МИНИСТЕРСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«БЕЛОРУССКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ СВЯЗИ»

ФАКУЛЬТЕТ ЭЛЕКТРОСВЯЗИ

КАФЕДРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СЕТЕЙ ТЕЛЕКОММУНИКАЦИЙ

Допущен к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Отметка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата и подпись преподавателя) (дата и подпись преподавателя)

ПРОГРАММА УЧЁТА ТОВАРОВ НА СКЛАДЕ

Пояснительная записка

к курсовому проекту

по дисциплине

«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»

|  |  |
| --- | --- |
| Выполнил учащийся гр. ТП312 | Г.В. Рудский |
| Руководитель | А.Б. Малолеткин |

Минск

2025

МИНИСТЕРСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«БЕЛОРУССКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ СВЯЗИ»

Обособленное подразделение

«КОЛЛЕДЖ ИНФОРМАЦИОННО-КОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ»

КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СЕТЕЙ ТЕЛЕКОММУНИКАЦИЙ

УТВЕРЖДАЮ

Председатель предметной

(цикловой) комиссии

\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**ЗАДАНИЕ**

**по курсовому проекту**

Обучающемуся\_\_\_\_\_\_\_\_\_\_\_Рудскому Геннадию\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, собственное имя,

\_Валерьевичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

отчество (если такое имеется)

курса\_\_\_\_\_3\_\_\_\_\_\_\_учебной группы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ТП312\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

по учебному предмету, модулю\_\_\_\_Объектно-ориентрованного программирования \_\_\_\_\_\_\_\_

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

Тема курсового проекта\_\_\_\_\_\_ Программа учёта товаров на складе\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Исходные данные \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

**АННОТАЦИЯ**

Данный курсовой проект посвящен проектированию и разработке программного обеспечения для учёта товаров на складе.

Первый раздел посвящен анализу предметной области и формулированию функциональных требований. Здесь представлен обзор основных целей, стоящих перед разрабатываемой программой. Затем рассмотрена постановка задач, определенные для успешной реализации проекта.

В этом разделе представлено теоретическое обоснование выбранных технологий, паттернов и языка программирования, а также представлена краткая базовая информация о них.

В третьем разделе представлено описание программного кода в соответствии с порядком разработки, начиная с основных элементов главной страницы и заканчивая иерархией классов. Приводится подробное описание форм, включая особенности их реализации и взаимодействия.

В четвертом разделе представлены результаты тестирования пользовательского интерфейса разработанного программного обеспечения, а также результаты тестирования его функциональности.

Объем пояснительной записки составляет \_ листа и содержит \_ рисунков, \_ источников литературы, \_ таблицу и \_ приложений.

**СОДЕРЖАНИЕ**

[1 АНАЛИЗ И РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 8](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346974)

[1.1 Анализ предметной области и формулирование функциональных требований 8](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346975)

[1.2 Постановка задач 9](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346976)

[2 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ 10](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346977)

[2.1. Выбор языка программирования 10](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346978)

[2.2 Объектно-ориентированное программирование 11](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346979)

[2.3 Наследование 11](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346980)

[2.4 Инкапсуляция 12](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346981)

[2.5 Конструкторы 13](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346982)

[2.6 Деструкторы 14](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346983)

[2.7 Библиотеки 14](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346984)

[2.8 Работа с формами Windows Forms (WinForms) 15](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346985)

[2.9 Создание приложения Windows Forms в среде Microsoft Visual Studio 2021 15](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346986)

[2.10 Отображение данных в Windows Forms 17](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346987)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 18](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346988)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42](file:///D:\docs\study3kurs\ООП\курсач\Мищенко2.docx#_Toc148346996)

ВВЕДЕНИЕ

В условиях цифровизации бизнес-процессов автоматизация складского учёта становится неотъемлемой частью эффективного управления логистикой и товарными запасами. Ручной учёт подвержен ошибкам, требует значительных временных затрат и не обеспечивает оперативного доступа к информации. Разработка программного решения для учёта товаров на складе позволяет повысить точность, прозрачность и скорость обработки данных, что особенно актуально для предприятий малого и среднего бизнеса.

Объектом исследования является процесс учёта товарных единиц на складе.

Предметом исследования являются программные средства и алгоритмы, обеспечивающие автоматизированный учёт товаров, включая их регистрацию, сортировку, поиск и доступ по ролям

Целью курсовой работы является разработка программного обеспечения для учёта товаров на складе с реализацией функционала авторизации, разграничения прав доступа и фильтрации товаров по заданным критериям.

Для достижения поставленной цели необходимо решить следующие задачи:

* определить требования к функционалу системы учёта товаров;
* разработать архитектуру программы с учётом ролей администратора и пользователя;
* реализовать механизм авторизации и разграничения прав доступа;
* разработать интерфейс ввода, хранения и отображения информации о товарах;
* реализовать индивидуальную функцию фильтрации товаров по сроку хранения и стоимости;
* протестировать работоспособность программы, удобность использования и оценить её эффективность.

В процессе выполнения курсовой работы применялись следующие методы:

* общенаучные: анализ, синтез, моделирование;
* специальные: объектно-ориентированное программирование, тестирование программного обеспечения, сравнительный анализ существующих решений.

1 АНАЛИЗ И РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

**1.1 Анализ предметной области и формулирование функциональных требований**

В условиях цифровизации бизнес-процессов автоматизация складского учёта становится важным элементом эффективного управления логистикой и товарными запасами. Ручной учёт, применяемый на многих предприятиях, характеризуется высокой трудоёмкостью, подвержен ошибкам и не обеспечивает оперативного доступа к актуальной информации. Особенно остро эта проблема стоит перед малым и средним бизнесом, где ресурсы ограничены, а точность учёта критична для финансовой устойчивости.

Анализ предметной области показал, что ключевыми задачами складского учёта являются:

* регистрация поступающих товаров;
* хранение информации о товарных единицах;
* сортировка и фильтрация по заданным критериям;
* разграничение прав доступа к данным;
* обеспечение безопасности и надёжности хранения информации.

На основе анализа были сформулированы следующие функциональные требования к программному продукту:

* регистрация товаров с указанием наименования, количества, цены, даты поступления и ФИО ответственного лица;
* хранение и отображение информации в структурированном виде;
* реализация авторизации пользователей с разграничением прав доступа (администратор/пользователь);
* возможность фильтрации товаров по сроку хранения и стоимости;
* сортировка списка товаров в алфавитном порядке;
* обеспечение удобного пользовательского интерфейса;
* сохранение данных между сессиями работы программы.

Таким образом, программный продукт должен представлять собой систему, способную автоматизировать ключевые процессы складского учёта, повысить точность обработки информации и упростить доступ к ней для разных категорий пользователей.

1.2 Постановка задач

Целью курсовой работы является проектирование и реализация программного обеспечения, обеспечивающего автоматизированный учёт товаров на складе с возможностью авторизации, фильтрации и сортировки данных.

Для достижения поставленной цели необходимо решить следующие задачи:

* Изучить особенности складского учёта и выявить основные проблемы, решаемые с помощью автоматизации;
* Определить требования к функционалу системы, включая роли пользователей и администраторов;
* Разработать архитектуру программы, отражающую структуру предметной области;
* Реализовать механизм авторизации и разграничения прав доступа;
* Разработать интерфейс для ввода, хранения и отображения информации о товарах;
* Реализовать функцию фильтрации товаров, хранящихся более x месяцев и стоимостью более y рублей;
* Обеспечить сортировку списка товаров в алфавитном порядке;
* Реализовать программный продукт на языке C++, с применением принципов объектно-ориентированного программирования;
* Провести тестирование программы на корректность работы, удобство использования и соответствие функциональным требованиям.

Решение этих задач позволит создать надёжную, удобную и эффективную систему складского учёта, адаптированную под реальные условия эксплуатации.

2 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

Для реализации моих целей и задач мне необходимо знания следующего материала: язык программирования C++, ООП, инкапсуляция, наследование, полиморфизм, классы и методы, агрегация и композиция, перегрузки, виртуальные функции, обработка исключительных операций, шаблоны функций и классов.

2.1 Выбор языка программирования

Для реализации программного продукта, предназначенного для автоматизации учета товаров на складе, был выбран язык программирования C++. Данный язык является одним из наиболее распространённых и мощных инструментов промышленной разработки, особенно в задачах, требующих высокой производительности, гибкости архитектуры и точного контроля над ресурсами.[1]

Одним из ключевых факторов выбора C++ стало наличие полноценной поддержки объектно-ориентированного программирования (ООП), что позволяет структурировать систему учета в виде логически связанных сущностей: товары, пользователи, администраторы, складские операции. Благодаря принципам инкапсуляции, наследования и полиморфизма, можно эффективно реализовать разграничение прав доступа, обработку товарных данных и расширяемость функционала.

Кроме того, механизм шаблонов и STL (Standard Template Library) предоставляет готовые решения для хранения, сортировки и фильтрации данных – что особенно важно для реализации индивидуального задания: вывод списка товаров, хранящихся более x месяцев и стоимостью более y рублей. Использование контейнеров STL (таких как vector, map, set) и алгоритмов (sort, find\_if, copy\_if) позволяет сократить время разработки и повысить надёжность кода.[1]

Язык C++ также обеспечивает:

* Высокую производительность, что важно при работе с большими объёмами складских данных;
* Точный контроль над памятью, включая работу с указателями и динамическими структурами;
* Строгую типизацию, способствующую снижению количества ошибок;
* Поддержку исключений, что позволяет грамотно обрабатывать ошибки при вводе данных и работе с файлами.

Дополнительным аргументом в пользу C++ является его стандартизация ANSI/ISO, что гарантирует стабильность синтаксиса и совместимость с современными компиляторами. Это делает язык надёжным выбором для разработки программ, которые могут использоваться в реальных условиях складского учета.

Таким образом, C++ предоставляет все необходимые инструменты для реализации функционала авторизации, работы с товарными данными, фильтрации по заданным критериям и построения масштабируемой архитектуры приложения.

**2.2 Объектно-ориентированное программирование (ООП)**

Объектно-ориентированное программирование (ООП) – это методология программирования, которая основана на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса. Классы образуют иерархию наследования, а объекты взаимодействуют друг с другом через методы и свойства. ООП позволяет создавать более гибкие и масштабируемые программы, которые легче поддерживать и расширять. Основные принципы ООП включают в себя инкапсуляцию, наследование и полиморфизм. Инкапсуляция позволяет скрыть детали реализации объекта от других объектов, наследование позволяет создавать новые классы на основе существующих, а полиморфизм позволяет использовать объекты разных классов как если бы они были одного класса. ООП используется в различных языках программирования, таких как Java, C++, Python, C#, Ruby, Swift, Kotlin, Objective-C, JavaScript, PHP, Perl, Lua, R, Scala, Go, Dart, Rust, TypeScript, Julia, Groovy, Smalltalk, Eiffel, Ada, Delphi, Visual Basic и другие [3].

ООП способствует:

* логическому структурированию программы;
* упрощению сопровождения и расширения функционала;
* повторному использованию кода;
* повышению читаемости и надёжности системы.

Для задач курсовой работы это означает, что каждый товар может быть представлен как объект с набором характеристик (наименование, количество, цена, дата поступления и т. д.), а действия над ним – как методы, встроенные в класс.

**2.3 Классы и методы**

Классы – это шаблоны, описывающие структуру объектов, а методы – действия, которые эти объекты могут выполнять. В системе учета товаров классы позволяют описать сущности (товар, пользователь, склад), а методы – реализовать действия (добавление, фильтрация, авторизация).

Преимущества:

* чёткая организация логики программы;
* возможность повторного использования;
* удобство в сопровождении и масштабировании.

Классы и методы являются основой всей архитектуры программы, обеспечивая её функциональность и структурную целостность. Пример

class Product {

private:

int quantity;

public:

void setQuantity(int q) {

quantity = q;

}

int getQuantity() const {

return quantity;

}

};

Класс Product содержит свойства и методы. Методы управляют доступом к данным — это основа ООП.

**2.4 Инкапсуляция**

Инкапсуляция ­– один из краеугольных камней объектно-ориентированного программирования. Он подчёркивает, что инкапсуляция – это **объединение данных и функций, которые с ними работают, в единый объект**,а также **сокрытие деталей реализации от внешнего мира**.[2] Это позволяет пользователю взаимодействовать с объектом, не зная, как он устроен внутри, и при этом быть уверенным в корректности работы.

В рамках складской системы инкапсуляция позволяет:

* ограничить прямой доступ к данным о товаре;
* обеспечить корректную обработку информации через проверенные методы;
* повысить безопасность и надёжность хранения данных;
* избежать ошибок, связанных с неконтролируемым вмешательством в структуру объекта.

В примере рассмотрен принцип инкапсуляции:

#include <iostream>

class Product {

private:

int quantity; // Скрытое поле

public:

Product(int q) : quantity(q) {}

int getQuantity() const {

return quantity;

}

void setQuantity(int q) {

if (q >= 0)

quantity = q;

else

std::cout << "Ошибка: отрицательное количество!" << std::endl;

}

};

int main() {

Product item(10);

// Прямой доступ невозможен: item.quantity = 100; // ошибка компиляции

std::cout << "Изначально: " << item.getQuantity() << std::endl;

item.setQuantity(15); // через метод

std::cout << "После изменения: " << item.getQuantity() << std::endl;

item.setQuantity(-5); // некорректное значение

}

Пользователь не может напрямую изменить количество товара – он должен использовать предусмотренную функцию, которая проверит корректность действия.

**2.5 Наследование**

Наследование – это механизм, позволяющий создавать новые классы на основе уже существующих. Это особенно полезно при построении иерархии ролей в системе. Например, базовый класс может описывать общие свойства и действия пользователя, а производный класс администратора – расширять эти возможности, добавляя функции управления складом.[3]

Преимущества наследования:

* повторное использование кода;
* упрощение архитектуры;
* возможность расширения функционала без изменения базовой логики;
* логическая организация ролей и прав доступа.

Механизм работы наследования отображён в примере:

class Figure {

protected:

double x, y; // координаты центра

public:

void setPosition(double newX, double newY) {

x = newX;

y = newY;

}

};

class Circle : public Figure {

public:

double radius;

void setRadius(double r) {

radius = r;

}

};

class Square : public Figure {

public:

double side;

void setSide(double s) {

side = s;

}

};

Класс фигура – родительский, он имеет свои методы и свойства. Классы круг и квадрат – дочерние классы, они имеют те же самые методы и свойства, что и родителя, а кроме них – свои методы и свойства.

**2.6 Полиморфизм**

Полиморфизм – это способность объектов разных классов реагировать на одинаковые команды по-разному. Это позволяет использовать единый интерфейс для работы с различными типами объектов, что делает код гибким и универсальным.[3]

В складской системе полиморфизм позволяет:

* реализовать единые методы для разных ролей;
* обрабатывать действия пользователя и администратора через общие механизмы;
* упростить добавление новых ролей или функций без переписывания существующего кода.

Как полиморфизм работает на примере?

class Figure {

public:

void draw() {

std::cout << "Рисуем фигуру\n";

}

};

class Circle : public Figure {

public:

void draw() {

std::cout << "Рисуем круг\n";

}

};

class Square : public Figure {

public:

void draw() {

std::cout << "Рисуем квадрат\n";

}

};

Метод draw() переопределяется в дочерних классах. При вызове через указатель на Figure будет работать версия, соответствующая типу объекта — это и есть полиморфизм.

**2.7 Агрегация и композиция**

Агрегация и композиция – это способы описания связей между объектами. Агрегация означает, что один объект содержит другой, но тот может существовать независимо. Композиция – более тесная связь, при которой составной объект не может существовать без основного.[3]

В системе учёта товаров:

* склад агрегирует множество товаров;
* пользователь может быть связан с действиями, но существовать независимо от конкретного товара.

Эти принципы позволяют моделировать реальные отношения между сущностями и обеспечивают гибкость архитектуры. Пример агрегации:

class Engine {

public:

void start() {}

};

class Car {

public:

Engine\* engine; // агрегируется, может существовать отдельно

};

Car использует Engine, но не владеет им. Engine может существовать вне Car – это агрегация. Пример композиции:

class Engine {

public:

void start() {}

};

class Car {

private:

Engine engine; // создаётся внутри, уничтожается вместе с Car

};

Car содержит Engine как часть себя. Без Car нет Engine – это композиция.

**2.8 Перегрузка функций и операторов**

Перегрузка – это возможность использовать одну и ту же функцию или оператор для разных типов данных или ситуаций. Это особенно полезно при реализации фильтрации товаров по различным критериям: дате, цене, наименованию.[7]

Преимущества:

* повышение удобства использования;
* сокращение количества уникальных функций;
* улучшение читаемости и логики программы.

В задачах курсовой работы это позволяет реализовать универсальные механизмы сортировки и поиска. Пример перегрузки оператора:

class Point {

public:

int x, y;

Point operator+(const Point& other) {

return {x + other.x, y + other.y};

}

};

Оператор + перегружен для сложения точек. Это делает работу с объектами интуитивной.

**2.9 Виртуальные функции**

Виртуальные функции – это механизм, позволяющий переопределять поведение методов в производных классах. Это важно для реализации полиморфизма и гибкой архитектуры.[4]

В системе учета:

* базовый класс может содержать общие методы;
* производные классы могут изменять их поведение в зависимости от роли пользователя.

Это делает программу адаптируемой к изменениям и расширениям. Пример виртуальной функции:

class Figure {

public:

virtual void draw() {

std::cout << "Фигура\n";

}

};

class Circle : public Figure {

public:

void draw() override {

std::cout << "Круг\n";

}

};

Функция draw() — виртуальная. При вызове через указатель на Figure будет использоваться версия Circle.

**2.10 Обработка исключений**

Обработка исключений – это механизм, позволяющий реагировать на ошибки во время выполнения программы. Это важно для обеспечения устойчивости и надёжности системы.[6]

В программе учёта товаров:

* ошибки при вводе данных;
* проблемы с доступом к файлам;
* некорректные действия пользователя – всё это может быть обработано без аварийного завершения работы.

Преимущества:

* повышение стабильности;
* улучшение пользовательского опыта;
* упрощение отладки.

Пример обработки исключений:

void divide(int a, int b) {

if (b == 0)

throw std::runtime\_error("Деление на ноль!");

}

Если b == 0, выбрасывается исключение. Это защищает от аварийных ситуаций.

**2.11 Шаблоны функций и классов**

Шаблоны – это универсальные конструкции, позволяющие создавать функции и классы, работающие с разными типами данных. Это особенно полезно при реализации универсальных алгоритмов сортировки, поиска, фильтрации.[7]

Преимущества:

* сокращение объёма кода;
* повышение гибкости;
* возможность повторного использования.

Пример шаблона функции:

template <typename T>

T add(T a, T b) {

return a + b;

}

Функция add работает с любым типом T. Это шаблон — универсальный код.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бьерн Страуструп – «Язык программирования C++» - 2013 год.
2. **Стивен Прата – «C++ Лекции и упражнения» (C++ Primer Plus)** - 2020 год.
3. Роберт Лафоре. "Объектно-ориентированное программирование в Turbo C++" - 2017 год.

4. Стэнли Б. Липпман, Жози А. Лаутер, Барбара Э. Му. "C++ Primer. Практика применения языка" - 2019 год.

5. Ш. Мейерс. "Эффективное использование C++. 55 верных способов улучшить структуру и дизайн ваших программ" - 2016 год.

6. Герб Саттер. "Исключения и ошибки в C++" - 2017 год.

7. Скотт Мейерс. "Эффективное и современное C++. 42 рекомендации по использованию C++11 и C++14" - 2016 год.