СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данное издание содержит набор лабораторных работ по первой части курса «Программирование» для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника», выполнение которых направлено на формирование знаний и умений в области объектно-ориентированного программирования на языке C++.

Каждая лабораторная работа включает индивидуальные задания и список контрольных вопросов по соответствующей теме. В ходе выполнения лабораторной работы студент должен:

- изучить теоретическую часть работы, частично приведенную в данном издании, частично предлагаемую на лекциях, а также самостоятельно проанализировать литературу по теме;
- выполнить индивидуальные задания, предполагающие разработку алгоритмов и кодирование;
- оформить и защитить отчет по лабораторной работе.

Оформленный отчет по лабораторной работе должен включать:

- название темы;
- цель лабораторной работы;
- листинги программ / скриптов;
- экранные формы с результатами (при необходимости);
- письменные ответы на контрольные вопросы, приведенные в конце каждой работы.

После выполнения всех лабораторных работ курса студент должен приобрести следующие навыки и умения:

- проектировать и кодировать структуры данных и алгоритмы в рамках объектно-ориентированной парадигмы на языке высокого уровня C++;
- корректно работать с памятью в C++, указателями и ссылками, идентифицировать участки кода с утечкой памяти;
- осуществлять консольный и файловый ввод/вывод;
- производить объектно-ориентированный анализ предметной области, программировать классы, реализовывать инкапсуляцию, наследование, полиморфизм, перегрузку операторов;
- писать безопасный код, включающий корректную обработку ошибок и перехват исключений;
- применять стандартную библиотеку шаблонов STL для работы с различными коллекциями данных;
- разрабатывать программы на языке C++ для решения широкого круга практических и научно-исследовательских задач.

Лабораторная работа №1

Тема: Основы языка C++. Работа с памятью.

Цель: Научиться программировать на языке C++ базовые операции с данными разных типов, разветвляющиеся и циклические алгоритмы, научиться работать с динамической памятью в C++, осуществлять консольный ввод-вывод данных.

Темы для предварительной проработки:

- Среда разработки MS Visual Studio.
- Типы данных языка C++. Переменные, константы, функции.
- Арифметические операции, ветвление и циклы на языке C++.
- Консольный ввод-вывод (iostream и cstdio).
- Работа с памятью в C++. Указатели, ссылки, массивы, строки.

Индивидуальные задания:

1. Написать программу обработки одномерного массива с N элементами, в соответствии с вариантом (*приложение А*). Необходимые действия должны производиться в функции *processArray()*, возвращающей определенное значение и формирующей выходной массив данных по алгоритму, указанному в варианте. Память под массивы выделять статически (объявлять массивы локально в функции *main*) и для доступа к элементам использовать **индексы**. Ввод-вывод данных организовать средствами *cstdio*.
2. Написать программу, которая преобразует одномерный массив (1D) в двумерный (2D) (или наоборот), в соответствии с вариантом. Необходимо оформить в отдельных функциях код следующих действий: 1) инициализация массива; 2) вывод массива; 3) преобразование массива (создание нового массива с другой структурой). Память под массивы выделять динамически и для доступа к элементам использовать **указатели**. Ввод-вывод данных организовать средствами *iostream* и *iomanip*.
3. Написать свой аналог стандартной функции обработки строк из файла *cstring*, в соответствии с вариантом. В функции *main* на тестовых данных продемонстрировать результат работы как стандартной функции, так и собственной версии. Ввод-вывод данных организовать средствами *cstdio*.

Контрольные вопросы:

1. Перечислите типы данных C++ с примерами объявления переменных.
2. Перечислите и кратко опишите классы памяти в C++.
3. Что из себя представляют указатели и ссылки? Приведите пример их взаимосвязи.

4. Что из себя представляет арифметика указателей (pointer arithmetic)?
5. Область действия переменных в C++. Приведите пример кода с memory leak.
6. Синтаксис функций на C++. Что такое сигнатура функции?
7. В чем заключается особенность переменных, объявленных в функциях с ключевым словом static?
8. Как можно организовать возврат значения в параметре функции?
9. Организация ввода-вывода с помощью cstdio.
10. Организация ввода-вывода с помощью iostream и iomanip.

Пример выполнения задания 1.

Объявить массив из N = 20 вещественных чисел, проинициализировать нулями. Функция *processArray()* должна заполнить массив случайными числами от -15.0 до 15.0, подсчитать и вернуть среднее значение *average* элементов массива и сформировать выходной булев массив, каждый элемент которого равен true (T), если соответствующий элемент входного массива больше среднего значения и false (F) в противоположном случае. Вывести на экран результирующие массивы. Пример:

```
Вход:
[-7 -6 13 -8 -10 -8 10 -10 -5 -3 -12 5 -13 -4 2 6 -10 -7 2 -5]

Выход:
[F F T F F F T F F T F T F F T T F F T F]
average = -3
```

Решение. В главной функции создаем статические массивы и вызываем функцию *processArray()*.

```
int main()
{
    const unsigned int LEN = 20;
    // создаем массив и инициализируем его нулями
    double a[LEN] = { 0.0 };
    // булев массив – значениями false
    bool b[LEN] = { false };

    // инициализация генератора случайных чисел
    srand(0);

    // подсчет среднего значения в массиве a;
    // изменение массива b
    double result = processArray(a, LEN, b);

    // вывод массивов на экран
    printArray(a, LEN);
    printArrayBool(b, LEN);
```

```
    // ...и среднего значения элементов массива a
    printf("Average: %g\n\n", result);

    return 0;
}
```

В функции processArray() заполняем массив случайными числами, считаем среднее и формируем массив outs булевых значений. Обратите внимание на сигнатуру: запись «double arr[]» эквивалентна записи «double* arr» (т.е. передаче параметра по указателю).

```
double processArray(double arr[], unsigned int N, bool outs[])
{
    // заполнение массива случайными числами [-15.0..15.0]
    for (unsigned int i = 0; i < N; ++i)
    {
        arr[i] = (double)(rand() % 30) - 15.0;
    }

    double average = 0.0;

    // подсчет среднего значения элементов
    for (unsigned int i = 0; i < N; ++i)
    {
        average += arr[i];
    }
    average /= N;

    // формирование булевого массива (по заданию)
    for (unsigned int i = 0; i < N; ++i)
    {
        outs[i] = arr[i] > average;
    }

    return average;     // возвращаем среднее
}
```

Пример функции вывода на экран всех значений булевого массива:

```
void printArrayBool(bool* arr, unsigned int N)
{
    for (unsigned int i = 0; i < N; ++i)
    {
        arr[i] ? printf("T ") : printf("F ");
    }
    printf("\n");
}
```

Пример выполнения задания 2.

Преобразование: 1D → 2D. Одномерный массив из 20 целых чисел необходимо разложить по двумерной сетке 4x5 слева направо и сверху вниз.

Инициализация: заполнить массив числами Фибоначчи:

$$a[n] = a[n–1] + a[n–2].$$

Вывод на экран: на каждый элемент массива отвести 7 позиций.

```
[1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584]

                                =>

     [   1        1         2         3         5
         8       13        21        34        55
        89      144       233       377       610
       987     1597      2584      4181      6765  ]
```

Решение. В главной функции создаем динамически входной одномерный массив int* arr = new int [LEN].

Двумерный массив создаем из одномерного массива arr в функции makeArray2D(arr, N, M).

```
int main()
{
     // задаем все параметры
     const unsigned int LEN = 20;
     const unsigned int N = 4;
     const unsigned int M = 5;

     // выделяем память под входной массив
     int* arr = new int [LEN];

     // инициализируем входной массив (по заданию)
     initializeArray1D(arr, LEN);

     // выводим на консоль массив
     printArray1D(arr, LEN);

     // конвертируем 1-мерный массив в 2-мерный (по заданию)
     int** arr2D = makeArray2D(arr, N, M);

     // выводим на консоль 2-мерный массив-результат
     printArray2D(arr2D, N, M);

     // освобождаем память 1-мерного массива
     freeArray1D(arr);
     // освобождаем память 2-мерного массива
     freeArray2D(arr2D, N);

     return 0;
}
```

Код функции makeArray2D может быть таким:

```
int** makeArray2D(int* arr1D, unsigned int rows, unsigned int
cols)
{
     // выделяем память под выходной массив
     int** arr2D = new int* [rows];

     // и еще под каждый ряд массива в отдельности
     for (unsigned int i = 0; i < rows; ++i)
     {
          *(arr2D + i) = new int [cols];
     }

     for (unsigned int i = 0; i < rows; ++i)
     {
          for (unsigned int j = 0; j < cols; ++j)
          {
               *(*(arr2D + i) + j) = *(arr1D + i*cols + j);
          }
     }

     // возвращаем указатель на выделенную память
     // под 2-мерный массив
     return arr2D;
}
```

По заданию, здесь работа с массивами производится через указатели. Запись

```
*(*(arr2D + i) + j) = *(arr1D + i*cols + j )
```

эквивалентна более понятной «индексной» записи

```
arr2D[i][j] = arr1D[i*cols + j]
```

Код инициализации массива может выглядеть, например, так:

```
void initializeArray1D(int* arr, unsigned int len)
{
     if (len > 0)
     {
          *arr = 1.0;
     }

     if (len > 1)
     {
          *(arr + 1) = 1.0;
     }
```

```
    for (unsigned int i = 2; i < len; ++i)
    {
        *(arr + i) = *(arr + i-1) + *(arr + i-2);
        // эквивалентно: arr[i] = arr[i-1] + arr[i-2]
    }
}
```

Остальные функции (printArray1D, printArray2D, freeArray1D, freeArray2D) предлагаются на самостоятельную проработку.

Пример выполнения задания 3.

Функция: *strrchr*.

Формат: const char* strrchr(const char* s, int c).

Существует также перегруженная версия char strrchr(char* s, int c). В чем отличие?*

Описание: функция находит в строке s последнее вхождение символа c и возвращает подстроку, начинающуюся с этого символа.

Решение. В главной функции объявляем тестовую строку и вызываем поочередно стандартную strrchr и свою версию – reverseChar.

```
int main()
{
    // тестовая строка
    char s[] = "www.some_address.and_something_else.com";

    // проверяем стандартную функцию
    // (ищем последнее вхождение символа ".")
    const char* standard_result = strrchr(s, '.');

    // проверяем свою версию стандартной функции
    const char* my_result = reverseChar(s, '.');

    // выводим на экран строку,
    // затем результат стандартной функции
    // и затем результат своей версии
    printf("Initial string: %s\nStandard: %s\nMy version: %s\n",
                            s, standard_result, my_result);
    return 0;
}


const char* reverseChar(const char* s, int c)
{
    int len = 0;   // стартовый индекс

    // указатель на оставшуюся часть строки (его нужно вернуть)
    const char* rest_of_string = NULL;

    // проходим до конца строки (пока не встретим символ '\0')
    while (s[len] != '\0')
    {
```

```
            // если очередной символ оказался равен искомому c,
            if (s[len] == c)
                 // то запоминаем указатель на него
                 rest_of_string = s + len;
            len++;
      }
      // возвращаем последний присвоенный указатель
      return rest_of_string;
}
```

Вообще, возможны разные алгоритмы, приводящие к одним и тем же результатам. Приведем еще пример (менее эффективный, не рекомендуется применять) возможной реализации той же функции strrchr():

```
const char* reverseChar2(const char* s, int c)
{
      // стартовый индекс
      int len = 0;

      // проходим до конца строки (пока не встретим символ 0)
      while (s[len] != '\0') len++;

      // затем идем обратно, пока не найдем искомый символ
      while (s[len] != c && len >= 0) len--;

      // возвращаем указатель на часть строки,
      // начиная с найденной позиции
      return s + len;
}
```

Наконец, третий вариант через указатели. Например, функция strrchr() может быть реализована на языках C/C++ следующим образом (более компактная версия функции reverseChar, приведенной выше):

```
const char *strrchr(const char *s, int c)
{
      const char* ret=0;
      do
      {
         if (*s == (char)c)
             ret = s;
      }
      while (*s++);

      return ret;
}
```

Лабораторная работа №2

Тема: Классы и объекты. Инкапсуляция. Перегрузка операторов.

Цель: Научиться программировать на языке C++ классы, инкапсулировать данные и методы, перегружать унарные и бинарные операторы.

Темы для предварительной проработки:

- Классы. Создание и удаление объектов классов.
- Принцип инкапсуляции в ООП. Спецификаторы доступа.
- Статические члены и функции класса.
- Перегрузка операторов.
- Файловый ввод-вывод.

Индивидуальные задания:

1. Написать классы Vector и Matrix для хранения и обработки одномерных и двумерных массивов, соответственно. Реализовать задание 2 лабораторной работы №1 с помощью созданных классов. Все функции оформить в виде методов классов.
   В коде отразить следующее:
   - Выделение и освобождение динамической памяти производить в конструкторах и деструкторах классов, соответственно.
   - В классе Vector перегрузить оператор индексации []. В классе Matrix добавить методы *T at(int i, int j) const* и *setAt(int i, int j, T val)*, которые позволяют получить и установить значение элемента массива с индексом [i][j], T – это тип элементов массива по варианту (int или double).

   Перегрузить операторы инкремента и декремента (как префиксного, так и постфиксного). Смысл инкремента / декремента всего массива заключается в инкременте / декременте каждого элемента массива.
2. Написать класс Fraction для представления обыкновенных дробей (как отношения двух целых чисел x/y). Перегрузить операторы +, -, *, / для дробей. Реализовать метод void reduce() для сокращения дроби, а также статические методы:
   - int gcd(int n1, int n2)
     функция для нахождения наибольшего общего делителя (Greatest Common Divisor) чисел n1 и n2;
   - void printAsFraction(double dec_fraction)
   - void printAsFraction(char* dec_fraction)
     перегруженные методы вывода десятичной дроби в виде обыкновенной (например, при значении dec_fraction = 0.43 на экране должно вывестись 43/100, при 0.25 – 1/4 и т.д.).

Также реализовать в виде статического члена класса счетчик всех созданных на данный момент в программе экземпляров дробей. Продемонстрировать работу созданного класса. Создать несколько объектов дробей. Произвести операции сложения, вычитания, умножения и деления дробей. Вывести на экран результаты. Показать также результаты работы статических методов класса.

3. Написать собственный класс, в соответствии с вариантом. Продемонстрировать в коде инкапсуляцию данных, применение конструкторов без параметров и с параметрами для инициализации данных. Класс должен содержать метод *serialize()* для сохранения данных класса в файл и метод *deserialize()* для чтения данных класса из файла по умолчанию в текущей директории, а также перегруженные методы *serialize(std::string filename)* и *deserialize(std::string filename)* для работы с файлом с указанным в параметре именем.
   Примечание. Можно вводить дополнительные закрытые свойства и метода класса, не указанные явно в задании, но помогающие решить поставленные задачи.

Контрольные вопросы:

1. Чем отличаются классы и структуры в C++ технически и концептуально?
2. Чем отличаются понятия «класс» и «объект»? Приведите примеры.
3. Опишите традиционную файловую структуру ООП проекта.
4. Что такое header guards (include guards)? Какую проблему они позволяют решить, и какой есть второй способ ее решить?
5. Что такое инкапсуляция? Какие есть спецификаторы доступа и что они означают?
6. Что такое конструктор класса? Деструктор класса? Укажите типы конструкторов.
7. Что означает указатель this?
8. В чем заключаются особенности статических членов и функций классов?
9. Как производится перегрузка методов и перегрузка операторов?
10. Укажите особенности работы с константными объектами классов.
11. Как можно запретить создание объекта класса?
12. В чем отличие глубокого копирования от поверхностного (deep and shallow copying)?

Пример выполнения задания 3.

Класс РОБОТ.

Данные: имя (название), текущие координаты робота, уровень заряда батареи, скорость передвижения. Робот может двигаться в плоскости влево, вправо, вперед, назад, а также ускоряться или замедляться. При увеличении скорости каждый шаг робота увеличивается на единицу и отнимает на 3

единицы больше заряда. В основной функции создать 3 роботов со 100%-ным уровнем заряда. Первому роботу задать начальные координаты при создании, второму – в сеттере после создания, третьему – присвоить по умолчанию при создании. Пять раз переместить всех роботов вправо, при этом первого робота ускорить на первых трех шагах, второго – на втором и дважды на четвертом шагах, а третьего – на пятом шаге. Вывести на консоль уровень заряда и координаты всех роботов до и после перемещения.

Решение. Код заголовочного файла класса РОБОТА (robot.h).

```
#pragma once
#include <string>

class Robot
{
public:
	// конструктор
	Robot();
	// конструктор с параметрами
	Robot(int startX, int startY);
	// деструктор
	~Robot();

	// пример сеттеров
	void setName(std::string name);
	void setPos(int xPos, int yPos);
	// пример геттеров
	int getSpeed();
	double getCharge();

	// переместиться влево / вправо
	void moveLeft();
	void moveRight();

	// ускориться / замедлиться
	void accelerate();
	void slowDown();

	// сериализация / десериализация
	void serialize();
	void serialize(std::string filename);
	void deserialize();
	void deserialize(std::string filename);

	// вывод информации о себе
	void printInfo();

private:
	// имя робота
	std::string name;
	// координаты робота
	int x, y;
```

```
    // уровень заряда батареи
    double charge;
    // скорость (на сколько робот передвигается за один шаг)
    int speed;

};
```

Код файла реализации класса РОБОТА (robot.cpp):

```
#include "robot.h"
#include <iostream>

Robot::Robot(): x(10), y(10)
{
    speed = 0;
    charge = 100.0;
}

Robot::Robot(int startX, int startY): x(startX), y(startY)
{
    speed = 0;
    charge = 100.0;
}

Robot::~Robot()
{
}

void Robot::setName(std::string name)
{
    this->name = name;
}

void Robot::setPos(int xPos, int yPos)
{
    x = xPos;
    y = yPos;
}

double Robot::getCharge()
{
    return charge;
}

int Robot::getSpeed()
{
    return speed;
}

void Robot::moveLeft()
{
    x -= speed;
```

```
     charge -= speed * 3;
}

void Robot::moveRight()
{
     x += speed;
     charge -= speed * 3;
}

void Robot::accelerate()
{
     speed++;
}

void Robot::slowDown()
{
     if (speed > 1)
          speed--;
}

void Robot::printInfo()
{
     std::cout << name << " : ";
     std::cout << "( " << x << "," << y << " ) ";
     std::cout << "CHARGE: " << charge << std::endl;
}

void Robot::serialize()
{
     // здесь код для сохранения данных класса в файл...
}

void Robot::serialize(std::string filename)
{
     //...
}

void Robot::deserialize()
{
     // ...
}

void Robot::deserialize(std::string filename)
{
     //...
}
```

В главной функции можно написать, например, так:

```
     Robot r1(300, 400), r2, r3;   // создаем трех роботов

     r2.setPos(150, 200);
```

```
r1.setName("R2D2");
r2.setName("C3PO");
r3.setName("Sheldon");

std::cout << "Before:" << std::endl;
r1.printInfo();
r2.printInfo();
r3.printInfo();

// --------- имитация перемещений и ускорений роботов
r1.accelerate();
r1.moveRight();     r2.moveRight();     r3.moveRight();

r1.accelerate();
r2.accelerate();
r1.moveRight();     r2.moveRight();     r3.moveRight();

r1.accelerate();
r1.moveRight();     r2.moveRight();     r3.moveRight();

r2.accelerate();
r2.accelerate();
r1.moveRight();     r2.moveRight();     r3.moveRight();

r3.accelerate();
r1.moveRight();     r2.moveRight();     r3.moveRight();
// -------------------------------------------------

std::cout << std::endl << "After:" << std::endl;
r1.printInfo();
r2.printInfo();
r3.printInfo();

r1.serialize("D:\\Robots\\1.txt");
r2.serialize("D:\\Robots\\2.txt");
r3.serialize("D:\\Robots\\3.txt");
```

Результат работы программы:

```
Before:
R2D2 : ( 300,400 ) CHARGE: 100
C3PO : ( 150,200 ) CHARGE: 100
Sheldon : ( 10,10 ) CHARGE: 100

After:
R2D2 : ( 312,400 ) CHARGE: 64
C3PO : ( 158,200 ) CHARGE: 76
Sheldon : ( 11,10 ) CHARGE: 97
```

Лабораторная работа №3

Тема: Наследование и полиморфизм. Абстрактные классы и интерфейсы.

Цель: Научиться реализовывать на C++ наследование классов, программировать абстрактные классы и интерфейсы, виртуальные методы, а также динамически определять тип объекта во время выполнения программы.

Темы для предварительной проработки:

- Наследование.
- Виртуальные функции. Полиморфизм.
- Чисто виртуальные функции, абстрактные классы и интерфейсы.
- Динамическое определение типа в C++ (RTTI).

Индивидуальные задания:

1. Написать в ООП-стиле код программы, позволяющей работать с арифметическими выражениями разного вида, оперирующими вещественными числами: вычислять результат выражения, выводить запись выражения на консоль и в файл лога. Например, для вычисления выражений вида (10+4+2+3+7+1) и (1+2.5) будет использоваться класс *Summator*, выражений вида (2·3·7·1) – класс *Multiplier*, и т.д.
   В коде необходимо отразить следующее:
   - Создать интерфейс *ILoggable* с двумя методами (функционал логирования):

     Запись лога всего выражения на консоль:
     void logToScreen()

     Добавление записи лога всего выражения в файл лога:
     void logToFile(std::string filename).

   - Создать абстрактный класс *ExpressionEvaluator*, реализующий интерфейс *ILoggable* и предоставляющий чисто виртуальный метод *double calculate()* для вычисления результата произвольного выражения. Количество операндов должно храниться в отдельном члене класса. Сами операнды x1, x2, x3 и т.д. должны храниться в члене данного класса – массиве, в куче (динамической памяти).
   - Класс *ExpressionEvaluator* должен предоставлять два конструктора и виртуальный деструктор. В конструкторе без параметров выделять память под 20 операндов и инициализировать их нулями, в конструкторе с параметром N – выделять память под N элементов и

инициализировать нулями. Также необходимо реализовать 2 метода, позволяющие присвоить операндам конкретные значения:

Присвоить значение value одному операнду на позиции pos:
void setOperand(int pos, double value)

Заполнить сразу группу из N операндов массивом значений ops:
void setOperands(double ops[], int N)

- В деструкторе должна освобождаться память, выделенная в конструкторе.
- Создать два подкласса класса *ExpressionEvaluator*, работающих со стандартными выражениями, в соответствии с вариантом, из четырех возможных:

Summator – сумма всех операндов (x1 + x2 + x3 + x4 + ...)
Subtractor – разность всех операндов (x1 – x2 – x3 – x4 – ...)
Multiplier – произведение всех операндов (x1 · x2 · x3 · x4 · ...)
Divisor – частное всех операндов (x1/x2/x3/x4/...), но если хоть один операнд равен 0, то результату выражения присвоить также 0.

- Создать подкласс *CustomExpressionEvaluator*, работающий со специфическими выражениями, вид которых приведен в варианте.
- Подклассы *ExpressionEvaluator*, для которых порядок следования операндов важен, должны также реализовывать интерфейс *IShuffle*. Данный интерфейс объявляет 2 перегруженных метода (функционал перемешивания операндов):

Произвольно перемешать операнды:
void shuffle()
Перемешать операнды, находящиеся на позициях i и j:
void shuffle(int i, int j)

В функции *main()* необходимо продемонстрировать работу созданных классов:

- Создать массив из трех указателей на класс обработки арифметических выражений.
- В соответствии с вариантом, создать в куче три объекта конкретных подклассов обработки арифметических выражений и установить на них указатели; присвоить их операндам значения двумя способами (поэлементным и групповым).

- Продемонстрировать полиморфизм: организовать проход в цикле по указателям и вывести лог выражения на консоль и в файл (в консоли отобразить еще и сам результат выражения).
- Организовать цикл по указателям, в теле которого средствами C++ проверить, реализует ли текущий объект интерфейс *IShuffle*. Если да, то вызвать один из методов *shuffle()* этого объекта и отобразить на экране запись выражения после перемешивания операндов, а также вычислить и отобразить результат нового выражения.

На рис.1 приведена желательная структура проекта (например, для случая, когда нужно использовать классы *Summator* и *Multiplier*):

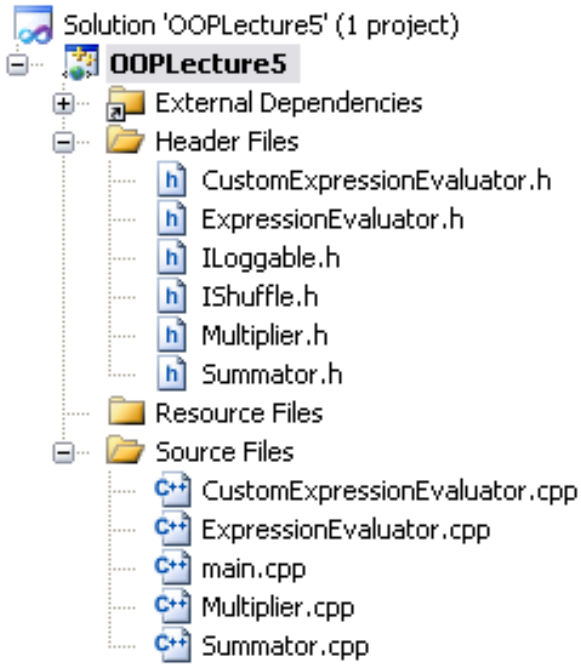


Рисунок 1 – Структура проекта

Примечание. Отрицательные операнды при выводе записи выражения на экран и в файл должны автоматически браться в скобки.

2. Дополнить код задания 3 лабораторной работы №2, написав еще два класса по предметной области, в соответствии с вариантом. Продумать и корректно реализовать схему наследования классов. В главной функции продемонстрировать применение интерфейса, полиморфизм и RTTI на примере 3-4 объектов классов, по аналогии с заданием 1.
   Примечание. Данное задание творческое. Вы можете свободно вводить различные свойства и метода классов, не указанные явно в задании.

Контрольные вопросы:

1. Перечислите механизмы повторного использования кода в ООП.
2. Что такое наследование? Приведите три примера схем наследования классов.
3. Типы и виды наследования в C++.
4. В чем заключается разница между ранним и поздним связыванием?
5. Что такое полиморфизм? Виртуальные методы.
6. Что такое абстрактный класс? Чисто виртуальные методы.
7. Что из себя представляет класс-интерфейс? Для чего он используется?
8. Порядок создания и удаления подклассов. Зачем нужен виртуальный деструктор?
9. Что означает RTTI? Какие есть возможности RTTI в C++?

Пример выполнения задания 1.

Вид выражения CustomExpression: result = x1 * x2 + x3 + x4 + ...

Порядок создания и инициализации объектов подклассов:

CustomExpressionEvaluator:

3 операнда, присвоить поэлементно 5, 4, -2.

Summator:

10 операндов, присвоить группой 5,-2.5, 3.5, 8, 1.5, -9.5, 3, -4, 6.5,-2.5.

Multiplier:

3 операнда, присвоить поэлементно 0.25, 8, 3.5.

Метод shuffle() – поменять местами первый и последний операнды.

Метод shuffle(int i, int j) – поменять местами i-ый и j-ый операнды.

Формат вывода:

```
[3]
5 * 4 + (-2)
18
```

Решение. Необходимый минимум функционала, который должен быть отражен в функции *main()*:

```
int main()
{
     // Создать массив указателей на абстрактный класс
     // обработки арифметических выражений. Например, так:
     ExpressionEvaluator* evaluators[3];

     // создать объект первого типа (CustomExpressionEvaluator)
     // ( x1*x2 + x3 + x4 + ... )
     // и заполнить его данными первым способом
     // (установить отдельно каждый операнд)

     evaluators[0] = new CustomExpressionEvaluator(3);

     // должно вычисляться 5*4 + (-2) = 18
     evaluators[0]->setOperand(0, 5);
```

```
    evaluators[0]->setOperand(1, 4);
    evaluators[0]->setOperand(2, -2);

    // создать объект второго типа (Summator)
    // (x1 + x2 + x3 + x4 + ...)
    // и заполнить его данными вторым способом
    // (массивом операндов)

    evaluators[1] = new Summator(10);

    double sum_operands[] =
        { 5, -2.5, 3.5, 8, 1.5, -9.5, 3, -4, 6.5, -2.5 };
    evaluators[1]->setOperands(sum_operands, 10);

    // должно вычисляться 5 + (-2.5) + 3.5 + 8 + 1.5 +
    // (-9.5) + 3 + (-4) + 6.5 + (-2.5) = 9


    // создание объекта третьего типа (Multiplier)
    // (x1 * x2 * x3 * x4 * ...)
    // и заполнить его данными вторым способом

    evaluators[2] = new Multiplier(3);

    // должно вычисляться 0.25 * 8 * 3.5 = 7
    double operands[] = { 0.25, 8, 3.5 };
    evaluators[2]->setOperands(operands, 3);

    // проход в цикле по указателям evaluators
    // и вывод на консоль и в файл лога выражения
    // (в консоли еще сам результат выражения)

    // демонстрация полиморфизма
    for (int i = 0; i < 3; ++i)
    {
        evaluators[i]->logToFile("Lab3.log");
        evaluators[i]->logToScreen();
        std::cout << evaluators[i]->calculate();
        std::cout << std::endl << std::endl;
    }

    /* здесь организовать еще цикл по указателям evaluators, в
теле которого проверить тип текущего объекта, и если он
реализует интерфейс IShuffle, то вызвать метод shuffle() этого
объекта, после чего метод calculate() и затем отобразить на
экране результат перемешивания и вычисления выражения
    */

    // ... освобождение памяти

    return 0;
}
```

Результат работы программы (последние три строки отражают результат перемешивания операндов выражения *CustomExpression*: первый и последний поменялись местами):

```
[3]
5 * 4 + (-2)
18

[10]
5 + (-2.5) + 3.5 + 8 + 1.5 + (-9.5) + 3 + (-4) + 6.5 + (-2.5)
9

[3]
0.25 * 8 * 3.5
7

[3]
(-2) * 4 + 5
-3
```

Демонстрация режима «дозаписи» в файл ios::app в fstream:

```
#include <ios>
#include <fstream>

int main()
{
    std::ofstream log("logfile.txt",
            std::ios_base::app | std::ios_base::out);

    // при повторном запуске строка допишется в файл
    log << "Log Record\n";

    return 0;
}
```

Лабораторная работа №4

Тема: Агрегация и композиция. Дружественные функции и классы. Исключения.

Цель: Научиться реализовывать на C++ межклассовое отношение агрегации / композиции, писать код генерации и обработки исключений, перегружать операторы с помощью дружественных функций.

Темы для предварительной проработки [УСТНО]:

- Агрегация и композиция.
- Дружественные функции и классы.
- Исключения.

Индивидуальное задание:

Написать простой консольный вариант карточной игры BlackJack для игры один-на-один с дилером, в соответствии с вариантом. В приложении А приведены вариации и особенности правил игры для каждого варианта. В ходе работы необходимо сделать как минимум следующее:

- создать и связать отношением агрегации/композиции и/или наследования классы КАРТА, КОЛОДА, ДИЛЕР, ИГРОК, РУКА, ИГРА (ШАФФЛ). В целом, Вы можете предлагать здесь свои варианты объектно-ориентированного проектирования;
- в начале игры генерировать случайным образом 4 колоды с 36 или 52 картами, в зависимости от варианта;
- имитировать действия дилера, в соответствии с вариантом игры;
- запрограммировать обработку всех потенциально возможных вариантов исхода: блек-джек, перебор, ровно, выигрыш по очкам, проигрыш по очкам;
- бросать и перехватывать исключение при «переборе» («перебор» рассматривать как исключительную ситуацию);
- перегрузить операцию потокового вывода объекта класса карты на экран с помощью дружественной функции. Выводить карту в виде 2♠, Q♦ и т.д. (символы карточных мастей имеют ASCII-коды 3, 4, 5, 6).

Контрольные вопросы:

1. Придумайте и приведите три примера пар классов, находящихся в отношении «has-a».
2. Укажите сходства и различия композиции и агрегации.
3. Опишите суть паттерна делегирования и приведите пример с композицией.
4. Какими способами можно реализовать композицию в C++?

5. Что такое исключение? Как генерируются и обрабатываются исключения в C++?
6. Какие есть стандартные типы исключений в C++?
7. Что такое дружественный класс и дружественная функция?

Демонстрация работы программы.

Пример игры:

- Тип: базовый
- Доп. правила: сплит

Пример консольного диалога:

```
Ваша ставка?
10000

Колоды:     [52] [51] [50] [51]
Дилер:      A♠   ??
Вы:         6♦   2♦

1. Хватит
2. Еще
2

Колоды:     [52] [51] [49] [51]
Дилер:      A♠   ??
Вы:         6♦   2♦   2♠

1. Хватит
2. Еще
3. Сплит?
2

Колоды:     [51] [51] [49] [50]
Дилер:      A♠   ??
Вы:         6♦   2♦   2♠   K♠

1. Хватит
2. Еще
1

Колоды:     [51] [51] [49] [50]
Дилер:      A♠   3♠
Вы:         6♦   2♦   2♠   K♠

Поздравляем! Вы выиграли! Ваш выигрыш: 10000. Всего: 20000.
```

Примечание. Возможно появление дополнительных вопросов в диалоге, в зависимости от ситуации и варианта игры.

Если игрок выбирает сплит, то добавляется еще одна «рука», которой дилер также раздает карты. Если обе «руки» выигрывают у дилера, выигрыш удваивается, если одна из них – игрок остается при своей ставке, если обе проигрывают – ставка теряется.

Пример игры:

- Тип: европейский
- Доп. правила: саррендер

Пример консольного диалога:

```
Ваша ставка?
10000

Колоды:     [52] [51] [50] [52]
Дилер:      A♠
Вы:         7♦   J♦

1. Хватит
2. Еще
3. Снять половину ставки?
3

1. Хватит
2. Еще
2

Колоды:     [51] [51] [50] [52]
Дилер:      A♠
Вы:         7♦   J♦   8♠

Перебор! Вы проиграли! Ваш проигрыш: 5000. Всего: 5000.
```

Примечание. При отсутствии перебора и выборе пользователем пункта «достаточно», дилер должен взять карту себе из колоды, что должно отразиться в диалоге (европейский тип).

Лабораторная работа №5

Тема: Шаблоны. Библиотека STL.

Цель: Получить навыки обобщенного программирования в C++, научиться использовать библиотеку STL для решения различных практических задач.

Темы для предварительной проработки:

- Шаблонные функции и классы.
- Библиотека STL. Основные классы. Итераторы.
- Библиотека STL. Функторы, алгоритмы.

Индивидуальные задания:

1. Написать шаблонный класс *DataManager<T>* для специфической работы с набором однотипных данных (максимальная вместимость равна 64 элементам). В набор можно добавлять данные (метод *push(T elem)* для добавления одного элемента и метод *push(T elems[], int n)* для добавления группы из n элементов), считывать без извлечения (метод *T peek()*) и извлекать (метод *T pop()*) по определенным алгоритмам (в соответствии с вариантом, приложение А). Если набор заполнен на 100% и поступает команда добавления нового элемента(ов), то данные полностью выгружаются (дописываются) в специальный файл dump.dat, а сам массив очищается и новые данные записываются уже в обновленный набор. Если из массива удаляется последний элемент, то он заполняется ранее выгруженными в файл данными (если файл не пуст).

   Необходимо также реализовать явную специализацию шаблонного класса для символьного типа. В ней надо запрограммировать следующее: при добавлении символа в набор все знаки пунктуации должны автоматически заменяться на символ подчеркивания; добавить методы *char popUpper()* и *char popLower()*, которые позволяют при извлечении символа из набора привести его к верхнему или нижнему регистру, соответственно.

   В функции main() продемонстрировать применение шаблонного класса *DataManager* для типов int, double и char.
2. Написать код для чтения произвольного текстового файла и вывода на экран всех слов размером более 3 букв, встречающихся в нем не менее 7 раз, в порядке убывания частоты встречаемости. В качестве разделителей слов рассматривать пробел, точку, запятую, тире, двоеточие, восклицательный знак, точку с запятой. Использовать контейнер std::map.
3. Создать класс книги Book, в котором хранится название, автор и год издания книги. В главной функции создать коллекцию книг.

Продемонстрировать сортировку книг по автору (первичный ключ) и названию (вторичный ключ). Продемонстрировать поиск в коллекции: найти все книги, год издания которых находится в указанном диапазоне. Использовать контейнер std::vector и функторы.

4. Модифицировать код игры «Блек-джек» из лабораторной работы №4: использовать библиотеку STL для работы с коллекциями объектов.

Контрольные вопросы:

1. Что такое шаблонная функция? Шаблонный класс?
2. Как компилятор работает с шаблонами? Статические члены шаблонных классов.
3. Что означает специализация шаблона? Частичная специализация шаблона?
4. Кратко опишите типы контейнеров STL.
5. Чем отличаются контейнеры vector, deque и list? Как они реализованы в STL?
6. Кратко опишите типы итераторов STL и методологию работы с ними.
7. Что такое ассоциативный массив? Кратко опишите контейнеры map, multimap, set.
8. Кратко опишите адаптеры контейнеров stack, queue, priority_queue.
9. Что такое функтор? Где и как функторы используются в библиотеке STL?
10. Что означает аббревиатура RAII? Что такое умный указатель?

Пример выполнения задания 1.

Пример кода в функции main():

```
DataManager<int> manager;
manager.push(-9);              // уже в наборе 1 элемент

int a[60] = {0};
manager.push(a, 60);           // уже в наборе 61 элемент

// узнаем последний элемент (без извлечения)
int x = manager.peek();

for (int i = 1; i < 15; ++i)
     manager.push(i);
     /* здесь на четвертой итерации должна
     произойти выгрузка 64 элементов в файл dump.dat */

// сейчас в наборе 11 элементов

x = manager.pop();
// после pop() будет 10
```

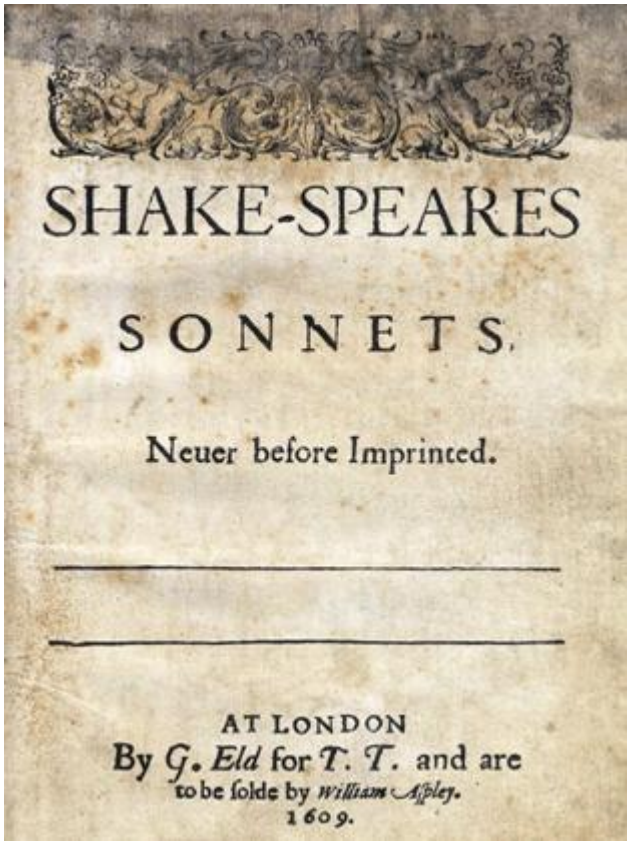
```
DataManager<char> char_manager;
char_manager.push('h');
char_manager.push('e');
char_manager.push('l');
char_manager.push('l');
char_manager.push('o');

// этот метод есть только для char
char C = char_manager.popUpper();
std::cout << C << std::endl;
```

Пример выполнения задания 2.

На примере файла с 20 сонетами Шекспира Sonnets.txt вывод на консоль должен быть таким:

```
thou 55
that 33
with 27
thee 25
when 16
thine 15
time 15
your 15
this 13
from 13
which 13
beauty 12
thyself 12
make 11
sweet 10
shall 10
should 10
were 10
beauty's 9
eyes 9
fair 9
then 9
love 9
world 9
live 8
more 7
yourself 7
than 7
where 7
```



Ниже приведен фрагмент кода с использованием функции strtok() для разбиения строки на подстроки, разделенные определенными символами (strtok() работает с char*, а не std::string):

```
// этот фрагмент кода должен находиться во внешнем цикле
// считывания строк из файла

char szLine[1000];
f.getline(szLine, 1000);

char* substr = strtok(szLine, ".,:!;? ");

while (substr != 0)
{
     std::string word = substr;

     // преобразование всех символов строки к нижнему регистру
     std::transform(word.begin(), word.end(),
                              word.begin(), ::tolower);
     // ... здесь необходимая работа со словом word ...
     substr = strtok(NULL, ".,:!;? ");
}
```

Пример выполнения задания 3.

Основной код функции main() приведен ниже. Вам необходимо написать класс Book с геттерами и два функтора BookSorter и BookFinder, а также уметь объяснить весь код.

```
setlocale(LC_ALL, "RUSSIAN");

std::vector<Book*> books;

books.push_back(
     new Book("Война и мир", "Толстой Л.Н.", 2010));
books.push_back(
     new Book("Подросток", "Достоевский Ф.М.", 2004));
books.push_back(
     new Book("Обрыв", "Гончаров И.А.", 2010));
books.push_back(
     new Book("Анна Каренина", "Толстой Л.Н.", 1999));
books.push_back(
     new Book("Обыкновенная история", "Гончаров И.А.", 2011));
books.push_back(
     new Book("Утраченные иллюзии", "Бальзак О.", 2009));
books.push_back(
     new Book("Оливер Твист", "Диккенс Ч.", 2001));
books.push_back(
     new Book("Фауст", "Гёте И.В.", 2010));
books.push_back(
     new Book("Лилия долины", "Бальзак О.", 1998));

std::cout << std::endl << "Книги в алфавитном порядке: ";
std::cout << std::endl << std::endl;

BookSorter book_sorter;
std::sort(books.begin(), books.end(), book_sorter);
```

```
std::vector<Book*>::iterator i;

for (i = books.begin(); i != books.end(); ++i)
{
     std::cout << (*i)->getAuthor() << " \""
          << (*i)->getName() << "\"" << std::endl;
}


BookFinder book_finder(2005, 2014);
std::vector<Book*>::iterator finder =
          std::find_if( books.begin(), books.end(), book_finder);

std::cout << "Книги в диапазоне года издания 2005 - 2014: ";
std::cout << std::endl << std::endl;

while (finder != books.end())
{
     std::cout << (*finder)->getAuthor() << " \""
          << (*finder)->getName() << "\"" << std::endl;

     finder = std::find_if(++finder, books.end(), book_finder);
}

for (i = books.begin(); i != books.end(); ++i)
{
     delete (*i);
}
```

ПЕРЕЧЕНЬ ЭКЗАМЕНАЦИОННЫХ ВОПРОСОВ

1. ООП. Основные принципы. Механизмы повторного использования кода. Структура проекта. Header guards.
2. Классы и объекты. Инициализация объекта класса. Конструкторы и деструкторы. Методы класса, создаваемые компилятором по умолчанию.
3. Классы и объекты. Особенности константных объектов классов. Указатель this. Статические методы класса.
4. Перегрузка операторов. Оператор +, оператор <, оператор [], оператор инкремента.
5. Принцип инкапсуляции. Спецификаторы доступа. Сеттеры, геттеры. Запрет создания объекта класса. Запрет копирования объекта класса.
6. Наследование. Переопределение, доопределение и сокрытие. Спецификаторы наследования. Порядок создания и удаления подклассов.
7. Полиморфизм. Виртуальные функции.
8. Абстрактные классы и интерфейсы. Чисто виртуальные функции. Виртуальный деструктор.
9. RTTI в C++ (dynamic_cast, typeid). Исключение bad_cast.
10. Композиция и агрегация. Способы реализации на C++. Паттерн делегирования.
11. Дружественные функции и классы. Перегрузка оператора << для потокового вывода произвольного объекта.
12. Исключения в C++. Раскручивание стека.
13. Шаблонные функции и классы. Особенности компиляции шаблонов. Явная и частичная специализация шаблона.
14. Библиотека STL. Контейнеры последовательностей и итераторы.
15. Библиотека STL. Адаптеры контейнеров и итераторы.
16. Библиотека STL. Ассоциативные контейнеры и итераторы.
17. Библиотека STL. Алгоритмы и функторы.
18. Методы класса. Сигнатура, прототип. Перегрузка методов. Перегрузка конструкторов. Списки инициализации.
19. Потоковый ввод-вывод в C++. Использование <iostream> и <fstream>. Перегрузка оператора << для потокового вывода произвольного объекта.
20. Работа с памятью в C++. Указатели и ссылки. Типы памяти. Утечки памяти. Массивы.

Пример экзаменационного билета

Донецкий национальный университет

Образовательно-квалификационный уровень Бакалавр
Направление подготовки 09.03.01 Информатика и вычислительная техника
Учебная дисциплина Программирование Семестр 3

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 3

1. Классы и объекты. Особенности константных объектов классов. Указатель this. Статические методы класса.

2. Написать интерфейс "Парсер" с методом парсинга (разбора) файлов **parse(std::string filename)**. Написать два класса, реализующие данный интерфейс: "XML-парсер" и "MIDI-парсер". В первом случае необходимо считать все строки из текстового файла и вывести их количество на консоль; во втором случае – просто вывести на консоль длину полного имени файла (параметра метода). Если указываемого файла не существует, бросать исключение. Продемонстрировать перехват исключения и полиморфизм в главной функции.

ЛИТЕРАТУРА

1. Лафоре Р. Объектно-ориентированное программирование в C++. – СПб.: Питер, 2013. – 928 с.
2. Дейтел Х.М., Дейтел П.Дж. Как программировать на C++. – М.: Издательство «БИНОМ», 2000. – 1024 с.
3. Страуструп Б. Язык программирования C++. 3-е изд. – СПб.; М.: Невский диалект; Издательство «БИНОМ», 1999. – 991 с.
4. Прата С. Язык программирования C++. Лекции и упражнения, 6-ое изд.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2012. – 1248 с.
5. Мейерс С. Эффективный и современный C++: 42 рекомендации по использованию C++11 и C++14. – М.: Издательский дом “Вильямс”, 2016. – 304 с.
6. Мейерс С. Эффективное использование STL. – СПб: Питер, 2002. – 224 с.
7. Саттер Г. Решение сложных задач на C++. Серия *C++* In-Depth: Пер. с англ. – М.: Издательский дом “Вильямс”, 2008. – 400 с.
8. Александреску А. Современное проектирование на C++: Обобщенное программирование и прикладные шаблоны проектирования. — СПб.: Вильямс, 2008. – 336 с.
9. Документация C++. URL: http://en.cppreference.com/w/ (дата обращения 03.01.2016).

ПРИЛОЖЕНИЕ А.
ВАРИАНТЫ ЗАДАНИЙ

Лабораторная работа №1

Вариант 1

1. Объявить массив из N = 15 вещественных чисел, проинициализировать единицами. Функция processArray() должна домножить все элементы с четными индексами на случайное число из диапазона [a, b] (a и b ввести с клавиатуры, a<0), подсчитать и вернуть количество отрицательных элементов массива и сформировать выходной массив, содержащий только отрицательные элементы входного (т.е. размерность уменьшится). Вывести на экран результирующие массивы.
   Пример:

```
Вход:
[13 -6 13 12 -20 22 -10 10 -25 -13 -22 -25 17 6 12]
            a = -25   b = 25
Выход:
[-6 -20 -10 -25 -13 -22 -25]
cnt = 7
```

2. *Преобразование*: 1D → 2D. Одномерный массив из 25 вещественных чисел необходимо разложить по двумерной сетке 5x5 слева направо и сверху вниз, но первый элемент на каждой строке должен содержать сумму остальных четырех.
   Инициализация: заполнить массив числами x[n] = n·sin(πn/25), n – индекс элемента.
   Вывод на экран: на каждый элемент массива отвести 10 позиций.

```
[0 0.12533 0.49738 1.1044 1.927 2.9389 4.1073 5.3936 6.7546
8.1434 9.5106 10.805 11.976 12.974 13.752 14.266 14.477 14.354
13.869 13.006 11.756 10.117 8.0987 5.7199 3.008]

                                =>

[   3.6541   0.12533   0.49738    1.1044     1.927
    24.399    4.1073    5.3936    6.7546    8.1434
    49.508    10.805    11.976    12.974    13.752
    55.706    14.477    14.354    13.869    13.006
    26.943    10.117    8.0987    5.7199     3.008 ]
```

3. Функция *strchr*.
   Формат const char* strchr(const char* s, int c).
   Функция находит в строке s первое вхождение символа c и возвращает подстроку, начинающуюся с этого символа.

Вариант 2

1. Объявить массив из N = 18 вещественных чисел, проинициализировать нулями. Функция processArray() должна заполнить массив случайными числами от -20.0 до 70.0, а затем отнормировать массив, т.е. поделить каждый элемент массива на максимальное по модулю значение в массиве. Это значение она также должна вернуть на выходе. Сформировать выходной вещественный массив, в котором все элементы, стоящие до позиции максимального элемента включительно, повторяют элементы входного массива, а остальные равны X (число X ввести с клавиатуры). Вывести на экран результирующие массивы.
   Пример:

```
Вход:
  [18 49 68 -13 15 -13 65 45 -10 -8 -17 0 42 -9 -3 61 15 -15]
                              X = 2.7
Выход:
max Item = 68

[0.264706 0.720588 1 -0.191176 0.220588 -0.191176 0.955882
0.661765 -0.147059 -0.117647 -0.25 0 0.617647 -0.132353 -
0.0441176 0.897059 0.220588 -0.220588]

[0.264706 0.720588 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7
2.7 2.7 2.7 2.7 2.7]
```

2. *Преобразование*: 2D → 1D. Двумерный массив 5x5 целых чисел необходимо выложить в один ряд по элементам слева направо и сверху вниз, исключая все элементы на четных строках.
   Инициализация: заполнить массив факториалами: x[i][j] = (i+j)! .
   Вывод на экран: на каждый элемент массива отвести 8 позиций.

```
 [       1        1        2        6       24
         1        2        6       24      120
         2        6       24      120      720
         6       24      120      720     5040
        24      120      720     5040    40320   ]

                                =>

[1 1 2 6 24 2 6 24 120 720 24 120 720 5040 40320]
```

3. Функция *strcmp*.
   Формат int strcmp (const char* s1, const char* s2).
   Функция сравнивает строку s1 со строкой s2 в лексикографическом порядке. Если s1<s2, то результат равен -1, если s1=s2, то результат равен 0, если s1>s2, то результат равен 1.

Вариант 3

1. Объявить массив из N=10 целых чисел, проинициализировать нулями. Функция processArray() должна заполнить массив членами геометрической прогрессии с начальным элементом b_1 и шагом q (ввести с клавиатуры), подсчитать и вернуть cnt – количество всех трехзначных элементов массива, а также сформировать выходной массив, который содержит все элементы исходного, кроме тех, которые делятся на 3 с остатком 1. Вывести на экран массивы.

```
Вход:
b1 = 2 q = 5
[2 10 50 250 1250 6250 31250 156250 781250 3906250]
Выход:
[2 50 1250 31250 781250]
cnt = 1
```

2. *Преобразование*: 2D → 1D. Двумерный массив 4x4 вещественных чисел необходимо выложить в один ряд по элементам *справа налево* и сверху вниз.
   Инициализация: заполнить массив числами x[i][j] = √(i+j+1).
   Вывод на экран: каждый элемент одномерного массива вывести с точностью 4 знака после запятой; каждый элемент двумерного массива – с точностью 2 знака.

```
[    1   1.41   1.73      2
  1.41   1.73      2   2.24
  1.73      2   2.24   2.45
     2   2.24   2.45   2.65      ]

                                =>

[2 1.7321 1.4142 1 2.2361 2 1.7321 1.4142 2.4495 2.2361 2
1.7321 2.6458 2.4495 2.2361 2]
```

3. Функция *strcat*.
   Формат char* strcat(char* dest, const char* src).
   Функция приписывает строку src к строке dest.

Вариант 4

1. Объявить массив из N=20 целых чисел, проинициализировать нулями. Функция processArray() должна заполнить массив случайными числами от 1 до 10, вычислить и вернуть наиболее часто встречающееся значение в массиве (если таких несколько, вернуть наименьшее) и сформировать выходной массив из всех элементов, которые встречаются как минимум 2 раза во входном массиве. Вывести на экран массивы.

```
Вход:
[2 10 3 5 8 7 3 3 7 10 5 2 4 5 4 3 2 10 5 1]
Выход:
[2 3 4 5 7 10]
most_frequent_element = 3
```

2. *Преобразование*: 2D → 1D. Двумерный массив 5x3 вещественных чисел необходимо выложить в один ряд по элементам слева направо и *снизу вверх*.
   Инициализация: заполнить массив числами x[i][j] = sin(i–j) + cos(i+j).
   Вывод на экран: на каждый элемент массива отвести 14 позиций.

```
[             1        -0.3012         -1.325
          1.382        -0.4161         -1.831
         0.4932        -0.1485        -0.6536
        -0.8489         0.2557          1.125
          -1.41         0.4248          1.869        ]

                                =>

[-1.41  0.4248  1.869  -0.8489  0.2557  1.125  0.4932  -0.1485  -
0.6536 1.382 -0.4161 -1.831 1 -0.3012 -1.325]
```

3. Функция *strncat*.
   Формат char* strncat(char* dest, const char* src, size_t maxlen).
   Функция приписывает maxlen символов строки src к строке dest.

Вариант 5

1. Объявить массив из N=16 целых чисел, проинициализировать единицами. Функция processArray() должна заполнить элементы массива с *четными индексами* степенями двойки (2, 4, 8, 16, …), с нечетными индексами – степенями тройки (3, 9, 27, …). Также подсчитать и вернуть cnt – количество двузначных чисел в массиве и сформировать выходной массив, содержащий только такие числа. Вывести на экран результирующие массивы.

```
Вход:
[3 2 9 4 27 8 81 16 243 32 729 64 2187 128 6561 256]
Выход:
[27 81 16 32 64]
cnt = 5
```

2. *Преобразование*: 1D → 2D. Одномерный массив из 18 целых чисел необходимо отсортировать в порядке убывания и разложить по двумерной сетке 9x2 справа налево и сверху вниз.
   Инициализация: заполнить массив числами x[i] = i^2+1 и все элементы с четными индексами домножить на -1.
   Вывод на экран: на каждый элемент массива отвести 5 позиций.

```
[1 -2 5 -10 17 -26 37 -50 65 -82 101 -122 145 -170 197 -226
257 -290]

                                    =>

[ 257   197
  145   101
   65    37
   17     5
    1    -2
  -10   -26
  -50   -82
 -122  -170
 -226  -290 ]
```

3. Функция *strcpy*.
   Формат char* strcpy(char* dest, const char* src).
   Функция копирует строку src в строку dest.

Вариант 6

1. Объявить массив из N = 15 вещественных чисел, проинициализировать единицами. Функция process_array() должна домножить все элементы с четными индексами на случайное число из диапазона [a, b] (a и b ввести с клавиатуры, a<0), подсчитать и вернуть количество двузначных элементов массива и сформировать выходной массив, содержащий только двузначные элементы входного (т.е. размерность уменьшится). Вывести на экран результирующие массивы.
   Пример:

```
Вход:
[13  -6  13  12  -20  22  -10  10  -25  -13  -22  -25  17  6  12]
                    a = -25   b = 25
Выход:
[13 13 12 -20 22 -10 10 -25 -13 -22 -25 17 12]
cnt = 13
```

2. *Преобразование*: 1D → 2D. Одномерный массив из 16 вещественных чисел необходимо разложить по двумерной сетке 4x4 слева направо и сверху вниз, но первый элемент на каждой строке должен содержать сумму остальных трех.
   Инициализация: заполнить массив числами x[n] = n·exp(πn / 100), n – индекс элемента.
   Вывод на экран: на каждый элемент массива отвести 10 позиций.

```
[0 1.0319 2.1297 3.2965 4.5356 5.8504 7.2446 8.7218 10.286
11.941 13.691 15.541 17.495 19.557 21.734 24.03]

                                =>

[   6.4581    1.0319    2.1297    3.2965
    21.817    5.8504    7.2446    8.7218
    41.173    11.941    13.691    15.541
    65.321    19.557    21.734    24.03    ]
```

3. Функция *strncpy*.
   Формат char* strncpy(char* dest, const char* src, size_t num).
   Функция копирует первые num символов из строки src в строку dest.

Вариант 7

1. Объявить массив из N=15 вещественных чисел, проинициализировать нулями. Функция processArray() должна заполнить массив случайными числами от 20.0 до 100.0, а затем вычесть из каждого элемента массива минимальное значение в массиве. Это значение она также должна вернуть на выходе. Сформировать выходной вещественный массив, в котором все элементы, стоящие до позиции минимального элемента включительно, повторяют элементы входного массива, а остальные равны минимальному элементу массива. Вывести на экран результирующие массивы. Пример:

```
Вход:
[38 79 68 29 65 43 85 93 72 80 37 23 31 92 83]

Выход:
min Item = 23

[15 56 45 6 42 20 62 70 49 57 14 0 8 69 60]

[38 79 68 29 65 43 85 93 72 80 37 23 23 23 23]
```

2. *Преобразование*: 2D → 1D. Двумерный массив 5x5 целых чисел необходимо выложить в один ряд по элементам слева направо и сверху вниз, исключая все элементы на нечетных строках.
   Инициализация: заполнить массив факториалами: x[i][j] = i! + j! .
   Вывод на экран: на каждый элемент массива отвести 8 позиций.

```
[      2         2         3         7        25
       2         2         3         7        25
       3         3         4         8        26
       7         7         8        12        30
      25        25        26        30        48          ]

                                  =>

[2 2 3 7 25 7 7 8 12 30]
```

3. Функция *strchr*.
   Формат const char* strchr(const char* s, int c).
   Функция находит в строке s первое вхождение символа c и возвращает подстроку, начинающуюся с этого символа.

Вариант 8

1. Объявить массив из N=15 целых чисел, проинициализировать нулями. Функция processArray() должна заполнить массив членами арифметической прогрессии с начальным элементом a_1 и шагом d (ввести с клавиатуры), подсчитать и вернуть cnt – количество всех двузначных элементов массива, а также сформировать выходной массив, который содержит все элементы исходного, кроме тех, сумма цифр которых равна 10. Вывести на экран массивы.

```
Вход:
a1 = 7 d = 12
[7 19 31 43 55 67 79 91 103 115 127 139 151 163 175]

Выход:
[7 31 43 67 79 103 115 139 151 175]
cnt = 7
```

2. Преобразование: 2D → 1D. Двумерный массив 4x4 вещественных чисел необходимо выложить в один ряд по элементам *слева направо* и *снизу вверх*.
   Инициализация: заполнить массив числами x[i][j] = √(i+j+1).
   Вывод на экран: каждый элемент одномерного массива вывести с точностью 4 знака после запятой; каждый элемент двумерного массива – с точностью 2 знака.

```
[    1   1.41   1.73      2
  1.41   1.73      2   2.24
  1.73      2   2.24   2.45
     2   2.24   2.45   2.65     ]

                                =>

[2 2.2361 2.4495 2.6458 1.7321 2 2.2361 2.4495 1.4142 1.7321 2
2.2361 1 1.4142 1.7321 2]
```

3. Функция *strncat*.
   Формат char* strncat(char* dest, const char* src, size_t maxlen).
   Функция приписывает maxlen символов строки src к строке dest

Вариант 9

1. Объявить массив из N=20 целых чисел, проинициализировать нулями. Функция processArray() должна заполнить массив случайными числами от 1 до 5, вычислить и вернуть наименее часто встречающееся значение в массиве (если таких несколько, вернуть наименьшее из них) и сформировать выходной массив из всех элементов входного, отсортированных в порядке возрастания. Вывести на экран массивы.

```
Вход:
[1 3 2 2 1 5 4 2 3 1 2 3 1 2 5 4 3 3 2 4]

Выход:
[1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 4 4 4 5 5]
Least_frequent_element = 5
```

2. *Преобразование*: 2D → 1D. Двумерный массив 5x3 вещественных чисел необходимо выложить в один ряд по элементам *по столбцам сверху вниз*.
Инициализация: заполнить массив числами x[i][j] = sin(i–j) + cos(i+j).
Вывод на экран: на каждый элемент массива отвести 10 позиций.

```
[            1        -0.3012         -1.325
         1.382        -0.4161         -1.831
        0.4932        -0.1485        -0.6536
       -0.8489         0.2557          1.125
         -1.41         0.4248          1.869        ]

                                =>

[1 1.382 0.4932 -0.8489 -1.41 -0.3012 -0.4161 -0.1485 0.2557
   0.4248 -1.325 -1.831 -0.6536 1.125 1.869]
```

3. Функция *strcat*.
Формат char* strcat(char* dest, const char* src).
Функция приписывает строку src к строке dest.

Вариант 10

1. Объявить массив из N=16 целых чисел, проинициализировать единицами. Функция processArray() должна заполнить элементы массива с *четными индексами* степенями двойки (1, 2, 4, 8, 16, …), с нечетными индексами – степенями тройки (3, 9, 27, …). Также подсчитать и вернуть cnt – количество чисел в массиве, содержащих цифру 1, и сформировать выходной массив, являющийся отсортированной версией входного. Вывести на экран результирующие массивы.

```
Вход:
[3 2 9 4 27 8 81 16 243 32 729 64 2187 128 6561 256]

Выход:
[2 3 4 8 9 16 27 32 64 81 128 243 256 729 2187 6561]
cnt = 5
```

2. *Преобразование*: 1D → 2D. Одномерный массив из 18 целых чисел необходимо отсортировать в порядке убывания и разложить по двумерной сетке 6x3 слева направо и сверху вниз по столбцам.
   Инициализация: заполнить массив числами x[i] = i^2 – 1 и все элементы с нечетными индексами домножить на -1.
   Вывод на экран: на каждый элемент массива отвести 5 позиций.

```
[1 0 -3 8 -15 24 -35 48 -63 80 -99 120 -143 168 -195 224 -255
288]

                                    =>

[   1   -35  -143
    0    48   168
   -3   -63  -195
    8    80   224
  -15   -99  -255
   24   120   288   ]
```

3. Функция *strncpy*.
   Формат char* strncpy(char* dest, const char* src, size_t num).
   Функция копирует первые num символов из строки src в строку dest.

Лабораторная работа №2

Вариант 1

Класс СТУДЕНТ.
Данные: ФИО, пол, год рождения, год поступления, номер зачетки, средний балл.
Создать массив из 3 студентов и установить их личные данные с помощью сеттеров. Еще одного студента создать отдельно в куче и установить его данные в конструкторе с параметрами. В главной функции проимитировать три сессии – случайным образом сформировать по 4 новые оценки и пересчитать в отдельном методе средний балл. Вывести результаты студентов, отсортировав их в порядке убывания среднего балла.

Вариант 2

Класс МАГАЗИН.
Данные: название, адрес, год основания, номер, суммарная прибыль.
Создать два объекта-магазина в куче и один в стеке. Данные первых двух заполнить с помощью сеттеров, а третий проинициализировать с помощью конструктора с параметрами. В главной функции проимитировать отдельно продажи за сентябрь, октябрь и ноябрь. Вывести все магазины отдельно в двух рейтинговых списках (сначала один, затем второй): 1) отсортировать в порядке убывания общей прибыли за 3 месяца; 2) отсортировать в порядке убывания среднего прироста прибыли за 3 месяца.

Вариант 3

Класс МУЗЫКАНТ.
Данные: имя, фамилия, пол, год рождения, инструмент, рейтинг.
Создать трех музыкантов в стеке и одного в куче. Данные первых трех заполнить с помощью сеттеров, а четвертый проинициализировать с помощью конструктора с параметрами. В главной функции проимитировать три концерта и голосование зрителей по их результатам (нарастить рейтинг каждого музыканта на случайное число из диапазона 1..5). Вывести список музыкантов в порядке убывания суммарного рейтинга, но клавишники должны идти впереди, независимо от рейтинга.

Вариант 4

Класс ТЕЛЕФОН.
Данные: модель, номер телефона, последний набранный номер, остаток на счету.

Создать два телефона в куче. Установить данные первого телефона с помощью сеттеров, второго – в конструкторе с параметрами. В главной функции проимитировать десять звонков – позвонить по нескольку раз на 3 номера, в том числе на номер другого телефона в программе. За каждую минуту разговора снимается 0,5 грн. Вывести всю информацию о телефонах после проведенных звонков.

Вариант 5

Класс КОНДИЦИОНЕР.
Данные: фирма, модель, вес, температура, режим, год выпуска.
Создать 2 кондиционера в куче и проинициализировать их с помощью конструкторов с параметрами. Еще один кондиционер создать отдельно в стеке и установить его данные с помощью сеттеров. В главной функции проимитировать настройку кондиционеров персоналом помещений – установить каждому режим; если выбран режим охлаждения, то установить также температуру. Проделать эту процедуру три раза. Вывести информацию об использовании кондиционеров – режим, в котором на данный момент работает техника, среднее изменение температуры за весь период настройки.

Вариант 6

Класс АВТОМОБИЛЬ.
Данные: фирма, модель, номер, цена, год выпуска, пробег.
Создать массив из 3 автомобилей и установить их данные с помощью сеттеров. Еще один автомобиль создать отдельно в куче и установить его данные в конструкторе с параметрами. В главной функции проимитировать три дальние поездки на всех машинах – случайным образом сформировать расстояния и нарастить суммарный пробег. Вывести все автомобили в порядке убывания пробега, но машины с годом выпуска от 2011 вывести первыми.

Вариант 7

Класс СТАДИОН.
Данные: адрес, футбольный клуб, количество секторов, вместимость, посещаемость.
Создать 2 стадиона в куче и один в стеке. Установить данные первых двух с помощью сеттеров, данные третьего установить в конструкторе с параметрами. В главной функции проимитировать четыре матча – случайным образом сформировать количество пришедших посетителей и просчитать процент заполнения каждого стадиона. Вывести информацию о стадионах, отсортировав их в порядке убывания среднего процента заполнения за четыре матча.

Вариант 8

Класс ФОТОГРАФ.
Данные: имя, фамилия, пол, год рождения, год начала деятельности, рейтинг.
Создать 2 фотографов в куче и одного в стеке. Данные первых двух заполнить с помощью сеттеров, а третий проинициализировать с помощью конструктора с параметрами. В главной функции проимитировать три фотосессии и голосование зрителей по их результатам (нарастить рейтинг каждого фотографа на случайное число из диапазона 0.0..1.0; если количество проголосовавших людей меньше 10, то не наращивать рейтинг). Вывести список фотографов в порядке убывания суммарного рейтинга.

Вариант 9

Класс САМОЛЕТ.
Данные: модель, авиалинии, год выпуска, вместимость, количество пассажиров.
Создать массив из 3 самолетов и установить их данные с помощью сеттеров. Еще один самолет создать отдельно в куче и установить его данные в конструкторе с параметрами. В главной функции проимитировать три рейса – случайным образом сформировать фактическое количество пассажиров и просчитать процент заполнения каждого самолета. Вывести информацию о самолетах, отсортировав их в порядке убывания среднего процента заполнения за три рейса.

Вариант 10

Класс РЕСТОРАН.
Данные: название, адрес, год основания, рейтинг, количество посетителей за месяц.
Создать два ресторана в куче и один в стеке. Данные первых двух заполнить с помощью конструктора с параметрами, а третий – с помощью сеттеров. В главной функции проимитировать отдельно посещение и голосование посетителей за март, апрель и май. Вывести все рестораны отдельно в двух рейтинговых списках (сначала один, затем второй): 1) отсортировать в порядке убывания суммарного количества посетителей за 3 месяца; 2) отсортировать в порядке убывания среднего рейтинга за 3 месяца.

Лабораторная работа №3

Задание 1

Вариант 1

Вид выражения CustomExpression: result = x1 + x2/2 + x3/3 + x4/4 + ...
Порядок создания и инициализации объектов подклассов:
CustomExpressionEvaluator: 6 операндов, присвоить группой 5, 16, -3, 10, 12.
Subtractor: 4 операнда, присвоить группой 5.6,-2.1, 3.2, 1.5.
Multiplier: 3 операнда, присвоить поэлементно 1.5, -8, 2.5.
Метод shuffle() – поменять местами первый и последний операнды и изменить их знак.
Метод shuffle(int i, int j) – поменять местами i-ый и j-ый операнды и изменить их знак.
Формат вывода:

```
Operands: 4
5.6-(-2.1)-3.2-1.5
Result = 3
```

Вариант 2

Вид выражения CustomExpression: result = x1 – x2 + x3 – x4 + ...
Порядок создания и инициализации объектов подклассов:
Summator: 7 операндов, присвоить группой 5, 12.5, 9, –1.5, -9.5, 0, 11.
Divisor: 4 операнда, присвоить поэлементно 100, -4, 2.5, -4.
CustomExpressionEvaluator: 5 операндов, присвоить группой 5, 4, -2, 9, 3.
Метод shuffle() – переставить операнды так, чтобы сначала шли все отрицательные, а затем положительные.
Метод shuffle(int i, int j) – если i-ый операнд отрицателен, а j-ый – нет, то поменять их местами, иначе – оставить без изменений.
Формат вывода:

```
Expression [4] : 100 / (-4) / 2.5 / (-4)
= 2.5
```

Вариант 3

Вид выражения CustomExpression: result = (x1 + x2) · x3 · x4 · ...
Порядок создания и инициализации объектов подклассов:
Subtractor: 2 операнда, присвоить поэлементно 23.65, -12.15.
CustomExpressionEvaluator: 5 операндов, присвоить группой 2.5, -5, -3, 2, 10.
Multiplier: 4 операнда, присвоить поэлементно 2.5, -3, 4, -1.

Метод shuffle() – поменять местами первый и последний операнды, имеющие дробную часть.
Метод shuffle(int i, int j) – если хотя бы один из i-ого и j-ого операндов имеет дробную часть, поменять их местами, иначе – не менять.
Формат вывода:

```
Total : 4
( 2.5 + (-5) ) * (-3) * 2 * 10
Equals to 150
```

Вариант 4

Вид выражения CustomExpression: result = x1 + 2·x2 + 3·x3 + 4·x4 + ...
Порядок создания и инициализации объектов подклассов:
CustomExpressionEvaluator: 4 операнда, присвоить поэлементно 50, 40, -10, -2.
Multiplier: 4 операнда, присвоить группой -0.5, -8, 1.5, 16.
Summator: 6 операндов, присвоить группой 2.6,-8.1, 13.2, 1.5, 3.4, -4.
Метод shuffle() – отсортировать все операнды в порядке убывания.
Метод shuffle(int i, int j) – отсортировать все операнды между i-ым и j-ым.
Формат вывода:

```
4 operands :
50 + 2*40 + 3*(-10) + (-2)
-> 98
```

Вариант 5

Вид выражения CustomExpression: result = x1 + x2 · x3 + x4 · x5 + ...
Порядок создания и инициализации объектов подклассов:
Divisor: 4 операнда, присвоить поэлементно 150, -3, 10, –2.5.
CustomExpressionEvaluator: 5 операндов, присвоить группой 5, 16, -3, 10, 12.
Multiplier: 5 операндов, присвоить группой 1.5, 4, -2.5 -8, -15.
Метод shuffle() – отсортировать все операнды в порядке возрастания.
Метод shuffle(int i, int j) – если хотя бы один из i-ого и j-ого операндов имеет дробную часть, поменять их местами, иначе – не менять.
Формат вывода:

```
5 + 16*(-3) + 10*12 < Total 5 >
< Result 77 >
```

Вариант 6

Вид выражения CustomExpression: result = (x1 + x2)/2 + x3 + x4 + ...
Порядок создания и инициализации объектов подклассов:
Subtractor: 4 операнда, присвоить группой 10.5, 2.5, -3, 1.5.
CustomExpressionEvaluator: 6 операндов, присвоить группой 5, 15, -8, 1, 2, 3.

Summator: 3 операнда, присвоить поэлементно 1.5, -4, 2.5.
Метод shuffle() – отсортировать все положительные операнды в порядке убывания.
Метод shuffle(int i, int j) – поменять местами i-ый и j-ый операнды.
Формат вывода:

```
Operand1, Operand2, Operand3, Operand4
10.5 minus 2.5 minus (-3) minus 1.5
Result = 3.5
```

Вариант 7

Вид выражения CustomExpression: result = x1 – x2/2 + x3/3 – x4/4 + ...
Порядок создания и инициализации объектов подклассов:
Summator: 2 операнда, присвоить поэлементно 39.1, -12.7.
Multiplier: 4 операнда, присвоить группой -4.5, 2, 3, -10.
CustomExpressionEvaluator: 6 операндов, присвоить группой 5, 16, -3, 10, 12.
Метод shuffle() – отсортировать все отрицательные операнды в порядке убывания.
Метод shuffle(int i, int j) – отсортировать все операнды между i-ым и j-ым.
Формат вывода:

```
Op1, Op2, Op3, Op4 : (-4.5) x 2 x 3 x (-10)
-> 270
```

Вариант 8

Вид выражения CustomExpression: result = x1 · x2 / x3 · x4 / x5 · ...
Порядок создания и инициализации объектов подклассов:
CustomExpressionEvaluator: 5 операндов, присвоить группой 5, 10, -2.5, -40, -2.
Subtractor: 9 операндов, присвоить группой 120,-12, 83.2, -1.5, 5, 7, 2, 18.5, 76.
Multiplier: 2 операнда, присвоить поэлементно -1.5, 80.
Метод shuffle() – поменять местами первый и последний операнды, имеющие дробную часть.
Метод shuffle(int i, int j) – если хотя бы один из i-ого и j-ого операндов имеет дробную часть, поменять их местами, иначе – не менять.
Формат вывода:

```
<4>
5 times 10 divided by (-2.5) times (-40) divided by (-2)
<Result: -400>
```

Вариант 9

Вид выражения CustomExpression: result = x1 / x2 + x3 + x4 + ...
Порядок создания и инициализации объектов подклассов:

Summator: 7 операндов, присвоить группой 15, -3.5, 10.5, –2.1, 3.3, 4, 6.3.
CustomExpressionEvaluator: 5 операндов, присвоить группой 15, 10, -3, 12, -6.5.
Subtractor: 3 операнда, присвоить поэлементно 1.5, 4, -2.5.
Метод shuffle() – переставить операнды так, чтобы сначала шли все отрицательные, а затем положительные.
Метод shuffle(int i, int j) – если i-ый операнд отрицателен, а j-ый – нет, то поменять их местами, иначе – оставить без изменений.
Формат вывода:

```
[ -7- operands]
15 plus (-3.5) plus 10.5 plus (-2.1) plus 3.3 plus 4 plus 6.3
[ -RESULT- 33.5 ]
```

Вариант 10

Вид выражения CustomExpression: result = x1 – x2 / x3 – x4 / x5 – ...
Порядок создания и инициализации объектов подклассов:
CustomExpressionEvaluator: 5 операндов, присвоить группой 1, -5, 2.5, 10, 8.
Summator: 6 операндов, присвоить группой 15,-7.5, 3.2, 8.7, -1.5, -9.5.
Multiplier: 3 операнда, присвоить поэлементно -8, 7, -0.5.
Метод shuffle() – отсортировать все операнды в порядке возрастания по модулю.
Метод shuffle(int i, int j) – отсортировать все операнды между i-ым и j-ым.
Формат вывода:

```
Expression: [5]
1 - (-5)/2.5 - 10/8
Result: [1.75]
```

Задание 2

Вариант 1

Класс СТУДЕНТ + классы ПРЕПОДАВАТЕЛЬ, ЛИЧНОСТЬ.
Реализовать схему наследования классов и корректно распределить по классам данные: ФИО, пол, год рождения, год поступления, номер зачетки, средний балл, стаж, год начала работы в университете, должность, ученая степень, ученое звание.
Интерфейс начислителя суммы оплаты труда ISalaryCalculation с методом calculate() – рассчитать сумму за месяц. Реализация метода в классе преподавателей: сумма равна 5000 + 700 за научную степень кандидата наук (или + 1200 за степень доктора наук) + 2200 за звание доцента (или + 3500 за звание профессора) + 700 за каждые 5 лет стажа. Реализация метода в классе студентов – если средний балл выше 4.5, то начислить 1000, иначе – 700. В

main() создать 2 преподавателей и 2 студентов, продемонстрировать полиморфизм calculate().

Вариант 2

Класс МАГАЗИН + классы АПТЕКА, БУТИК.
Реализовать схему наследования классов и корректно распределить по классам данные: название, адрес, год основания, номер, суммарная прибыль, доход, время работы, количество клиентов со скидкой, тип (круглосуточно или нет).
Интерфейс налогоплательщика ITaxPayment с методом payTax(). Для бутика налог рассчитывается с учетом земельного налога, для аптеки – без (коэффициенты задайте произвольно). Сумма налогов должна быть вычтена из суммарной прибыли. В main() создать 2 аптеки и 1 бутик, продемонстрировать полиморфизм payTax().

Вариант 3

Класс МУЗЫКАНТ + классы ЧЕЛОВЕК, ЗРИТЕЛЬ.
Реализовать схему наследования классов и корректно распределить по классам данные: имя, фамилия, пол, год рождения, инструмент, рейтинг, место в зале, стаж (количество концертов).
Интерфейс посетителя концерта IVisitor с методом visit(). Для музыканта посещение концерта означает наращивание количества концертов на 1, а также изменение рейтинга на некоторую величину (по результатам концерта). Для зрителя посещение означает назначение ему первого свободного места (список свободных мест хранится в файле; при резервировании места оно удаляется из списка в файле). В main() создать 1 музыканта и 3 зрителей, продемонстрировать полиморфизм visit().

Вариант 4

Класс ТЕЛЕФОН + классы МОБИЛЬНОЕ УСТРОЙСТВО, ПЛАНШЕТ.
Реализовать схему наследования классов и корректно распределить по классам данные: фирма-производитель, модель, номер телефона, последний набранный номер, остаток на счету, вес, цвет, цена, уровень заряда.
Интерфейс возможности звонка ICallable с методом void call(std::string recepient) – позвонить по номеру recipient. Реализация метода в классе телефона: проверка корректности номера телефона (содержит только цифры), снятие суммы со счета и расходование 3% заряда. Реализация метода в классе планшета: израсходовать 10% заряда (звонок идет по скайпу). В main() создать 2 телефона и 2 планшета, продемонстрировать полиморфизм call().

Вариант 5

Класс КОНДИЦИОНЕР + классы БЫТОВОЕ УСТРОЙСТВО, ОБОГРЕВАТЕЛЬ.
Реализовать схему наследования классов и корректно распределить по классам данные: фирма, модель, вес, температура, режим, год выпуска, мощность.
Интерфейс возможности управления / регулировки устройства IControllable с методом void control(int temperature) – отрегулировать устройство в зависимости от установленной в параметре температуры. Реализация метода в классе кондиционера: если температура задана меньше 10 градусов, то выдать сообщение и выключиться, иначе присвоить текущему режиму разный номер в зависимости от температуры (т.е. выставить режим). Реализация метода в классе обогревателя: если задана температура выше 45 градусов, то выдать сообщение и выключиться, иначе присвоить текущему режиму разный номер в зависимости от температуры. В main() создать 2 кондиционера и 1 обогреватель, продемонстрировать полиморфизм control().

Вариант 6

Класс АВТОМОБИЛЬ + классы ЛЕГКОВОЙ АВТОМОБИЛЬ, МИКРОАВТОБУС.
Реализовать схему наследования классов и корректно распределить по классам данные: фирма, модель, номер, цена, год выпуска, пробег.
Интерфейс расходователя топлива IFuelConsumer с методом double consume(int kilometers) – рассчитать расход топлива по пройденной дистанции. Реализация метода в классе легкового автомобиля и микроавтобуса отличается коэффициентами (ввести произвольно). В main() создать 2 легковых автомобиля и 2 микроавтобуса, продемонстрировать полиморфизм consume().

Вариант 7

Класс СТАДИОН + классы КОРТ, СПОРТИВНОЕ СООРУЖЕНИЕ.
Реализовать схему наследования классов и корректно распределить по классам данные: адрес, футбольный клуб, количество секторов, вместимость, посещаемость, площадь, тип покрытия, количество абонементов.
Интерфейс предоставителя абонементов ISeasonTicketProvider с методом newTicket() – сгенерировать новый абонемент с выдачей сообщения на экран номера абонемента. Реализация метода в классе стадиона: при выдаче абонемента (все свободные места для абонементов хранятся в файле) уменьшается количество свободных абонементов; максимум – 100 билетов. Реализация метода в классе корта: выдать абонемент можно только на летние месяцы (если текущий месяц – не летний, то не выдать билет); максимум – 15 билетов. В main() создать 1 корт и 2 стадиона, продемонстрировать полиморфизм newTicket().

Вариант 8

Класс ФОТОГРАФ + классы ДИЗАЙНЕР, СОТРУДНИК.
Реализовать схему наследования классов и корректно распределить по классам данные: имя, фамилия, пол, год рождения, год начала деятельности, рейтинг, телефон, количество фотографий в портфолио.
Интерфейс обработки компьютерных изображений IPictureProcessor с методом process(std::string photo). Реализация метода в классе фотографа: вывод на экран сообщения “Photo was processed by /имя_фотографа/”, а также наращивание количества фотографий в портфолио. Реализация метода в классе дизайнера: вывод на экран сообщения “Image was produced by /имя_дизайнера/”, а также изменение рейтинга на некоторую величину. В main() создать 2 фотографов и 2 дизайнеров, продемонстрировать полиморфизм process().

Вариант 9

Класс САМОЛЕТ (ПАССАЖИРСКИЙ) + классы ГРУЗОВОЙ САМОЛЕТ, САМОЛЕТ.
Реализовать схему наследования классов и корректно распределить по классам данные: модель, авиалинии, год выпуска, вместимость, количество пассажиров, размеры и вес, грузовместимость.
Интерфейс загрузки транспортного средства ILoadable с методом void load(int kilograms) – поместить груз на самолет. В грузовой самолет можно помещать груз вплоть до максимальной грузовместимости, иначе – выдавать сообщение об ошибке; в пассажирский – обратно пропорционально фактическому количеству пассажиров на борту (коэффициент зависимости можете задать произвольный); в случае превышения лимита – также выдавать сообщение об ошибке. В main() создать 2 грузовых и 2 пассажирских самолета, продемонстрировать полиморфизм load().

Вариант 10

Класс РЕСТОРАН + классы ЗАВЕДЕНИЕ ОБЩЕПИТА, КАФЕ.
Реализовать схему наследования классов и корректно распределить по классам данные: название, адрес, год основания, рейтинг, количество посетителей за месяц, количество мест, количество блюд в меню, суммарная прибыль.
Интерфейс заведения с возможностью резервирования стола IReservable с методом reserve(). Для ресторана резервирование производится путем уменьшения количества свободных мест на 1; при этом происходит прирост прибыли на 100 грн, для кафе – также путем уменьшения количества свободных мест на 1 и наращивание рейтинга на величину 0.1. В main() создать 2 ресторана и 2 кафе, продемонстрировать полиморфизм reserve().

Лабораторная работа №4

Таблица А.1 – Тип и правила игры «Блекджек» по вариантам

Вар.	Особый вариант игры	Тип игры	Доп. правила
1	Пара тузов	Базовый	Сплит
2	777	Европейский	Дабл
3	Одномастный блек-джек	Базовый	Трипл
4	Максимум карт	Европейский	Саррендер
5	17 + 4	Базовый	Сплит
6	Пара тузов	Европейский	Дабл
7	777	Базовый	Трипл
8	Одномастный блек-джек	Европейский	Саррендер
9	Максимум карт	Базовый	Сплит
10	17 + 4	Европейский	Дабл

Лабораторная работа №5

Задание 1

Вариант 1

push(): данные пишутся на первое свободное место в наборе;
peek(): возвращает предпоследний элемент в наборе или 0, если элементов в наборе меньше 2;
pop(): извлекает средний элемент из набора (если элементов четное число, то первый элемент слева от центра раздела набора).

Вариант 2

push(): данные пишутся в начало набора, остальные смещаются вправо;
peek(): возвращает второй элемент в наборе или 0, если элементов в наборе меньше 2;
pop(): извлекает последний элемент.

Вариант 3

push(): данные пишутся на первое свободное место в наборе, начиная с конца;
peek(): возвращает центральный элемент в наборе или 0, если число элементов четно;
pop(): извлекает первый элемент.

Вариант 4

push(): данные пишутся на первое свободное место в наборе;
peek(): возвращает второй элемент в наборе или 0, если элементов в наборе меньше 2;
pop(): извлекает первый элемент.

Вариант 5

push(): данные пишутся в начало набора, остальные смещаются вправо;
peek(): возвращает центральный элемент в наборе или 0, если число элементов четно;
pop(): извлекает средний элемент из набора (если элементов четное число, то первый элемент слева от центра раздела набора).

Вариант 6

push(): данные пишутся на свое порядковое место в наборе (данные должны упорядочиваться по возрастанию);
peek(): возвращает предпоследний элемент в наборе или 0, если элементов меньше 2;
pop(): извлекает последний элемент.

Вариант 7

push(): данные пишутся на свое порядковое место в наборе (данные должны упорядочиваться по убыванию);
peek(): возвращает второй элемент в наборе или 0, если элементов в наборе меньше 2;
pop(): извлекает первый элемент.

Вариант 8

push():данные пишутся на свое порядковое место в наборе (данные должны упорядочиваться по возрастанию);
peek(): возвращает второй элемент в наборе или 0, если элементов в наборе меньше 2;
pop(): извлекает первый элемент.

Вариант 9

push():данные пишутся на свое порядковое место в наборе (данные должны упорядочиваться по убыванию);
peek(): возвращает последний элемент;
pop(): извлекает предпоследний элемент или последний, если элементов в наборе меньше 2.

Вариант 10

push(): данные пишутся на первое свободное место в наборе, начиная с конца;
peek(): возвращает самый большой элемент в наборе;
pop(): извлекает последний элемент.