();

getline(cin, title);

cout << "Enter the document content: ";

getline(cin, content);

SmartPointer<Document<string>> newDocument = documentManager.CreateDocument(title, content);

cout << "Document created with ID: " << newDocument->getID() << endl;

}

break;

case 2:

{

int docID;

cout << "Enter the ID of the document to edit: ";

docID = GetIntNumber();

documentManager.EditDocument(docID);

}

break;

case 3:

cout << "\nHere are your documents: \n";

documentManager.PrintAllDocuments();

break;

case 4:

{

int docID;

cout << "Enter the ID of the document to delete: ";

docID = GetIntNumber();

documentManager.DeleteDocument(docID);

}

break;

case 5:

cout << "Exiting the program." << endl;

return;

default:

cout << "Invalid choice. Please try again." << endl;

}

}

}

//Document

template<typename T>

int Document<T>::count = 0;

template<typename T>

Document<T>::Document() : title(""), content(0) {

++count;

id = count;

}

template<typename T>

Document<T>::Document(const string& title, const T& content) : title(title), content(content) {

++count;

id = count;

}

template<typename T>

Document<T>::Document(const Document& other) : title(other.title), content(other.content) {

++count;

id = count;

}

template<typename T>

Document<T>::~Document() {

}

template<typename T>

string Document<T>::getTitle() const {

return title;

}

template<typename T>

void Document<T>::setTitle(const std::string& title) {

this->title = title;

}

template<typename T>

T Document<T>::getContent() const {

return content;

}

template<typename T>

void Document<T>::setContent(const T& content) {

this->content = content;

}

template<typename T>

int Document<T>::getID() const {

return id;

}

**Скриншоты выполнения индивидуального задания**

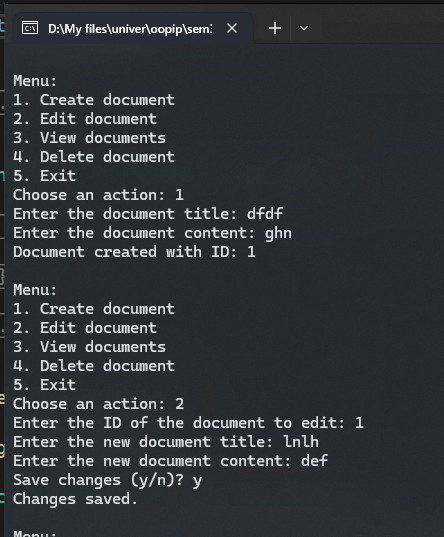


Рисунок 1 - создание и редактирование нового объекта

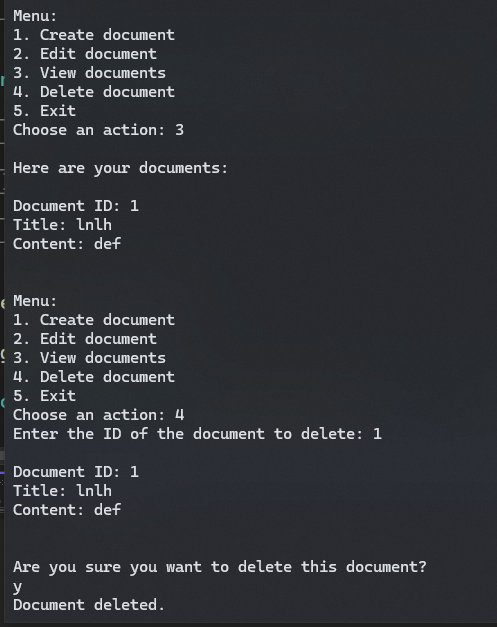
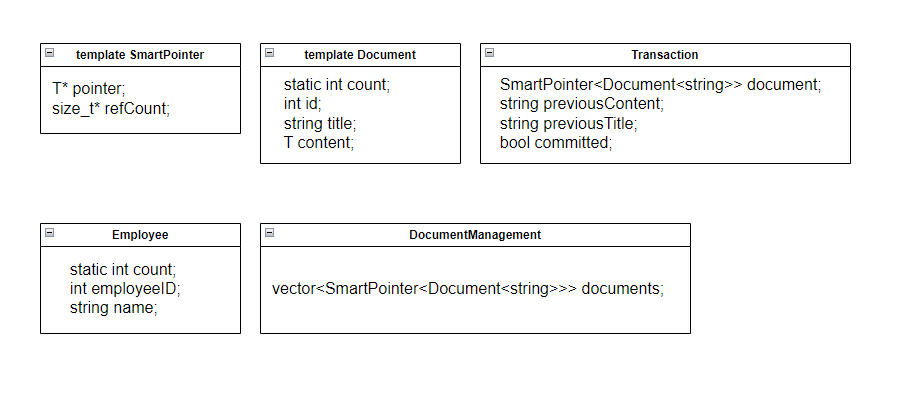


Рисунок 2 – просмотр и удаление объекта

**Диаграмма всех созданных классов**



**Выводы:** изучили принципы и получили практические навыки при использовании шаблонных классов; рассмотрели случаи, когда необходимо использовать транзакции; решили предоставленные задачи.