МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания
к лабораторным работам по дисциплине
для студентов дневной и заочной форм обучения направлений
09.03.02 – «Информационные системы и технологии»
09.03.03 – «Прикладная информатика»

В двух частях

Часть 2

Севастополь
СевГУ
2024

УДК:004.434:519.682.2(076.5)
ББК 32.973.2я73
О-294

Р е ц е н з е н т ы:
В. Ю. Карлусов - канд. техн. наук,
доцент кафедры «Информационные системы»
И. В. Кудрявченко - канд. техн. наук, доцент кафедры
«Радиоэлектронные системы и технологии»

Ответственный за выпуск:
И. П. Шумейко – канд. физ.-мат. наук, доцент, заведующий
кафедрой информационных систем

Составитель: Т. И. Сметанина

О-294 Объектно-ориентированное программирование : методические указания к выполнению лабораторных работ по дисциплине для студентов дневной и заочной форм обучения направлений подготовки 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика». В 2-х частях. Ч. 2 / Севастопольский государственный университет ; сост. Т. И. Сметанина. – Севастополь : СевГУ, 2024. – 49 с. – Текст : электронный.

УДК 004.434:519.682.2(076.5)
ББК 32.973.2я73

Рассмотрены и утверждены на заседании кафедры информационных систем, протокол № 08 от 16 апреля 2024 г.

Рассмотрены и рекомендованы к изданию на заседании Учёного совета Института информационных технологий, протокол № 05 от 18 апреля 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения лабораторных работ – приобретение практических навыков написания программ на объектно-ориентированном языке программирования – C++. В результате выполнения лабораторных работ студенты углубляют знания основных теоретических положений дисциплины «Объектно-ориентированное программирование», решая практические задачи на ЭВМ.

По окончанию курса студенты должны овладеть объектно-ориентированным подходом решения практических задач, овладеть инструментами, сопровождающими разработку программных систем с использованием этого подхода.

Выбор вариантов и график выполнения лабораторных работ

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены в каждой лабораторной работе и уточняются преподавателем.

Лабораторная работа выполняется в два этапа. На первом этапе, выполняемом самостоятельно дома, студенту необходимо выполнить следующее:

- проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и ответить на контрольные вопросы;
- разработать алгоритм решения задачи;
- оформить результаты первого этапа в виде заготовки отчета по выполнению лабораторной работы (студенты-заочники первый этап выполнения лабораторных работ оформляют в виде контрольной работы, которую сдают на кафедру до начала зачетно-экзаменационной сессии).

На втором этапе, выполняемом в лабораториях кафедры, студенту следует:

- написать программу на языке C++, реализующую разработанный алгоритм;
- отладить программу;
- разработать и выполнить тестовые примеры (не менее двух);
- подготовить и защитить отчёт по лабораторной работе.

Студенты должны выполнять и защищать работы строго по графику.

График уточняется преподавателем, ведущим лабораторные занятия.

Требования к оформлению отчета

Отчёты по лабораторной работе оформляются каждым студентом индивидуально. В общем случае отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановку задачи; разработку и описание алгоритма решения задачи; разработку и описание программы; выводы по работе; приложения. Содержание отчета приведено в методических указаниях к каждой лабораторной работе.

Постановка задачи представляет изложение содержания задачи, определение списка подзадач для её решения. На этом этапе должны быть полностью определены

исходные данные, перечислены задачи по преобразованию и обработке входных данных, описаны выходные данные, дана характеристика результатов.

Разработка и описание программы включает описание вычислительного процесса на языке C++. Кроме текста программы, в отчёте предоставляется её пооператорное описание (комментарии в тексте программы), даётся характеристика конструкций языка, обеспечивающих решение задачи конкретной лабораторной работы.

В приложениях приводятся текст программы, результат выполнения тестовых примеров.

Лабораторная работа № 5
ИССЛЕДОВАНИЕ МЕХАНИЗМА ДРУЖЕСТВЕННОСТИ

5.1 Цель работы

Исследование основных средств описания дружественных функций и дружественных классов, и особенности использования их при написании объектно-ориентированных программ.

5.2 Краткие теоретические сведения

5.2.1 Дружественные функции

Обычный способ доступа к закрытым элементам класса – использование методов с открытым доступом. Однако C++ поддерживает и другой способ получения доступа к закрытым элементам класса – с помощью дружественных функций. Дружественные функции не являются элементами класса, но, тем не менее, имеют доступ к его закрытым элементам. Более того, одна такая функция может иметь доступ к закрытым элементам нескольких классов.

Чтобы объявить функцию дружественной некоторому классу, в определение этого класса включают ее прототип, перед которым ставится ключевое слово `friend`.

Рассмотрим пример дружественной функции к нескольким классам.

```
#include<iostream>
using namespace std;
class B;                                  // неполное описание класса B
class A {
    int a;                                // скрытое поле
    public: A(int _a) {  a=_a; }          // конструктор
    friend int f(A ob_a, B ob_b);         // объявление друж. функции
};
class B {
    int b;                                // скрытое поле
    public: B(int _b) {  b=_b;}           // конструктор
    friend int f(A ob_a, B ob_b);         // объявление друж. функции
};
int f(A ob_a, B ob_b) {                   // описание друж. функции
    return (ob_a.a + ob_b.b); }
int main() {
    A ob_a(2);   B ob_b(3);               // создание объектов
    cout<<f(ob_a, ob_b);                  // вызов друж. функции
}
```

Также эта программа демонстрирует важный случай применения неполного объявления класса – без применения этой конструкции в данном случае было бы невозможно объявить дружественную функцию для двух классов. Неполное объявление класса `class B;` дает возможность использовать его имя в объявлении дружественной функции еще до его определения.

Дружественная функция не является элементом класса, в котором она объявлена. Поэтому, вызывая дружественную функцию, не нужно указывать имя объекта или указатель на объект и операцию доступа к члену класса (точку или стрелку). Доступ к закрытым элементам класса дружественная функция получает только через объект класса, который, в силу этого, должен быть либо объявлен внутри функции, либо передан ей в качестве аргумента. Дружественная функция не наследуется, то есть она не является таковой для производных классов. С другой стороны, функция может быть дружественной сразу нескольким классам. Использования дружественных функций нужно по возможности избегать, поскольку они нарушают принцип инкапсуляции и, таким образом, затрудняют отладку и модификацию программы.

5.2.2 Дружественные классы

Для создания дружественного класса достаточно включить в объявление класса имя другого класса, объявляемого дружественным, перед которым ставится ключевое слово `friend`.

Класс не может объявить сам себя другом некоторого другого класса. Для того чтобы механизм дружественности сработал, он должен быть объявлен дружественным в другом классе, к которому нужно получить доступ.

В приведенном ниже примере класс B объявляется дружественным классу A:

```
#include<iostream>
using namespace std;
class A{
 friend class B;            //  класс B объявлен другом класса A
    int x;                   // скрытое поле т.к. по умолчанию private
    void inc_x(){x++;}
public:                       // открытые поля
    A(){x=0;}                // конструктор по умолчанию
    A(int _x){x=_x;}         // конструктор с параметрами
};
class B{
    A obA;
    public: void show();  //объявление метода show
};
void B::show(){        //описание метода show
   cout<<obA.x<<endl;
   obA.inc_x();
   cout<<obA.x;
```

```
 }
int main(){     B obB;   //создание объекта класса B
  obB.show(); // печать скрытых полей
}
```

Два класса могут объявить друг друга друзьями. С практической точки зрения такая ситуация свидетельствует о плохой продуманности иерархии классов, тем не менее, язык C++ допускает и такую возможность. В этом случае объявления классов должны иметь вид:

```
class B; //Неполное объявление класса
class A { friend class B;  /*...*/ };
class B { friend class A;  /*...*/ };
```

Неполное объявление класса, которое приведено в данном фрагменте, может понадобиться, только если в классе A имеется ссылка на класс B, например, в параметре метода класса.

5.2.3 Свойства дружественности

C++ предоставляет возможность обойти (или нарушить) одну из основополагающих концепций ООП – концепцию инкапсуляции данных – с помощью друзей. Но использовать ее без веских причин не стоит. Обычно такой метод доступа используется, например, когда из функции необходимо получить доступ к приватным полям и методам сразу нескольких классов одновременно. Однако, если есть возможность решить проблему без использования дружественности – то лучше так и сделать.

По отношению к дружественным классам действуют следующие правила:

- дружественность не является взаимным свойством: если класс B друг класса A, то это не означает, что класс A является другом для класса B;
- дружественность не наследуется: если класс B друг класса A, то классы, производные от B, не являются друзьями A;
- дружественность не распространяется на потомков базового класса: если класс B друг класса A, то B не является другом для классов, производных от A.

5.3 Порядок выполнения лабораторной работы

5.3.1 В ходе самостоятельной подготовки изучить основы работы с «дружественными» классами и функциями.

5.3.2 Разработать программу на языке C++, которая демонстрирует использование дружественной функции. Программа должна содержать описание трех или более классов и вызов дружественной функции.

5.3.3 Разработать тестовые примеры.

5.3.4 Выполнить отладку программы.

5.3.5 Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

5.3.6 Оформить отчет по проделанной работе.

5.4 Варианты заданий

Для класса, заданного по варианту в лабораторной работе №4, описать дружественную функцию, получающую доступ к закрытым полям и, при наличии, к закрытым методам.

5.5 Содержание отчета о выполнении лабораторной работы

Титульный лист, наименование работы, цель работы, задание, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

5.6 Контрольные вопросы

5.6.1 Объясните понятие «дружественность». Перечислите виды «друзей» класса.

5.6.2 Расскажите о дружественных функциях. Приведите пример.

5.6.3 Расскажите о дружественных классах. Приведите пример.

5.6.4 Объясните понятие «неполное объявление класса» и его применение.

5.6.5 Назовите положительные и отрицательные стороны использования механизма дружественности.

5.6.6 Перечислите свойства данного механизма.

Лабораторная работа № 6
ИССЛЕДОВАНИЕ ПЕРЕГРУЗКИ ОПЕРАТОРОВ

6.1 Цель работы

Исследование назначения и средств создания перегруженных операторов при написании объектно-ориентированных программ.

6.2 Краткие теоретические сведения

6.2.1 Перегрузка операторов

C++ позволяет переопределить действие большинства операций так, чтобы при использовании с объектами класса, заданного пользователем, они выполняли свои стандартные функции. Это позволяет использовать привычные операторы совместно с объектами объявленных пользователем типов так же просто, как и с переменными стандартных типов языка C++.

Перегруженные операторы реализуются как функции с помощью ключевого слова `operator`. Обозначения собственных операций вводить нельзя. Можно перегружать любые операции, существующие в C++, за исключением:

Таблица 6.1 – Операции, не подлежащие перегрузке

Оператор	Название
`.`	выбор элемента
`.*`	выбор элемента по указателю
`::`	оператор расширения области видимости
`?:`	оператор условия
`#`	препроцессорный символ
`##`	препроцессорный символ

Функции перегруженных операторов, за исключением `new` и `delete`, должны подчиняться следующим правилам:

- операторная функция должна быть либо нестатическим методом класса, либо принимать аргумент типа класса, или аргумент, который является ссылкой на тип класса;

- операторная функция унарного оператора, объявленная как метод, не должна иметь параметров; если же она объявлена как глобальная функция, она должна иметь один параметр;

- операторная функция бинарного оператора, объявленная как метод, должна иметь один параметр; если же она объявлена как глобальная функция, она должна иметь два параметра;

- операторная функция не может изменять число аргументов или приоритеты операций и их порядок выполнения по сравнению с использованием соответствующего оператора для встроенных типов данных;

- операторная функция не может иметь параметров по умолчанию;

- операторная функция не может быть переопределена для стандартных типов данных;

- операторные функции `=, (), [] и ->` должны быть нестатическими методами (и не могут быть глобальными функциями);

- за исключением оператора присваивания**, все операторные функции класса наследуются его потомками.**

6.2.2 Перегрузка унарных операторов

Унарная функция-операция, определяемая внутри класса, должна быть представлена с помощью нестатического метода без параметров, при этом неявным операндом является вызвавший ее объект, который передается ей через указатель `*this`, например:

```
class time {
  int hour, min;
public:
    time() {hour=min=12;}
    show(){cout<<"hour="<<hour<<" min="<<min<<endl;
    }                    // унарный оператор – метод класса
    time operator--(){
        hour=min=-2;return *this;}
   //…
};
// вызов:
int main(){
time ob1;
    --ob1;               //  вызов 1 способ
ob1.operator--();        //  вызов 2 способ
}
```

Если функция определяется вне класса как дружественная, то она должна иметь один параметр типа класса:

```
class time {
    //…       // объявление друж. функции
    friend bool operator!(time x);
    //…
};
bool operator!(time x){          // описание друж. функции
    if (x.hour==x.min) return true;
    else return false;
}
```

```
// вызов:
time ob1;
     if(!ob1) cout<< "=="; // 1 способ
     else cout<<"!=";
      if(operator!(ob1)) cout<< "=="; // 2 способ
       else cout<<"!=";
}
```

Операции постфиксного инкремента и декремента должны иметь первый параметр типа `int`. Он используется только для того, чтобы отличить их от префиксной формы:

```
class time {
     //…
     time operator--(int){
         hour=-5;min=-7;return *this;
     }
     //…
};
// вызов:
int main(){
     time ob1;
     ob1--;                  // вызов 1 способ
     ob1.operator--(5);      // вызов 2 способ
}
```

6.2.3 Перегрузка бинарных операторов

Бинарная функция-операция, определяемая внутри класса, должна быть представлена с помощью нестатического метода с одним параметром, при этом вызвавший ее объект передается в функцию в качестве неявного параметра с помощью указателя `*this`:

```
class time {
     //…
     bool operator>(time ob){return hour>ob.hour;}
     //…
};
// вызов:
int main(){
  time ob1, ob2;
  ob2--;
  if(ob1>ob2) cout<<"yes"; // 1 способ
  else cout<<"no";
```

```
    if(ob1.operator>(ob2)) cout<<"yes";   // 2 способ
    else cout<<"no";
}
```

Если функция определяется вне класса, она должна иметь два параметра, один из которых обязательно должен быть типа класса:

```
class time {
    //…
    friend int operator+(time x, time y);
    //…
};
// описание:
int operator+(time x, time y){return x.hour+y.hour;}
// вызов:
int main(){
  time ob1, ob2;
  ob2--;
    cout<<ob1+ob2;                    // 1 способ
    cout<<operator+(ob1,ob2);         // 2 способ
}
```

6.2.4 Перегрузка операции присваивания

Операция присваивания определена в любом классе по умолчанию как поэлементное копирование. Эта операция вызывается каждый раз, когда одному существующему объекту присваивается значение другого. Если класс содержит поля, память под которые выделяется динамически, необходимо определить собственную операцию присваивания. Чтобы сохранить семантику присваивания, операция-функция должна возвращать ссылку на объект, для которого она вызвана, и принимать в качестве параметра единственный аргумент — ссылку на присваиваемый объект.

```
const time& operator = (const time &ob){
    // Процесс копирования…
return *this;
}
```

Проще говоря, операция присваивания **всегда** переопределяется как метод класса. Кроме того, из логики работы оператора следует, что операция присвоения **не наследуется –** в самом деле, у наследника могут быть свои собственные поля, с которыми операция базового класса не сможет работать корректно.

6.2.5 Особенности перегрузки операторов

Перегрузка операторов создавалась как механизм, позволяющий упростить написание кода и сделать его более простым и понятным при чтении. Однако чрезмерное и бездумное применение данного механизма приводит к противоположному результату. Чтобы избежать этого, необходимо учитывать следующие рекомендации:

- если семантически нет разницы, как определять оператор, то лучше его оформить в виде метода класса. Это более целесообразно с точки зрения объектного подхода и несет смысл связывания оператора с конкретным классом, которому он принадлежит;

- крайне не рекомендуется менять семантический смысл оператора. Делать подобное не запрещено, однако это сильно затрудняет чтение и понимание кода. Поэтому, если необходимо выполнить действие, которому нет аналога среди множества операторов языка – лучше написать отдельную функцию;

- тип возвращаемого значения сильно зависит от сути оператора. Логические операторы должны возвращать в худшем случае `int`, а в лучшем `bool`. Для операторов присваивания необходимо возвращать ссылку на измененный элемент. Унарные операторы возвращают сам объект или ссылку на него.

Следуя подобным простым правилам, можно существенно улучшить качество кода своей программы.

6.3 Порядок выполнения лабораторной работы

6.3.1 В ходе самостоятельной подготовки изучить основы работы с перегруженными операторами.

6.3.2 Разработать программу на языке C++, которая должна содержать:

- описание заданного класса с переопределенными для него операторами;
- два объекта полученного класса;
- демонстрацию работы унарных операторов на одном объекте;
- демонстрацию работы бинарных операторов над созданными объектами.

Ввод и вывод данных должен осуществляться перегруженными операторами работы с потоками.

6.3.3 Разработать тестовые примеры.

6.3.4 Выполнить отладку программы.

6.3.5 Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

6.3.6 Оформить отчет по проделанной работе.

6.4 Варианты заданий

Для класса, заданного в 1 лабораторной работе, выполнить следующие действия:

- описать конструкторы и деструктор;

- переопределить оператор вывода в поток <<;
- переопределить оператор ввода из потока >>;
- переопределить заданные по варианту операторы;
- предусмотреть обработку ошибок.

Создать два объекта и на их примере продемонстрировать корректную работу всех перегруженных операторов.

Вариант 1

Создать класс `clerk`, содержащий следующие поля: порядковый номер, фамилия, имя, должность, стаж, подразделение, оклад. Перегрузить операторы:

1) ! как унарный метод класса, проверяющий значение стажа – четное число;

2) -- (префиксный) как унарную дружественную функцию, уменьшающую значение оклада на 3000;

3) + как бинарный метод класса для нахождения суммы окладов двух клерков;

4) == как бинарную дружественную функцию, проверяющую на равенство должностей двух клерков.

Вариант 2

Создать класс `product`, содержащий следующие поля: порядковый номер, название продукта, дата изготовления, срок годности, производитель, цена. Перегрузить операторы:

1) -- (префиксный) как унарный метод класса, уменьшающий значение цены на 10%.

2) ! как унарную дружественную функцию, проверяющую, срок годности больше ли он 3-х месяцев;

3) + как бинарный метод класса, вычисляющий сумму цен двух продуктов;

4) < как бинарную дружественную функцию, сравнивающую на равенство названия двух продуктов.

Вариант 3

Создать класс вектор `student`, содержащий следующие поля: порядковый номер, фамилия, имя, пол, рост, группа. Перегрузить операторы:

1) ++ (префиксный) как унарный метод класса, увеличивающий значение поля рост на 2 см;

2) ! (префиксный) как унарную дружественную функцию, проверяющую, студент поступил в 2022;

3) + как бинарный метод класса, суммирующий значение поля рост у двух студентов;

4) > как бинарную дружественную функцию, сравнивающую имена двух студентов.

Вариант 4

Создать класс `man`, содержащий следующие поля: порядковый номер, фамилия, имя, пол, рост, вес, возраст. Перегрузить операторы:

1) ! как унарный метод класса, проверяющий значение поля вес – больше 80 кг;

2) ++ как унарную дружественную функцию увеличения значения поля возраст на 1;

3) / как бинарный метод класса, который возвращает среднее арифметическое значение по полю рост двух людей;

4) != как бинарную дружественную функцию, проверяющую на неравенство фамилий двух людей.

Вариант 5

Создать класс `books`, содержащий следующие поля: автор книги, название книги, издательство, жанр, дата поступления, стоимость книги. Перегрузить операторы:

1) ++ как унарный метод класса, увеличивающий значение поля стоимость книги на 2%;

2) ! как унарную дружественную функцию, проверяющую, что книга не старше 10 лет;

3) + как бинарный метод класса, складывающий значение полей стоимость двух книг;

4) >= как бинарную дружественную функцию, сравнивающую обе книги напечатаны в одном издательстве.

Вариант 6

Создать класс `hostel`, содержащий следующие поля: номер зачетной книжки, ФИО студента, направление, группа, № общежития, № комнаты, долг за общежитие. Перегрузить операторы:

1) ! как унарный метод класса, проверяющий долг за общежитие отличен от нуля;

2) – как унарную дружественную функцию, уменьшающую значение поля долг на 3000 рублей;

3) + как бинарный метод класса проверки на равенство значений полей направление у двух объектов класса `hostel`;

4) == как бинарную дружественную функцию, проверяющую верно: что оба объекта класса `hostel` проживают в одной комнате.

Вариант 7

Создать класс `route`, содержащий следующие поля: номер маршрута, начальный пункт, конечный пункт, стоимость проезда, время в пути. Перегрузить операторы:

1) + как унарный метод класса, изменяющий значение поля конечный пункт;

2) ! как унарную дружественную функцию, проверяющую на совпадение названий начального и конечного пунктов;

3) <= как бинарный метод класса, сравнивающий значение полей время в пути у двух маршрутов;

4) + как бинарную дружественную функцию, нахождения суммы значений полей время в пути у двух маршрутов.

Вариант 8

Создать класс `bank_client`, содержащий следующие поля: порядковый номер, фамилия, имя, отчество, город, улица, дом, квартира. Перегрузить операторы:

1) ! как унарный метод класса, проверяющий на равенство значение полей номер дома и номер квартиры;

2) – как унарную дружественную функцию изменяющую значение поля город;

3) == как бинарный метод класса, сравнивающий значение поля номер квартиры у двух людей;

4) + как бинарную дружественную функцию проверки: оба клиента банка живут по одному адресу.

Вариант 9

Создать класс `stipends`, содержащий следующие поля: номер зачетной книжки, фамилия, имя, отчество, номер группы, курс, средний балл за последнюю сессию, размер стипендии. Перегрузить операторы:

1) ! как унарный метод класса, увеличивающий значение поля курс на 1;

2) -- как унарную дружественную функцию, проверки значения поля размер стипендии больше 1500 рублей;

3) + как бинарный метод класса, суммирующий значение полей размер стипендии двух студентов;

4) < как бинарную дружественную функцию, сравнивающую размер стипендии двух студентов.

Вариант 10

Создать класс `clerk`, содержащий следующие поля: порядковый номер, фамилия, имя, должность, стаж, подразделение, оклад. Перегрузить операторы:

1) -- как унарный метод класса, изменяющий значение поля подразделение;

2) ++ как унарную дружественную функцию проверки на соответствие «должности программист»;

3) + как бинарный метод класса, суммирующий значение поля стаж у двух клерков;

4) >= как бинарную дружественную функцию, сравнивающую значение поля подразделение у двух клерков.

Вариант 11

Создать класс `books`, содержащий следующие поля: автор книги, название книги, издательство, жанр, дата поступления, стоимость книги. Перегрузить операторы:

1) ! как унарный метод класса, изменяющий значение поля жанр;

2) ++ как унарную дружественную функцию, уменьшающую стоимость книги на 100 рублей;

3) * как бинарный метод класса, проверяющий на равенство полей «автор» двух книг;

4) - как бинарную дружественную функцию нахождения разности значений поля «стоимость» двух книг.

Вариант 12

Создать класс `student`, содержащий следующие поля: порядковый номер, фамилия, имя, пол, рост, группа. Перегрузить операторы:

1) ! как унарный метод класса, изменяющий значение поля группа;

2) -- как унарную дружественную функцию, проверяющую, рост студентки больше 170 см;

3) += как бинарный метод класса, суммирующий значение поля номер у двух студентов;

4) == как бинарную дружественную функцию, проверяющую, верно: что оба студента учатся в одной группе.

Вариант 13

Создать класс `product`, содержащий следующие поля: порядковый номер, название продукта, дата изготовления, срок годности (количество суток), производитель, цена. Перегрузить операции:

1) ! как унарный метод класса, изменяющий значение поля название продукта;

2) ++ как унарную дружественную функцию, проверяющую, цена продукта больше 1000 рублей;

3) - как бинарный метод класса, вычисляющий разность срока годности у двух продуктов;

4) == как бинарную дружественную функцию, сравнивающую даты изготовления двух продуктов.

Вариант 14

Создать класс `clerk`, содержащий следующие поля: порядковый номер, фамилия, имя, должность, стаж, подразделение, оклад. Перегрузить операции:

1) -- как унарный метод класса, уменьшающий оклад на 5 лет;

2) ++ как унарную дружественную функцию проверки, что стаж больше 10 лет;

3) - как бинарный метод класса, вычисляющий разность окладов двух клерков;

4) != как бинарную дружественную функцию, сравнивающую имена двух клерков.

Вариант 15

Создать класс `bank_client`, содержащий следующие поля: порядковый номер, фамилия, имя, отчество, город, улица, дом, квартира. Перегрузить операции:

1) ++ как унарный метод класса, увеличивающий номер дома на 20;

2) -- как унарную дружественную функцию проверки, что № квартиры больше № дома;

3) - как бинарный метод класса, вычисляющий разность значений полей номер квартиры у двух клиентов;

4) < как бинарную дружественную функцию, сравнивающую номера домов у двух клиентов.

Вариант 16

Создать класс `hostel`, содержащий следующие поля: номер зачетной книжки, ФИО студента, направление, группа, № общежития, № комнаты, долг за общежитие. Перегрузить операторы:

1) ! как унарный метод класса, изменяющий значение поля группа;

2) ++ как унарную дружественную функцию проверки на равенство полей № общежития и № комнаты;

3) == как бинарный метод класса, который сравнивает номера комнат у двух студентов;

4) + как бинарную дружественную функцию, суммирующую долг по общежитию двух студентов.

Вариант 17

Создать класс `route`, содержащий следующие поля: номер маршрута, начальный пункт, конечный пункт, стоимость проезда, время в пути. Перегрузить операторы:

1) ++ как унарный метод класса, увеличивающий значение поля стоимость проезда на 5%;

2) ! как унарную дружественную функцию, проверяющую стоимость проезда больше 2500 рублей;

3) + как бинарный метод класса, складывающий стоимость проезда двух маршрутов;

4) >= как бинарную дружественную функцию, сравнивающую время в пути двух маршрутов.

Вариант 18

Создать класс `stipends`, содержащий следующие поля: номер зачетной книжки, фамилия, имя, отчество, номер группы, курс, средний балл за последнюю сессию, размер стипендии. Перегрузить операторы:

1) ! как унарный метод класса, изменяющий значение поля отчество;

2) ++ как унарную дружественную функцию, проверяющую, рост студента мужского пола меньше 180 см;

3) - как бинарный метод класса для определения разности размера стипендии у двух студентов;

4) == как бинарную дружественную функцию, проверяющую, верно: что оба студента учатся на одном курсе.

Вариант 19

Создать класс `clerk`, содержащий следующие поля: порядковый номер, фамилия, имя, должность, стаж, подразделение, оклад. Перегрузить операторы:

1) + как унарный метод класса, изменяющий значение поля должность;

2) ! как унарную дружественную функцию, проверяющую значение оклада меньше 35 лет;

3) / как бинарный метод класса, определяющий среднее арифметическое значение оклада двух клерков;

4) == как бинарную дружественную функцию, проверяющую равенство фамилий двух клерков.

Вариант 20

Создать класс `books`, содержащий следующие поля: автор книги, название книги, издательство, жанр, дата поступления, стоимость книги. Перегрузить операторы:

1) ! как унарный метод класса, изменяющий значение поля издательство;

2) – как унарную дружественную функцию, уменьшающую стоимость книги на 2,5%;

3) == как бинарный метод класса, сравнивающий авторов двух книг;

4) / как бинарную дружественную функцию определения среднего арифметического значения стоимости двух книг.

Вариант 21

Создать класс `route`, содержащий следующие поля: номер маршрута, начальный пункт, конечный пункт, стоимость проезда, время в пути. Перегрузить операторы:

1) -- как унарный метод класса, уменьшающий значение поля время пути на 1,5%;

2) ! как унарную дружественную функцию, проверяющую номер маршрута равен 14;

3) + как бинарный метод класса, определяющий из двух маршрутов номер маршрута с наибольшей стоимостью;

4) < как бинарную дружественную функцию, определения среднего арифметического значения стоимости проезда двух маршрутов.

Вариант 22

Создать класс `bank_client`, содержащий следующие поля: порядковый номер, фамилия, имя, отчество, город, улица, дом, квартира. Перегрузить операторы:

1) -- как унарный метод класса, уменьшающий номер квартиры на 10;

2) ++ как унарную дружественную функцию проверки, что сумма значений полей «№ дома» и «№ квартиры» равна значению поля «номер»;

3) + как бинарный метод класса, проверяющий, что двое людей живут в одном доме, но в разных квартирах;

4) >= как бинарную дружественную функцию проверки сравнения значения полей «город» у двух людей.

Вариант 23

Создать класс `books`, содержащий следующие поля: автор книги, название книги, издательство, жанр, дата поступления, стоимость книги. Перегрузить операторы:

1) ! как унарный метод класса, изменяющий значение поля жанр;

2) ++ как унарную дружественную функцию, проверяющую, что книга поступила после 2000 года;

3) * как бинарный метод класса, умножающий на 2 сумму стоимости двух книг;

4) != как бинарную дружественную функцию проверки, что обе книги написаны в одном жанре.

Вариант 24

Создать класс `route`, содержащий следующие поля: номер маршрута, начальный пункт, конечный пункт, стоимость проезда, время в пути. Перегрузить операторы:

1) ! как унарный метод класса, изменяющий название конечного пункта;

2) -- как унарную дружественную функцию, уменьшающий значение поля стоимость проезда на 3%;

3) -= как бинарный метод класса, определяющий разность стоимости проезда двух маршрутов;

4) == как бинарную дружественную функцию, проверки, что у обоих маршрутов начальные и конечные пункты совпадают.

Вариант 25

Создать класс `hostel`, содержащий следующие поля: номер зачетной книжки, ФИО студента, направление, группа, № общежития, № комнаты, долг за общежитие. Перегрузить операции:

1) ! как унарный метод класса, изменяющий значение поля «направление»;

2) + как унарную дружественную функцию, увеличивающую значение поля «долг» на 3%;

3) += как бинарный метод класса, вычисляющий сумму долга за общежитие двух студентов;

4) == как бинарную дружественную функцию, проверяющую верно, что оба студента учатся в одной группе и оба не имеют долгов по общежитию.

6.5 Содержание отчета о выполнении лабораторной работы

Титульный лист, наименование работы, цель работы, задание, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

6.6 Контрольные вопросы

6.6.1 Опишите механизм перегрузки операторов и его предназначение.

6.6.2 Перечислите операторы, которые можно перегрузить.

6.6.3 Перечислите операторы, которые нельзя перегрузить.

6.6.4 Приведите примеры всех возможных вариантов перегрузки операторов, дайте подробные пояснения.

6.6.5 Перечислите ограничения механизма перегрузки операторов.

6.6.6 Перечислите достоинства и недостатки механизма перегрузки операторов, особенностях его применения.

6.6.7 Приведите примеры перегрузки унарного оператора.

6.6.8 Приведите примеры перегрузки бинарного оператора.

Лабораторная работа № 7
ИССЛЕДОВАНИЕ ШАБЛОНОВ ФУНКЦИЙ

7.1 Цель работы

Исследование назначения и способа описания шаблонов функций, применение их при написании объектно-ориентированных программ.

7.2 Краткие теоретические сведения

7.2.1 Шаблоны и их применение

Шаблоны позволяют давать обобщенные, в смысле произвольности используемых типов данных, определения функций и классов. Поэтому их часто называют параметризованными функциями и параметризованными классами. Часто также используются термины «шаблонные функции» и «шаблонные классы».

Шаблон функции представляет собой обобщенное определение функции, из которого компилятор автоматически создает представитель функции для заданного пользователем типа (или типов) данных. Когда компилятор создает по шаблону функции конкретную ее реализацию, то говорят, что он создал **порожденную функцию**. Синтаксис объявления шаблона функции имеет следующий вид:

```
template <class T1|T1 идент1, class T2|T2 идент2, …,
         class Tn|Tn идентn>
 возвр_тип имя_функции (параметры){
      /*тело функции*/
  }
```

За ключевым словом **`template`** следуют один или несколько параметров, заключенных в угловые скобки, разделенные между собой запятыми. Каждый параметр является либо ключевым словом **`class`**, за которым следует имя типа, либо именем типа, за которым следует идентификатор.

Для задания параметризованных типов данных вместо ключевого слова class может также использоваться ключевое слово **`typename`**. Параметры шаблона, следующие за ключевыми словами **`class`** или **`typename`**, называют **параметризованными типами**. Они информируют компилятор о том, что некоторый (пока что неизвестный) тип данных используется в шаблоне в качестве параметра (в момент вызова шаблона на место такого параметризованного типа станет тип, заданный программистом). Параметры шаблона, состоящие из имени типа и следующего за ним имени идентификатора, информируют компилятор о том, что параметром шаблона является константа указанного типа. Шаблонную функцию можно вызывать как обычную, никакого специального синтаксиса не требуется.

Рассмотрим пример определения и вызова шаблонных функций:

```
#include <iostream>
using namespace std;
template <class T>    // шаблон функции, вычисляющей квадрат
// введенного значения
T Sqr(T x)   {        // здесь T — некий общий тип, который будет
    return x*x;       // задан при вызове функции в теле основной
}                     // программы

template < class T>   // шаблон функции обмена двух элементов
                      // в массиве по значению их индексов
T* Swap(T* t, int ind1, int ind2) {
    T tmp = t[ind1];      // Собственно обмен
    t[ind1] = t[ind2];
    t[ind2] = tmp;
    return t;
}
int main() {
    int n=10,             // для возведения в квадрат
        i = 2, j = 5      // индексы в массиве
    double d = 10.21      // для возведения в квадрат
    char* str = "Шаблон";     // массив букв
    cout << "Значение n = " << n << endl
         << "Его квадрат = " << Sqr (n) << endl
                          // вызов шаблона для возведения
         << "Значение d = " << d << endl
                          // в квадрат целого числа
         <<"Его квадрат = " << Sqr(d) << endl
                          // вызов шаблона для возведения
         << "Исходная строка = " << str << endl
                          // в квадрат веществ. числа
                          // преобразование массива
  << "Преобразованная строка = " << Swap(str, i, j) << endl;
  return 0;
}
```

При выполнении программа выводит на экран следующее:

```
Значение n = 10
Его квадрат = 100
Значение d = 10.21
Его квадрат = 104.2441
```

```
Исходная строка = Шаблон
Преобразованная строка = Шанлоб
```

Обратите внимание на вызовы шаблонных функций, сделанные в предыдущем примере: `Sqr(n)` и `Sqr(d)`. Каждый такой вызов приводит к построению конкретной **порожденной функции**. В данном случае первый вызов приводит к построению компилятором функции `Sqr()` для целого типа данных, во втором — для типа `double`. В связи с этим процесс построения порожденной функции компилятором называют **конкретизацией шаблонной функции.**

Как и для обычных функций, можно создать прототип шаблона функции в виде его предварительного объявления. Например:

```
template < class T>
T* Swap(T* t, int ind1, int ind2);
```

Каждый типовой параметр шаблона должен появиться в списке формальных параметров функции. Например, следующее объявление приведет к ошибке во время компиляции:

```
template < class T, class U>
T FuncName(U);
```

7.2.3 Недостатки шаблонов

Шаблоны предоставляют определенные выгоды при программировании, связанные с широкой применимостью кода и легким его сопровождением. В общем, этот механизм позволяет решать те же задачи, для которых используется полиморфизм. С другой стороны, в отличие от макросов, они позволяют обеспечить безопасное использование типов данных. Однако с их использованием связаны и некоторые недостатки:

- программа содержит полный код для всех порожденных представителей шаблонного класса или функций;
- не для всех типов данных предусмотренная реализация класса или функции оптимальна.

Преодоление второго недостатка возможно с помощью специализации шаблонов.

7.3 Порядок выполнения лабораторной работы

7.3.1 В ходе самостоятельной подготовки изучить основы работы с шаблонами функций и классов.

7.3.2 Разработать программу на языке C++, которая обрабатывает данные разных типов (`int, char`, и др.). Функция обработки данных должна быть реализована как шаблон.

7.3.3 Разработать тестовые примеры.

7.3.4 Выполнить отладку программы.

7.3.5 Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

7.3.6 Оформить отчет по проделанной работе.

7.4 Варианты заданий

Описать функцию-шаблон, заданную по варианту. Проиллюстрировать ее корректную работу на различных по типу наборах данных (не менее трех: **`int`**, **`char`** и др.). А также написать функцию-шаблон для заполнения массива и функцию-шаблон для вывода элементов массива на экран.

Вариант 1

Написать функцию-шаблон, выполняющую транспонирование матрицы.

Вариант 2

Написать функцию-шаблон, меняющую местами *i*-ю строку и *j*-й столбец в квадратной матрице.

Вариант 3

Написать функцию-шаблон сортировки массива по возрастанию методом пузырька.

Вариант 4

Написать функцию-шаблон, определяющую, существуют ли в матрице повторяющиеся строки.

Вариант 5

Написать функцию-шаблон, определяющую элемент, который встречается в массиве максимальное число раз.

Вариант 6

Написать функцию-шаблон, определяющую, существуют ли в матрице повторяющиеся столбцы.

Вариант 7

Написать функцию-шаблон, вычисляющую максимальное значение элемента в массиве.

Вариант 8

Написать функцию-шаблон, переставляющую *i*-ю и *j*-ю строки в матрице.

Вариант 9

Написать функцию-шаблон, записывающую массив в обратном порядке.

Вариант 10

Написать функцию-шаблон, меняющую диагонали матрицы местами.

Вариант 11

Написать функцию-шаблон последовательного поиска в массиве по ключу. Функция возвращает индекс первого найденного элемента в массиве, равного ключу.

Вариант 12

Написать функцию-шаблон, заменяющую в матрице все элементы A на элемент B (значения A и B передаются параметрами в функцию-шаблон).

Вариант 13

Написать функцию-шаблон, циклически сдвигающую одномерный массив элементов n раз (1-й элемент становится 2-м, 2-й – 3-м …*n*-й элемент становится первым).

Вариант 14

Написать функцию-шаблон, меняющую в одномерном массиве соседние элементы (поменять элементы с четными индексами на элементы с нечетными индексами).

Вариант 15

Написать функцию-шаблон, проверяющую матрицу на симметричность.

Вариант 16

Написать функцию-шаблон, проверяющую: верно, что все элементы матрицы одинаковые.

Вариант 17

Написать функцию-шаблон, проверяющую: верно, что элементы в одномерном массиве расположены по убыванию.

Вариант 18

Написать функцию-шаблон, проверяющую: верно, что максимальный и минимальный элемент матрицы расположены в одной строке.

Вариант 19

Написать функцию-шаблон, проверяющую: верно, что в одномерном массиве есть хотя бы два одинаковых элемента.

Вариант 20

Написать функцию-шаблон, проверяющую на равенство две строки матрицы (номера строк вводит пользователь).

Вариант 21

Написать функцию-шаблон, меняющую местами минимальный и максимальный элементы матрицы.

Вариант 22

Написать функцию-шаблон, проверяющую на равенство значений двух элементов массива, индексы которых задает пользователь.

Вариант 23

Написать функцию-шаблон, подсчитывающую количество нулевых элементов.

Вариант 24

Написать функцию-шаблон, сдвигающую элементы одномерного массива вправо на n позиций (значение n вводит пользователь).

Вариант 25

Написать функцию-шаблон, проверяющую: верно, что элементы на главной и вспомогательной диагоналях в квадратной матрице равны.

7.5 Содержание отчета о выполнении лабораторной работы

Титульный лист, цель работы, вариант задания, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

7.6 Контрольные вопросы

7.6.1 Дайте определение понятию «шаблон», приведите пример.
7.6.2 Опишите случаи в которых выгодно применение шаблонов.
7.6.3 Перечислить особенности работы шаблонов.
7.6.4 Дайте определение понятию «порожденная функция».
7.6.5 Перечислить достоинства и недостатки шаблонов
7.6.6 Дайте определение понятию «конкретизация шаблонной функции».
7.6.7 Дайте определение понятию «параметризованные типы».

Лабораторная работа № 8
ИССЛЕДОВАНИЕ СРЕДСТВ УПРАВЛЕНИЯ ПОТОКАМИ ВВОДА-ВЫВОДА. ИССЛЕДОВАНИЕ МЕХАНИЗМА ОБРАБОТКИ ИСКЛЮЧЕНИЙ

8.1 Цель работы

Изучить способы реализации и особенности управления потоками ввода/вывода, исследовать способы генерации и обработки исключений.

8.2 Краткие теоретические сведения

8.2.1 Классы потокового ввода/вывода

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику. Потоки C++, в отличие от функций ввода/вывода в стандартном C, обеспечивают надежную работу как со стандартными, так и с определенными пользователем типами данных, а также единообразный и понятный синтаксис.

Пример:

```
#include <iostream>
using namespace std;
int main(){
  int i;
  cin >> i;                    // ввод переменной
  cout << "Вы ввели " << i; // вывод (цепочка вывода)
 return 0;
}
```

Чтение данных из потока называется извлечением, вывод в поток – помещением, или включением. Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти – буфер. Фактическая передача данных выполняется при выводе после заполнения буфера, а при вводе – если буфер исчерпан.

По направлению обмена потоки можно разделить на входные (данные вводятся в память), выходные (данные выводятся из памяти) и двунаправленные (допускающие как извлечение, так и включение).

По виду устройств, с которыми работает поток, можно разделить потоки на стандартные, файловые и строковые.

Стандартные потоки предназначены для передачи данных от клавиатуры и на экран дисплея, файловые потоки – для обмена информацией с файлами на внешних носителях данных (например, на магнитном диске), а строковые потоки – для работы с массивами символов в оперативной памяти.

Для поддержки потоков библиотека C++ содержит иерархию классов, построенную на основе двух базовых классов – `ios` и `streambuf`. Класс `ios` содержит общие для ввода и вывода поля и методы, класс `streambuf` обеспечивает буферизацию потоков и их взаимодействие с физическими устройствами.

Таблица 8.1 – Классы потокового ввода/вывода и буферизации потоков

`ios`	базовый класс потоков
`istream`	класс входных потоков
`ostream`	класс выходных потоков
`iostream`	класс двунаправленных потоков
`istringstream`	класс входных строковых потоков
`ostringrstream`	класс выходных строковых потоков
`stringstream`	класс двунаправленных строковых потоков
`ifstream`	класс входных файловых потоков
`ofstream`	класс выходных файловых потоков
`fstream`	класс двунаправленных файловых потоков
`iomanip`	класс манипуляторов двунаправленных потоков
`bitset`	класс для работы с двоичными числами
`streambuf`	базовый класс буферизации потоков
`filebuf`	класс буферизации файловых потоков

Основным преимуществом потоков по сравнению с функциями ввода/вывода, унаследованными из библиотеки C, является контроль типов, а также расширяемость, то есть возможность работать с типами, определенными пользователем. Для этого требуется переопределить операции потоков.

К недостаткам потоков можно отнести снижение быстродействия программы, которое, в зависимости от реализации компилятора, может быть весьма значительным.

Заголовочный файл <`iostream`> содержит, кроме описания классов для ввода/вывода, четыре предопределенных объекта потокового ввода/вывода (табл. 8.2).

Эти объекты создаются при включении в программу заголовочного файла <`iostream`>, при этом становятся доступными связанные с ними средства ввода/вывода. Имена этих объектов можно переназначить на другие файлы или символьные буферы.

В классах `istream` и `ostream` операции извлечения из потока >> и помещения в поток << определены путем перегрузки операций сдвига.

Операции извлечения и чтения в качестве результата своего выполнения формируют ссылку на объект типа `istream` для извлечения и ссылку на `ostream` – для включения. Это позволяет формировать цепочки операций, что проиллюстрировано последним оператором приведенного ранее примера. Вывод при этом выполняется слева направо.

Как и для других перегруженных операций, для вставки и извлечения невозможно изменить приоритеты, поэтому в необходимых случаях используются скобки:

```
cout << i + j;      // Скобки не требуются – приоритет сложения
                    // больше, чем <<
cout << (i < j);    // Скобки необходимы – приоритет операции
                    // отношения  меньше, чем <<:
cout << (i << j);   // Правая операция << означает сдвиг
```

Таблица 8.2 – Стандартные потоки для ввода/вывода

Объект	Класс	Описание
`cin`	`istream`	Связывается с клавиатурой (стандартным буферизованным вводом)
`cout`	`ostream`	Связывается с экраном (стандартным буферизованным выводом)
`cerr`	`ostream`	Связывается с экраном (стандартным не буферизованным выводом), куда направляются сообщения об ошибках
`clog`	`ostream`	Связывается с экраном (стандартным буферизованным выводом), куда направляются сообщения об ошибках

8.2.2 Флаги и форматирующие методы

Флаги представляют собой отдельные биты, объединенные в поле `x_flags` типа `long` класса `ios`. Флаги перечислены в таблице 8.3. Форматирующие методы приведены в таблице 8.4.

Таблица 8.3 – Флаги форматирования

Флаг	Умол-чание	Описание действия при установленном бите
`skipws`	+	При извлечении пробельные символы игнорируются
`left`		Выравнивание по левому краю поля
`right`	+	Выравнивание по правому краю поля
`internal`		Знак числа выводится по левому краю, число – по правому.
`dec`	+	Десятичная система счисления
`oct`		Восьмеричная система счисления
`hex`		Шестнадцатеричная система счисления
`showbase`		Выводится основание системы счисления (0x для шестнадцатеричных чисел и 0 для восьмеричных)
`showpoint`		При выводе вещественных чисел печатать десятичную точку и дробную часть
`uppercase`		При выводе использовать символы верхнего регистра
`showpos`		Печатать знак при выводе положительных чисел
`scientific`		Печатать вещественные числа в форме мантиссы с порядком
`fixed`		Печатать вещественные числа в форме с фиксированной точкой
`unitbuf`		Выгружать буферы всех потоков после каждого вывода
`stdio`		Выгружать буферы потоков stdout и stderr после каждого вывода

Примечание: Флаги (`left`, `right` и `internal`), (`dec`, `oct` и `hex`), а также (`scientific` и `fixed`) взаимно исключают друг друга, то есть в каждый момент может быть установлен только один флаг из каждой группы.

Таблица 8.4 – Функции форматирования

Функция	Описание действия
`int width(int w);`	устанавливает ширину ноля вывода в соответствии со значением параметра
`int width();`	возвращает значение ширины поля вывода
`char fill(char);`	устанавливает значение текущего символа заполнения, возвращает старое значение символа
`char fill();`	возвращает текущий символ заполнения
`int precision(int);`	устанавливает значение точности представления при выводе вещественных чисел, возвращает старое значение точности
`int precision();`	возвращает значение точности представления при выводе вещественных чисел
`long flags(long f);`	установить состояния всех флагов
`long flags();`	вернуть состояния всех флагов
`long setf (long setbits, long field);`	присваивает флагам, биты которых установлены в первом параметре, значение соответствующих битов второго параметра
`long setf(long);`	устанавливает флаги, биты которых установлены в параметре
`long unsetf(long);`	сбрасывает флаги, биты которых установлены в параметре

Пример форматирования при выводе с помощью флагов и методов:

```
#include <iostream>
using namespace std;
int main() {
    long a = 1000, b = 077;
    cout.width(7);
    cout.setf(ios::uppercase);
    cout << hex<<a;
    cout.width(7);
    cout << b << endl;
    double d = 0.12, c = 1.3e-4;
    cout.setf(ios::left);
    cout << d << endl;
    cout << c;
    return 0;
}
```

В результате работы программы (рис. 8.1) в первой строке будут прописными буквами выведены переменные `a` и `b` в шестнадцатеричном представлении, под каждую из них отводится по 7 позиций (функция `width` действует только на одно выводимое значение, поэтому ее вызов требуется повторить дважды). Значения переменных `c` и `d` прижаты к левому краю поля.

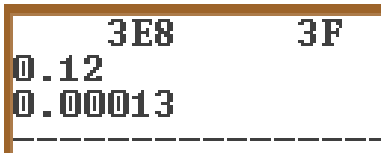

Рисунок 8.1 – Результат работы программы

8.2.3 Манипуляторы

Манипуляторами называются функции, которые можно включать в цепочку операций помещения и извлечения для форматирования данных. Манипуляторы делятся на простые, не требующие указания аргументов, и параметризованные. Пользоваться манипуляторами более удобно, чем методами установки флагов форматирования (табл. 8.5).

Таблица 8.5 – Манипуляторы

Манипулятор	Назначение	Поток
`dec`	Вывод чисел в десятичной системе счисления	Вывод
`endl`	Вывод символа новой строки и флэширование	Вывод
`ends`	Вывод нуля (`NULL`)	Ввод\вывод
`flush`	Флэширование	Вывод
`hex`	Вывод чисел в шестнадцатеричной системе счисления	Вывод
`oct`	Вывод чисел в восьмеричной системе счисления	Вывод
`resetiosflags(long f)`	Сбросить флаги, определяемые f	Ввод\вывод
`setbase(int base)`	Установить основание системы счисления	Вывод
`setfill(char ch)`	Установить символ заполнения ch	Вывод
`setiosflags(long f)`	Установить флаги, задаваемые f	Ввод\вывод
`setprecision(int p)`	Установить точность, равную p	Вывод
`setw(int w)`	Установить ширину поля, равную w	Вывод
`ws`	Пропускает начальный символ-разделитель	

8.2.4 Методы обмена с потоками

В потоковых классах наряду с операциями извлечения >> и включения << определены методы для неформатированного чтения и записи в поток (при этом преобразования данных не выполняются).

В таблице 8.6 приведены функции чтения, определенные в классе `istream`.

Таблица 8.6 – Функции чтения

Функция	Описание действия
`get ()`	возвращает код извлеченного из потока символа или `EOF`
`get (c)`	возвращает ссылку на поток, из которого выполнялось чтение, и записывает извлеченный символ в с
`get(buf,num,lim='\n')`	считывает `num-1` символов или пока не встретится символ `lim` и копирует их в символьную строку `buf`. Вместо символа `lim` в строку записывается признак конца строки (`'\0'`). Символ `lim` остается в потоке. Возвращает ссылку на текущий поток
`getline(buf, num, lim='\n')`	аналогична функции `get`, но копирует в `buf` и символ `lim`
`ignore(num = 1, lim = EOF)`	считывает и пропускает символы до тех пор, пока не будет прочитано `num` символов или не встретится разделитель, заданный параметром `lim`. Возвращает ссылку на текущий поток
`seekg(pos)`	устанавливает текущую позицию чтения в значение `pos` байт от начала файла
`seekg(offs, org)`	перемещает текущую позицию чтения на `offs` байтов, считая от трех позиций, определяемых параметром `org`: `ios::beg` (от начала файла), `ios::cur` (от текущей позиции) или `ios::end` (от конца файла)
`flush()`	записывает содержимое буфера потока вывода на физическое устройство
`put (c)`	выводит в поток символ `c` и возвращает ссылку на поток
`seekg(pos)`	устанавливает текущую позицию записи в значение `pos`

В классе `ostream` определены аналогичные функции для вывода (табл. 8.7).

Таблица 8.7 – Функции вывода

Функция	Описание действия
`seekg (offs, org)`	перемещает текущую позицию записи на `offs` байтов, считая от трех позиций, определяемых параметром `org: ios::beg` (от начала файла), `ios::cur` (от текущей позиции) или `ios::end` (от конца файла)
`write(buf, num)`	записывает в поток `num` символов из массива `buf` и возвращает ссылку на поток

8.2.5 Файловые потоки

Чтобы открыть для вывода файл `myfile.txt` с помощью объекта `ofstream`, необходимо создать экземпляр объекта класса `ofstream` и передать ему имя файла в качестве параметра: `ofstream fout («myfile.txt»)`.

Чтобы открыть этот файл для ввода, применяется та же методика, за исключением того, что используется объект класса `ifstream: ifstream fin («myfile.txt»)`.

Обратите внимание, что `fout` и `fin` не более чем имена объектов; здесь `fout` использовался для вывода в файл подобно тому, как `cout` используется для вывода на экран; `fin` аналогичен `cin`.

Очень важным методом, используемым в файловых потоках, является функция-член `close`(). Каждый раз при открытии файла для чтения или записи (или и того и другого) создается соответствующий объект файлового потока. По завершении работы файл необходимо закрыть (например: `fout.close();`), чтобы впоследствии не повредить его и записанные в нем данные. На практике нередко бывают непредвиденные случаи, когда не закрытые файлы теряют всю внесенную в них информацию.

После того как объекты потока будут ассоциированы с файлами, они используются наравне с другими объектами потока ввода и вывода. Например:

```
#include <fstream>
#include <iostream>
using namespace std;
int main() {
 char buffer[255];        // для ввода данных пользовате-
лем
 cout << "File name: ";
 cin.getline (buffer,255);
 ofstream fout(buffer);    // открыть для записи
  fout << "This line written directly to the file\n";
  cin.getline (buffer,255);    //   получить   данные   от
пользователя
  fout << buffer;              // и запись их в файл
  fout.close();           // закрыть файл
return 0;                 // уходя, гасите свет
}
```

Обычно объект класса `ofstream`, открывая файл для записи, создает новый файл, если таковой не существует, или усекает его длину до нуля, если файл с этим именем уже существует (то есть удаляет все его содержимое). Изменить стандартное поведение объекта `ofstream` можно с помощью второго аргумента конструктора, заданного явно.

Допустимыми аргументами являются:

- `ios::app` – добавляет данные в конец файла, не усекая прежнее его содержимое (`append` - добавлять);

- `ios::ate` – осуществляет переход в конец файла, но запись допускает в любом месте файла (`at end` – в конец);

- `ios::trunc` – задано по умолчанию; усекает существующий файл полностью (`truncate` - обрезать);

- ios::nocreate – открывает существующий файл, если его нет – ошибка (не создавать);

- ios::noreplace – открывает несуществующий файл, иначе выдаст ошибку (не заменять).

8.2.6 Ошибки потоков

Каждый поток (istream или ostream) имеет связанное с ним состояние. Установкой и соответствующей проверкой этого состояния вылавливаются ошибки и нестандартные ситуации. Состояние потока вводится в basic_ios, базовом классе класса basic_stream, в <ios> (табл. 8.8).

Таблица 8.8 – Функции работы с ошибками потоков

Функция	Описание действия
`bool good() const;`	следующая операция может выполниться
`bool fail() const;`	следующая операция не выполнится
`bool eof() const;`	виден конец ввода
`bool bad() const;`	поток испорчен
`iostate rdstate() const;`	получение флагов состояния ввода/вывода
`void clear(iostate f=goodbit);`	сбрасываются флаги ошибок (по умолчанию - все). Обычно после возникновения ошибки нужно сбросить ошибочное состояние, чтобы дальше пользоваться потоком, но этого недостаточно – для восстановления работоспособности еще нужно вручную очистить буфер ввода/вывода.
`void setstate(iostate f) {clear(rdstate() | f);}`	добавление f к флагам состояния
`operator void*() const;`	не ноль, если `!fail()`
`bool operator!() const {return fail();}`	не `good()`

Состояние потока представляется набором флагов. Эти флаги определены в базовом классе ios:

```
enum io_state{
 goodbit = 0x00, //  если бит не установлен, то ошибок нет
 eofbit = 0x01, //  обнаружен конец файла
 failbit = 0x02, //  сбой в последней операции ввода-вывода
 badbit = 0x04, // попытка недопустимой операции
 hardfail = 0x80 //  в потоке невосстанавливаемая ошибка
};
```

Пример: проще всего состояние потока можно проверить как обычное логическое выражение. Обычно нужно проверить, ввел ли пользователь то, что ожидает программа:

```
double d; cin>>d;
/* 1) 2 - преобразуется к d=2.0
   2) 2A – d=2.0 - значение сформировано, но <A> еще осталось в буфере потока и ждет приема char
   3) 3,3 вместо 3.3 – d=3.0, но запятая и вторая 3 осталась в буфере ввода и ждет ввода
4) вводим AAA - поток испорчен, вырабатывается флаг ошибки, пользоваться d нельзя! Значение d не изменилось!!! Весь последующий ввод (cin>>) будет проигнорирован! */
  if(!cin) { /* Сюда попадем, если только поток ввода Ваш ввод никаким образом проинтерпретировать не может (случай 4). Для того, чтобы программу можно было выполнять дальше необходимо:
1) сбросить ошибки */
cin.clear(); // иначе весь последующий ввод будет проигнорирован
// 2) очистить буфер ввода
cin.ignore(MAXINT,'\n');
cin>>d;} // теперь можно вводить снова
```

8.2.7 Обработка исключительных ситуаций

Исключительная ситуация, или *исключение* – это возникновение непредвиденного или аварийного события, которое может порождаться некорректным использованием аппаратуры.

Например, это деление на ноль или обращение по несуществующему адресу памяти. Обычно эти события приводят к завершению программы с системным сообщением об ошибке. C++ дает программисту возможность восстанавливать программу и продолжать ее выполнение.

Исключения C++ **не поддерживают** обработку асинхронных событий, таких, как ошибки оборудования или обработку прерываний, например, нажатие клавиш Ctrl+C. Механизм исключений **предназначен** только для событий, которые происходят в результате работы самой программы и указываются явным образом.

Для управления исключениями в C++ используются три ключевых слова: `try, catch` и `throw`.

Ключевое слово try служит для обозначения блока кода, который может генерировать исключение. При этом соответствующий блок заключается в фигурные скобки и называется защищенным или try-блоком:

```
try {
   // Защищенный блок кода
}
```

Тело всякой функции, вызываемой из try-блока, также принадлежит try-блоку. Предполагается, что одна или несколько функций или инструкций из защищенного блока могут выбрасывать (или генерировать) исключение. Если выброшено исключение, выполнение соответствующей функции или инструкции приостанавливается, все оставшиеся инструкции try-блока игнорируются, а управление передается вовне блока try.

Ключевое слово catch следует непосредственно за try-блоком и обозначает секцию кода, в которую передается управление, если произойдет исключение. За ключевым словом catch следует описание исключения, заключенное в круглые скобки. Описание исключения состоит из имени типа исключения и необязательной переменной:

```
catch (<имя типа>[<перменная>])
{
     //Обработчик исключения
}
```

Имя типа исключения идентифицирует обслуживаемый тип исключений. Блок кода, обрабатывающего исключение, заключается в фигурные скобки и называется catch-блоком или обработчиком исключения. При этом говорят, что данный catch-блок перехватывает исключения описанного в нем типа. Если исключение перехвачено, переменная получает его значение. Если вам не нужен доступ к самому исключению, указывать эту переменную не обязательно. Переменная исключения может иметь любой тип данных, включая созданные пользователем типы классов.

За одним try-блоком могут следовать несколько catch-блоков. Оператор catch, с указанным вместо типа исключения многоточием, перехватывает исключения любого типа и должен быть последним из операторов catch, следующих за try-блоком. Рассмотрим простой пример обработки исключения:

```
int main()
{   double x, y;
    cout << "Введите x: ";
 try {
      cin >> x;
      if (!cin)    throw '1';
      if (x == 0)  throw 1;
      if (1/x < 0) throw 1.01;
      cout << "y =sqrt(1/x) ";
     y=sqrt(1/x);
      cout << "y=" << y << endl;
  }
  catch(char) {cout << "Ошибка ввода" << endl;}
```

```
      catch(int)  {cout << "Делить на 0 нельзя" << endl;}
      catch(double d) {cout << "Извлекать корень из отр.
числа нельзя d=" <<d<< endl;}
     catch(...) {cout << "Ошибка" << endl;}
    cout << "продолжение работы программы";
    return 0;
    }
```

В try-блоке располагается код, который потенциально может вызвать ошибку в работе программы, а именно ошибку в 3-х случаях (вводе символа, деления на 0, извлечения корня из отрицательного числа).

Задаем условие если – if(!cin) – поток испорчен, то будет сгенерировано исключение типа char. В этом случае try-блок сразу прекращает выполнение дальнейших команд, а '1' «падает» в catch. В нашем примере он выводит "Ошибка ввода". При этом программа продолжает работать и выполнять команды, размещенные ниже.

Если же x будет равен нулю, то в try-блоке будет сгенерировано исключение типа int, этом случае try-блок прекращает выполнение дальнейших команд, а 1 «падает» в catch и на экран будет выведено "Делить на 0 нельзя".

Если же x будет меньше нуля, то в try-блоке будет сгенерировано исключение типа double, этом случае try-блок прекращает выполнение дальнейших команд, а 1.0 «падает» в catch и на экран будет выведено "Извлекать корень из отр. числа нельзя d=1.0".

Иначе выполнится команда y=sqrt(1/x), а catch не сработает.

Часто вложенные try/catch блоки возникают неявно, в результате создания защищенного блока в функции, которая сама находится в защищенном блоке.

К вложенным try/catch-блокам приходится прибегать потому, что какая-то функция может непредвиденно привести к исключению. В этом случае простейший выход – заключить весь код в функции main() в такой блок, причем для перехвата исключения использовать обработчик вида catch(...). Затем можно использовать средства, предоставляемые компилятором для определения места в программе, вызвавшего появление исключения.

8.2.8 Перехватывание исключений

Когда с помощью throw генерируется исключение, функции исполнительной библиотеки C++ выполняют следующие действия:

1) создают копию параметра throw в виде статического объекта, который существует до тех пор, пока исключение не будет обработано;

2) в поисках подходящего обработчика раскручивают стек, вызывая деструкторы локальных объектов, выходящих из области действия;

3) передают объект и управление обработчику, имеющему параметр, совместимый по типу с этим объектом.

При раскручивании стека все обработчики на каждом уровне просматриваются последовательно, от внутреннего блока к внешнему, пока не будет найден подходящий обработчик.

Обработчик считается найденным, если тип объекта, указанного после `throw`:

- тот же, что и указанный в параметре `catch` (параметр может быть записан в форме `T`, `const T`, `T&` или `const T&`, где `T`- тип исключения);

- является производным от указанного в параметре `catch` (если наследование производилось с ключом доступа `public`);

- является указателем, который может быть преобразован по стандартным правилам преобразования указателей к типу указателя в параметре `catch`.

Обработчики производных классов следует размещать до обработчиков базовых, поскольку в противном случае им никогда не будет передано управление. Обработчик указателя типа `void` автоматически скрывает указатель любого другого типа, поэтому его также следует размещать после обработчиков указателей конкретного типа.

8.2.9 Возможности механизма исключений

Язык C++ не позволяет возвращать значение из конструктора и деструктора. Механизм исключений дает возможность сообщить об ошибке, возникшей в конструкторе или деструкторе объекта.

Если в конструкторе объекта генерируется исключение, автоматически вызываются деструкторы для полностью созданных в этом блоке к текущему моменту объектов, а также для полей данных текущего объекта, являющихся объектами, и для его базовых классов. Например, если исключение возникло при создании массива объектов, деструкторы будут вызваны только для успешно созданных элементов. Если объект создается в динамической памяти с помощью операции new и в конструкторе возникнет исключение, память из-под объекта корректно освобождается.

8.3 Порядок выполнения лабораторной работы

8.3.1. В ходе самостоятельной подготовки изучить основы работы с потоковым вводом/выводом и исключениями.

8.3.2. Разработать согласно варианту программу на языке C++, состоящую из двух частей: первая демонстрирует умение управлять потоками ввода-вывода, вторая демонстрирует умение генерировать и перехватывать исключения.

8.3.3. Разработать тестовые примеры и выполнить отладку программы.

8.3.4. Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

8.3.5. Оформить отчет по проделанной работе.

8.4 Варианты заданий

Вариант 1

Описать класс геометрическая фигура **прямоугольник**, содержащий следующие поля: ширина, высота, цвет прямоугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади прямоугольника по формуле

$S = a \times b$, где a, b – ширина и высота прямоугольника. Результат вычисления вывести в 17-ти позициях с точностью 5 знаков, пробелы необходимо заменить знаком «&», выравнивание выполнить по левому краю.

Вариант 2

Описать класс геометрическая фигура **ромб**, содержащий следующие поля: сторона, диагональ, угол между сторонами, цвет ромба.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади ромба по формуле

$S = a^2 \times \sin(\alpha)$, где a – известная сторона, α – угол между сторонами. Результат вычисления вывести в 20-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком «@», выравнивание выполнить по правому краю.

Вариант 3

Описать класс геометрическая фигура **круг**, содержащий следующие поля: d – диаметр, цвет круга.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади круга по формуле

$S = \pi \times d^2 : 4$, где d – это диаметр, π – это константа, равная 3.14. Результат вычисления вывести в 17-ти позициях с точностью 6 знаков, пробелы необходимо заменить знаком «^», выравнивание выполнить по левому краю.

Вариант 4

Описать класс геометрическая фигура **треугольник**, содержащий следующие поля: a, b и c – стороны треугольника, а R – радиус описанной окружности, цвет треугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади треугольника по формуле

$S = (a \times b \times c) : 4 \times R$, где a, b и c – стороны треугольника, а R – радиус описанной окружности. Результат вычисления вывести в 12-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком «~», выравнивание выполнить по правому краю.

Вариант 5

Описать класс геометрическая фигура **прямоугольник**, содержащий следующие поля: диагональ, угол между диагоналями, цвет прямоугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади прямоугольника по формуле

$S = 0{,}5 \times d^2 \times sin(a)$, где d – диагональ прямоугольника. Результат вычисления вывести в 12-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком «*», выравнивание выполнить по левому краю.

Вариант 6

Описать класс геометрическая фигура **треугольник**, содержащий следующие поля: a, b и c – стороны треугольника, а r – радиус вписанной окружности, цвет треугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади треугольника по формуле

$S = p \times r$, где p – полупериметр треугольника, r – радиус вписанной окружности. Результат вычисления вывести в 22-х позициях с точностью 5 знаков,

пробелы необходимо заменить знаком «+», выравнивание выполнить по правому краю.

Вариант 7

Описать класс геометрическая фигура **равнобедренная трапеция**, содержащий следующие поля: a, b – параллельные стороны, c – длина одинаковых сторон, цвет трапеции.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления периметра трапеции по формуле

$P = a + b + 2 \times c$, где a, b – параллельные стороны, c – две длины одинаковых сторон Результат вычисления вывести в 12-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком , выравнивание выполнить по левому краю.

Вариант 8

Описать класс геометрическая фигура **круг**, содержащий следующие поля: L – длина окружности, цвет круга.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади круга по формуле

$S = L^2 : (4 \times п)$, где L – это длина окружности, п – это константа, равная 3.14. Результат вычисления вывести в 14-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком «/», выравнивание выполнить по правому краю.

Вариант 9

Описать класс геометрическая фигура **прямоугольник**, содержащий следующие поля: ширина, высота, диагональ, цвет прямоугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления периметра прямоугольника по формуле

P = 2 × (a + b), где a – ширина, b – высота. Результат вычисления вывести в 18-ти позициях с точностью 5 знаков, пробелы необходимо заменить знаком «!», выравнивание выполнить по левому краю.

Вариант 10

Описать класс геометрическая фигура **треугольник**, содержащий следующие поля: длина основания, высота, цвет треугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади треугольника по формуле

S = 0,5 × a × h, где a – длина основания, h – высота, проведенная к основанию. Результат вычисления вывести в 19-ти позициях с точностью 5 знаков, пробелы необходимо заменить знаком «&», выравнивание выполнить по правому краю.

Вариант 11

Описать класс геометрическая фигура **ромб**, содержащий следующие поля: диагональ1, диагональ2, цвет ромба.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади ромба по формуле

S = 0,5 × (d1 × d2), где d1, d2 – две диагонали. Результат вычисления вывести в 19-ти позициях с точностью 4 знака, пробелы необходимо заменить знаком «?», выравнивание выполнить по левому краю.

Вариант 12

Описать класс геометрическая фигура **круг**, содержащий следующие поля: d – диаметр, цвет круга.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления длины окружности по формуле

L = 2 × r × π, где r – радиус, π – это константа, равная 3.14. Результат вычисления вывести в 14-ти позициях с точностью 8 знаков, пробелы необходимо заменить знаком «[», выравнивание выполнить по правому краю.

Вариант 13

Описать класс геометрическая фигура **треугольник**, содержащий следующие поля: a и b – две стороны, α – угол между ними, цвет треугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади треугольника по формуле

S = 0,5 × a × b × sin(α), где a и b – две стороны, α – угол между ними. Результат вычисления вывести в 12-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком «%», выравнивание выполнить по левому краю.

Вариант 14

Описать класс геометрическая фигура **параллелограмм**, содержащий следующие поля: a и b – две стороны, α – угол между ними, цвет параллелограмма.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади параллелограмма по формуле

S = a × b × sin(α), где a и b – две стороны, α – угол между ними. Результат вычисления вывести в 17-ти позициях с точностью 5 знаков, пробелы необходимо заменить знаком «`», выравнивание выполнить по правому краю.

Вариант 15

Описать класс геометрическая фигура **трапеция**, содержащий следующие поля: a, b – два разных основания, h – высота трапеции, цвет трапеции.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади трапеции по формуле

S = (a + b) : 2 × h, где a, b – два разных основания, h – высота трапеции. Результат вычисления вывести в 17-ти позициях с точностью 4 знака, пробелы необходимо заменить знаком «.», выравнивание выполнить по левому краю.

Вариант 16

Описать класс геометрическая фигура **круг**, содержащий следующие поля: r – радиус, цвет круга.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади круга по формуле

$S = \pi \times r^2$, где r – это радиус, π – это константа, равная 3.14. Результат вычисления вывести в 17-ти позициях с точностью 6 знаков, пробелы необходимо заменить знаком «-», выравнивание выполнить по правому краю.

Вариант 17

Описать класс геометрическая фигура **параллелограмм**, содержащий следующие поля: $d1$, $d2$ – диагонали, β – угол между диагоналями, цвет параллелограмма.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади параллелограмма по формуле

$S = 0,5 \times (d1 \times d2) \times \sin(\beta)$, где $d1$, $d2$ – диагонали, β – угол между диагоналями. Результат вычисления вывести в 18-ти позициях с точностью 5 знаков, пробелы необходимо заменить знаком «”», выравнивание выполнить по левому краю.

Вариант 18

Описать класс геометрическая фигура **треугольник**, содержащий следующие поля: a, b и c – стороны треугольника, цвет треугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления периметра треугольника по формуле

$P = a + b + c$, где a, b, c – длина стороны. Результат вычисления вывести в 15-ти позициях с точностью 4 знака, пробелы необходимо заменить знаком «_», выравнивание выполнить по правому краю.

Вариант 19

Описать класс геометрическая фигура **параллелограмм**, содержащий следующие поля: a – сторона, h – высота, цвет параллелограмма.

Написать следующие методы:

-- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади параллелограмма по формуле

`S = a × h`, где `a` – сторона, `h` – высота. Результат вычисления вывести в 18-ти позициях с точностью 5 знаков, пробелы необходимо заменить знаком «”», выравнивание выполнить по левому краю.

Вариант 20

Описать класс геометрическая фигура **эллипс**, содержащий следующие поля: `a` – длина большей полуоси эллипса, `b` – длина меньшей полуоси эллипса, цвет эллипса.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления периметра эллипса по формуле

`P = 4 × (π×a×b + (a-b))/(a+b)`, где `a` - длина большей полуоси эллипса, `b` - длина меньшей полуоси эллипса, `π` – это константа, равная 3.14. Результат вычисления вывести в 17-ти позициях с точностью 5 знаков, пробелы необходимо заменить знаком «:», выравнивание выполнить по правому краю.

Вариант 21

Описать класс геометрическая фигура **параллелограмм**, содержащий следующие поля: ширина, высота, цвет параллелограмма.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления периметра параллелограмма по формуле

`P = 2 × (a + b)`, где `a` – ширина, `b` – высота. Результат вычисления вывести в 12-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком «@», выравнивание выполнить по левому краю.

Вариант 22

Описать класс геометрическая фигура **трапеция**, содержащий следующие поля: `d1, d2` – диагонали, β – угол между диагоналями, цвет трапеции.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади трапеции по формуле

`S = 1/2 × d1 × d2× sin(β)`, где `d1, d2` – диагонали, β – угол между диагоналями. Результат вычисления вывести в 12-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком «#», выравнивание выполнить по правому краю.

Вариант 23

Описать класс геометрическая фигура **эллипс**, содержащий следующие поля: `a` – длина большей полуоси эллипса, `b` – длина меньшей полуоси эллипса, цвет эллипса.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади эллипса по формуле

`S = π × a × b`, где `a` - длина большей полуоси эллипса, `b` - длина меньшей полуоси эллипса, `π` – это константа, равная 3.14. Результат вычисления вывести в 14-ти позициях с точностью 8 знаков, пробелы необходимо заменить знаком «`,`», выравнивание выполнить по левому краю.

Вариант 24

Описать класс геометрическая фигура прямоугольный **треугольник**, содержащий следующие поля: `a, b` – катеты, цвет треугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади треугольника по формуле

`S = 0,5 × a × b`, где `a, b` – катеты. Результат вычисления вывести в 19-ти позициях с точностью 5 знаков, пробелы необходимо заменить знаком «}», выравнивание выполнить по правому краю.

Вариант 25

Описать класс геометрическая фигура **прямоугольник**, содержащий следующие поля: ширина, высота, диагональ, цвет прямоугольника.

Написать следующие методы:

- метод заполнения приватных полей класса считанными с клавиатуры данными; состояние потока после ввода каждого поля нужно проверять и генерировать исключения в случае ошибки в потоке и в случае ввода «некорректных» данных (не положительное число, несоответствие радиуса вписанной окружности длинам сторон треугольника и т.д.);

- метод вычисления площади прямоугольника по формуле

$S = a * \sqrt{(d^2 - a^2)}$, где a – известная сторона, d – диагональ прямоугольника. Результат вычисления вывести в 12-ти позициях с точностью 3 знака, пробелы необходимо заменить знаком «@», выравнивание выполнить по левому краю.

8.5 Содержание отчета о выполнении лабораторной работы

Титульный лист, цель работы, вариант задания, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

8.6 Контрольные вопросы

8.6.1 Объясните понятие «поток ввода/вывода».

8.6.2 Поясните механизм буферизации потоков ввода/вывода.

8.6.3 Приведите примеры работы потоков ввода/вывода с различными типами данных.

8.6.4 Перечислите виды средств форматирования потоков.

8.6.5 Расскажите о функциях и флагах форматирования потоков ввода/вывода.

8.6.6 Расскажите о манипуляторах.

8.6.7 Опишите файловые потоки и режимы работы с файлами.

8.6.8 Дайте определение термину «исключение», опишите ситуации возникновения исключений.

8.6.9 Опишите работу механизма обработки исключений.

8.6.10 Расскажите о вложенных конструкциях `try/catch`, о правилах оформления `catch`-блоков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Зайцев, М. Г. Объектно-ориентированный анализ и программирование : учебное пособие / М. Г. Зайцев. — Новосибирск : НГТУ, 2017. — 84 с. — URL: https://e.lanbook.com/book/118271 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — ISBN 978-5-7782-3308-9. — Текст : электронный.

2. Мурлин, А. Г. Объектно-ориентированное программирование : учебное пособие / А. Г. Мурлин, В. А. Мурлина, М. В. Янаева. — Краснодар : КубГТУ, 2021. — 151 с. — URL: https://e.lanbook.com/book/231569 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — ISBN 978-5-8333-1059-5. — Текст : электронный.

3. Скворцова, Л. А. Объектно-ориентированное программирование на языке C++: Практикум : учебное пособие / Л. А. Скворцова, А. А. Бирюкова, К. В. Гусев. — Москва : РТУ МИРЭА, 2021. — 146 с. — URL: https://e.lanbook.com/book/176540 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — Текст : электронный // Лань : электронно-библиотечная система.

4. Скворцова, Л. А. Объектно-ориентированное программирование на языке C++ : учебное пособие / Л. А. Скворцова. — Москва : РТУ МИРЭА, 2020. — 246 с. — URL: https://e.lanbook.com/book/163862 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — Текст : электронный.

5. Теплоухов, С. В. Основы объектно-ориентированного программирования на языке C++ : учебное пособие / С. В. Теплоухов. — Майкоп : АГУ, 2021. — 92 с. — URL: https://e.lanbook.com/book/231416 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — Текст : электронный.

6. Юрина, Т. А. Объектно-ориентированное программирование : учебно-методическое пособие / Т. А. Юрина. — Омск : СибАДИ, 2023. — 72 с. — URL: https://e.lanbook.com/book/338576 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — Текст : электронный.

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания
к лабораторным работам

Часть 2

Составитель: **Сметанина** Татьяна Ивановна

В авторской редакции

Изд. № 66/2024. Объем 3,25 п.л. Усл. печ. л. 3,02. Уч.-изд. л. 3,185.
Издательский центр ФГАОУ ВО «Севастопольский государственный университет»