# Введение

Методическое пособие содержит фрагменты теоретического материала по прикладному программированию на языке С++, задания для разработки программ, объяснения алгоритмов и прокомментированные исходные тексты программ по различным темам.

Цель пособия – помочь студентам усовершенствовать свои знания языка С++ и освоить рациональные приемы составления программ на этом объектно-ориентированном языке.

Объектно-ориентированная технология (парадигма) программирования наиболее распространена и востребована в настоящее время. При объектно-ориентированном подходе к программированию программа представляет собой совокупность взаимодействующих между собой данных – объектов. Функциональную возможность и структуру объектов задают классы – типы данных, определенные пользователем.

Изучение ООП целесообразно начать на примере объектно-ориентированного языка программирования C++ ([5, 8-9]), как наиболее теоретически выдержанного в этой части. Другие языки, поддерживающие идеи ООП, такие, как Object Pascal [4], Java [7], разрабатывались в первую очередь с учетом удобства программирования задач в соответствующих предметных областях.

Язык C++, унаследовав язык C (стандарт С90 [1,2]), был разработан его автором Б. Страуструпом [7] с наиболее возможной реализацией теоретических концепций ООП. Язык C++ вобрал в себя не только некоторые концепции языка С, а также и некоторых других языков. Например, концепция классов взята автором языка С++ из языка Simula [6], а концепция наследования – из языка Smalltalk.

В данном учебном пособии C++ рассматривается в строго теоретическом аспекте.

**Объектно-ориентированная парадигма**, при которой программа рассматривается как совокупность фрагментов кода, обрабатывающих отдельные совокупности данных – **объекты**. Эти объекты взаимодействуют друг с другом посредством так называемых **интерфейсов**. При этом данные управляют доступом к коду.

Выбор тем и изложение материала таковы, что студент освоит:

* синтаксис, семантику и выразительные возможности языка С++;
* три основных принципа объектно-ориентированного программирования (ООП);
* особенности реализации на языке С++ классических алгоритмов обработки

# 1. Основные принципы объектно-ориентированного программирования

Центральной идеей ООП является реализация понятия «**абстракция**»[7]. Смысл абстракции заключается в том, что *сущность* произвольной сложности можно рассматривать, а также производить определенные действия над ней, как над **единым целым**, не вдаваясь в детали внутреннего построения и функционирования.

При создании программного комплекса необходимо разработать определенные абстракции.

Одним из основных способов создания абстракции является использование концепции **иерархической классификации**. Ее суть заключается в том, что сложные системы разбиваются на более простые фрагменты.

Практически все сложные системы иерархичны, и уровни их иерархии отражают различные уровни абстракции. Для каждой конкретной задачи рассматривается соответствующий уровень. Выбор низшего уровня абстракции достаточно произволен. Выбранный уровень в одном случае в качестве низшего уровня может оказаться уровнем достаточно высокой абстракции в другом проекте.

Различают **типовую** иерархию и **структурную** иерархию, которые далее будут называться соответственно **структурой классов** и **структурой объектов**.

Во всех объектно-ориентированных языках программирования реализованы следующие **основные механизмы** (**постулаты) ООП**[5-6]:

* инкапсуляция,
* наследование,
* полиморфизм.

Все эти механизмы важны для разработки и использования абстракций.

1)**Инкапсуляция** – механизм, связывающий вместе код и данные, которыми он манипулирует, и одновременно защищающий их от произвольного доступа со стороны другого кода, внешнего по отношению к рассматриваемому. Доступ к коду и данным жестко контролируется интерфейсом.

Основой инкапсуляции при ООП является **класс**.

Механизм инкапсуляции позволяет оставлять скрытыми от пользователя некоторые детали реализации класса (то есть инкапсулировать их в классе), что упрощает работу с объектами этого класса.

2)**Наследование** – механизм, с помощью которого один объект (производного класса) приобретает свойства другого объекта (родительского, базового класса). При использовании наследования новый объект не обязательно описывать, начиная с нуля, что существенно упрощает работу программиста. Наследование позволяет какому-либо объекту наследовать от своего родителя общие атрибуты, а для себя определять только те характеристики, которые делают его уникальным внутри класса.

Наследование есть очень важное понятие, поддерживающее концепцию иерархической классификации.

3)**Полиморфизм** – механизм, позволяющий использовать один и тот же интерфейс для общего класса действий.

# 2. Классы и объекты в языке С++

Центральным понятием ООП на С++ является **класс**. Класс используется для опи­сания типа, на основе которого создаются **объекты этого типа**(переменные типа **класс**).

Класс, как и любой тип данных, характеризуется множеством значений, которые могут принимать объекты класса, и **множеством функций**, задающих операции над объектами.

Класс полноценно определяет тип данных как совокупность множества значений и набора операций над этими значениями.

## 2.1. Синтаксис описания класса

**class** *имя\_класса { определение\_членов\_класса* };

Члены класса можно разделить на информационные члены и функции-члены (методы) класса. Информационные члены описывают внутреннюю структуру информации, хранящейся в объекте, который создается на основе класса. Методы класса описывают алгоритмы обработки этой информации.

Данные, хранящиеся в информационных членах, описывают **состояние** объекта, созданного на основе класса. Состояние объекта изменяется на основе изменения хранящихся данных с помощью методов класса. Алгоритмы, заложенные в реализации методов класса, определяют **поведение** объекта, то есть реагирование объекта на поступающие внешние воздействия в виде входных данных [6, 8-9].

## 2.2. Управление доступом к членам класса

Принцип **инкапсуляции** обеспечивается вводом в класс **областей доступа**:

* ***private*** (закрытый, доступный только собственным методам),
* ***public*** (открытый, доступный любым функциям),
* ***protected*** (защищенный, доступный только собственным методам и методам производных классов).

Члены класса, находящиеся в закрытой области (***private***), недоступны для использования со стороны внешнего кода. Напротив, члены класса, находящиеся в открытой секции (***public***), доступны для использования со стороны внешнего кода. При описании класса каждый член класса помещается в одну из перечисленных выше областей доступа следующим образом:

**class** *имя\_класса* {

**private:**

*определение\_закрытых\_членов\_класса*

**public:**

*определение\_открытых\_членов\_класса*

**protected:**

*определение\_защищенных\_членов\_класса*};

Порядок следования областей доступа и их количество в классе – произвольны.

Служебное слово, определяющее первую область доступа, может отсутствовать. **По умолчанию** эта область считается ***private***.

В закрытую (***private***) область обычно помещаются информационные члены, а в открытую (***public***) область – методы класса, реализующие интерфейс объектов класса с внешней средой. Если какой-либо метод имеет вспомогательное значение для других методов класса, являясь подпрограммой для них, то его также следует поместить в закрытую область. Это обеспечивает логическую целостность информации [5, 8-9].

После описания класса его имя можно использовать для описания объектов этого типа.

Доступ к информационным членам и методам объекта, описанным в открытой секции, осуществляется через объект или ссылку на объект с помощью **операции выбора члена класса**‘.’. Если работа с объектом выполняется с помощью указателя на объект, то доступ к соответствующим членам класса осуществляется на основе **указателя на член класса**‘->’. Объекты класса можно определять совместно с описанием класса.

## 2.3. Класс как область видимости

Класс является областью видимости описанных в нем членов класса. Идентификатор члена класса локален по отношению к данному классу. Одноименные идентификаторы членов класса закрывают **видимость** соответствующих внешних идентификаторов.

Операция ‘*::*’позволяет получить **доступ** к одноименным объектам, внешним по отношению к текущей области видимости, в частности, к глобальным функциям и переменным, следующим образом [5, 8-9]:

*имя\_класса :: имя\_члена\_класса* или

*:: имя* – для имен глобальных функций и переменных.

## 2.4. Объявление и определение методов класса. Спецификатор inline

Каждый метод класса, должен быть **определен** в программе. Определить метод класса можно либо непосредственно в классе (если тело метода не слишком сложно и громоздко), либо вынести определение вне класса, а в классе только **объявить** соответствующий метод, указав его прототип.

При определении метода класса вне класса для указания области видимости соответствующего имени метода используется операции ‘*::*’.

Это позволяет повысить наглядность текста, особенно, в случае значительного объема кода в методах. При определении метода вне класса с использованием операции ‘*::*’прототипы объявления и определения функции должны совпадать.

Метод класса и любую функцию, не связанную ни с каким классом, можно определить со спецификатором ***inline***:

**inline int** func1();

Такие функции называются *встроенными*.

Спецификатор ***inline*** указывает компилятору, что необходимо по возможности генерировать в точке вызова код функции, а не команды вызова функции, находящейся в отдельном месте выполняемого кода программы. Это позволяет уменьшить время выполнения программы за счет отсутствия команд вызова функции и возврата из функции, которые кроме передачи управления выполняют действия соответственно по сохранению и восстановлению *контекста* (содержимого основных регистров процессора). По определению методы класса, определенные непосредственно в классе, являются ***inline***-функциям.

## 2.5. Указатель this

В классах С++ неявно введен специальный указатель ***this*** – указатель на текущий объект. Каждый метод класса при обращении к нему получает данный указатель в качестве **неявного параметра**[5, 8-9]. Через него методы класса могут получить доступ к другим членам класса.

Указатель ***this*** можно рассматривать как локальную константу, имеющую тип *X\**, если *X* – имя описываемого класса. Нет необходимости использовать его явно. Он используется явно, например, в том случае, когда выходным значением для метода является текущий объект.

Данный указатель, как и другие указатели, может быть разыменован.

При передаче возвращаемого значения метода класса в виде ссылки на текущий объект используется разыменованный указатель ***this***, так как ссылка, как уже было указано, инициализируется непосредственным значением.

# 3. Конструкторы и деструкторы

**Конструкторы** и **деструкторы** являются специальными методами класса. **Конструкторы** вызываются при создании объектов класса. **Деструкторы** вызываются при уничтожении объектов.

В большинстве случаев конструкторы и деструкторы вызываются автоматически (неявно) соответственно при определении объекта (в момент отведения памяти под него) и при уничтожении объекта [5, 8-9]. Конструктор (как и деструктор) может вызываться и явно, например, при создании объекта в динамической области памяти с помощью операции ***new,*** а деструктор – с помощью операции ***delete.***

Так как конструкторы и деструкторы неявно входят в интерфейс объекта, их обычно располагают в открытой области декларации класса. Приватные конструкторы применяют для создания объектов класса в его статических методах, которые в этом случае можно назвать именованными конструкторами (см. лабораторную работу № 1).

Отличия и особенности описания конструктораот обычной функции, метода класса:

1. имя конструктора совпадает с именем класса;
2. при описании конструктора не указывается тип возвращаемого значения.

Следует отметить, что и обычная процедура может не возвращать значения, а только перерабатывать имеющиеся данные. В этом случае при описании соответствующей функции указывается специальный тип возвращаемого значения ***void***.

В описании конструктора тип возвращаемого значения не указывается не потому, что возвращаемого значения нет. Оно как раз есть. Ведь результатом работы конструктора в соответствии с его названием является **созданный объект** того типа, который описывается данным классом. Б. Страуструп отмечал, что конструктор – это то, что область памяти превращает в объект.

Конструкторы можно классифицировать разными способами:

1) по наличию параметров:

* без параметров,
* с параметрами;

2) по количеству и типу параметров:

* конструктор **умолчания**,
* конструктор **преобразования**,
* конструктор **копирования**,
* конструктор **с двумя и более параметрами.**

Набор и типы параметров зависят от того, на основе каких данных создается объект.

**Деструкторы** применяются для корректного уничтожения объектов. Часто процесс уничтожения объектов включает в себя действия по освобождению выделенной для них по операциям ***new*** памяти.

Обозначение деструктора: **~***имя\_класса()*

У деструкторов нет параметров и возвращаемого значения. В отличие от конструкторов деструктор в классе может быть только один.

## 3.1. Конструктор умолчания

Конструктор без параметров называется **конструктором умолчания (конструктор по умолчанию)**.

Если для создания объекта не требуется каких-либо параметров, то используется конструктор умолчания. При определении таких объектов после имени класса указывается только идентификатор переменной:

**class** Х{ **…** };

Х x1;

## 3.2. Конструктор преобразования и конструкторы с двумя и более параметрами

Если для создания объекта необходимы параметры, то они указываются в **круглых скобках** после идентификатора переменной:

box b2(1,2,3); box b3(5);

Указываемые параметры являются параметрами конструктора класса. Если у конструктора имеется **ровно один** входной параметр, который **не** представляет собой ссылку на свой собственный класс, то соответствующий конструктор называется **конструктором преобразования**. Этот конструктор называется так в связи с тем, что в результате его работы на основе объекта одного типа создается объект другого типа (типа описываемого класса).

Если уже описан класс *T* и описывается новый класс *X*, то его конструкторы преобразования могут иметь любой из следующих прототипов:

X(T);

X(T&);

X(**const** T&);

Последний прототип служит для защиты от изменения передаваемого фактического параметра в теле конструктора, так как при получении ссылки на фактический параметр используется собственно передаваемый объект, а не его локальная копия.

## 3.3. Конструктор копирования-инициализации

При создании объекта его информационные члены могут быть проинициализированы значениями полей другого объекта этого же типа, то есть объект создается как копия другого объекта.

Для такого создания объекта используется **конструктор копирования**.

Инициализация может быть выполнена аналогично инициализации переменных встроенных типов с использованием операции присваивания совместно с объявлением объекта:

box b5(2,4,6); // создание объекта типа box с использованием числовых данных

box b6 = b5; // создание объекта b6 – копии объекта b5

Если инициализация производится объектом такого же типа, то объект-инициализатор также может быть указан в круглых скобках после идентификатора создаваемого объекта:

box b7(b5);

Если класс не предусматривает создания внутренних динамических структур, например, массивов, создаваемых с использованием операции **new,** то в конструкторе копирования достаточно предусмотреть **поверхностное копирование**, то есть почленное копирование информационных членов класса.

Конструктор копирования, осуществляющий поверхностное копирование, можно явно не описывать, он сгенерируется **автоматически**.

Если же в классе предусмотрено создание внутренних динамических структур, использование только поверхностного копирования будет ошибочным, так как информационные члены-указатели, находящиеся в разных объектах, будут иметь одинаковые значения и указывать на одну и ту же размещенную в динамической памяти структуру. Автоматически сгенерированный конструктор копирования в данном классе не позволит корректно создавать объекты такого типа на основе других объектов этого же типа.

В подобных случаях необходимо **глубокое копирование**, осуществляющее не только копирование информационных членов объекта, но и самих динамических структур. При этом, естественно, информационные члены-указатели в создаваемом объекте должны не механически копироваться из объекта-инициализатора, а указывать на вновь созданные динамические структуры нового объекта.

Работа конструктора копирования заключается в следующих действиях.

1) Входной параметр является внешним объектом по отношению к создаваемому объекту. Тем не менее, имеется возможность прямого обращения к закрытым членам этого внешнего объекта. Это возможно только потому, что входной параметр имеет тип, совпадающий с типом создаваемого в результате работы конструктора копирования объекта. Если бы на вход конструктора поступал бы объект другого типа, то для доступа к закрытым членам объекта-параметра необходимо было бы применять специальные средства. Это связано с тем, что **единицей защиты является не объект, а тип**, то есть методы объекта могут обращаться к закрытым членам не только данного объекта, но и к закрытым членам любого объекта данного типа.

2) В момент описания конструктора копирования класс, как тип данных, еще не описан до конца. Тем не менее, идентификатор класса уже используется в качестве полноценного типа данных при описании входного параметра конструктора копирования. Такая технология схожа с описанием рекурсивной функции, когда тело описываемой функции содержит вызов этой же функции.

В отличие от конструктора преобразования, входной параметр конструктора копирования имеет тип, описываемый данным классом. Таким образом, если описывается класс *Х*, то его конструктор копирования может иметь один из следующих прототипов:

Х(Х&);

Х(**const** Х&);

Объект, создаваемый с использованием конструктора копирования, может инициализироваться не только именованными объектами, но и временно созданными объектами.

## 3.4. Работа с динамической памятью

В языке C++ для работы с динамической памятью введены операции ***new*** и ***delete***[5, 8-9], которыми можно пользоваться наряду с функцией стандартной библиотеки С ***malloc, calloc*** и ***free***.

Операция ***new*** используется как для выделения памяти для одного объекта (при этом возможна инициализация выделенной памяти передаваемым значением), так и для массива однородных объектов. Операция ***new*** возвращает адрес начала выделенной динамической памяти соответствующего типа.

Ее синтаксис: **new** *тип;*

**new** *тип* (*выражение-инициализатор*);

**new** *тип* [*выражение\_размерность\_массива*];

Операция ***delete*** освобождает распределенную операцией ***new*** память. Ее синтаксис:

**delete** *указатель\_на\_объект*;

**delete [ ]** *указатель\_на\_массив\_объектов*;

Первая форма используется, если операцией ***new*** размещался единичный (скалярный) объект. Векторная форма используется**,** если операцией ***new*** создан массив объектов, при удалении которого для каждого из объектов необходим вызов деструктора (деструкторы описываются далее). Такими объектами являются объекты пользовательского типа.

# 4. Практическое изучение первого принципа ООП – механизма инкапсуляции

## 4.1. Подготовка к лабораторной работе № 1

**Программирование точечных конфигураций на плоскости**

В первой лабораторной работе используются абстрактные типы данных (АТД). АТД являются математическим понятием, пригодным на этапе подготовки спецификации – в процессе анализа. Понятие класса, предусматривая частичную или полную реализацию, обеспечивает необходимую связь с разработкой ПО на этапах проектирования и программирования. Класс называется эффективным, если его реализация полна, и отложенным - при частичной реализации. Рассмотрим два типовых задания.

**Вычисление декартовых координат вершин правильного треугольника**

Разработать ООП для вычисления декартовых координат вершин правильного треугольника с заданной длиной стороны, центр которого лежит на оси Х (рисунок 1). Длина стороны треугольника должна передаваться программе через аргумент командной строки, а полученные координаты должны отображаться через стандартный вывод. Программная реализация вычислений должна быть основана на разработке класса точки с приватными полями ее декартовых координат и публичными методами доступа к ним и конструктором инициализации их значений. Кроме того нужно предусмотреть статический метод конструирования точки по ее полярным координатам (программа polar.cc).

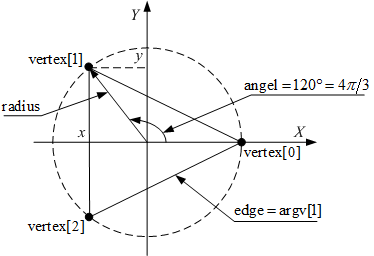


Рисунок 1 – К постановке задачи вычисления декартовых координат правильного треугольника

**//** Программа polar.cc

#include <stdlib.h>

#include <math.h>

#include <iostream.h>

// Класс Point2 точки плоскости

class Point2 {

private:

float x;

float y;

Point2(float \_x, float \_y) : x(\_x), y(\_y) { };// приватный конструктор

public:

float getx() { return x; }; // доступ к x

float gety() { return y; }; // доступ к y

static Point2 Polar(float, float); **//** именованный конструктор точки - статический метод

Point2() { x=y=0.0; }; // конструктор по умолчанию (для массива точек)

}; // Point2

// конструирование точки по полярным координатам

inline Point2 Point2 :: Polar(float R, float F) {

Point2 p(R\*cos(F), R\*sin(F)); // перевод полярных координат в декартовы

return p; // возврат объекта класса точки

}; //polar

// основная функция

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

float edge = 1.0; // сторона треугольника (1.0 – длина по умолчанию)

float angle = 0.0; // полярный угол вершины

float pi; // число pi

pi = acos (-1.0 ); // вычислить pi (можно M\_PI из math.h)

float radius; // радиус-вектор вершины

int i = 0; // счетчик вершин треугольника

Point2 vertex [3]; // массив вершин треугольника vertex[0], vertex[1], vertex[2]

if(argc > 1)

edge = atof (argv[1]);// получить длину стороны из командной строки

cout << "pi=" << pi << endl;

radius = edge / (2\*sin(pi/3.0)); // радиус описанной окружности

while (i<3) { // цикл вычисления декартовых координат вершин

vertex[ i ] = Point2::Polar(radius, angle);

angle += ((4.0\*pi)/3.0); // +120 градусов =4\* pi/3

i++;

}// while

do { // Печать декартовых координат вершин в обратном порядке

--i;

cout << vertex[ i ].getx() << ';' << vertex[ i ].gety() << endl;

} while(i > 0); //do- while

return (0); // Корректное завершение программы

}// main

**Вычисление средних расстояний между точками на плоскости**

Разработать ООП для вычисления среднего расстояния от каждой точки заданного набора до остальных точек этого набора. Декартовы координаты точек набора должны передаваться программе парами целых чисел через аргументы командной строки. Полученные значения средних расстояний должны отображаться строками стандартного вывода. Программная реализация вычислений должна быть основана на разработке класса точки с приватными полями ее декартовых координат и конструктором инициализации их значений, публичными методами доступа к ним, а также дружественная функция вычисления расстояния между 2-мя точками. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массива их адресов (программа average.cc).

//Программа average.cc

#include <stdio.h>

#include <math.h>

// класс точки плоскости

class Point2 {

private:

int x;

int y;

public:

Point2(int \_x, int \_y) : x(\_x), y(\_y) { }; // конструктор точки

int getx() { return x; }; // доступ к x

int gety() { return y; }; // доступ к y

friend double distance(Point2\*, Point2\*); // расстояние между 2-мя точками

}//Point2

// дружественная функция вычисления расстояния между двумя точками

double distance(Point2\* r, Point2\* q) {

double d2=(r->x-q->x)\*(r->x-q->x)+(r->y-q->y)\*(r->y-q->y);

return (sqrt(d2));

} // distance

//Основная функция

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

double sum; // суммарное расстояние от первой до остальных точек

Point2\*\* p; // адрес массива адресов точек набора

Point2\* t; // адрес текущей точки

int x,y; // координаты точки

int i=0;

int j=0; // номера точек набора

if(argc < 2)

return(-1); //анализ числа аргументов командной строки

p = new Point2\*[argc]; // распределить память под массив адресов точек

while(++i < argc) { //цикл ввода точек из командной строки

sscanf(argv[ i ], "(%d%\*c%d)", &x, &y)); // получение координат точки

p[ j ]=new Point2(x,y); // динамическое конструирование точки

j ++;

} // while

p[ j ]=NULL; // маркировка конца массива адресов точек набора

for (i=0; p[ i ] != NULL; i++) { // цикл вычисления средних расстояний

sum = 0.0; // инициализировать сумму расстояний до текущей точки

t = p[ i ]; // фиксировать адрес текущей точки

// цикл суммирования расстояний от текущей точки до остальных

for (j=0; p[ j ] != NULL; j++)

sum + = distance(t, p[ j ]); //вычисление среднего расстояния до текущей точки

printf("Dcp(%d;%d)=%f\n", t->getx(), t->gety(), sum/(argc-2)); // ->

} // for-i

// освобождить дин. память точек набора по их адресам

for(j=0; p[ j ] != NULL; j++)

delete p[ j ];

delete [ ]p; // освобождить дин. память массива адресов точек

return 0; //корректное завершение программы

}// main

**Варианты заданий для первой лабораторной работы**

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления декартовых координат вершин правильного шестиугольника с заданной длиной стороны, центр которого совпадает с началом координат, а две стороны параллельны оси X. Длина стороны шестиугольника должна передаваться программе аргументом командной строки ее вызова, а полученные координаты его вершин должны отображаться в потоке стандартного вывода. Программная реализация вычислений должна быть основана на разработке класса точки с приватными полями ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Кроме того, в классе должен быть предусмотрен статический метод для конструирования точки по полярным координатам. P01

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления декартовых координат вершин правильного шестиугольника с заданной длиной стороны, центр которого совпадает с началом координат, а две стороны параллельны оси Y. Длина стороны шестиугольника должна передаваться программе аргументом командной строки ее вызова, а полученные координаты его вершин должны отображаться в потоке стандартного вывода. Программная реализация вычислений должна быть основана на разработке класса точки с приватными полями ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Кроме того, в классе должен быть предусмотрен статический метод для конструирования точки по полярным координатам. P02

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления декартовых координат вершин квадрата с заданной длиной стороны и центром в начале координат, диагонали которого отклонены относительно координатных осей на заданный угол. Длина стороны квадрата и угол поворота его диагоналей должны передаваться программе аргументами командной строки ее вызова, а полученные координаты его вершин должны отображаться в потоке стандартного вывода. Программная реализация вычислений должна быть основана на разработке класса точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Необходимо также предусмотреть статический метод для конструирования точки по полярным координатам. P03

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления декартовых координат вершин правильного восьмиугольника с заданной длиной стороны и центром в начале координат, у которого четыре вершины лежат на осях координат. Длина стороны этого восьмиугольника должна передаваться программе аргументом командной строки ее вызова, а полученные координаты его вершин должны отображаться в потоке стандартного вывода. Программная реализация вычислений должна быть основана на разработке класса точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Кроме того, в этом классе нужно предусмотреть статический метод для конструирования точки по полярным координатам. P04

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для поиска точки из любого заданного набора точек на плоскости, суммарное расстояние которой до остальных точек этого набора будет минимально. Декартовые координаты всех точек должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной точки должны отображаться строкой стандартного вывода. В программе должен быть реализован класс точки с приватными полями ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Вычисление расстояния между парами любых точек должна обеспечивать внешняя функция. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех заданных точек и массива их адресов. P05

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для поиска точки из любого заданного набора точек на плоскости, суммарное расстояние которой до остальных точек этого набора будет максимально. Декартовые координаты всех точек должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной точки должны отображаться строкой стандартного вывода. В программе должен быть реализован класс точки с приватными полями ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Вычисление расстояния между парами любых точек должна обеспечивать внешняя функция. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех заданных точек и массива их адресов. P06

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для поиска пары наиболее близких точек из любого заданного набора точек на плоскости. Декартовые координаты всех его точек должны быть специфицированы парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары ближайших точек набора и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений, а также метод для вычисления расстояния до любой другой заданной точки. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массива их адресов. P07

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска пары наиболее отдаленных точек из любого заданного набора точек на плоскости. Декартовые координаты всех его точек должны быть специфицированы парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары наиболее отдаленных точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений, а также метод для вычисления расстояния до любой другой заданной точки. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массива их адресов. P08

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска точки любого заданного набора точек на плоскости, которая наиболее близка к началу координат. Декартовые координаты всех точек набора должны быть заданы парами целых чисел в аргументах командной строки вызова программы. Результатом выполнения программы должен быть стандартный вывод координат искомой точки и ее расстояния от начала координат. В программе необходимо реализовать класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Кроме того, в этом классе нужно предусмотреть компонентный метод вычисления расстояния точки от начала координат и обеспечить динамическое распределение памяти для всех заданных точек и массива их адресов. P09

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска точки любого заданного набора точек на плоскости, которая наиболее удалена от начала координат. Декартовые координаты всех точек набора должны быть заданы парами целых чисел в аргументах командной строки вызова программы. Результатом выполнения программы должен быть стандартный вывод координат искомой точки и ее расстояния от начала координат. В программе необходимо реализовать класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Кроме того, в этом классе нужно предусмотреть компонентный метод вычисления расстояния точки от начала координат и обеспечить динамическое распределение памяти для всех заданных точек и массива их адресов. P10

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу, которая для каждой точки из заданного набора точек на плоскости определяет ближайшую к ней точку этого набора. Декартовые координаты его точек должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученных пар точек и расстояния между ними должны отображаться строками стандартного вывода. В программе должен быть реализован класс точки с приватными полями ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Вычисление расстояния между парами точек должна обеспечивать дружественная функция этого класса. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех заданных точек и массива их адресов. P11

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу, которая для каждой точки из заданного набора точек на плоскости ищет наиболее удаленную от нее точку этого набора. Декартовые координаты его точек должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученных пар точек и расстояния между ними должны отображаться строками стандартного вывода. В программе должен быть реализован класс точки с приватными полями ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Вычисление расстояния между парами точек должна обеспечивать дружественная функция этого класса. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех заданных точек и массива их адресов. P12

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска в любом заданном наборе точек плоскости пары наиболее близких точек, которые находятся по разные стороны от оси Y. Декартовые координаты всех точек заданного набора должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений, а также компонентный метод вычисления расстояния до любой другой заданной точки. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массивов их адресов. P13

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска в любом заданном наборе точек плоскости пары наиболее отдаленных точек, которые находятся по разные стороны от оси Y. Декартовые координаты всех точек заданного набора должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений, а также компонентный метод вычисления расстояния до любой другой заданной точки. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массивов их адресов. P14

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска в любом заданном наборе точек плоскости пары наиболее близких точек, которые находятся по разные стороны от оси X. Декартовые координаты всех точек заданного набора должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Вычисление расстояния между парами точек должна обеспечивать дружественная функция этого класса. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массивов их адресов. P15

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска в любом заданном наборе точек плоскости пары наиболее отдаленных точек, которые находятся по разные стороны от оси X. Декартовые координаты всех точек заданного набора должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Вычисление расстояния между парами точек должна обеспечивать дружественная функция этого класса. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массивов их адресов. P16

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска в любом заданном наборе точек плоскости пары наиболее близких точек, которые разделены прямой с уравнением *y = x*. Декартовые координаты всех точек заданного набора должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений, а также компонентный метод вычисления расстояния до любой другой заданной точки. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массивов их адресов. P17

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска в любом заданном наборе точек плоскости пары наиболее отдаленных точек, которые разделены прямой с уравнением *y = x*. Декартовые координаты всех точек заданного набора должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений, а также компонентный метод вычисления расстояния до любой другой заданной точки. Кроме того, в программе должно быть предусмотрено динамическое распределение памяти для всех точек заданного набора и массивов их адресов. P18

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска в любом заданном наборе точек плоскости пары наиболее близких точек, которые разделены прямой с уравнением *y = −x*. Декартовые координаты всех точек заданного набора должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Вычисление расстояния между любыми парами точек должна выполнять внешняя функция. Кроме того, в программе нужно реализовать динамическое распределение памяти для всех точек заданного набора и массивов их адресов. P19

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска в любом заданном наборе точек плоскости пары наиболее удаленных точек, которые разделены прямой с уравнением *y = −x*. Декартовые координаты всех точек заданного набора должны быть указаны парами целых чисел в аргументах командной строки вызова программы. Координаты полученной пары точек и расстояние между ними должны отображаться строкой потока стандартного вывода. В программе должен быть реализован класс точки с приватными полями для ее декартовых координат, публичными методами доступа к ним и конструктором инициализации их значений. Вычисление расстояния между любыми парами точек должна выполнять внешняя функция. Кроме того, в программе нужно реализовать динамическое распределение памяти для всех точек заданного набора и массивов их адресов. P20

--------------------------------------------------------------------------------------------------------------------

## 4.2. Подготовка к лабораторной работе № 2

**Программирование магических квадратов**

Разработать ООП для построения магического квадрата из последовательных натуральных чисел по методу коня в следующем варианте. Начальное значение 1 нужно записать в среднюю клетку нижней строки квадрата. Запись чисел в остальные клетки должна происходить по ходу шахматного коня на 2 клетки вверх и на 1 клетку вправо. Если эта клетка уже занята, то следующее число нужно записать на 1 клетку ниже, чем предыдущее. Когда число оказывается за границей квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата (рисунок 2). Результат построения квадрата должен быть отображен через стандартный вывод. Значение порядка квадрата должно передаваться программе через аргумент командной строки. При разработке программы нужно реализовать класс магического квадрата с приватными полями для матрицы и порядка квадрата, а также публичными методами его заполнения и отображения. Конструктор и деструктор класса должны обеспечивать динамическое распределение памяти квадрата (программа magic2.cc).

На рисунке 3 представлен результирующий магический квадрат, построенный по алгоритму коня.

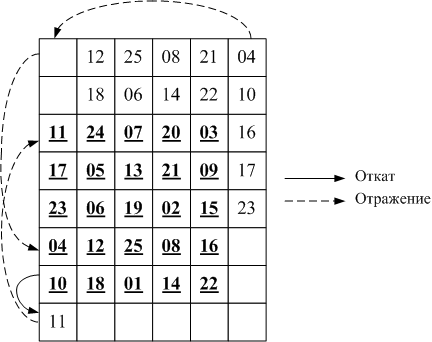


Рисунок 2 – К постановке задачи построения магического квадрата

//Программирование магического квадрата нечетного порядка, некратного 3, //Византийским методом с индийским откатом

#include <stdlib.h>

#include <stdio.h>

class Magic {//класс магического квадрата

private:

unsigned\*\* tab; // адрес таблицы квадрата

int row; // номер строки квадрата

int col; // номер столбца квадрата

int degree; // порядок квадрата

public:

Magic(int); // конструктор квадрата

~Magic(); // деструктор квадрата

void print();// печать таблицы квадрата

void horse3();//способ заполнения квадрата методом коня

int reflect(int); // правило отражения

}; // Magic

Magic::Magic(int n) {// конструктор магического квадрата

degree = n; // инициализация порядка квадрата

tab = new unsigned\* [degree]; // распределить массив адресов строк

for(row=0; row < degree; row++)// динамически распределить память по

tab[row] = new unsigned [degree]; // адресам строк квадрата

for(row=0; row < degree; row++)// инициализация клеток квадрата

for(col=0; col < degree; col++)//по строкам

tab[row][col] = 0; // нулями

} // Magic

Magic::~Magic() {// деструктор магического квадрата

for(row=0; row < degree; row++)//освободить динамическую память

delete [] tab[row]; // по адресам строк квадрата

delete [] tab; // освободить динамическую память массива адресов строк

} // ~Magic

void Magic::print() {//Метод печати квадрата

int degree2; // максимальное число квадрата

int len = 0; // ширина поля вывода чисел

degree2 = (degree \* degree);

while(degree2 > 0) {

degree2 /= 10;

len++;

} // while

for(row=0; row < degree; row++) {//построчная печать чисел квадрата

for(col=0; col < degree; col++)//с лидирующими нулями

printf("%0\*d ", len, tab[row][col]); //в поле вывода

putchar('\n');

} // for-row

putchar('\n');

return; //завершение метода вывода

} // print

int Magic::reflect(int k) {//метод отражения чисел внутрь квадрата

if(k < 0) // выход за правую или левую границу квадрата

return(k + degree); // k += degree; //отражение вниз или вправо

if(k > (degree - 1)) // выход за правую или нижнюю границу квадрата

return(k - degree); // k = 0; //отражение влево или вверх

return(k); //нет нарушения границ квадрата и нет отражения

} // reflect

// метод византийского коня с индийским откатом

void Magic::horse3() {

int i; // расчетный индекс строк

int j; // расчетный индекс столбцов

int degree2 = (degree\*degree); // максимальное число квадрата.

int z = 1;// текущее число квадрата 1<=z<=degree2

row = (degree - 1); //начальная строка заполнения

col = (degree / 2); //начальный столбец заполнения

tab[row][col] = z; //записать в начальную позицию значение z=1

while(z < degree2) {//цикл заполнения квадрата числами z>1

i = reflect(row - 2); // получить позицию числа по ходу коня (2 вверх)

j = reflect(col + 1); // и 1 вправо с отражением при нарушении границ

if(tab[i][j] > 0) { // проверка занятости клетки в расчетной позиции

i = reflect(row + 1);// индийский откат (1 вниз) от позиции числа z

j = reflect(col); //с отражением при нарушении границ

} // if

row = i; col = j; //фиксировать позицию следующего числа

++z; //взять следующее число

tab[row][col] = z; //записать очередное значение в квадрат

} // while

return; //завершить заполнение квадрата

} // horse3

int main(int argc, char\* argv[]) {//основная функция

if(argc != 2) //проверка формата командной строки

return(puts("Usage: magic degree"));

int n = atoi(argv[1]); //получить значение порядка квадрата

if((n % 2) == 0) //исключить четный порядок

{

puts("Usage: magic 5 (or 7, 11, 17, 19, 23, ...)");

return(n);

} // if

Magic mag(n); //конструирование магического квадрата

mag.horse3();//заполнение квадрата методом коня

mag.print();//печать квадрата

return(n); //корректное завершение программы

} // main

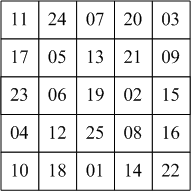


Рисунок 3 – Результирующий магический квадрат

**Варианты заданий для второй лабораторной работы**

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по индийскому методу в следующем варианте. Начальное значение 1 необходимо записать в среднюю клетку верхней строки квадрата. Заполнение всех следующих клеток последовательными числами должно происходить направо и вверх по диагонали. Если очередная клетка уже занята, то следующее число нужно записать под предыдущим. Когда число оказывается за границами квадрата, его следует перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по указанным правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы необходимо реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

M1

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по индийскому методу в следующем варианте. Начальное значение 1 необходимо записать в среднюю клетку верхней строки квадрата. Заполнение всех следующих клеток последовательными числами должно происходить налево и вверх по диагонали. Если очередная клетка уже занята, то следующее число нужно записать под предыдущим. Когда число оказывается за границами квадрата, его следует перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по указанным правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы необходимо реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

M2

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по индийскому методу в следующем варианте. Начальное значение 1 необходимо записать в среднюю клетку нижней строки квадрата. Заполнение всех следующих клеток последовательными числами должно происходить налево и вниз по диагонали. Если очередная клетка уже занята, то следующее число нужно записать над предыдущим. Когда число оказывается за границами квадрата, его следует перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по указанным правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы необходимо реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М3

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по индийскому методу в следующем варианте. Начальное значение 1 необходимо записать в среднюю клетку нижней строки квадрата. Заполнение всех следующих клеток последовательными числами должно происходить направо и вниз по диагонали. Если очередная клетка уже занята, то следующее число нужно записать над предыдущим. Когда число оказывается за границами квадрата, его следует перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по указанным правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы необходимо реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М4

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по сиамскому методу в следующем варианте. Начальное значение 1 необходимо записать в клетку справа от центра квадрата. Заполнение следующих клеток последовательными числами должно происходить направо и вверх по диагонали. Если очередная клетка уже занята, то следующее число нужно записать на 2 клетки справа от предыдущего. Когда число оказывается за границей квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по указанным правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М5

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по сиамскому методу в следующем варианте. Начальное значение 1 необходимо записать в клетку над центром квадрата. Заполнение следующих клеток последовательными числами должно происходить налево и вверх по диагонали. Если очередная клетка уже занята, то следующее число нужно записать на 2 клетки вверх от предыдущего. Когда число оказывается за границей квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по указанным правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М6

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по сиамскому методу в следующем варианте. Начальное значение 1 необходимо записать в клетку слева от центра квадрата. Заполнение следующих клеток последовательными числами должно происходить налево и вниз по диагонали. Если очередная клетка уже занята, то следующее число нужно записать на 2 клетки слева от предыдущего. Когда число оказывается за границей квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по указанным правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М7

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по сиамскому методу в следующем варианте. Начальное значение 1 необходимо записать в клетку под центром квадрата. Заполнение следующих клеток последовательными числами должно происходить направо и вниз по диагонали. Если очередная клетка уже занята, то следующее число нужно записать на 2 клетки вниз от предыдущего. Когда число оказывается за границей квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по указанным правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М8

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу Москопула в следующем варианте. Начальное значение 1 необходимо записать в среднюю клетку нижней строки квадрата. Запись чисел в остальные клетки должна происходить по ходу шахматного коня на 2 клетки вверх и на 1 клетку вправо. Если эта клетка уже занята, то следующее число следует записать на 4 клетки выше предыдущего. Когда число оказывается за границами квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по этим правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. В производном классе следует предусмотреть компонентные методы его заполнения и отображения.

М9

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу Москопула в следующем варианте. Начальное значение 1 необходимо записать в среднюю клетку нижней строки квадрата. Запись чисел в остальные клетки должна происходить по ходу шахматного коня на 2 клетки вверх и на 1 клетку влево. Если эта клетка уже занята, то следующее число следует записать на 4 клетки выше предыдущего. Когда число оказывается за границами квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по этим правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. В производном классе следует предусмотреть компонентные методы его заполнения и отображения.

М10

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу Москопула в следующем варианте. Начальное значение 1 необходимо записать в среднюю клетку верхней строки квадрата. Запись чисел в остальные клетки должна происходить по ходу шахматного коня на 2 клетки вниз и на 1 клетку вправо. Если эта клетка уже занята, то следующее число следует записать на 4 клетки ниже предыдущего. Когда число оказывается за границами квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по этим правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. В производном классе следует предусмотреть компонентные методы его заполнения и отображения.

М11

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу Москопула в следующем варианте. Начальное значение 1 необходимо записать в среднюю клетку верхней строки квадрата. Запись чисел в остальные клетки должна происходить по ходу шахматного коня на 2 клетки вниз и на 1 клетку влево. Если эта клетка уже занята, то следующее число следует записать на 4 клетки ниже предыдущего. Когда число оказывается за границами квадрата, его необходимо перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по этим правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. В производном классе следует предусмотреть компонентные методы его заполнения и отображения.

М12

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу альфила в следующем варианте. Начальное значение 1 нужно записать над клеткой в левом нижнем углу квадрата. Запись чисел в остальные клетки должна происходить направо и вверх по диагонали через 1 клетку. Если эта клетка уже занята, то следующее число нужно записать на 1 клетку правее и на 3 клетки выше предыдущего. Когда число оказывается за границами квадрата, его следует перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по таким правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М13

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу альфила в следующем варианте. Начальное значение 1 нужно записать над клеткой в правом нижнем углу квадрата. Запись чисел в остальные клетки должна происходить налево и вверх по диагонали через 1 клетку. Если эта клетка уже занята, то следующее число нужно записать на 1 клетку левее и на 3 клетки выше предыдущего. Когда число оказывается за границами квадрата, его следует перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по таким правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М14

-------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу альфила в следующем варианте. Начальное значение 1 нужно записать под клеткой в левом верхнем углу квадрата. Запись чисел в остальные клетки должна происходить направо и вниз по диагонали через 1 клетку. Если эта клетка уже занята, то следующее число нужно записать на 1 клетку правее и на 3 клетки ниже предыдущего. Когда число оказывается за границами квадрата, его следует перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по таким правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует предусмотреть в производном классе.

М15

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу альфила в следующем варианте. Начальное значение 1 нужно записать под клеткой в правом верхнем углу квадрата. Запись чисел в остальные клетки должна происходить налево и вниз по диагонали через 1 клетку. Если эта клетка уже занята, то следующее число нужно записать на 1 клетку левее и на 3 клетки ниже предыдущего. Когда число оказывается за границами квадрата, его следует перенести внутрь квадрата, изменив заграничную координату на порядок квадрата. Результат построения магического квадрата по таким правилам должен отображаться в поток стандартного вывода. Значение порядка квадрата должно передаваться программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует реализовать в производном классе.

М16

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу террас Баше в следующем варианте. Сначала пустой квадрат заполняется снизу вверх по вертикалям последовательными натуральными числами. Затем его нужно повернуть на 45 градусов по часовой стрелке и наложить на пустой квадрат в основной ориентации. В результате клетки пустого квадрата будут заполнены в шахматном порядке числами из развернутого квадрата. В пустые клетки нужно отобразить все остальные числа развернутого квадрата, изменив их заграничные координаты на величину порядка квадрата. Полученный магический квадрат должен отображаться в поток стандартного вывода. Значение его порядка нужно передавать программе через аргумент командной строки. При разработке программы нужно реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует реализовать в производном классе.

М17

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу террас Баше в следующем варианте. Сначала пустой квадрат заполняется по горизонталям слева направо последовательными натуральными числами. Затем его нужно повернуть на 45 градусов по часовой стрелке и наложить на пустой квадрат в основной ориентации. В результате клетки пустого квадрата будут заполнены в шахматном порядке числами из развернутого квадрата. В пустые клетки нужно отобразить все остальные числа развернутого квадрата, изменив их заграничные координаты на величину порядка квадрата. Полученный магический квадрат должен отображаться в поток стандартного вывода. Значение его порядка нужно передавать программе через аргумент командной строки. При разработке программы нужно реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует реализовать в производном классе.

М18

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу террас Баше в следующем варианте. Сначала столбцы пустого квадрата заполняются последовательными натуральными числами снизу вверх. Затем квадрат нужно повернуть на 45 градусов против часовой стрелки и наложить на пустой квадрат в основной ориентации. В результате клетки пустого квадрата будут заполнены в шахматном порядке числами из развернутого квадрата. В пустые клетки нужно отобразить все остальные числа развернутого квадрата, изменив их заграничные координаты на величину порядка квадрата. Полученный магический квадрат должен отображаться в поток стандартного вывода. Значение его порядка нужно передавать программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует реализовать в производном классе.

М19

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для построения магического квадрата любого нечетного порядка из последовательных натуральных чисел по методу террас Баше в следующем варианте. Сначала строки пустого квадрата в порядке снизу вверх заполняются натуральными числами справа налево. Затем квадрат нужно повернуть на 45 градусов по часовой стрелке и наложить на пустой квадрат в основной ориентации. В результате клетки пустого квадрата будут заполнены в шахматном порядке числами из развернутого квадрата. В пустые клетки нужно отобразить все остальные числа развернутого квадрата, изменив их заграничные координаты на величину порядка квадрата. Полученный магический квадрат должен отображаться в поток стандартного вывода. Значение его порядка нужно передавать программе аргументом командной строки. При разработке программы следует реализовать базовый класс квадратной матрицы и производный от него класс магического квадрата. Конструктор базового класса должен обеспечивать динамическое распределение памяти по защищенному адресу для двумерного массива. Компонентные методы для его заполнения и отображения следует реализовать в производном классе.

М20

--------------------------------------------------------------------------------------------------------------------

# 5. Статический полиморфизм

Статический полиморфизм реализуется на этапе компиляции с помощью **перегрузки** функций и операций. Под перегрузкой функций в С++ понимается описание **в одной области видимости** нескольких функций с одним и тем же именем. О перегрузке операций в С++ говорят в том случае, если в некоторой области видимости по-является описание функции с именем **operator <обозначение\_операции\_С++>**, задающее интерпретацию заданной операции [5-6,8-9].

## 5.1. Перегрузка бинарных операций

Для перегрузки операций используется ключевое слово **operator.** Прототип перегруженной операции:

*тип\_возвращаемого значения* ***operator*** *символ оператора* (*операнды*) { *тело*\_*функции*};

Перегружать операции можно с помощью:

* функции-члена;
* функции-друга;
* глобальной функции (как правило, менее эффективно).

Можно перегружать любые операции языка С++, кроме следующих операций:

* . – операция выбора члена класса
* :: – операция разрешения области видимости
* ? : – условная операция (например, *j = i>0 ? 1 : 0;*)
* .\* – операция разыменования указателя на член класса
* # – директива препроцессора
* ***sizeof***
* ***typeid***

При перегрузке операции с помощью метода класса число формальных параметров оказывается на единицу меньше числа фактических операндов операции. В этом случае первый операнд операции соответствует объекту типа класса, в котором перегружается операция. В случае бинарной операции входной параметр соответствует второму операнду перегружаемой операции.

При перегрузке операции с помощью функции-друга число формальных параметров совпадает с числом операндов операции, так как в этом случае операнды операции, представленные формальными параметрами, являются внешними объектами для такой функции.

Тип выходного параметра является встроенным типом или типом, определенным пользователем (то есть классом).

Если при перегрузке операции методом класса результатом применения операции является изменение первого (или единственного) операнда, то рекомендуется объявлять выходной параметр в виде ссылки на текущий объект. Это необходимо для оптимизации использования результата операции в других операциях, совмещенных в одном операторе, например: *z = x += y;*

Если при перегрузке операции функцией-членом результатом применения перегружаемой операции является вычисление значения, не изменяющего первый операнд, а также при перегрузке операции функцией-другом, выходной параметр не может быть ссылкой (если выходной параметр требуется). Это связано с тем, что вычисляемое значение помещается во временный объект, который уничтожается при завершении работы алгоритма перегруженной операции и выходе из области видимости этого временного объекта.

**Перегрузка операции присваивания может быть произведена только методом класса** и не может быть перегружена функцией-другом.

В отличие от операции присваивания **операция *‘+=’*** (и другие подобные операции) **может быть перегружена как методом класса, так и функцией-другом**.

Прототип перегрузки операции присваивания:

X & ***operator*** = (***const*** X &); или X & ***operator*** = (X&);

Операцию следует перегружать функцией членом того класса, который является типом первого операнда. Если первый операнд имеет встроенный или библиотечный тип, в описание которого невозможно вставить описание дружественной функции, то такую операцию можно перегружать только функцией-другом класса, к которому относится второй операнд [8-9].

**Перегрузка операции вывода**

В файле внешней стандартной библиотеки *iostream* стандартная операция языка ‘***<<***’, осуществляющая побитовый сдвиг, перегружена в классе *ostream* как операция вывода. Операция ‘***<<***’перегружена для вывода объектов стандартных типов: ***int***, ***char***, ***double***, ***char\* –***и других встроенных типов. Формат использования данной операции:

cout << *переменная\_стандартного\_типа*;

Таким образом, первый операнд операции ‘***<<***’должен иметь тип *ostream*. Если необходимо перегрузить данную операцию для структурированного вывода объекта астрактного пользовательского типа, то это можно сделать только функцией-другом разработанного класса. Она должна иметь два аргумента. Один типа ссылки на объекты класса *ostream*, а другой ссылка на объект класса, для которого нужно перегрузить эту операцию. Код возврата должен иметь тип ссылки на объект класса *ostream*. Код этой оператор-функции должен обеспечивать вывод требуемых полей класса как данных стандарного типа [8-9].

## 5.2. Перегрузка унарных операций

Если для унарной операции имеется только одна форма, то ее перегрузка реализуется по общим описанным выше правилам. При этом, как уже было описано, для оптимизации использования результата операции в других операциях, совмещенных в одном операторе с данной операцией, рекомендуется объявлять выходной параметр в виде ссылки на текущий объект.

При перегрузке унарной операции в том случае, если для нее в языке определены две формы – **префиксная** и **постфиксная**, имеются особенности.

Для того чтобы отличать постфиксную форму от префиксной, при перегрузке операции в постфиксной форме в списке формальных параметров указывается дополнительный, неиспользуемый в алгоритме операции, параметр (точнее, тип параметра).

Операция индексирования является бинарной: ее операнды – объект с нумерованными элементами (массив, вектор и т. д.) и целое число – индекс элемента.

# 6. Практическое изучение статического полиморфизма с помощью перегрузки операций

## 6.1. Подготовка к лабораторной работе № 3

**Программирование рациональных дробей**

Разработать ООП вычисления обыкновенной дроби до ближайшего целого числа, значение которого будет не меньше ее величины по модулю. Символьная запись дроби должна передаваться программе в качестве аргумента командной строки, где числитель и знаменатель разделены знаком ‘/’. Результирующее дополнение должно отображаться строкой стандартного вывода в том же символьном формате. Программная реализация вычислений должна быть основана на разработке класса обыкновенной дроби с приватными полями целочисленных данных для ее числителя и знаменателя, а также публичным компонентным методом перегрузки оператора ‘~’. Конструкторы этого класса должны обеспечивать преобразование символьной записи дроби в числовой формат и инициализацию значений ее числителя и знаменателя. Кроме того, нужно предусмотреть компонентный метод приведения дроби к несократимому виду, использующий алгоритм Евклида для вычисления наибольшего общего делителя (НОД) ее числителя и знаменателя, а также компонентный оператор преобразования числовой записи дроби в формат символьной строки для стандартного вывода результатов вычислений.

//Дополнение дроби до целого числа

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <iostream>

using std::cout;

using std::endl;

int euclide(int, int);//функция алгоритма Евклида

class Fraction {//класс обыкновенной дроби

private:

int nom; //числитель(nominator)дроби

int den; //знаменатель (denominator) дроби

public:

Fraction(char\*);//символьный конструктор

Fraction(int n=0, int m=1) : nom(n), den(m) {};

Fraction operator~();//оператор дополнения

operator char\*();// оператор преобразования типа записи дроби

void reduce();//метод сокращения дроби

}; // class Fraction

//конструирование дроби по символьной записи «n/m=s»

Fraction::Fraction(char\* s) {

char\* p = strchr(s, '/'); //адресовать '/' в символьной записи дроби

den = 1; //инициализировать знаменатель дроби

if(p != NULL) {//вычисление знаменателя дроби

\*p++ = '\0'; // адресовать знаменатель и заменить '/' на '\0'

den = atoi(p); //преобразовать запись знаменателя в целое число

} // if

nom = atoi(s); //преобразовать запись числителя в целое число

}// Fraction

//оператор дополнения дроби до целого:

Fraction Fraction::operator~() {

int k = 0; //кратность знаменателя

int sign = (nom < 0) ? -1 : 1; //фиксировать знак числителя

int n = sign \* nom; //модуль числителя дроби

while(k < n) //Найти наименьшее число, которое не меньше числителя

k += den; //по модулю кратно знаменателю дроби

n = (k - n); //модуль числителя дополнения

return Fraction(n \* sign, den); //анонимный возврат дополнения

} // operator~

//оператор преобразования дроби в символьный формат ”n/m”

Fraction::operator char\* () {

static char s[32]; //строка для символьной записи дроби

sprintf(s, "%d/%d", nom, den); //символьная запись дроби

return s; //возврат адреса символьной записи дроби

} // operator char\*

//метод приведения дроби к несократимому виду

void Fraction::reduce() {

int gcd; //наименьший общий делитель (НОД)

gcd = euclide(abs(nom), den);//вычисление НОД знаменателя и модуля числителя

nom /= gcd; //сокращение числителя на НОД

den /= gcd; // сокращение знаменателя на НОД

return; //возврат из метода сокращения дроби

}// reduce

//вычисление наибольшего общего делителя НОД(m,n) по алгоритму Евклида

int euclide(int n, int m) {

int r = 1; //остаток деления

while(n != 0) {//цикл уменьшения чисел m и n

r = m % n; //найти остаток целочисленного деления m на n

m = n; //уменьшение m и n без

n = r; //изменения НОД

} // while

return(m); //возврат НОД(m,n)

} // euclide

int main(int argc, char\* argv[]) {//основная функция

if(argc < 2) //проверка числа аргументов командной строки

return(puts("Usage: complement nomerator/denomerator"));

Fraction x(argv[1]); //получить исходную дробь в числовом формате

Fraction y; //инициализация дополнения конструктором по умолчанию

x.reduce();//сокращение исходной дроби

y = ~x; //вычисление дополнения дроби

cout << (char\*) y << endl; //стандартный вывод дополнения дроби

return(0); //корректное завершение программы

} // main

Примеры дополнений: ~6/7=1/7; ~-23/5=-2/5.

**Варианты заданий для третьей лабораторной работы**

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу преобразования заданной обыкновенной дроби в конечную десятичную дробь. Обыкновенная дробь должна передаваться программе аргументом командной строки, где записи числителя и знаменателя десятичными цифрами разделены знаком **'/'**. Результат преобразований должна отображать запись вещественного числа с фиксированной точкой строкой стандартного вывода. Программная реализация этого преобразования должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также оператором преобразования типа для получения вещественного числа по результату их деления. Конструктор класса должен обеспечивать представление исходной символьной записи обыкновенной дроби парой взаимно простых целочисленных значений ее числителя и знаменателя. Для приведения обыкновенной дроби к несократимому виду необходимо предусмотреть компонентный метод, реализующий алгоритм Евклида.

R1

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для округления заданной обыкновенной дроби до ближайшего целого числа, значение которого не превосходит ее величину. Запись обыкновенной дроби должна передаваться программе аргументом командной строки, где числитель и знаменатель разделены знаком **'/'**. Результат округления должен отображаться строкой потока стандартного вывода. Программная реализация округления должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также оператором преобразования типа, который преобразует результат их деления в целое число. Конструктор класса должен обеспечивать представление исходной символьной записи обыкновенной дроби парой взаимно простых целочисленных значений ее числителя и знаменателя. Для приведения обыкновенной дроби к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида.

R2

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для округления заданной обыкновенной дроби до ближайшего целого числа, значение которого должно быть не меньше ее величины. Запись обыкновенной дроби должна передаваться программе аргументом командной строки, где числитель и знаменатель разделяет знак **'/'**. Результат округления должен отображаться строкой потока стандартного вывода. Программная реализация округления должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также оператором преобразования типа, который преобразует результат их деления в целое число. Конструктор класса должен обеспечивать представление исходной символьной записи обыкновенной дроби парой взаимно простых целочисленных значений ее числителя и знаменателя. Для приведения обыкновенной дроби к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида.

R3

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу представления заданной обыкновенной дроби в виде суммы целой и правильной дробной части. Обыкновенная дробь должна передаваться программе аргументом командной строки, где записи числителя и знаменателя десятичными цифрами разделяет символ **'/'**. Результат преобразований должен отображаться строкой стандартного вывода, где записи целой и правильной дробной части обыкновенной дроби разделены символом **'+'**. Программная реализация преобразований должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя. Получение их значений по заданной символьной записи обыкновенной дроби должен обеспечивать конструктор класса. Их функциональную обработку должен обеспечивать компонентный оператор преобразования типа для вычисления целой части обыкновенной дроби и компонентный оператор **'−'** для перегрузки операции вычитания его из исходной дроби.

R4

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу представления заданной обыкновенной дроби в виде разности целой и правильной дробной части. Обыкновенная дробь должна передаваться программе аргументом командной строки, где записи числителя и знаменателя десятичными цифрами разделяет символ **'/'**. Результат преобразований должен отображаться строкой стандартного вывода, где записи целой и правильной дробной части обыкновенной дроби разделены символом **'−'**. Программная реализация преобразований должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя. Получение их значений по заданной символьной записи обыкновенной дроби должен обеспечивать конструктор класса. Их функциональную обработку должен обеспечивать компонентный оператор преобразования типа для вычисления целой части обыкновенной дроби и компонентный оператор **'−'** для перегрузки операции вычитания ее из исходной дроби.

R5

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления суммы обыкновенной дроби и целого числа, которые должны передаваться ей через аргументы командной строки. Результат сложения должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. Во всех случаях для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'+'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходной и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R6

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления разности обыкновенной дроби и целого числа, которые должны передаваться ей через аргументы командной строки. Результат вычитания должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. Во всех случаях для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'−'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходной и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R7

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перемножения обыкновенной дроби и целого числа. Оба операнда должны передаваться ей через аргументы командной строки. Результат умножения должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. Во всех случаях для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'\*'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходной и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R8

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для деления обыкновенной дроби на целое число. Оба операнда должны передаваться ей через аргументы командной строки. Результат деления должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. Во всех случаях для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'/'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходной и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R9

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления суммы обыкновенных дробей, которые должны передаваться ей через два аргумента командной строки. Результат суммирования должен отображаться строкой стандартного вывода в формате обыкновенной дроби. Во всех случаях для записи обыкновенных дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений суммы должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'+'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходных и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R10

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления разности обыкновенных дробей, которые должны передаваться ей через два аргумента командной строки. Результат вычитания должен отображаться строкой стандартного вывода в формате обыкновенной дроби. Во всех случаях для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений разности должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'−'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходных и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R11

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления произведения двух обыкновенных дробей, которые должны передаваться ей через аргументы командной строки. Результат умножения должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. Во всех случаях для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'\*'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходных и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R12

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления отношения обыкновенных дробей, которые должны передаваться ей через два аргумента командной строки. Результат этой операции деления должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. Во всех случаях для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'/'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходных и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R13

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления суммы целого числа и обыкновенной дроби, которые должны передаваться ей через аргументы командной строки. Результат операции должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. В любом случае для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также дружественным методом перегрузки оператора **'+'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходной и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R14

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления разности целого числа и обыкновенной дроби, которые должны передаваться ей через аргументы командной строки. Результат операции должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. В любом случае для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также дружественным методом перегрузки оператора **'−'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходной и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R15

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перемножения целого числа и обыкновенной дроби, которые должны передаваться ей через аргументы командной строки. Результат операции должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. В любом случае для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также дружественным методом перегрузки оператора **'\*'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходной и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R16

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу деления целого числа на обыкновенную дробь. Оба операнда должны передаваться ей через аргументы командной строки. Результат деления должен отображаться строкой потока стандартного вывода в формате обыкновенной дроби. В любом случае для записи обыкновенной дроби должен применяться символьный формат, где ее числитель и знаменатель разделены знаком **'/'**. Программная реализация требуемых вычислений должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений числителя и знаменателя, а также компонентным методом перегрузки оператора **'/'**. Конструкторы класса должны выполнять преобразования обоих операндов в указанный числовой формат обыкновенных дробей. Для приведения исходной и результирующей обыкновенных дробей к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида, и перегрузку оператора присваивания.

R17

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу сравнения двух обыкновенных дробей, которые должны передаваться ей через аргументы командной строки. Результат сравнения должен отображаться строкой потока стандартного вывода, где заданные дроби разделяет знак отношения порядка между их величинами. Во всех случаях для записи обыкновенных дробей должен применяться символьный формат, где их числители и знаменатели разделены знаком **'/'**. Программная реализация сравнения должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений их числителя и знаменателя, а также компонентными методами перегрузки операторов **'>'** и **'<'** в условных выражениях. Конструктор этого класса должен обеспечивать преобразование символьной записи сравниваемых обыкновенных дробей в указанный числовой формат.Для приведения обеих обыкновенных дробей к несократимому виду необходимо предусмотреть компонентный метод, реализующий алгоритм Евклида.

R18

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для сравнения обыкновенной дроби и целого числа, которые должны передаваться ей аргументами командной строки. Результат сравнения должен отображаться строкой потока стандартного вывода, где заданные дробь и целое число разделяет знак отношения порядка между их величинами. Во всех случаях для записи обыкновенных дробей должен применяться символьный формат, где их числители и знаменатели разделены знаком **'/'**. Программная реализация сравнения должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений их числителя и знаменателя, а также компонентными методами перегрузки операторов **'>'** и **'<'** в условных выражениях. Конструктор этого класса должен выполнять преобразование символьной записи обыкновенной дроби в указанный числовой формат.Для приведения обыкновенной дроби к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида.

R19

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу сравнения целого числа и обыкновенной дроби, которые должны передаваться ей двумя аргументами командной строки. Результат сравнения должен отображаться строкой потока стандартного вывода, где заданные целое число и дробь разделяет знак отношения порядка между их величинами. Во всех случаях для записи обыкновенных дробей должен применяться символьный формат, где их числители и знаменатели разделены знаком **'/'**. Программная реализация сравнения должна быть основана на разработке класса обыкновенных дробей с приватными полями данных для целочисленных значений их числителя и знаменателя, а также дружественными методами перегрузки операторов **'>'** и **'<'** в условных выражениях. Конструкторы этого класса должны обеспечивать преобразование обеих сравниваемых величин в указанный числовой формат обыкновенной дроби.Для приведения обыкновенной дроби к несократимому виду следует предусмотреть компонентный метод, реализующий алгоритм Евклида.

R20

--------------------------------------------------------------------------------------------------------------------

## 6.2. Подготовка к лабораторной работе № 4

**Программирование логических операций на множестве латинских букв**

Разработать ООП для объединения 2-х любых подмножеств латинских букв, которые передаются ей через аргументы командной строки. Полученный результат должен отображаться строкой стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация вычислений должна быть основана на разработке класса множества латинских букв. При этом состав множества кодируется двоичными разрядами целого числа без знака в приватном поле данных этого класса, а публичный компонентный метод предназначается для перегрузки оператора ‘**+’** в операции объединения 2-х множеств. Конструктор класса должен обеспечивать преобразование символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть дружественную перегрузку оператора ‘**<<’** потока стандартного вывода или компонентный оператор преобразования типа множества в символьную строку для отображения объектов класса множества латинских букв.

//Перегрузка объединения подмножеств латинских букв

#include <ctype>

#include <iostream>

using namespace std;

class Alpha{//класс множества латинских букв

private:

unsigned bin; //бинарный код подмножества латинских букв

public:

Alpha() {bin=0;};

Alpha(Alpha& y) {bin=y.bin;};

Alpha(char\*);

Alpha operator +(Alpha&);

operator char\*();

friend ostream& operator << (ostream&,Alpha&);

};// Alpha

//конструктор множества букв по строке

Alpha::Alpha(char\* s){

bin=0;

while(\*s){

bin | = (1 << (tolower(\*s)-'a')); s++;}

}//Alpha

//оператор объединения подмножеств букв

Alpha Alpha:: operator+(Alpha& y){

Alpha z;

z.bin=bin | y.bin;

return(z);}//operator+

//оператор вывода подмножеств букв

ostream& operator << (ostream& out,Alpha& z){

unsigned bit=1;

int i;

for(i=0; i<26;i++){

if((z.bin& bit)>0)

out<<(char)('a'+i);

bit=bit<<1;

}

return out;}// operator <<

//оператор преобразования множества в строку

Alpha::operator char\*(){

static char s[32];

unsigned b=bin;

int i=0;

int j=0;

int w;

while(w>0){

if(w & 1)

s[j++]='a'+i;

i++;

w>>1;

}//while

s[j]='\0';

return (s);

}//operator char\*()

//основная функция

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

Alpha x(argv[1]); //буквы 1-го аргумента

Alpha y(argv[2]); //буквы 2-го аргумента

Alpha z; //буквы результата

z=x+y; // объединение подмножеств букв

cout<<z; // или cout<<(char\*)z;

return (0);

}//main

**Варианты заданий для четвертой лабораторной работы**

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления разности двух любых подмножеств латинских букв, которые заданы двумя символьными строками аргументов команды ее вызова. Искомую разность образуют все буквы первого подмножества, которые отсутствуют во втором. Полученный результат должен отображаться строкой стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация вычислений разности должна быть основана на разработке класса множества латинских букв с компонентным методом перегрузки оператора '−' и полем приватных данных, где состав множества фиксируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B1

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления симметрической разности двух любых подмножеств латинских букв, которые передаются ей аргументами командной строки. Такую разность образуют все различные буквы из двух заданных подмножеств. Полученный результат должен отображаться строкой стандартного вывода, где все буквы перечислены в алфавитном порядке. Программный код должен быть основан на разработке класса множества латинских букв с компонентными методами перегрузки операторов '+' и '−' для выполнения теоретико-множественных операций объединения двух разностей. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируют двоичные разряды целого числа. Конструктор этого класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B2

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления дополнения до полного алфавита любого подмножества латинских букв, которое задается символьной строкой в аргументе командной строки ее вызова. Результатом этой операции должны быть все буквы латинского алфавита, кроме тех, которые указаны в аргументе программы. Полученное дополнение должна отображать строка стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация таких вычислений должна быть основана на разработке класса множества латинских букв с компонентным методом для перегрузки оператора '~' и полем приватных данных, где состав множества кодируется двоичными разрядами целого числа. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B3

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв, которые отсутствуют в двух любых символьных строках, заданных аргументами команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация таких перечислений должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '~' и '+' для выполнения теоретико-множественных операций дополнения и объединения. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B4

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв кроме тех, которые совпадают в двух любых символьных строках, заданных аргументами команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация таких перечислений должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '\*' и '~' для выполнения теоретико-множественных операций пересечения и дополнения двух множеств. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B5

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления дополнения до полного алфавита разности двух заданных подмножеств латинских букв, которые передаются ей через аргументы командной строки. Результатом этой операции должны быть все латинские буквы, кроме тех, которые имеются только в первом аргументе. Они должны отображаться строкой стандартного вывода, где все буквы перечислены в алфавитном порядке. Программа должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '~' и '−' для выполнения операций дополнения и разности. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B6

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для исключения всех латинских букв, по которым совпадают два любых их подмножества, заданные двумя символьными строками в аргументах команды ее вызова. Все оставшиеся различные буквы должны отображаться строкой потока стандартного вывода, где они перечисляются в алфавитном порядке. Программная реализация такого исключения совпадений должна базироваться на разработке класса множества латинских букв с компонентным методом перегрузки оператора '^' и полем приватных данных, где состав множества кодируется двоичными разрядами целого числа без знака. Конструктор этого класса должен обеспечивать преобразование исходной символьной записи подмножества латинских букв в бинарный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B7

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв, которые одновременно отсутствуют или совпадают в двух любых их подмножествах, заданных двумя символьными строками аргументов команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация таких перечислений должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '^' и '~' для выполнения теоретико-множественных операций исключения совпадений и дополнения. Обе они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B8

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления пересечения двух любых подмножеств латинских букв, которые передаются ей через аргументы командной строки. Полученный результат должен отображаться строкой потока стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация вычислений должна быть основана на разработке класса множества латинских букв с компонентным методом перегрузки оператора '\*' для выполнения теоретико-множественной операций пересечения двух множеств. Она должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B9

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв, которые отсутствуют в двух любых символьных строках, заданных аргументами команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация таких перечислений должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '\*' и '~' для выполнения теоретико-множественных операций пересечения дополнений двух множеств. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B10

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв кроме тех, которые совпадают в двух любых символьных строках, заданных аргументами команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация таких перечислений должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '+' и '~' для выполнения теоретико-множественных операций объединения дополнений двух множеств. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B11

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв, кроме тех, которые есть только в первой из двух символьных строк, заданных аргументами команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечисляются в алфавитном порядке. Программная реализация должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '~' и '+' для выполнения теоретико-множественных операций дополнения и объединения. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B12

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления разности двух любых подмножеств латинских букв, которые задают 2 символьные строки аргументов команды ее вызова. Искомую разность образуют все буквы первого подмножества, которые отсутствуют во втором. Полученный результат должен отображаться строкой потока стандартного вывода, где все буквы перечисляются в обратном алфавитном порядке. Программная реализация вычисления разности должна быть основана на разработке класса множества латинских букв с компонентным методом перегрузки оператора '−' и полем приватных данных, где состав множества фиксируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B13

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления симметрической разности двух любых подмножеств латинских букв, которые передаются ей аргументами командной строки. Такую разность образуют все различные буквы из двух заданных подмножеств. Полученный результат должен отображаться строкой стандартного вывода, где все буквы перечислены в обратном алфавитном порядке. Программный код должен быть основан на разработке класса множества латинских букв с компонентными методами перегрузки операторов '+' и '−' для выполнения теоретико-множественных операций объединения двух разностей. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируют двоичные разряды целого числа. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный формат целого числа. Нужно также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B14

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления дополнения до полного алфавита любого подмножества латинских букв, которое задается символьной строкой в аргументе командной строки ее вызова. Результатом этой операции должны быть все буквы латинского алфавита, кроме тех, которые указаны в аргументе программы. Полученное дополнение должно отображаться строкой стандартного вывода, где все буквы перечислены в обратном алфавитном порядке. Программная реализация этих вычислений должна быть основана на разработке класса множества латинских букв с компонентным методом перегрузки оператора '~' и полем приватных данных, где состав множества кодируется двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B15

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв, которые отсутствуют в двух любых символьных строках, заданных аргументами команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечислены в обратном алфавитном порядке. Программная реализация таких перечислений должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '~' и '+' для выполнения теоретико-множественных операций дополнения и объединения. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Следует также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B16

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв кроме тех, которые совпадают в двух любых символьных строках, заданных аргументами команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечислены в обратном алфавитном порядке. Программная реализация таких перечислений должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '\*' и '~' для выполнения операций пересечения и дополнения двух множеств. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B17

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления дополнения до полного алфавита разности двух заданных подмножеств латинских букв, которые передаются ей аргументами командной строки. Результатом являются все латинские буквы, кроме тех, которые есть только в первом аргументе. Они должны отображаться строкой стандартного вывода, где все буквы перечислены в обратном алфавитном порядке. Программа должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '~' и '−' для выполнения операций дополнения и разности. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B18

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для исключения всех латинских букв, по которым совпадают два любых их подмножества, заданные двумя символьными строками в аргументах команды ее вызова. Все оставшиеся различные буквы должны отображаться строкой потока стандартного вывода, где они перечислены в обратном алфавитном порядке. Программная реализация такого исключения совпадений должна базироваться на разработке класса множества латинских букв с компонентным методом перегрузки оператора '^' и полем приватных данных, где состав множества кодируется двоичными разрядами целого числа без знака. Конструктор этого класса должен обеспечивать преобразование исходной символьной записи подмножества латинских букв в бинарный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B19

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для перечисления всех латинских букв, которые одновременно отсутствуют или совпадают в двух любых их подмножествах, заданных двумя символьными строками аргументов команды ее вызова. Искомый результат должна отображать строка стандартного вывода, где все буквы перечислены в обратном алфавитном порядке. Программная реализация таких перечислений должна быть основана на разработке класса множества латинских букв с компонентными методами перегрузки операторов '^' и '~' для выполнения операций исключения совпадений и дополнения. Они должны быть ориентированы на обработку поля приватных данных, где состав множества фиксируется двоичными разрядами целого числа. Конструктор класса должен обеспечивать преобразование исходной символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть перегрузку оператора класса потока стандартного вывода для отображения объектов класса множества латинских букв.

B20

--------------------------------------------------------------------------------------------------------------------

# 7. Виды отношений между классами

**Ассоциация** представляет смысловую связь между сущностями (объектами), создаваемыми на основе классов. *Ассоциация (association)* определяется некоторой связью между классами. Когда в системе создаются представители ассоциированных классов, они связываются так, как определяет данная ассоциация [5-9].

Ассоциации между классами разрабатываются в процессе так называемого **семантического моделирования**: моделирования структуры данных исходя из их смысла. Для этого полезно использовать **ER-диаграммы** (Entity – Relationship: Сущность – Связь). ER-диаграммы используются в разных аспектах проектирования сложных программных комплексов, причем, не только комплексов, разрабатываемых с использованием объектно-ориентированной парадигмы, но и при разработке баз данных и во многих других приложениях.

Основными понятиями, используемыми при построении ER-диаграмм, являются следующие.

1)**Сущность** – класс однотипных объектов, информация о которых должна быть учтена в модели. Сущность в рамках ООП представляется классом.

2)**Экземпляр сущности** – объект, создаваемый на основе класса.

3) **Атрибут сущности** – именованная характеристика. В ООП – информационный член класса.

4) **Ключ сущности** – совокупность атрибутов, однозначно определяющих объект.

5) **Связь** – ассоциация между сущностями.

Типы связей:

* один к одному;
* один ко многим;
* многие ко многим.

Пример связи: группа – студенты. Связь может иметь одну из двух модальностей:

* может (может быть, а может и не быть);
* должен.

При разработке ER-модели определяется следующая информация:

* список сущностей;
* список атрибутов;
* описание связей.

Связи между сущностями реализуются с помощью механизмов наследования, агрегирования и использования.

**Наследование**– отношение между классами, при котором один класс повторяет структуру и поведение другого класса (одиночное наследование) или других (множественное наследование) классов.

Класс, поведение и структура которого наследуется, называется базовым (родительским) классом, а класс, который наследует – производным классом.

В производном классе структура и поведение базового класса (информационные члены и методы), дополняются и переопределяются. В производном классе указываются только дополнительные и переопределяемые члены класса. Производный класс является уточнением базового класса:

**class** z: **public** y{. . . };

**Агрегация**– это отношение между классами типа целое/часть. Агрегируемый класс в той или иной форме является частью агрегата. Объект класса-агрегата может хранить объект агрегируемого класса, или хранить ссылку (указатель) на него.

**Композиция** является специальным видом агрегирования (так называемое сильное агрегирование). Композиция объектов заключается в использовании объектов типов разработанных классов в качестве информационных членов при описании других классов [5].

**Использование**– отношение между классами, при котором один класс в своей реализации использует в той или иной форме реализацию объектов другого класса.

Использование одним классом объектов другого класса может проявляться одним из следующих образов:

* имя одного класса используется в профиле метода другого класса;
* в теле метода одного класса создаётся локальный объект другого класса;
* метод одного класса обращается к методу другого класса (не совсем частный случай предыдущего способа использования, так как при вызове статических членов классов локальный объект не создаётся).

## 8. Практическое изучение композиции с помощью использования композитных классов

## 8.1. Подготовка к лабораторной работе № 5

**Программирование геометрических объектов на плоскоти**

Разработать ООП для поиска точек Сх и Сy осей координат, которые находятся на минимальном расстоянии от пары заданных точек A и B плоскости XY. Координаты пары заданных точек A и B должны передаваться программе через поток стандартного ввода. Искомые точки Сх и Сy образуют пересечения с осями координат либо отрезка AB, либо отрезка с симметричным отражением точки A или B относительно оси, которую отрезок AB не пересекает. Координаты полученных точек Сх и Сy должны отображаться через поток стандартного вывода. Программная реализация этих геометрических построений должна быть основана на разработке композитного класса отрезка прямой. Его приватные данные должны вллючать подобъекты класса точки для концов отрезка, а публичные методы должны обеспечивать построение точек пересечения отрезка с осями координат. Конструкторы классов отрезка и точки должны использовать списки инициализации своих приватных данных. Необходимо также предусмотреть перегрузку операторов ввода-вывода точек и обеспечить доступ к их приватным координатам.

//Минимальное расстояние пары точек плоскости от осей координат

include <iostream>//заголовок потоков ввода-вывода

using namespace std; //разрешить доступ к стандартному пространству имен

class Line; //превентивное объявление класса отрезка прямой линии

class Dot {//декларация класса точки

friend class Line; //разрешить доступ в Line к приватным данным Dot

friend istream& operator>>(istream&, Dot&);//перегрузка оператора ввода

friend ostream& operator<<(ostream&, Dot&);// перегрузка оператора вывода

private:

float x; //координата точки по оси X

float y; // координата точки по оси Y

public:

Dot(float xx=0.0, float yy=0.0) : x(xx), y(yy) { };//конструктор //точки

// Dot(Dot& p) { x=p.x; y=p.y; };

float getx() { return x; };//получить координату точки по оси X

float gety() { return y; };// получить координату точки по оси Y

Dot symx() { return Dot(-x, y); };//отразить точку относительно оси Y

Dot symy() { return Dot(x, -y); };// отразить точку относительно оси X

}; // Dot

class Line {// декларация класса отрезка линии

private:

Dot p1; //начальная точка отрезка

Dot p2; //конечная точка отрезка

public:

Line(Dot& p, Dot& q) : p1(p), p2(q) {};// конструктор отрезка по двум //точкам

Line(float x1, float y1, float x2, float y2) : p1(x1,y1), p2(x2,y2) {};// конструктор //отрезка по координатам точек его концов

Dot clipx();//отсечение отрезка осью X

Dot clipy();//отсечение отрезка осью Y

}; // Line

//пересечение отрезка (y-y2)(x1-x2)=(y1-y2)(x-x2) и прямой y=0 (ось X)

Dot Line::clipx() {

float dx = p2.x - p1.x; //разность координат концов отрезка по оси X

float dy = p1.y - p2.y; // разность координат концов отрезка по оси Y

float x0 = (p2.y \* dx)/dy + p2.x; //отсечение по X

return Dot(x0, 0.0); //возврат точки пересечения отрезка с осью X

} // clipx()

//пересечение отрезка (y-y2)(x1-x2)=(y1-y2)(x-x2) и прямой x=0 (ось Y)

Dot Line::clipy() {

float dx = p2.x - p1.x; //разность координат концов отрезка по оси X

float dy = p1.y - p2.y; // разность координат концов отрезка по оси Y

float y0 = (p2.x \* dy)/dx + p2.y; //возврат точки пересечения отрезка с осью Y

return Dot(0.0, y0); //возврат точки пересечения отрезка с осью Y

} // clipy()

//дружественная перегрузка оператора стандартного ввода точки

istream& operator>>(istream& input, Dot& p) {

char semicolon; ; //символ(;) разделения координат при вводе

input >> p.x >> semicolon >> p.y; //ввод пары координат точки

return input; //возврат ссылки на объект потока ввода

} // >>operator>>()

//дружественная перегрузка оператора стандартного вывода точки

ostream& operator<<(ostream& output, Dot& p) {

char semicolon = ; //символ(;) разделения координат при выводе

output << p.x << semicolon << p.y; //вывод пары координат точки

return output; //возврат ссылки на объект потока вывода

} // << operator<<()

int main() {//основная функция

Dot A, B, \_A, \_B, C; //объекты для пары задаваемых точек

float xa, ya, xb, yb, \_x, \_y;

cout << "Input xa;ya xb;yb or exit" << endl; //запрос ввода координат пары точек

while(cin >> A >> B) {//цикл ввода и обработки точек

//поиск точки оси Х, ближайшей к заданным точкам А и В

ya = A.gety();

yb = B.gety();

//отражение т.А относительно оси Х, когда А и В обе выше или ниже оси Х

\_A = (ya \* yb) > 0 ? A.symy() : A;

Line lx(\_A, B); //построение отрезка\_АВ,пересекающего ось Х

C = lx.clipx(); //точка пересечения отрезка\_АВ с осью Х

cout << C << endl; //вывод координат полученной точки на оси Х

//поиск точки оси Y, ближайшей к заданным точкам А и В

xa = A.getx();

xb = B.getx();

\_x = (xa\*xb > 0) ? -xb : xb; //отражение т.А относительно оси

// Y, когда А и В обе выше или ниже оси Y

Line ly(xa, ya, \_x, yb); //построение отрезка\_АВ, пересекающего ось Y

C = ly.clipy();//точка пересечения отрезка\_АВ с осью Y

cout << C << endl; //вывод координат полученной точки на оси Y

cout << "Input xa;ya xb;yb or exit" << endl;

} // while

return(0); //завершение программы

} // main

**Варианты заданий для пятой лабораторной работы**

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления длины любых отрезков прямых линий. Отрезки должны задаваться парами координат своих концов на плоскости через поток стандартного ввода. Результат вычисления длины каждого заданного отрезка должен отображаться через поток стандартного вывода. Программная реализация этих вычислений должна быть основана на разработке контейнерного класса отрезка прямой. Его компонентные данные должны включать подобъекты класса точки для хранения координат концов отрезка, а компонентный метод должен обеспечивать вычисление длины отрезка по ним. Конструкторы классов отрезка и точки должны использовать списки инициализации своих компонентных данных.

K1

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для деления на 2 равные части отрезков прямых линий. Отрезки должны задаваться парами координат своих концов на плоскости через поток стандартного ввода. Результат вычисления координат центра каждого заданного отрезка должен отображаться через поток стандартного вывода. Программная реализация этих вычислений должна быть основана на разработке контейнерного класса отрезка прямой линии. Его компонентные данные должны включать подобъекты класса точки для хранения координат концов отрезка, а компонентный метод должен определять середину отрезка по ним. Конструкторы классов отрезка и точки должны использовать списки инициализации своих компонентных данных.

K2

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для деления отрезков прямых линий на 2 части в пропорциях золотого сечения. Отрезки должны задаваться парами координат своих концов на плоскости в строках потока стандартного ввода. Результат вычисления координат золотого сечения каждого заданного отрезка должен отображаться через поток стандартного вывода. Программная реализация этих вычислений должна быть основана на разработке контейнерного класса отрезка прямой линии. Его компонентные данные должны включать подобъекты класса точки для хранения координат концов отрезка, а компонентный метод должен строить золотое сечение отрезка по ним. Конструкторы классов отрезка и точки должны использовать списки инициализации своих компонентных данных.

K3

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для деления отрезков прямых линий на 2 части в отношении, указанном аргументом командной строки ее вызова. Отрезки должны задаваться парами координат своих концов на плоскости через поток стандартного ввода. Координаты точки деления каждого заданного отрезка должны отображаться через поток стандартного вывода. Программная реализация этих вычислений должна быть основана на разработке контейнерного класса отрезка прямой линии. Его компонентные данные должны включать подобъекты класса точки для хранения координат концов отрезка, а компонентный метод должен делить отрезок в заданном отношении. Конструкторы классов отрезка и точки должны использовать списки инициализации своих компонентных данных.

K4

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска точек отрезков прямых линий на плоскости XY, которые имеют обе целочисленные координаты. Отрезки должны задаваться парами координат своих концов в строках потока стандартного ввода. Пары целочисленных координат точек каждого заданного отрезка должны отображаться через поток стандартного вывода. Программная реализация поиска должна основываться на разработке контейнерного класса отрезка. Его компонентные данные должны включать подобъекты класса точки для координат концов отрезка, а компонентный метод должен перечислять все целочисленные точки отрезка в порядке увеличения координаты X. Конструкторы классов отрезка и точки должны использовать инициализаторы своих компонентных данных.

K5

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска точек отрезков прямых линий на плоскости XY, которые имеют обе целочисленные координаты. Отрезки должны задаваться парами координат своих концов в строках потока стандартного ввода. Пары целочисленных координат точек каждого заданного отрезка должны отображаться через поток стандартного вывода. Программная реализация поиска должна основываться на разработке контейнерного класса отрезка. Его компонентные данные должны включать подобъекты класса точки для координат концов отрезка, а компонентный метод должен перечислять все целочисленные точки отрезка в порядке увеличения координаты Y. Конструкторы классов отрезка и точки должны использовать инициализаторы своих компонентных данных.

K6

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска точек отрезков прямых линий на плоскости XY, которые имеют обе целочисленные координаты. Отрезки должны задаваться парами координат своих концов в строках потока стандартного ввода. Пары целочисленных координат точек каждого заданного отрезка должны отображаться через поток стандартного вывода. Программная реализация поиска должна основываться на разработке контейнерного класса отрезка. Его компонентные данные должны включать подобъекты класса точки для координат концов отрезка, а компонентный метод должен перечислять все целочисленные точки отрезка в порядке уменьшения координаты Y. Конструкторы классов отрезка и точки должны использовать инициализаторы своих компонентных данных.

K7

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска точек отрезков прямых линий на плоскости XY, которые имеют обе целочисленные координаты. Отрезки должны задаваться парами координат своих концов в строках потока стандартного ввода. Пары целочисленных координат точек каждого заданного отрезка должны отображаться через поток стандартного вывода. Программная реализация поиска должна основываться на разработке контейнерного класса отрезка. Его компонентные данные должны включать подобъекты класса точки для координат концов отрезка, а компонентный метод должен перечислять все целочисленные точки отрезка в порядке уменьшения координаты X. Конструкторы классов отрезка и точки должны использовать инициализаторы своих компонентных данных.

K8

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления следа на вертикальной оси координат и коэффициента углового наклона прямых линий. Каждая прямая линия должна задаваться парами координат двух своих точек в строках потока стандартного ввода. Результаты вычислений наклона и следа каждой прямой должны отображаться через поток стандартного вывода. Программная реализация этих вычислений должна быть основана на разработке контейнерного класса прямой линии. Его компонентные данные должны быть подобъекты класса точки для координат двух точек прямой, а компонентные методы должны вычислять значения ее наклона и следа по ним. Конструкторы классов прямой линии и точки должны использовать списки инициализации своих компонентных данных.

K9

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления значений следа прямых линий на осях координат. Каждая прямая линия должна задаваться парами координат двух своих точек на плоскости через поток стандартного ввода. Результаты вычислений следов каждой прямой на осях координат должны отображаться через поток стандартного вывода. Программная реализация вычислений должна быть основана на разработке контейнерного класса прямой линии. Его компонентные данные должны включать подобъекты класса точки для хранения координат двух точек прямой, а компонентные методы должны вычислять значения ее следов на осях координат. Конструкторы классов прямой линии и точки должны использовать списки инициализации своих компонентных данных.

K10

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления коэффициентов уравнения AX+BY+C=0 для прямых линий, заданных координатами двух своих точек на плоскости XY. Координаты пар точек должны передаваться программе через поток стандартного ввода. Записи уравнений прямых по ним должны отображаться в строках потока стандартного вывода. Программная реализация вычислений их коэффициентов должна быть основана на разработке контейнерного класса прямой линии. Его компоненты-данные должны включать подобъекты класса точки для координат двух точек прямой, а компонентные методы должны вычислять коэффициенты для ее уравнения. Конструкторы классов прямой линии и точки должны использовать списки инициализации своих компонентных данных.

K11

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления расстояний любых прямых линий на плоскости от начала координат. Каждая прямая линия должна задаваться парами координат двух своих точек через поток стандартного ввода. Результирующая величина ее расстояния от начала координат должна отображаться строкой потока стандартного ввода. Программная реализация вычислений этих расстояний должна быть основана на разработке контейнерного класса прямой линии. Его компоненты-данные должны включать подобъекты класса точки для координат двух точек прямой, а компонентный метод должен вычислять по ним ее расстояние от начала координат. Конструкторы классов прямой линии и точки должны использовать списки инициализации своих компонентных данных.

K12

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления периметров треугольников, которые задаются координатами вершин на плоскости. Координаты вершин треугольников должны передаваться программе строками потока стандартного ввода. Значения периметров соответствующих треугольников должны отображаться через поток стандартного вывода. Программная реализация вычислений должна быть основана на разработке контейнерного класса треугольника. Его компонентные данные должны включать подобъекты класса точки для координат вершин треугольника, а компонентный метод должен вычислять по ним периметр треугольника. Конструкторы классов треугольника и точки должны использовать списки инициализации своих компонентных данных.

K13

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления площадей треугольников, которые задаются координатами вершин на плоскости. Координаты вершин треугольников должны передаваться программе строками потока стандартного ввода. Значения площади соответствующих треугольников должны отображаться через поток стандартного вывода. Программная реализация вычислений должна быть основана на разработке контейнерного класса треугольника. Его компонентные данные должны включать подобъекты класса точки для координат вершин треугольника, а компонентный метод должен вычислять по ним площадь треугольника. Конструкторы классов треугольника и точки должны использовать списки инициализации своих компонентных данных.

К14

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления радиуса окружности, вписанной в любой треугольник, заданный координатами своих вершин на плоскости. Эти координаты должны передаваться программе строками потока стандартного ввода. Значения радиуса вписанной окружности для каждого заданного треугольника должно отображаться в потоке стандартного вывода. Программная реализация вычислений должна быть основана на разработке контейнерного класса треугольника. Его компоненты-данные должны включать подобъекты класса точки для координат вершин треугольника, а его компонентный метод должен вычислять радиус вписанной окружности. Конструкторы классов треугольника и точки должны использовать списки инициализации компонентных данных.

К15

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления радиуса окружности, в которую вписывается треугольник, заданный координатами своих вершин на плоскости. Эти координаты должны передаваться программе строками потока стандартного ввода. Значения радиуса описанной окружности для каждого заданного треугольника должно отображаться в потоке стандартного вывода. Программная реализация вычислений должна быть основана на разработке контейнерного класса треугольника. Его компоненты-данные должны включать подобъекты класса точки для координат вершин треугольника, а его компонентный метод должен вычислять радиус описанной окружности. Конструкторы классов треугольника и точки должны использовать списки инициализации компонентных данных.

К16

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу вычисления длины медиан для любых треугольников, которые задаются координатами вершин на плоскости. Координаты вершин треугольников должны передаваться программе строками потока стандартного ввода. Длины медиан для каждого заданного треугольника должны отображаться через поток стандартного вывода. Программная реализация этих вычислений должна быть основана на разработке контейнерного класса треугольника. Его компоненты-данные должны включать подобъекты класса точки для хранения координат вершин треугольника, а компоненты-методы должны вычислять длину медиан треугольника. Конструкторы классов треугольника и точки должны использовать списки инициализации компонентных данных.

К17

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления длины биссектрис углов треугольников, которые задаются координатами вершин на плоскости. Координаты вершин треугольников должны передаваться программе строками потока стандартного ввода. Длины биссектрис углов заданных треугольников должны отображаться через поток стандартного вывода. Программная реализация этих вычислений должна быть основана на разработке контейнерного класса треугольника. Его компоненты-данные должны включать подобъекты класса точки для хранения координат вершин треугольника, а компоненты-методы должны вычислять длины биссектрис углов. Конструкторы классов треугольника и точки должны использовать списки инициализации компонентных данных.

К18

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для вычисления длины высот в любых треугольниках, которые задаются координатами вершин на плоскости. Координаты вершин треугольников должны передаваться программе строками потока стандартного ввода. Длины высот для каждого заданного треугольника должны отображаться через поток стандартного вывода. Программная реализация этих вычислений должна быть основана на разработке контейнерного класса треугольника. Его компоненты-данные должны включать подобъекты класса точки для хранения координат вершин треугольника, а компоненты-методы должны вычислять длину высот треугольника. Конструкторы классов треугольника и точки должны использовать списки инициализации компонентных данных.

К19

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для определения углов треугольников, которые задаются координатами вершин на плоскости. Координаты вершин треугольников должны передаваться программе строками потока стандартного ввода. Значения углов для каждого заданного треугольника должны отображаться через поток стандартного вывода. Программная реализация вычислений должна быть основана на разработке контейнерного класса треугольника. Его компоненты-данные должны включать подобъекты класса точки для хранения координат вершин треугольника, а компонентные методы должны определять по ним значения углов треугольника. Конструкторы классов треугольника и точки должны использовать списки инициализации компонентных данных.

К20

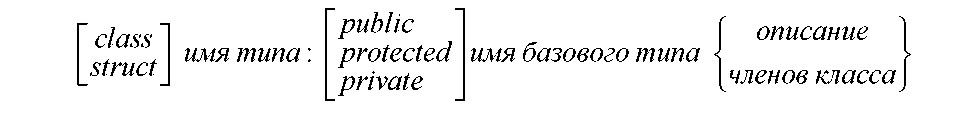
--------------------------------------------------------------------------------------------------------------------

# 9. Одиночное наследование

## 9.1. Правила наследования

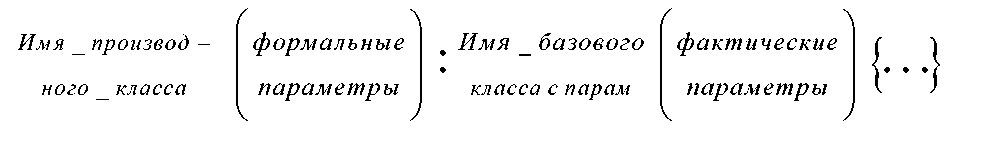
Наследование является одним из трех основных механизмов языка ООП [5]. В результате использования механизма наследования осуществляется формирование иерархических связей между описываемыми типами. Тип-наследник уточняет базовый тип.

Прототип объявления типа-наследника:



При наследовании наследуются не только информационные члены, но и методы класса [5,8-9]. Однако конструкторы, деструктор и операция присваивания не наследуются.

Единственный способ использования конструктора базового класса – **список инициализации**. Он записывается при описании конструктора производного класса:



При создании объекта производного типа сначала будет вызван конструктор базового типа. При этом если конструктору базового типа нужны параметры, то **его необходимо вызывать явно в списке инициализации**. Затем будет вызван конструктор производного класса.

Деструкторы вызываются в обратном порядке. При разрушении объекта производного типа сначала будет вызван деструктор этого типа, а затем деструктор базового типа.

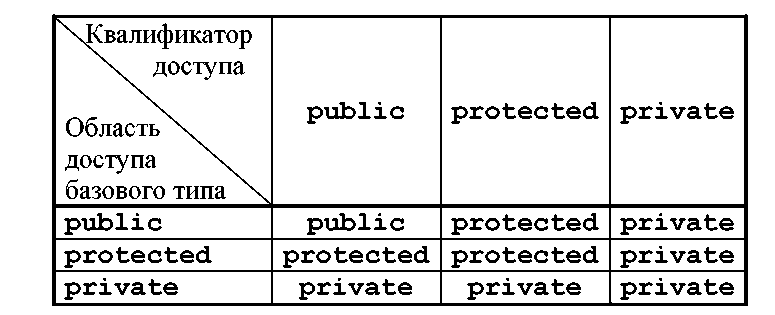
Допустимо присвоение объекту базового типа объекта производного типа. При этом объекту базового типа будет присвоена та часть объекта производного типа, которая структурно совпадает с базовым типом.

## 9.2. Правила видимости при наследовании

Наследование свойств и поведения могут контролироваться с помощью **квалификаторов доступа**, задаваемых при наследовании: ***public***, ***protected***, ***private***. Названия квалификаторов доступа совпадают с названиями ранее описанных областей доступа в классах и структурах. Квалификаторы доступа ограничивают видимость полностью или частично для полностью или частично открытых членов. Закрытые члены всегда остаются закрытыми. При наследовании можно уменьшить видимость членов, но не расширить их видимость.

Информация о виде доступа для членов базового типа в типе-наследнике представлена в таблице 1. Закрытый вид доступа в типе-наследнике для закрытых членов базового типа имеет особый статус, описываемый далее.

Таблица 1. Вид доступа в типе-наследнике



Если не указан тип наследования, то тип наследования по умолчанию определяется описанием типа наследника. Если тип-наследник описывается классом, то по умолчанию будет установлен закрытый (***private***) тип наследования, если же это структура, то наследование по умолчанию будет открытым (***public***).

При необходимости открытого наследования членов базового типа когда тип-наследник описывается с использованием класса, следует явно указывать квалификатор ***public***.

Защищенный вид доступа (***protected***) означает, что члены базового типа в типе-наследнике доступны только для методов своего (базового) типа, а также для методов производного типа. Во всех остальных случаях они ведут себя так же, как члены с закрытым видом доступа (***private***).

Ограничение видимости при наследовании ограничивает манипуляции с членами базового типа только в объектах типа-наследника и его потомках. Поэтому при преобразовании указателя типа-наследника к указателю на объекты базового типа работа с объектом осуществляется в соответствии с правилами видимости для базового класса.

## 9.3. Закрытое (private) наследование

Закрытые члены базового класса недоступны напрямую с использованием дополнительных методов класса-наследника (при любом способе наследования). Работа внутри класса-наследника с закрытыми членами базового класса возможна только с использованием открытых и защищенных методов базового класса.

Закрытые и защищенные методы недоступны для манипулирования с объектом вне класса. Они могут использоваться как подпрограммы другими методами класса.

При закрытом наследовании открытые и защищенные члены базового класса (любые) доступны только внутри производного класса и недоступны извне (через объекты производного класса), как и его собственные закрытые члены.

Таким образом, приведенная таблица показывает вид доступа для членов в типе наследнике для типа наследника следующего уровня. Но для текущего типа-наследника доступность зависит от вида доступа в базовом типе.

Закрытое наследование целесообразно в том случае, когда меняется сущность нового объекта.

Таким образом, закрытое наследование несколько напоминает **композицию** объектов, когда подобъект находится в закрытой области. Все же необходимо помнить, что наследование – это совсем другая концепция ассоциирования классов, по многим своим свойствам отличная от агрегации, даже в ее строгом варианте (композиции) [5-9].

# 10. Практическое изучение одиночного наследования с помощью использования производных классов

## 10.1. Подготовка к лабораторной работе № 6

**Программирование двусвязных списков**

Очень часто для хранения и обработки различных наборов данных удобно использовать 2-связные (двунаправленные) линейные списки, записи которых связаны посредством пары указателей, хранимых в адресных полях записей списка [5-6]. Логическую структуру 2-связного списка иллюстрирует диаграмма, представленная на рисунке 4.

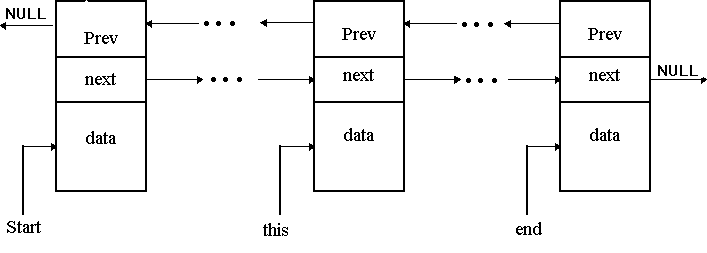


Рисунок 4 – Диаграмма 2-связного списка

В каждой записи поле ***prev*** содержит адрес предыдущей записи, поле ***next*** – адрес последующей записи, а ***data*** обозначает информационные поля. Для доступа к списку могут быть использованы адреса начальной (***start***), конечной (***end***) и текущей (***this***) записей списка. Индикатором начальной и конечной записей являются нулевые (NULL) значения адресных полей ***Start*** и ***end***, соответственно. Любая функциональная обработка списка строится на основе процедур модификации и просмотра. Процедуры модификации должны обеспечивать вставки и исключение записей списка. Частным случаем процедур вставки и исключения являются процедуры добавить и удалить начальную или конечную запись списка. На рисунке 5 представлена диаграмма, иллюстрирующая процедуру вставки новой записи Z после записи Y (или перед записью X).

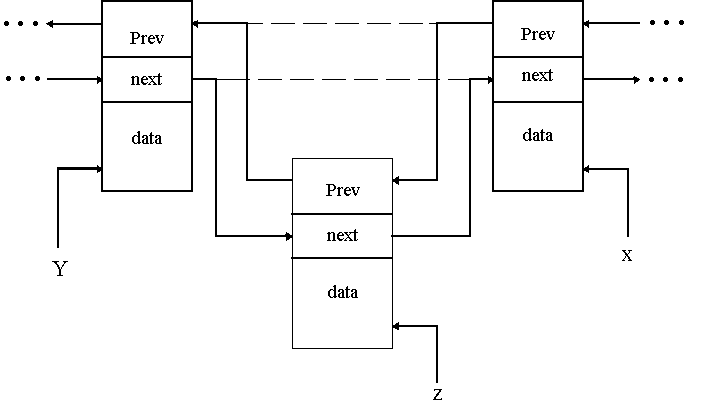


Рисунок 5 – Процедура вставки в 2-связном списке

Следующая диаграмма иллюстрирует процедуру исключения текущей записи ***this*** (рисунок 6).

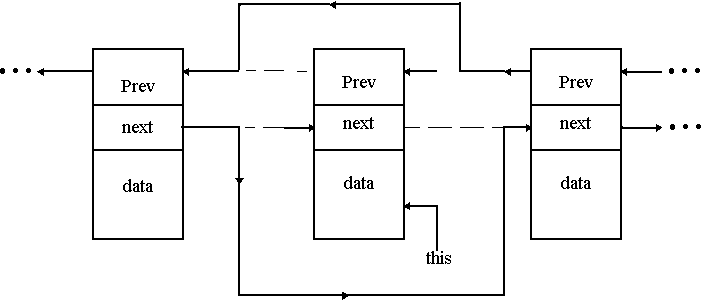


Рисунок 6 – Процедура исключения в 2-связном списке

Процедуры просмотра списка должны обеспечивать смещение указателя текущей записи (***this***) на требуемое число позиций в направлении начала или конца списка, как показано на следующей диаграмме (рисунок 7).

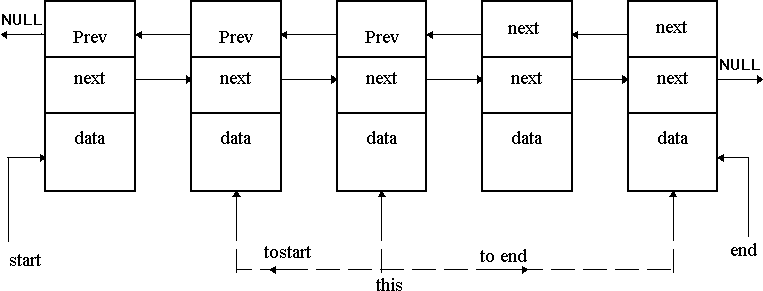


Рисунок 7 – Процедура просмотра списка

Частным случаем смещения указателя текущей записи является переход к соседней предыдущей или последующей записи (смещение на 1 позицию), а также нефиксированный переход к начальной или конечной записям списка, который позволяет принудительно инициализировать указатели ***start*** и ***end*** после модификаций в начале и конце списка.

Для организации объектно-ориентированной обработки данных, реализованных на основе списковых структур, целесообразно построить базовый класс записей 2-связного списка (***class Dlink***).

Базовый класс ***Dlink*** должен включать универсальные компоненты. Компоненты-данные класса ***Dlink*** должны обеспечивать двунаправленную защищенную (***protected***) связь записей списка посредством 2-х адресных полей, которые содержат адреса предыдущей и последующей записей списка, соответственно. Универсальную обработку записей списка должны обеспечивать общедоступные (***public***) компонентные методы. Компоненты базового класса ***Dlink*** могут наследоваться в различных производных классах с информационными полями данных и методами их обработки с обращением к базовым методам обработки списка.

**Программирование символьных списков**

Разработать ООП для перестановки символов заданной строки в случайном порядке. Зерно случайной последовательности генератора псевдослучайных чисел должно передаваться через аргумент командной строки. Исходная строка должна передаваться программе через поток стандартного ввода. Результирующую строку образует последовательное перемещение символов из случайно выбранных позиций строки в ее начало или конец. После каждой такой перестановки полученная строка должна отображаться через поток стандартного вывода, а очередной случайно выбранный символ в ней указывается знаком ^. Число случайных перестановок определяется по длине строки. Разработка программы должна быть основана на использовании связного списка с реализацией операций просмотра, удаления и вставки его элементов. Их публичные методы должны определить базовый класс элемента абстрактного списка с защищенными адресными полями данных. Ему должен наследовать производный класс элемента списка символов с приватным полем кода символов и публичной перегрузкой базовых методов с адресным возвратом. Кроме того в нем должны быть определенны собственные публичные методы. Они должны обеспечивать стандартный вывод списка символов и адресацию их позиций по смещению от концов списка. В основной функции программы должна быть реализована необходимая функциональная обработка списка символов с использованием динамического распределения памяти для его элементов.

Декларация класса ***Dlink*** может быть специфицирована в заголовочном файле ***Dlink.h*** и имеет следующий вид

// базовый класс Dlink, содержащий основные методы обработки записей списка

#ifndef DLINK

#define DLINK

class Dlink {

protected:

Dlink\* next; // адрес следующей записи

Dlink\* prev; // адрес предыдущей записи

public:

Dlink(); // инициализация ссылок

Dlink\* append(Dlink\*); // добавить новую запись

Dlink\* incr(); // получить адрес следующей записи

Dlink\* decr(); // получить адрес предыдущей записи

void excluse(); // исключить запись

Dlink\* after(Dlink\*); // вставить запись после текущей

Dlink\* before(Dlink\*); // вставить запись перед текущей

Dlink\* tohead(void); // адрес начала списка

Dlink\* tohead(int); // сдвиг в направлении начала списка

Dlink\* totail(void); // адрес конца списка

Dlink\* totail(int); // сдвиг в направлении конца списка

}; // Dlink

#endif

Исходный код методов класса ***Dlink*** специфицируется в файле dlink.ccи имеет следующий вид:

#include <stdio.h>

#include "dlink.h"

// init linked references

Dlink::Dlink() {

next=prev=NULL;

}

// инициализация адресов записей

Dlink\* Dlink::incr() {

return(next);

}

// получить адрес предыдущей записи

Dlink\* Dlink::decr() {

return(prev);

}

// добавить новую запись

Dlink\* Dlink::append(Dlink\* p) {

p->next=this;

if(this != NULL)

prev = p;

return(p);

}

// исключить следующую запись

void Dlink::excluse() {

if(next != NULL)

next->prev=prev;

if(prev != NULL)

prev->next=next;

return;

}

// вставить запись после текущей

Dlink\* Dlink::after(Dlink\* p) {

p->next = next;

p->prev = this;

if(next != NULL)

next->prev = p;

next = p;

return(p->next);

}

// вставить запись перед текущей

Dlink\* Dlink::before(Dlink\* p) {

p->next = this;

p->prev = prev;

if(prev != NULL)

prev->next = p;

prev = p;

return(p->prev);

}

// получить адрес начала списка

Dlink\* Dlink::tohead() {

Dlink\* p = this;

Dlink\* q = NULL;

while(p != NULL) {

q = p;

p = p->prev;

} // while

return(q);

}

// перейти на n записей в направлении начала списка

Dlink\* Dlink::tohead(int n) {

Dlink\* p = this;

Dlink\* q = this;

int i = 0;

while(p != NULL) {

q = p;

if(i == n)

break;

p = p->prev;

i++;

}

return(q);

}

// получить адрес конца списка

Dlink\* Dlink::totail() {

Dlink\* p = this;

Dlink\* q = NULL;

while(p != NULL) {

q = p;

p = p->next;

} // while

return(q);

}

// перейти на n записей в направлении конца списка

Dlink\* Dlink::totail(int n) {

Dlink\* p = this;

Dlink\* q = this;

int i = 0;

while(p != NULL) {

q = p;

if(i == n)

break;

p = p->next;

i++;

} // while

return(q);

}

Компоненты базового класса ***Dlink*** могут наследоваться в различных производных классах. В данном случае методы класса ***Dlink*** будут наследоваться в производном классе двусвязного списка ***SymLink***.

// Случайная перестановка строки символов

#include < stdio.h>

#include <stdlib.h>

#include "dlink.h"

// Класс элемент списка символов, производный от базового

// класса двусвязного списка

class SymLink : public Dlink {

private:

unsigned char sym; // код символа

public:

// инициализации кода символа и базовых адресов

SymLink(unsigned char c) : Dlink(), sym(c) {};

// перегрузка базового метода incr

SymLink\* incr() {return((SymLink\*) Dlink::incr());};

// перегрузка базового метода deer

SymLink\* deer() {return((SymLink\*) Dlink::deer());};

SymLink\* seek(int); // смещение указателя на заданное число элементов списка

int print(); // печать элементов списка

}; // SymLink

// метод сдвига к началу или к концу списка на n элементов

// в зависимости от знака аргумента

SymLink\* SymLink::seek(int n) {

if(n > 0) // сдвиг к хвосту списка от текущего адреса базовы м методом totail

return((SymLink\* ) Dlink::totail(n));

if(n < 0) // сдвиг к началу списка от текущего адреса

return((SymLink\* ) Dlink::tohead(abs(n))); // базовым методом tohead

return(this); // нулевой сдвиг

} // Symlink::seek

// Печать и подсчет элементов от начала до конца списка

int SymLink::print() {

SymLink\* p = this; // текущий адрес символа списка

SymLink\* q; // адрес следующего символа списка

int n= 0; // счетчик символов

while(p != NULL){ // цикл просмотра до конца списка

putchar(p->sym); // печать текущего символа списка

q = p->incr();// получить адрес следующего символа списка

p = q; // перейти к следующему символу списка

n++;// увеличить счетчик числа символов

}// while

return(n); // возврат числа напечатанных символов

} // SymLink::print

// основная функция

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

unsigned seed =0; // зерно псевдо-случайной последовательности

int count = 0; // счетчик случайных перестановок символов списка

int length; // длина списка символов

int ch; // код входного символа

unsigned pos; // смещение символа от начала списка

int side; // индекс сторожевого символа списка

SymLink\* watch[2]; // адреса сторожевых символов в начале и конце списка

SymLink\* head; // адрес начала списка символов

SymLink\* tail; // адрес конца списка символов

SymLink\* p, q; // текущие адреса символов списка

// массив адресов методов вставки символов

Dlink\* (Dlink::\* insert[])(Dlink\*) = {&Dlink::after, SDlink::before};

if(argc >1) // получить зерно случайной последовательности символов

seed = atoi(argv[1]); // из аргумента командной строки вызова программы

// распределить сторожевые символы в начале и конце списка

watch[0] = head = new SymLink('\n'); // адресовать LF в начале списка

watch[1] = tail = new SymLink('\n'); / / адресовать LF в конце списка

tail ->before(head); // связка 2 сторожевых символов в пустой список

// распределить список символов из строки стандартного ввода

while((ch = getchar()) != '\n') { // ввод символа до LF

q = new SymLink(ch); // распределить элемент списка

tail->before(q); //вставить символ в конец списка перед концевым сторожем

} // while

// вывод исходного списка символов и подсчет его длины

if((length = head->print ( ) - 1) < 2)

count = length; // блокировать обработку пустого списка

// случайное перемешивание символов в списке

srand(seed); // установить зерно псевдо-случайной последовательности

while(count < length) { // цикл перемешивания

side = rand() % 2; // случайный выбор начала или конца списка

while((pos = rand() % length) == 0); // случайный выбор символа в списке

printf("%\*c\n", pos, '^'); // индикация выбора символа

q = head->seek(pos); // адресация выбранного символа

q->excluse(); // исключить выбранный символ из списка

(watch[side]->\*insert[side])(q); // вставить его в начало или конец списка

head->print();// печать списка после перестановки символа

count++; // увеличить счетчик перестановок символов

} // while

// удалить список символов

р = tail;// адресовать конец списка

while(р != NULL) { // цикл удаления списка от конца к началу

q = p->decr(); // фиксировать адрес предыдущего символа списка

p->excluse(); // исключить символ списка по текущему адресу

delete р; // освободить память исключенного символа списка

р = q; // адресовать предыдущий символ списка

} // while

return(length+1);// завершение программы возвратом длины списка

} // main

**Варианты заданий для шестой лабораторной работы**

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу (ООП), которая в заданной строке символов переставляет каждый символ с максимальным кодом непосредственно перед ближайшим к нему символом с минимальным кодом без изменения взаимного расположения остальных символов. Исходная символьная строка должна передаваться программе через поток стандартного ввода. Результирующая символьная строка должна отображаться через поток стандартного вывода. В обеих строках все символы с максимальным и минимальным кодом должны указывать знаки ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L01

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в заданной строке символов переставляет каждый символ с максимальным кодом непосредственно после ближайшего к нему символа с минимальным кодом без изменения взаимного расположения остальных символов. Исходная символьная строка должна передаваться программе через поток стандартного ввода. Результирующая символьная строка должна отображаться через поток стандартного вывода. В обеих строках все символы с максимальным и минимальным кодом должны указывать знаки ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L02

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в заданной строке символов переставляет каждый символ с минимальным кодом непосредственно перед ближайшим к нему символом с максимальным кодом без изменения взаимного расположения остальных символов. Исходная символьная строка должна передаваться программе через поток стандартного ввода. Результирующая символьная строка должна отображаться через поток стандартного вывода. В обеих строках все символы с максимальным и минимальным кодом должны указывать знаки ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L03

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в заданной строке символов переставляет каждый символ с минимальным максимальным кодом непосредственно после ближайшего к нему символа с кодом без изменения взаимного расположения остальных символов. Исходная символьная строка должна передаваться программе через поток стандартного ввода. Результирующая символьная строка должна отображаться через поток стандартного вывода. В обеих строках все символы с максимальным и минимальным кодом должны указывать знаки ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L04

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в каждой входной символьной строке переставляет символы с максимальным кодом в конец строки без изменения взаимного расположения остальных символов. Все входные символьные строки должны передаваться программе через поток стандартного ввода. Результирующие строки символов должны отображаться через поток стандартного вывода. Во всех входных и результирующих строках на каждый символ с максимальным кодом должен указывать знак ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L05

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в каждой входной символьной строке переставляет символы с минимальным кодом в конец строки без изменения взаимного расположения остальных символов. Все входные символьные строки должны передаваться программе через поток стандартного ввода. Результирующие строки символов должны отображаться через поток стандартного вывода. Во всех входных и результирующих строках на каждый символ с минимальным кодом должен указывать знак ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L06

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в каждой входной символьной строке переставляет символы с максимальным кодом в начало строки без изменения взаимного расположения остальных символов. Все входные символьные строки должны передаваться программе через поток стандартного ввода. Результирующие строки символов должны отображаться через поток стандартного вывода. Во всех входных и результирующих строках на каждый символ с максимальным кодом должен указывать знак ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L07

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в каждой входной символьной строке переставляет символы с минимальным кодом в начало строки без изменения взаимного расположения остальных символов. Все входные символьные строки должны передаваться программе через поток стандартного ввода. Результирующие строки символов должны отображаться через поток стандартного вывода. Во всех входных и результирующих строках на каждый символ с минимальным кодом должен указывать знак ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L08

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в каждой входной символьной строке переставляет символы с наименьшим кодом в середину строки без изменения взаимного расположения остальных символов. Все входные символьные строки должны передаваться программе через поток стандартного ввода. Результирующие строки символов должны отображаться через поток стандартного вывода. Во всех входных и результирующих строках на каждый символ с наименьшим кодом должен указывать знак ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L09

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в каждой входной символьной строке переставляет символы с наибольшим кодом в середину строки без изменения взаимного расположения остальных символов. Все входные символьные строки должны передаваться программе через поток стандартного ввода. Результирующие строки символов должны отображаться через поток стандартного вывода. Во всех входных и результирующих строках на каждый символ с наибольшим кодом должен указывать знак ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L10

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в любой заданной символьной строке переставляет первый символ с минимальным кодом симметрично позиции наиболее удаленного от него символа с максимальным кодом без изменения взаимного расположения всех остальных символов. Исходная строка символов должна передаваться программе через поток стандартного ввода. Результирующая строка символов должны отображаться через поток стандартного вывода. В обеих строках символы с минимальным и максимальным кодом должны указывать знаки ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L11

--------------------------------------------------------------------------------------------------------------------

Разработать ООП, которая в любой заданной символьной строке переставляет первый символ с максимальным кодом симметрично позиции наиболее удаленного от него символа с минимальным кодом без изменения взаимного расположения всех остальных символов. Исходная строка символов должна передаваться программе через поток стандартного ввода. Результирующая строка символов должны отображаться через поток стандартного вывода. В обеих строках символы с минимальным и максимальным кодом должны указывать знаки ^. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L12

--------------------------------------------------------------------------------------------------------------------

Разработать ООП для разделения любой заданной символьной строки на группы символов, коды которых заканчиваются одной цифрой в системе счисления по основанию 10. Входная символьная строка должны передаваться программе через поток стандартного ввода. Результат группировки символов должен отображаться строкой потока стандартного вывода, где все символы каждой группы расположены рядом, а под начальным символом каждой группы указана крайняя правая цифра его кода. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L13

--------------------------------------------------------------------------------------------------------------------

Разработать ООП для разделения любой заданной символьной строки на группы символов, коды которых заканчиваются одной цифрой в системе счисления по основанию 16. Входная символьная строка должны передаваться программе через поток стандартного ввода. Результат группировки символов должен отображаться строкой потока стандартного вывода, где все символы каждой группы расположены рядом, а под начальным символом каждой группы указана крайняя правая цифра его кода. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L14

--------------------------------------------------------------------------------------------------------------------

Разработать ООП для разделения любой заданной символьной строки на группы символов, коды которых заканчиваются одной цифрой в системе счисления по основанию 8. Входная символьная строка должны передаваться программе через поток стандартного ввода. Результат группировки символов должен отображаться строкой потока стандартного вывода, где все символы каждой группы расположены рядом, а под начальным символом каждой группы указана крайняя правая цифра его кода. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L15

--------------------------------------------------------------------------------------------------------------------

Разработать ООП для разделения любой заданной символьной строки на 2 группы символов, бинарные коды которых заканчиваются нулевым и единичным разрядом. Входная строка символов должны передаваться программе через поток стандартного ввода. Результат группировки символов должен отображаться строкой потока стандартного вывода, где все символы каждой группы расположены рядом, а под начальным символом каждой группы указано значение младшего разряда его бинарного кода. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L16

--------------------------------------------------------------------------------------------------------------------

Разработать ООП для разделения любой заданной символьной строки на группы символов, коды которых начинаются одинаковой цифрой в системе счисления по основанию 10. Входная символьная строка должны передаваться программе через поток стандартного ввода. Результат группировки символов должен отображаться строкой потока стандартного вывода, где все символы каждой группы расположены рядом, а под начальным символом каждой группы указана первая цифра его кода. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L17

--------------------------------------------------------------------------------------------------------------------

Разработать ООП для разделения любой заданной символьной строки на группы символов, коды которых начинаются одинаковой цифрой в системе счисления по основанию 16. Входная символьная строка должны передаваться программе через поток стандартного ввода. Результат группировки символов должен отображаться строкой потока стандартного вывода, где все символы каждой группы расположены рядом, а под начальным символом каждой группы указана первая цифра его кода. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L18

--------------------------------------------------------------------------------------------------------------------

Разработать ООП для разделения любой заданной символьной строки на группы символов, коды которых начинаются с одинаковой цифры системы счисления по основанию 8. Входная символьная строка должны передаваться программе через поток стандартного ввода. Результат группировки символов должен отображаться строкой потока стандартного вывода, где все символы одной группы расположены рядом, а под начальным символом каждой группы указана первая цифра его кода. Разработка программы должна быть основана на использовании абстрактной структуры двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L19

--------------------------------------------------------------------------------------------------------------------

Разработать ООП для разделения любой заданной символьной строки на латинские буквы, десятичные цифры и знаки пунктуации. Заданная входная символьная строка должна передаваться программе через поток стандартного ввода. Результаты классифицирования ее символов должна отображать строка потока стандартного вывода, где все символы одного класса расположены рядом друг с другом. При этом первый символ каждого класса должен быть указан знаком ^. При разработке программы следует использовать абстрактную структуру двунаправленного связанного списка с реализацией операций просмотра, удаления и вставки его элементов с использованием механизма наследования, который определяет базовый класс элемента абстрактного списка и использует производный класс элемента списка символов.

L20

--------------------------------------------------------------------------------------------------------------------

# 11. Динамический полиморфизм, механизм виртуальных функций

В C++ введено понятие **виртуальных функций (методов)**. Механизм виртуальных методов заключается в том, что результат вызова виртуального метода с использованием указателя или ссылки зависит от типа объекта, на который указывает этот указатель [5-9].

Тип данных (класс), содержащий хотя бы одну виртуальную функцию, называется **полиморфным типом (классом)**, а объект этого типа – **полиморфным объектом**.

Таким образом, при вызове виртуальной функции через указатель на полиморфный объект осуществляется динамический выбор тела функции в зависимости от текущего тела объекта, а не от типа указателя. Тело функции в таком случае выбирается на этапе выполнения, а не компиляции. В этом и проявляется **динамический полиморфизм.**

В языке C++ виртуальные методы классов существуют наряду с невиртуальными методами. В некоторых объектно-ориентированных языках программирования, например, в языке Java, все методы в иерархиях классов являются виртуальными.

Виртуальная функция объявляется описателем ***virtual***. Во всех производных классах наследуемая виртуальная функция остается таковой (виртуальной). Таким образом, все типы-наследники полиморфного типа являются полиморфными типами.

## 11.1. Виртуальные деструкторы

Виртуальный деструктор – важная часть аппарата динамического полиморфизма. Если указатель типа базового класса указывает на объект производного класса, то при удалении объекта с использованием данного указателя в случае невиртуальности деструкторов сработает деструктор того типа, который был использован при объявлении указателя. При описании конструкторов и деструкторов уже было указано [5,8-9], что при удалении объекта во вторую очередь срабатывает деструктор базового типа, удаляя информационные члены базового типа, унаследованные в типе-наследнике, а сначала срабатывает деструктор текущего объекта, удаляя дополнительные члены типа-наследника. Таким образом, деструктор базового типа применяется к объектам производного типа. Но при его локальном срабатывании не будут удалены дополнительные члены типа-наследника.

Таким образом, при создании и удалении объектов производных типов с использованием указателей необходимо описывать деструкторы как виртуальные, если типы-наследники в своем составе имеют динамические структуры. Однако отметим следующее:

1. виртуальный деструктор необходим и для объекта без динамических структур в случае наличия динамических структур у типа-наследника, так как деструктор, автоматически генерируемый системой по умолчанию, является невиртуальным;
2. несмотря на то, что имя деструктора производного класса отличается от имени деструктора базового класса, достаточно объявления деструктора виртуальным только в базовом классе;
3. конструктор, в отличие от деструктора, нельзя описывать как виртуальный, так как всегда срабатывает конструктор именно того типа, который используется при создании объекта, и только после создания объекта его адрес передается для присвоения указателю.

## 11.2. Абстрактные классы. Чистые виртуальные функции

**Чистая виртуальная функция** – функция вида:

*virtual тип\_возвращаемого\_значения имя\_функции* (*формальные параметры*) = 0;

Такая форма записи функции означает, что данная функция (точнее – метод класса) не имеет тела, описывающего ее алгоритм.

**Абстрактный класс – это класс, содержащий хотя бы одну чистую виртуальную функцию.**

Нельзя создавать объекты на основе абстрактных классов, так как последние, имея в своем составе чистые виртуальные функции, не являются полноценными типами данных. Однако указатели на абстрактные классы создавать можно [5,8-9].

Несмотря на то что абстрактный класс не является полноценным типом, таблица виртуальных методов (ТВМ) для него создается. При этом в ТВМ перечисляются все виртуальные функции, в том числе и чистые виртуальные функции.

В классе-наследнике чистая виртуальная функция может быть переопределена обычной виртуальной функцией с соответствующей заменой пустого значения на адрес данной функции в ТВМ класса-наследника.

Класс-наследник абстрактного класса может также быть абстрактным классом, если в нем осталась (или была дополнительно введена) хотя бы одна чистая виртуальная функция.

# 12. Практическое изучение динамического полиморфизма с помощью использования абстрактного класса и чистой виртуальной функции

## 12.1. Подготовка к лабораторной работе № 7

**Разработка игровой программы**

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который вынудил противника взять последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются человек (Man) и компьютер (Pen), которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода размер кучи автоматически уменьшается на 1 предмет, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, чтобы остаток кучи после каждого его хода был равен (XN+X+1), где X обозначает наибольшее целое число, при котором этот остаток будет меньше текущего размера кучи. В любом случае результат каждого хода компьютера должен отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход осуществляется в данный момент. Игра должна завершаться диагностическим сообщением, которое показывает ее конечный результат. При разработке программы следует реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

#include <stdlib.h>

#include <iostream.h>

// Класс кучи предметов

class Heap {

private:

int size; // размер кучи

public:

Heap(int s) : size(s) {}; // инициализация кучи

// запрос размера и выбор предметов из кучи

int get(int n = 0) { size -= n; return(size); };

}; // Heap

// базовый класс игрока

class Gambler {

protected:

Heap\* heap; // адрес кучи

const int limit; // предел выборки за 1 ход

const char\* name; // имя игрока

public:

Gambler(Heap& h, int l) : limit(l) { heap = &h; }; // инициализация данных

virtual int move() = 0; // чистая виртуальная функция хода

int query(); // запрос хода

}; // Gambler

// класс человека

class Man : public Gambler {

public:

Man(Heap& h, int l, char\* n) : Gambler(h, l) { name = n; };

virtual int move(); // ход человека

}; // Man

// класс копьютера

class Pen : public Gambler {

public:

Pen(Heap& h, int l, char\* n) : Gambler(h, l) { name = n; };

virtual int move(); // ход копьютера

}; // Pen

// метод запроса хода

int Gambler::query() {

cout << "Heap = " << heap->get() << " " << name << " > ";

return(heap->get()); // возврат размера кучи

} // query()

// метод хода человека

int Man::move() {

int g; // число выбранных предметов из кучи

cin >> g; // ввод числа выбранных предметов

// анализ числа выбранных предметов

if((g < 1) || (g > (heap->get())) || (g > limit))

g = 1;

// выбор предметов из кучи с возвратом ее размера

return(heap->get(g));

} // Man::move()

// метод хода компьютера

int Pen::move() {

int rest = 0; // остаток кучи

int x = 0; // параметр выбора

int h; // текущий размер кучи

if((h = heap->get()) == 1)

h = heap->get(1);

// подбор выигрышного остатка кучи

while((rest = (x \*limit + x + 1)) < h)

x ++;

if((rest > h) || (rest == h))

rest = (x - 1)\*limit + x;

heap->get(h - rest); // выбор предметов из кучи

cout << h - rest << "\n"; // вывод хода компьютера

return(heap->get()); // возврат текущего размера кучи

} // Pen::move()

// основная функция

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

int i = 0; // индекс игрока

Heap h(atoi(argv[1])); // инициализация кучи

// инициализация и адресация игроков

Gambler\* g[] = { new Pen(h, atoi(argv[2]), "Pen"),

new Man(h, atoi(argv[2]), "Man")

};

// реализация партии игры

while(g[i]->query() != 0) {

g[i]->move();

if(++i > 1)

i = 0;

}

// идентификация победителя

cout << "Winner\n";

// освобождение памяти игроков

delete g[0];

delete g[1];

return(0); // корректное завершение программы

} // main()

Результат выполнения программы для кучи из S=11 предметов, из которой за 1 ход можно выбрать не более N=3 предмета

Heap=11 Pen>2

Heap=9 Man>1

Heap=8 Pen>3

Heap=5 Man>2

Heap=3 Pen>2

Heap=1 Man>1

Heap=0 Pen>Winner

**Варианты заданий для седьмой лабораторной работы**

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода сумма автоматически увеличивается на 1, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен увеличивать текущую сумму на 1. В любом случае приращение суммы на ходе компьютера должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V1

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода сумма автоматически увеличивается на 1, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен увеличивать текущую сумму на N. В любом случае приращение суммы на ходе компьютера должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V2

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода сумма автоматически увеличивается на N, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен увеличивать текущую сумму на 1. В любом случае приращение суммы на ходе компьютера должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V3

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода сумма автоматически увеличивается на N, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен увеличивать текущую сумму на N. В любом случае приращение суммы на ходе компьютера должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V4

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода игра должна аварийно завершаться соответствующим диагностическим сообщением. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен увеличивать текущую сумму на 1. В любом случае приращение суммы на ходе компьютера должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V5

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода игра должна аварийно завершаться соответствующим диагностическим сообщением. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен увеличивать текущую сумму на N. В любом случае приращение суммы на ходе компьютера должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V6

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода ему должна предоставляться возможность повторить ход. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен увеличивать текущую сумму на 1. В любом случае приращение суммы на ходе компьютера должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V7

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода ему должна предоставляться возможность повторить ход. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен увеличивать текущую сумму на N. В любом случае приращение суммы на ходе компьютера должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V8

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода сумма автоматически увеличивается на 1, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен повторить ход человека. В любом случае величина приращения суммы на ходе компьютера должна отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V9

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно наращивать сумму, достигнутую на предыдущем ходе противника, выбирая числовое приращение, которое не превышает заданной фиксированной величины N. Игра должна продолжаться, пока значение суммы, набранной обоими игроками, меньше заданной величины S>N. Победителем считается игрок, который на своем ходе смог первым набрать требуемую сумму M. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставлено право первого хода. Человек может делать любые допустимые ходы, сообщая величину приращения суммы через стандартный ввод. При любой ошибке ввода сумма автоматически увеличивается на N, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток до предельной суммы S после каждого его хода должен быть кратен (N+1). Если выигрышный ход сделать невозможно, компьютер должен повторить ход человека. В любом случае величина приращения суммы на ходе компьютера должна отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются значение суммы S, которую должны набрать игроки, и предельная величина N ее приращения за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать запрос, который отображает текущую величину суммы и идентифицирует игрока, чей ход ожидается в данный момент. Партию игры должно завершать диагностическое сообщение, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к значению набранной суммы от абстрактного базового класса.

V10

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода размер кучи автоматически уменьшается на 1, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать невозможно, компьютер должен взять из кучи 1 предмет. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V11

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода размер кучи автоматически уменьшается на 1, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать нельзя, компьютер должен взять из кучи N предметов. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V12

---------------------------------------------------------------------------------------------------------------------Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода размер кучи автоматически уменьшается на N, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать невозможно, компьютер должен взять из кучи 1 предмет. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V13

---------------------------------------------------------------------------------------------------------------------Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода размер кучи автоматически уменьшается на N, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать нельзя, компьютер должен взять из кучи N предметов. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V14

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода партия игры должна аварийно завершаться с соответствующей диагностикой. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать невозможно, компьютер должен взять из кучи 1 предмет. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V15

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода партия игры должна аварийно завершаться с соответствующей диагностикой. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать нельзя, компьютер должен взять из кучи N предметов. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V16

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода ему должна предоставляться возможность повторить ход. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать невозможно, компьютер должен взять из кучи 1 предмет. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход партии игры должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V17

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода ему должна предоставляться возможность повторить ход. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать нельзя, компьютер должен взять из кучи N предметов. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход партии игры должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V18

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода размер кучи автоматически уменьшается на 1, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать невозможно, компьютер должен повторить ход человека. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V19

---------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода размер кучи автоматически уменьшается на N, и ход передается компьютеру. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать нельзя, компьютер должен повторить ход человека. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

V20

---------------------------------------------------------------------------------------------------------------------

# 13. Классы потоков ввода-вывода языка С++

Библиотека потоков C++ предоставляет набор классов для управления вводом-выводом. Непосредственно в язык C++ (как и в язык Си) средства ввода-вывода не входят [5-9]. В программах часто используется препроцессорная директива

#include <iostream.h>

Назначение указанного в директиве заголовочного файла – связать компилируемую программу с одной из основных частей библиотеки ввода-вывода, построенной на основе механизма классов. Эта библиотека почти стандартная, так как включена практически во все компиляторы C++. Однако о стандарте здесь можно говорить только неформально: библиотека создана после появления языка. Она разрабатывалась в некотором смысле независимо от создания языка, не входит в формальное описание языка и написана на языке C++.

## 13.1. Потоки ввода-вывода

Библиотека обеспечивает программиста механизмами для извлечения данных из потоков и для помещения данных в потоки [5,8-9]. Каждый поток (за исключением строковых) ассоциируется при помощи операционной системы с определенным внешним устройством. При обмене с потоком используется вспомогательный участок памяти, называемый буфером потока. При вводе данных они сначала помещаются в буфер и только затем передаются выполняемой программе. При выводе данные заполняют буфер перед передачей внешнему устройству. Заполнение и очистку буферов операционная система выполняет без явного участия программиста, поэтому поток в прикладной программе можно рассматривать как последовательность байтов, не зависящую от конкретных устройств, с которыми ведется обмен данными.

Все потоки библиотеки последовательные, т.е. в каждый момент для потока определены позиции записи и (или) чтения, которые после обмена перемещаются по потоку на длину переданной порции данных.

В зависимости от реализуемого направления передачи данных потоки делят на три группы [5,8-9]:

* **входные**, из которых читается информация;
* **выходные**, в которые вводятся данные;
* **двунаправленные**, допускающие как чтение, так и запись.

В соответствии с особенностями обслуживаемого устройства потоки принято делить на следующие группы:

* **стандартные**, для передачи данных от клавиатуры к дисплею (во всех предыдущих программах использовались потоки данной группы);
* **файловые**, при размещении данных на внешнем носителе (например, диск, магнитная лента);
* **строковые**, позволяющие размещать данные потока в памяти (символьный массив или строка) и пользоваться при этом всеми средствами, предоставляемыми библиотекой потоков (например, форматный ввод-вывод данных).

## 13.2. Классы потоков C++

Библиотека ввода-вывода разработана средствами объектно-ориентированного программирования и является библиотекой классов. Основные преимущества такого подхода заключаются в следующем.

Во-первых, классам потоков доступен надежный механизм контроля типов передаваемых данных, основанный на перегрузке операций (что обеспечивает для каждого типа данных вызов соответствующей функции обработки). Таким образом, именно классами интерпретируются «сырые» последовательности байтов соответствующих потоков.

Во-вторых, классы потоков, благодаря полиморфизму, позволяют одним и тем же функциям работать с потоками различных типов, а библиотеке в целом – поддерживать единообразный интерфейс ввода-вывода. Например, интерфейс, используемый стандартным вводом-выводом, применим также к файловым и строковым потокам.

В-третьих, библиотека ввода-вывода расширяема. Это значит, что, с одной стороны, классы библиотеки могут работать с новыми типами данных, определенными программистом, используя упомянутый механизм перегрузки операций; а с другой – программист имеет возможность создавать собственные классы потоков на основе наследования свойств классов библиотеки.

Тем не менее, для прикладного программиста обычно достаточно лишь сведений об основных классах, показанных на рисунке 8.

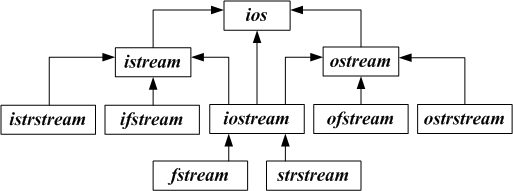


Рисунок 8 – Иерархия потоков ввода-вывода

Перечислим основные классы потоков ввода-вывода:

* ***ios*** — базовый потоковый класс;
* ***strstreambase*** — базовый класс строковых потоков;
* ***fstreambase*** — базовый класс файловых потоков;
* ***istream*** — класс входных потоков;
* ***ostream*** — класс выходных потоков;
* ***iostream*** — класс двунаправленных потоков ввода-вывода;
* ***ifstream*** — класс входных файловых потоков;
* ***ofstream*** — класс выходных файловых потоков;
* ***fstream*** — класс двунаправленных файловых потоков;
* ***istrstream*** — класс входных строковых потоков;
* ***ostrstream*** — класс выходных строковых потоков;
* ***strstream*** — класс двунаправленных строковых потоков.

Класс ***ios*** является базовым для всех классов потоков и, таким образом, содержит все общие средства потоков. Например, при помощи методов и данных класса ***ios*** осуществляется управление процессом передачи данных из буфера и в буфер. При выполнении этих действий необходимы, например, сведения о нужном основании счисления, о точности представления вещественных чисел и т.д. Класс ***ios*** содержит эти сведения, т.е. данные, относящиеся к состояниям потоков, и методы, позволяющие менять эти свойства.

Буферизация данных потоков реализована при помощи следующего механизма: класс ***ios***, а следовательно, и производные классы содержат указатель на объект класса ***streambuf*** библиотеки базового класса, обеспечивающего свои производные классы общими методами для буферизации данных. В свою очередь его производные классы (***filebuf*** и ***strstreambuf***) обеспечивают взаимодействие создаваемых потоков с физическими устройствами. Данная параллельная иерархическая структура классов-буферов не показана на рисунке 8 по причине достаточно редкой необходимости обращения к методам и данным класса ***streambuf*** напрямую из прикладных программ. Вместо этого чаще происходит опосредованное их использование вызовом методов классов-наследников ***ios***.

Классы потоков, их данные и методы становятся видимыми и доступными в программе, если в нее включен соответствующий заголовочный файл:

* **iostream.h** —для классов ***ios***, ***istream***, ***ostream***, ***stream***;
* **fstream.h** —для классов ***fstreambase***, ***ifstream***, ***ofstream***, ***fstream***;
* **strstrem.h** — для классов ***strstreambase***, ***istrstream***, ***ostrstream***, ***strstream***.

Поскольку класс ***ios*** является базовым для остальных потоковых классов, то включение в текст программы любого из заголовочных файлов **fstream.h** или **strstrem.h** автоматически подключает к программе и файл **iostream.h**. Соответствующие проверки выполняются на этапе препроцессорной обработки.

## 13.3. Стандартные объекты-потоки

При классификации потоков по типу обслуживаемого устройства в особую группу были выделены стандартные потоки для обмена с клавиатурой и дисплеем. Библиотека обслуживает стандартные потоки путем определения в заголовочном файле **iostream.h** четырех специальных объектов:

* ***cin*** – объект класса ***istream***, связанный со стандартным буферизованным входным потоком (обычно – клавиатура консоли);
* ***cout*** – объект класса ***ostream***, связанный со стандартным буферизованным выходным потоком (обычно – дисплей консоли);
* ***сеrr*** – объект класса ***ostream***, связанный со стандартным небуферизованным выходным потоком (обычно – дисплей консоли), в который направляются сообщения об ошибках;
* ***clog*** – объект класса ***ostream***, связанный со стандартным буферизованным выходным потоком (обычно – дисплей консоли), в который с буферизацией направляются сообщения об ошибках.

## 13.4. Операции помещения и извлечения

Как следует из схемы иерархии библиотеки потоков, для ввода и вывода предусмотрены два основных класса: соответственно ***istream*** и ***ostream***. Класс ***ostream*** использует для вывода операцию левого сдвига (<<), перегрузив ее. Если эта операция применяется к объектам-потокам (в качестве первого операнда), то ее называют операцией **помещения** в поток:

cout << "Hello!";

Класс ***istream*** перегружает для реализации ввода операцию правого сдвига (>>), которую в таком контексте называют операцией **извлечения** из потока:

char name[100];

cin >> name; // Извлечение строки из потока клавиатуры.

Классы ***istream*** и ***ostream*** перегружают соответственно операции извлечения и помещения для **всех встроенных типов данных.** Такая перегрузка позволяет использовать единообразный синтаксис для ввода и вывода символов, строк, целых и вещественных чисел и в то же время реализует строгий контроль типов данных.

Выполнение операции >> заключается в преобразовании последовательности символов потока (внешнее представление данных) в значение типизированного объекта (его внутреннее представление в кодах ЭВМ). При этом библиотека надежно защищена от ошибок интерпретации этой последовательности из-за несоответствия типов данных.

При выполнении операции << осуществляется обратное преобразование: внутреннее представление типизированного значения трансформируется в последовательность символов потока.

Перегруженные операции << и >> над классами ***ostream*** и ***istream*** возвращают ссылку на тот объект-поток, к которому они применяются. Следовательно, к результату выполнения операции можно вновь применить ту же операцию, как и к самому объекту-потоку. Это позволяет последовательно соединять несколько операций в цепочки**,** что весьма рационально.

Записывая цепочки операций помещения в поток, не нужно забывать о приоритете операций (операция помещения имеет тот же приоритет, что и перегруженная ею операция сдвига).

При перегрузке операций помещения (извлечения) данных абстрактных типов чаще всего соответствующие оператор-функции объявляются дружественными. Такое объявление обеспечивает операции доступ к частным элементам данных класса для организации обмена данными:

class TClass

{ friend istreams operator»(istreamS, TClassS);

friend ostreams operator« (ostreams, TClass); };

## 13.5. Форматирование данных при обмене с потоками

Непосредственное применение операций помещения (извлечения) к стандартным потокам ***cout***, ***cin***, ***cerr***, ***clog*** для данных базовых типов приводит к использованию «умалчиваемых» форматов внешнего представления пересылаемых значений. Например, при выводе чисел каждое из них занимает столько позиций, сколько необходимо для его представления. Это не всегда удобно. Форматы представления выводимой информации и правила восприятия данных, вводимых из потока, программистом могут быть изменены. Библиотека потоков C++ предусматривает три способа управления форматом данных: использование флагов форматирования, вызов компонентных функций потока и применение манипуляторов. Все эти средства определены для базового потокового класса ***ios*** и наследуются всеми потоками библиотеки. Рассмотрим каждый из этих методов.

Флаги форматирования определяют данные перечислительного типа в поле ***x-flags*** класса **ios**:

enum { // флаги форматирования

skipws = 0x0001, // пропуск ведущих пробелов

left = 0x0002, // выравнивание влево

right = 0x0004, // выравнивание вправо

internal = 0x0008, // заполнение после знака

dec = 0x0010, // десятичное преобразование

oct = 0x0020, // восьмеричное преобразование

hex = 0x0040, // шестнадцатеричное преобраз.

showbase = 0x0080, // выводить признак сист. счисл.

showpoint = 0x0100, // выводить десятичную точку

uppercase = 0x0200, // верхний регистр 16-ных чисел

showpos = 0x0400, // выводить знак ‘+’

scientific = 0x0800, // нотация с плавающей точкой

fixed = 0x1000, // нотация с фиксирован, точкой

unitbuf = 0x2000, // очищать поток при помещении

stdio = 0x4000 // очищать потоки stdout, stderr

Кроме того, в классе ios определены следующие статические константы:

static const long basefield=( dec | oct | hex);

static const long adjusttield=( left | right | internal);

static const long floatfield=(scientific | fixed);

Флаги реализованы в виде отдельных фиксированных битов чисел типа ***long***, поэтому несколько флагов с помощью логических битовых выражений можно объединять, тем самым по-разному комбинируя свойства потока. Ниже перечислим флаги, которые реализуют основные свойства потока:

* ***skipws***: операция извлечения из потока игнорирует ведущие пробелы и эквивалентные им символы (знаки табуляции ‘\t’ и ‘\v’, символы перевода строки ‘\n’, возврата каретки ‘\r’ и перевода страницы ‘\f’); данный флаг установлен по умолчанию;
* ***left*:** данные при помещении в поток выравниваются по левой границе поля вывода;
* ***right*:** данные при помещении в поток выравниваются по правой границе (по умолчанию);
* ***internal***: знак числа (либо признак основания системы счисления — см. флаг showbase) выводится с левого края поля, а число выравнивается по правому краю, промежуток заполняется символами-заполнителями (по умолчанию — пробел, см. ниже компонент ios::x\_fill);
* ***dec*:** числа выводятся в десятичной системе счисления (флаг установлен по умолчанию);
* ***oct***: числа выводятся в восьмеричной системе счисления;
* ***hex***: числа выводятся в шестнадцатеричной системе счисления;
* ***showbase***: при выводе чисел добавляется признак основания системы счисления (0х для шестнадцатеричных чисел, 0 — для восьмеричных);
* ***showpoint***: при выводе вещественных чисел обязательно показывать десятичную точку и следующие за ней нули (даже для чисел с нулевой дробной частью);
* ***uppercase***:при выводе чисел используются буквы верхнего регистра (символ х и буквы abcdef для шестнадцатеричных цифр), указатель порядка Е — для чисел с плавающей точкой;
* ***showpos***: для положительных значений выводится знак числа (символ +);
* ***scientific***: для вещественных чисел используется представление в формате с плавающей десятичной точкой (научная нотация), т.е. с указанием порядка и мантиссы, имеющей одну ненулевую (значащую) цифру перед точкой (например, 1.2345е2);
* ***fixed***: для вещественных чисел используется представление в формате с фиксированной десятичной точкой, причем количество цифр дробной части определяется заданной по умолчанию точностью;
* ***unitbuf***: поток очищается (выгружается содержимое буфера) после каждой операции помещения.

Среди флагов форматирования можно выделить три битовых поля, каждое из которых объединяет исключающие друг друга флаги (в каждый момент времени может быть установлен только один из них):

* поле основания системы счисления, состоящее из флагов ***dec***, ***oct*** и ***hex***;
* поле способа выравнивания — ***left***, ***right*** и ***internal***;
* поле нотации вещественных чисел — ***scientific*** и ***fixed***.

Класс ***ios*** дает имена этим полям, описывая три статические константы как дизъюнкции соответствующих флагов — ***basefield***, ***adjustfield*** и ***floatfield***. Поскольку константы, а также перечисленные имена флагов описаны внутри класса ***ios***, то обращение к ним требует применения операции разрешения видимости, например, ***ios::basefield***, ***ios::skipws***.

Установку необходимых флагов ввода-вывода обеспечивают компонентные методы:

* ***long flags()*** возвращает текущие флаги форматирования потока;
* ***long flags(long)*** присваивает флагам значение, сообщаемое параметром, и возвращает прежнее значение флагов;
* ***long setf(long, long)*** присваивает флагам, биты которых установлены во втором параметре, значения соответствующих битов первого параметра и возвращает прежнее значение всех флагов (в качестве второго параметра удобно использовать имена полей ***basefield***, ***adjustfield*** и ***floatfield***);
* ***long setf(long)*** устанавливает флаги, биты которых установлены в параметре, и возвращает прежнее значение всех флагов;
* ***long unsetf(long)*** сбрасывает флаги, биты которых установлены в параметре, и возвращает прежнее значение всех флагов.

## 13.6. Форматирующие функции класса ios

Рассмотрим способ управления форматом при помощи вызова **компонентных функций** класса ***ios***. Эти функции перегружены, чтобы обеспечить возможность как чтения, так и установки управляющего атрибута. Перечислим основные из этих атрибутов.

* **Ширина поля** (по умолчанию равна нулю). Для выходного потока задает минимальную ширину поля вывода для одной операции помещения. Если количество помещенных символов меньше заданного, то выход дополняется символами заполнения в соответствии с флагами ***left***, ***right***, ***internal***. Если количество символов больше указанного, то значение ширины поля игнорируется; после каждого помещения данных в поток ширина поля принимает значение по умолчанию (выход не дополняется и не обрезается). Для входного потока задает максимальное количество извлекаемых символов.
* **Символ заполнения** (по умолчанию — пробел). При помещении в поток дополняет поле вывода до заданной ширины, в соответствии со значениями флагов ***left***, ***right*** и ***internal*** (заполняет незанятые позиции поля вывода).
* **Точность представления вещественных чисел** (по умолчанию равна шести цифрам). При установленных флагах ***scientific*** или ***fixed*** задаст максимальное количество цифр, выводимых после десятичной точки. При «сброшенных» флагах ***scientific*** и ***fixed*** определяет общее количество выводимых значащих цифр.

Прямое обращение к полям, хранящим текущие установки перечисленных атрибутов потока, из прикладной программы закрыто. Вместо этого необходимо использовать шесть публичных компонентных форматирующих функций: ***int width()***, ***int width(int)***, ***char fill()***, ***char fill(char)***, ***int precision()***, ***int precision(int)***. Каждая из этих функций возвращает текущее значение соответствующего атрибута и при наличии параметра устанавливает новое значение атрибута.

## 13.7. Управление форматированием с помощью манипуляторов

Последний из трех рассматриваемых способов управления форматированием данных – применение **манипуляторов.** Манипуляторами называются функции, которые можно включать в цепочку последовательных операций помещения и извлечения. Манипуляторы обеспечивают удобный способ управления флагами и модификации атрибутов форматирования потока (частично дублируя возможности, предоставляемые рассмотренными функциями-элементами класса ***ios***), хотя их применение не ограничивается модификациями формата ввода-вывода. При выполнении манипулятора никаких обменов данными с потоком не происходит, но манипулятор в качестве побочного эффекта изменяет состояние потока.

Манипуляторы библиотеки потоков C++ делятся на две группы: **простые** (не имеющие аргументов) и **параметризованные** (требующие спецификации аргументов). Простые манипуляторы описаны в файле **iostream.h**. Параметризованные требуют включения в текст программы заголовочного файла **iomanip.h**, содержащего их описания.

Перечислим простые манипуляторы:

* ***dec***, ***hex*** и ***oct*** для входных и выходных потоков устанавливают соответственно флаги десятичной (***ios::dec***), шестнадцатеричной (***ios::hex***) и восьмеричной (***ios::oct***) систем счисления;
* ***ws*** действует только при вводе (определен для входных потоков в классе ***istream***) и устанавливает флаг ***ios::skipws***, тем самым заставляя поток игнорировать ведущие пробельные символы;
* ***ends*** действует только при выводе (определен для выходных потоков в классе ostream) и обеспечивает помещение в поток нулевого признака конца строки ‘\0’;
* ***flush*** действует только при выводе и очищает выходной поток (выгружает содержимое его буфера, записывая данные на соответствующие физические устройства);
* ***endl*** действует только при выводе и помещает в выходной поток символ конца строки ‘\n’, после чего очищает поток, как и манипулятор ***flush***.

Использование манипуляторов ***dec***, ***hex*** и ***oct*** не требует дополнительных пояснений:

cout « 15 « oct « 15 « hex « 15; // Выводит 1517f

Манипулятор ***endl*** рекомендуется использовать при каждом выводе, который должен быть незамедлительно воспринят пользователем, например:

cout << "Ждите! Идет выполнение операции." << endl;

При отсутствии ***endl*** здесь нельзя гарантировать, что сообщение не останется в буфере потока cout до окончания выполнения операции. По этой же причине рекомендуется с помощью манипулятора ***flush*** «сбрасывать» буфер потока при выводе на экран подсказки до последующего ввода информации:

cout << "Имя файла: " << flush; cin >> fileName;

Формат декларации манипуляторов без параметров:

оstream & endl(ostream&); // вставляет конец строки и flush

ostream & ends(ostream&); // вставляет нулевой конец строки

ostream & flush(ostream&);// очищает ostream

ios & dec(ios&); // устанавливает десятичное основание

ios & hex (ios&); // шестнадцатеричное основание

ios & oct(ios&); // восьмеричное основание

istream & ws (istream&); // извлекает пробельные символы

Таким образом, оба класса ***istream*** и ***ostream*** перегружают соответственно операции извлечения и помещения для второго операнда — указателя на функцию-манипулятор.

Кроме использования стандартных манипуляторов программист может создавать собственные манипуляторы. Их аргумент и код возврата должен иметь тип ссылки на соответствующий потоковый класс.

Параметризованные манипуляторы выполняют над потоком действия, специфицированные указанным значением аргумента.

* ***setbase(int)*** устанавливает основание системы счисления, манипулируя флагами форматирования ***ios::dec***, ***ios::oct*** и ***ios::hex*** потока; значениями параметра могут быть 0, 8, 10 или 16. При использовании параметра 0 основание счисления при выводе выбирается десятичным, а при вводе нулевой параметр означает, что целые десятичные цифры должны обрабатываться по правилам стандарта ANSI языка Си.
* ***resetiosflags(long)*** для входных и выходных потоков сбрасывает флаги форматирования, биты которых установлены в переданном параметре.
* ***setiosflags(long)*** для входных и выходных потоков устанавливает флаги форматирования, биты которых установлены в переданном параметре.
* ***setfill(int)*** устанавливает код символа заполнения для ***ios::x\_fill*** в соответствии со значением параметра.
* ***setprecision(int)*** задает точность представления вещественных чисел (значение ***ios::precision***).
* ***setw(int)*** устанавливает ширину поля входных и выходных потоков (значение ios::width). Действие распространяется до первой операции обмена данными с потоком.

## 13.8. Неформатируемый обмен данными с потоками

Библиотека потоков C++ предусматривает различные функции для неформатируемого ввода и вывода. Эти функции, в качестве альтернативы форматирующим операциям помещения (извлечения), позволяют читать и записывать байты данных без всякой модификации. При использовании этих функций никакие установки флагов и атрибутов форматирования не влияют иа процесс обмена с потоком. Чаще всего необходимость в подобных средствах возникает при работе с файловыми потоками, связанными с двоичными (нетекстовыми) файлами.

Класс ***ostream*** (базовый для всех выходных потоков) содержит две перегруженные функции для записи в поток «сырых» данных:

ostream & put (char);

ostream & write (char \*, int);

Первая функция помещает в выходной поток, для которого она вызвана, одиночный символ (байт); вторая предназначена для вывода в поток указанного количества символов из буфера. Обе функции возвращают ссылку на объект-поток, для которого они выполняются, поэтому можно организовать цепочку вызовов.

Функции для чтения из потока «сырых» данных определены в классе ***istream*** (заголовочный файл **iostream.h**), где они декларируются следующим образом

// ввод произвольных данных в массив символов

istream & read(char\*, int);

// читать символы в массив до символа-ограничителя

istream & get(char\*, int, char = ‘\n’);

istream & getline(char\*, int, char = ‘\n’);

// извлечь 1 символ c возвратного кода

int get();

Условно эти функции можно разделить на три группы. Функция ***istream& read(char\*, int)*** предназначена для извлечения из потока указанного количества символов (второй параметр) и переноса их в буфер (первый параметр), который должен иметь размер, достаточный для размещения указанного количества байтов. Обе функции ***istream& get(char\*, int, char=‘\n’)*** и ***istream& getline(char\*, int, char=‘\n’)*** преназначены для чтения символов вплоть до ограничителя (обычно используются при вводе символьных строк); дополнительно к описанным параметрам функции ***read*** эти функции имеют третий параметр, специфицирующий символ-ограничитель (по умолчанию – ‘\n’). Чтение из потока в буфер продолжается до наступления одного из следующих трех событий: найден ограничитель, извлечено заданное количество символов или встретился конец файла (для файловых потоков). В первом случае ограничитель в буфер (в формируемую строку) не переносится, а вместо него автоматически добавляется «концевой» символ строки ‘\0’, при этом сам символ-ограничитель извлекается из потока лишь функцией ***getline***, в то время как ***get*** оставляет его в потоке.

Функция ***int get()*** предназначена для извлечения одиночного символа. Эта функция возвращают код символа либо (при пустом потоке) код конца файла EOF.

# 14. Практическое изучение потоковых классов с помощью использования методов управления потоками ввода-вывода

## 14.1. Подготовка к лабораторной работе № 8

**Программирование потоков ввода-вывода**

Разработать ООП для формирования дампа любого заданного файла. Именаисходного файла и результирующего файла дампа должны передаваться программе через аргументы командной строки ее вызова. Исходный файл может иметь произвольный текстовый или бинарный формат. Результирующий файл дампа должен иметь текстовую структуру, идентичную стандартному выводу команды od из OS UNIX с ключом – x, где 2-х байтовые слова исходного файла кодируются короткими целыми числами без знака, которые записаны в системе счисления по основанию 16. При разработке программы нужно использовать методы бесформатного ввода и форматного вывода файловых потоков.

Файл дампа 16 имеет текстовый формат символьных строк фиксированной длины. Каждая строка кодирует 8 двухбайтовых слов целыми числами, которые разделены пробелом. Каждое число записано в системе счисления по основанию 16 и дополнено лидирующими нулями до 4 цифр. В начале каждой строки дампа указано смещение ее первого слова от начала исходного файла. Смещение измеряется в байтах и записано в системе счисления по основанию 8. Запись смещения дополнена лидирующими нулями до 7 цифр. Последняя строка дампа содержит запись размера исходного файла в формате, аналогичном смещению.

**Искомый текст**:

od -x OS UNIX dump\n

**Файл дампа 16**:

0000000 646f 2d20 2078 534f 5520 494e 2055 7564 706d

0000020 706d 000a

// Word hex dump as UNIX od -x

#include <iostream>

#include <fstream>

#include <ios>

#include <iomanip>

using namespace std; // объявление пространства имен

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

unsigned short i=0; // код слова

unsigned offset = 0; // смещение слова

// проверка числа аргументов

if(argc < 2) {

cerr << "Usage: asod source-file target-file" << endl;

return(0);

} // if

// задать входной поток

ifstream input(argv[1]);

if(input == 0) {

cerr << argv[1] << ": Can't open to read" << endl;

return(-1);

}

// задать выходной поток

ofstream output;

output.open(argv[2]);

if(output == 0) {

if(!argv[2])

argv[2] = "target";

cerr << argv[2] << ": Can't open to write" << endl;

return(-2);

}

// Установить выравнивание и заполнение полей вывода дампа

// input.unsetf(ios::skipws);

output.setf(ios::right, ios::adjustfield);

output.fill('0');

// Вывод начального смещения

output << oct << setw(7) << offset;

// Цикл ввода-вывода

while(input.read((char \*)&i, 2)) { // чтение слова

offset += 2; // увеличить счетчик смещения

output << " " << hex << setw(4) << i; // вывод очередного слова

i = 0; // инициализация кода слова

if((offset % 16) == 0) // вывод смещения

output << endl << oct << setw(7) << offset;

} // while

// вывод последнего слова

if(input.gcount() == 1) {

output << " " << hex << setw(4) << i;

offset++;

} // if

// вывод последнего смещения

output << endl;

if(offset % 16)

output << oct << setw(7) << offset << endl;

// закрыть потоки ввода-вывода

input.close();

output.close();

return(0); // корректное завершение программы

} // main

**Варианты заданий для восьмой лабораторной работы**

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для измерения размера заданного файла в байтах, килобайтах и мегабайтах. Имя измеряемого файла должно передаваться программе через аргумент командной строки ее вызова. Результат измерения размера файла должен отображаться информационным сообщением в потоке протокола стандартной диагностики (сerr). Эта диагностика должна иметь формат записи суммы числа мегабайтов, килобайтов и байтов, которые в совокупности составляют размер файла (M + K + B). При разработке программы следует использовать методы позиционирования файловых потоков ввода без чтения данных из них.

F01

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для подсчета количества непустых строк любого заданного текстового файла. При этом пустыми считаются строки, где отсутствуют любые символы, кроме необязательных пробелов и табуляций. Имя текстового файла для подсчета числа непустых строк должно передаваться программе через аргумент командной строки ее вызова. Результат подсчета должен отображаться информационным сообщением в потоке протокола стандартной диагностики (сerr). При разработке программы необходимо использовать методы бесформатного ввода файловых потоков и исключить ее зависимость от длины строк измеряемого файла.

F02

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для подсчета количества слов в любом заданном текстовом файле. При этом словом считается любая последовательность букв без различия регистра, которая отделена от других фрагментов текста символами пробелов, табуляций или знаками пунктуации и перевода строки. Имя текстового файла для подсчета числа слов должно передаваться программе через аргумент командной строки ее вызова. Результат подсчета должен отображаться информационном сообщением в потоке протокола стандартной диагностики (сerr). При разработке программы следует использовать оператор форматного ввода файловых потоков.

F03

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу подсчета количества абзацев в любом заданном текстовом файле. При этом считается, что абзацы разделены с помощью пустых строк, которые не содержат других символов, кроме необязательных пробелов и табуляций. Имя текстового файла в указанном формате должно передаваться программе через аргумент командной строки ее вызова. Результат подсчета числа абзацев в нем должен отображаться информационным сообщением через поток протокола стандартной диагностики (сerr). При разработке программы следует использовать методы бесформатного ввода класса файловых потоков, а также исключить ее зависимость от длины абзаца.

F04

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для измерения общего числа единичных разрядов во всех байтах любого заданного файла. Его имя должно передаваться программе через аргумент командной строк. Результат измерения должно отображать информационное сообщение в потоке протокола стандартной диагностики (сerr). При разработке программы следует использовать методы бесформатного ввода символов классов файловых потоков. Кроме того, в программе должен быть реализован эффективный вычислительный алгоритм, который минимизирует число сравнений разрядов для каждого байта. При этом необходимо также исключить повторные измерения одинаковых байтов.

F05

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для определения размера самой длинной и короткой непустой строки любого заданного текстового файла. При измерении строк в них не должны учитываться лидирующие и хвостовые пробелы или символы табуляции. Имя текстового файла для указанных измерений должно передаваться программе через аргумент командной строки ее вызова. Результат измерения и сравнения длин строк должен быть отображен информационным сообщением через поток стандартной диагностики (сerr). При разработке программы необходимо использовать методы бесформатного ввода файловых потоков и исключить ее зависимость от длины строк измеряемого файла.

F06

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для поиска всех строк любого заданного текстового файла, длина которых превышает установленную максимальную величину. Имя файла и предельный допустимый размер его строк должны передаваться программе через аргументы командной строки ее вызова. Итоговый результат поиска длинных строк должен отображаться информационным сообщением в потоке протокола стандартной диагностики (сerr), где перечислены их длины и порядковые номера в заданном текстовом файле. При разработке программы необходимо использовать методы бесформатного ввода файловых потоков.

F07

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу поиска самого длинного слова в любом заданном текстовом файле. При этом под словом понимается любая последовательность символов, которая состоит из букв и (или) цифр. Имя файла должно передаваться программе через аргумент командной строки ее вызова. Если в заданном файле имеется несколько различных слов равной максимальной длины, то программа должна перечислять все такие слова. Все найденные слова и число повторений каждого из них должны отображаться через поток стандартного вывода (сout). При разработке программы следует использовать методы форматного ввода файловых потоков.

F08

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для поиска всех различных слов любого заданного текстового файла, длины которых превышают установленную максимальную величину. При этом под словом понимается любая последовательность символов, которая состоит из букв и (или) цифр. Имя текстового файла и предельная длина слова в нем должны передаваться программе через два аргумента командной строки ее вызова. Все найденные слова без повторений должны отображаться через поток стандартного вывода (сout). При разработке программы следует использовать методы форматного ввода файловых потоков и исключить какие-либо ограничения по числу обнаруженных слов.

F09

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для поиска в любом заданном текстовом файле абзаца, который состоит из максимального числа строк. При этом считается, что абзацы разделены пустыми строками, где нет других символов, кроме необязательных пробелов и табуляций. Имя текстового файла для поиска таких абзацев должно передаваться программе через аргумент командной строки ее вызова. Число строк и смещение по строкам максимального абзаца от начала файла должно быть отображено через поток стандартной диагностики (сerr). При разработке программы следует применить методы бесформатного ввода файловых потоков и исключить ее зависимость от размера абзацев файла.

F10

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для поиска по образцу любой заданной последовательности символов в строках текстового файле. Имя текстового файла и образец искомой последовательность символов должны передаваться программе через 2 аргумента командной строки ее вызова. При этом поиск образца должен осуществляться с точностью до регистра букв и разделителей в строках текстового файла. Количество обнаруженных вхождений заданного образца в файл должно отображаться информационным сообщение в потоке протокола стандартной диагностики (сerr). При разработке программы необходимо применить методы бесформатного ввода файловых потоков.

F11

--------------------------------------------------------------------------------------------------------------------

Необходимо разработать объектно-ориентированную программу, которая отображает в потоке стандартного вывода (сout) требуемое число начальных строк любых заданных текстовых файлов. Имена всех файлов и число их начальных строк для вывода должны передаваться программе аргументами командной строки ее вызова. При этом требуемое число начальных строк должно быть указано после специального ключа (-n). Если такой ключ отсутствует, то из всех заданных файлов должна отображаться первая строка. При разработке программы необходимо применить методы бесформатного ввода файловых потоков и исключить ее зависимость от длины строк заданных файлов.

F12

--------------------------------------------------------------------------------------------------------------------

Необходимо разработать объектно-ориентированную программу, которая отображает через поток стандартного вывода (сout) требуемое число концевых строк из любого заданного текстового файла. Имя файла и число его концевых строк для вывода должны передаваться программе через аргументы командной строки ее вызова. Если число концевых строк не указано, то по умолчанию должна отображаться только одна последняя строка заданного файла. При разработке программы необходимо применить методы бесформатного ввода и позиционирования файловых потоков. Следует также обеспечить независимость программы от максимальной длины строк файла.

F13

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для поиска кодов символов в любом диапазоне от 0 до 255, которые наиболее часто встречаются в заданном файле. Имя файла и диапазон кодов должны передаваться программе аргументами командной строки ее вызова. При этом диапазон кодов должен быть задан парой чисел, которые записаны через тире. Результат поиска должен отображаться информационным сообщением в потоке протокола стандартной диагностики (сerr), где указаны коды и число повторений всех наиболее часто используемых символов файла. При разработке программы необходимо применить методы бесформатного ввода символов файловых потоков.

F14

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу разбиения любого файла на заданное число частей равной длины. Имя файла и требуемое количество частей должны передаваться программе через аргументы командной строки ее вызова. Все части, полученные в после разбиения исходного файла, должны сохраняться в отдельных файлах текущего каталога. При этом базовые имена всех частей должны совпадать с базовым именем исходного файла. Полное имя файла каждой части должно иметь цифровой суффикс, который определяет ее номер в порядке разбиения исходного файла. При разработке программы следует применить методы бесформатного ввода-вывода файловых потоков.

F15

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу конкатенации любого заданного набора файлов. Имена всех сливаемых файлов должны передаваться программе через аргументы командной строки ее вызова. Заданные файлы должны объединяться в порядке перечисления их имен в командной строке вызова программы. Результат слияния заданных файлов должен быть отображен через поток стандартного вывода (сout). При этом содержимое каждого следующего файла должно отображаться после предыдущего без какого-либо промежутка. При разработке программы необходимо применить методы бесформатного ввода и вывода файловых потоков.

F16

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для исключения из заданного текстового файла всех отображаемых символов, номера позиций которых в его строках находятся в указанном диапазоне. Имя файла и диапазон позиций исключаемых символов строк должны передаваться программе через аргументы командной строки ее вызова. При этом диапазон позиций должен быть задан парой чисел, которые разделены тире. Когда правая позиция диапазона превышает длину строки, должны быть исключены все символы до конца строки. Результаты исключений должны отображаться через поток стандартного вывода (сout). При разработке программы следует применить методы бесформатного ввода файловых потоков.

F17

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для разбиения любого файла на части заданной фиксированной длины. Имя файла и требуемый размер части в байтах должны передаваться программе через аргументы командной строки ее вызова. Полученные в результате части исходного файла должны быть сохранены в отдельных файлах текущего каталога. Базовые имена этих частей должны совпадать с базовым именем исходного файла. Полное имя файла каждой части должно иметь цифровой суффикс, который определяет ее номер в порядке разбиения исходного файла. При разработке программы следует применить методы бесформатного ввода-вывода файловых потоков.

F18

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу исключения лидирующих и хвостовых пробелов или табуляций в строках любого заданного текстового файла. Имя этого файла должно передаваться программе через аргумент командной строки ее вызова. Когда имя файла для указанной текстовой обработки не задано, должны рассматриваться строки потока стандартного ввода (cin). В любом случае результат обработки текста должен отображаться строками потока стандартного вывода (сout). При разработке программы следует применить методы бесформатного ввода-вывода файловых и стандартных потоков, а также исключить ее зависимость от длины строк любого исходного текста.

F19

--------------------------------------------------------------------------------------------------------------------

Разработать объектно-ориентированную программу для исключения из любого текстового файла повторений пустых строк, чтобы из каждой группы пустых соседних строк оставалась только одна. При этом пустыми считаются строки, где отсутствуют любые другие символы, кроме необязательных пробелов и табуляций. Имя исходного текстового файла должно передаваться программе через аргумент командной строки ее вызова. Результирующий текст после исключений повторов пустых строк должен отображаться через поток стандартного вывода (сout). При разработке программы необходимо использовать методы бесформатного ввода файловых потоков и исключить ее зависимость от длины строк заданного файла.

F20

--------------------------------------------------------------------------------------------------------------------

# Литература

1. Standard for the C Programming Language ISO/IEC 9899, 1990.

2. Standard for the C++ Programming Language ISO/IEC 14882, 1998.

3. Архангельский А.Я. Компоненты общего назначения библиотеки C++ Builder 5. М.: Издательство БИНОМ, 2001. 416 с.

4. Бокс Д. Сущность технологии COM. Библиотека программиста. 4-е изд. СПБ.: Питер, 2001. 400 с.

5. Пол Айра. Объектно - ориентированное программирование на C++: пер. с англ. 2-е изд. СПБ.: Невский Диалект; М.: Издательство БИНОМ, 2001. 462 с. [Pohl Ira. Object-Oriented Programming Using C++. 2nd ed. Addison-Wesley, 1996. 576 p.].

6. Пратт Т., Зелковиц М. Языки программирования: разработка и реализация. 4-е изд. СПБ.: Питер, 2002. 688 с.

7. Страуструп Б. Язык программирования C++: пер. с англ / под ред. Н.Н. Мартынова. Специальное изд. М.: Бином, 2011. 1035 c. [Stroustrup B. The C++ Programming Language. Special ed. Addison-Wesley, 2000. 1029 p.].

8. Шилдт Г. Самоучитель C++. 3-е изд. СПБ.: БХВ-Петербург, 2002. 688 с. [Schildt H. Teach Yourself C++. 3d ed. McGraw-Hill, 1998. 768 p.].

9. Волосатова Т.М., Родионов С.В. Объектно-ориентированное программирование на С++. Режим доступа: http://bigor.bmstu.ru/?cnt/?doc=VU/base.cou (дата обращения 01.06.2013).

Оглавление

[Введение 3](#_Toc366240867)

[1. Основные принципы объектно-ориентированного программирования 4](#_Toc366240868)

[2. Классы и объекты в языке С++ 5](#_Toc366240869)

[2.1. Синтаксис описания класса 5](#_Toc366240870)

[2.2. Управление доступом к членам класса 5](#_Toc366240871)

[2.3. Класс как область видимости 6](#_Toc366240872)

[2.4. Объявление и определение методов класса. Спецификатор inline 7](#_Toc366240873)

[2.5. Указатель this 7](#_Toc366240874)

[3. Конструкторы и деструкторы 8](#_Toc366240875)

[3.1. Конструктор умолчания 9](#_Toc366240876)

[3.2. Конструктор преобразования и конструкторы с двумя и более параметрами 9](#_Toc366240877)

[3.3. Конструктор копирования-инициализации 10](#_Toc366240878)

[3.4. Работа с динамической памятью 11](#_Toc366240879)

[4. Практическое изучение первого принципа ООП – механизма инкапсуляции 12](#_Toc366240880)

[4.1. Подготовка к лабораторной работе № 1 12](#_Toc366240881)

[4.2. Подготовка к лабораторной работе № 2 23](#_Toc366240882)

[5. Статический полиморфизм 36](#_Toc366240883)

[5.1. Перегрузка бинарных операций 37](#_Toc366240884)

[5.2. Перегрузка унарных операций 39](#_Toc366240885)

[6. Практическое изучение статического полиморфизма с помощью перегрузки операций 39](#_Toc366240886)

[6.1. Подготовка к лабораторной работе № 3 39](#_Toc366240887)

[6.2. Подготовка к лабораторной работе № 4 50](#_Toc366240888)

[7. Виды отношений между классами 60](#_Toc366240889)

[8. Практическое изучение композиции с помощью использования композитных классов 62](#_Toc366240890)

[8.1. Подготовка к лабораторной работе № 5 62](#_Toc366240891)

[9. Одиночное наследование 72](#_Toc366240892)

[9.1. Правила наследования 72](#_Toc366240893)

[9.2. Правила видимости при наследовании 73](#_Toc366240894)

[9.3. Закрытое (private) наследование 74](#_Toc366240895)

[10. Практическое изучение одиночного наследования с помощью использования производных классов 75](#_Toc366240896)

[10.1. Подготовка к лабораторной работе № 6 75](#_Toc366240897)

[11. Динамический полиморфизм, механизм виртуальных функций 91](#_Toc366240898)

[11.1. Виртуальные деструкторы 92](#_Toc366240899)

[11.2. Абстрактные классы. Чистые виртуальные функции 93](#_Toc366240900)

[12. Практическое изучение динамического полиморфизма с помощью использования абстрактного класса и чистой виртуальной функции 93](#_Toc366240901)

[12.1. Подготовка к лабораторной работе № 7 93](#_Toc366240902)

[13. Классы потоков ввода-вывода языка С++ 111](#_Toc366240903)

[13.1. Потоки ввода-вывода 111](#_Toc366240904)

[13.2. Классы потоков C++ 112](#_Toc366240905)

[13.3. Стандартные объекты-потоки 114](#_Toc366240906)

[13.4. Операции помещения и извлечения 114](#_Toc366240907)

[13.5. Форматирование данных при обмене с потоками 116](#_Toc366240908)

[13.6. Форматирующие функции класса ios 118](#_Toc366240909)

[13.7. Управление форматированием с помощью манипуляторов 119](#_Toc366240910)

[13.8. Неформатируемый обмен данными с потоками 121](#_Toc366240911)

[14. Практическое изучение потоковых классов с помощью использования методов управления потоками ввода-вывода 122](#_Toc366240912)

[14.1. Подготовка к лабораторной работе № 8 122](#_Toc366240913)

[Литература 132](#_Toc366240914)