ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)
Санкт-Петербургский колледж телекоммуникаций

ЛАБОРАТОРНЫЙ ПРАКТИКУМ «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»

по учебной дисциплине
«**Основы алгоритмизации и программирования»**
часть 2 (26-36)
для специальности: 09.02.07 Информационные системы и программирование
среднего профессионального образования
(базовый уровень)

Санкт-Петербург
2021

Лабораторный практикум содержит описания лабораторных работ на базе языка программирования Visual C++ по учебной дисциплине «Основы алгоритмизации и программирования». Практикум имеет практическую ценность для обучающихся по очной и заочной форме обучения. Самостоятельное изучение теории, самоконтроль по изученному материалу, выполнение заданий позволит хорошо разобраться с вопросами дисциплины «Основы алгоритмизации и программирования».

Выполнение лабораторных работ способствует формированию у учащихся общих и профессиональных компетенций:

- выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам;
- осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности;
- работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами;
- осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста;
- использовать информационные технологии в профессиональной деятельности;
- пользоваться профессиональной документацией на государственном и иностранном языках;
- формировать алгоритмы разработки программных модулей в соответствии с техническим заданием;
- осуществлять рефакторинг и оптимизацию программного кода;
- осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения;
- производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования;

В результате выполнения лабораторных работ студенты должны освоить общие принципы построения алгоритмов, основные алгоритмические конструкции; научиться использовать язык программирования высокого уровня Visual C++, строить логически правильные и эффективные программы.

Практикум составлен в соответствии с федеральным государственным образовательным стандартом среднего профессионального образования, учебным планом и рабочей программой по специальности 09.02.07 Информационные системы и программирование

Лабораторный практикум рассмотрен и одобрен предметной (цикловой) комиссией и методическим советом колледжа.

Составители: К.В.Лебедева, Обудовская А.А.

Согласовано:

Зам. директора по УР _____ Н.В.Калинина

ОГЛАВЛЕНИЕ

ЛАБОРАТОРНАЯ РАБОТА №26. РЕШЕНИЕ ЗАДАЧ НА СОЗДАНИЕ КЛАССОВ СО СВОЙСТВАМИ.

1. Краткие сведения из теории

Класс — это тип, определяемый программистом, в котором объединяются структуры данных и функции их обработки. Конкретные переменные типа данных «класс» называются экземплярами класса, или объектами.

Программы, разрабатываемые на основе концепций объектно-ориентированного программирования (ООП), реализуют алгоритмы, описывающие взаимодействие между объектами.

Класс содержит константы и переменные, называемые полями, а также выполняемые над ними операции и функции. Функции класса называются методами. Предполагается, что доступ к полям класса возможен только через вызов соответствующих методов. Поля и методы являются элементами, или членами класса.

Рассмотрим структуру объявления классов:

```
1    // объявление классов в C++
2    class /*имя класса*/
3    {
4      private:
5      /* список свойств и методов для использования внутри класса */
6      public:
7      /* список методов доступных другим функциям и объектам программы */
8      protected:
9      /*список средств, доступных при наследовании*/
10   }
     ;
```

Объявление класса начинается с зарезервированного ключевого слова **class,** после которого пишется имя класса.

В фигурных скобочках, **строки 3 — 10** объявляется тело класса, причём после закрывающейся скобочки обязательно нужно ставить точку с запятой, **строка 10**.

В теле класса объявляются три метки спецификации доступа, **строки 4, 6, 8,** после каждой метки нужно обязательно ставить двоеточие.

В **строке 4** объявлена метка спецификатора доступа **`private`.** Все методы и свойства класса, объявленные после спецификатора доступа **`private`** будут доступны только внутри класса.

В **строке 6** объявлен спецификатор доступа **`public`,** все методы и свойства класса, объявленные после спецификатора доступа **`public`** будут доступны другим функциям и объектам в программе.

Спецификатор доступа **`protected`** используется в случае наследования классов.

При объявлении класса, не обязательно объявлять три спецификатора доступа, и не обязательно их объявлять в таком порядке. Но лучше сразу определиться с порядком объявления спецификаторов доступа, и стараться его придерживаться.

Разработаем программу, в которой объявим простейший класс, в котором будет объявлена одна функция, печатающая сообщение.

```
1    // classes.cpp: определяет точку входа для консольного приложения.
2
3    #include "stdafx.h"
4    #include <iostream>
```

```
5     using namespace std;
6     // начало объявления класса
7     class CppStudio // имя класса
8     {
9     public: // спецификатор доступа
10        void message() // функция (метод класса) выводящая сообщение на экран
11        {
12            cout << "website: cppstudio.com\ntheme: Classes and Objects in C + +\n";
13        }
14    }; // конец объявления класса CppStudio
15
16    int main(int argc, char* argv[])
17    {
18        CppStudio objMessage; // объявление объекта
19        objMessage.message(); // вызов функции класса message
          system("pause");
          return 0;
      }
```

В **строках 7 — 14** мы определили класс с именем **CppStudio**. Имя класса принято начинать с большой буквы, последующие слова в имени также должны начинаться с большой буквы. Такое сочетание букв называют верблюжьим регистром, так как чередование больших и маленьких букв напоминает силуэт верблюда.

В теле класса объявлен спецификатор доступа **`public`**, который позволяет вызывать другим функциям методы класса, объявленные после **`public`**. Вот именно поэтому в главной функции, в **строке 19** мы смогли вызвать функцию **`message().`**

В классе **`CppStudio`** объявлена всего одна функция, которая не имеет параметров и выводит сообщение на экран, строка **12**. Методы класса — это те же функции, только объявлены они внутри класса, поэтому всё что относится к функциям актуально и для методов классов. Объявление классов выполняется аналогично объявлению функций.

В **строке 18** объявлена переменная **`objMessage`** типа **`CppStudio,`** так вот, переменная **`objMessage`** — это объект класса **`CppStudio`**. Таким образом, класс является сложным типом данных. После того как объект класса объявлен, можно воспользоваться его методами. Метод всего один — функция **`message().`** Для этого обращаемся к методу объекта **`objMessage`** через точку, как показано в **строке 19**, в результате программа выдаст текстовое сообщение:

```
website: cppstudio.com
theme: Classes and Objects in C + +
```

2.Порядок выполнения работы

2.1 Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

2.2 Содержание отчёта

1. Номер работы и её название.

2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.
6. Результат тестирования программы.

3.Вопросы к защите

1. Что в C++ называют классом?
2. Что такое объекты класса?
3. Каков формат определения классов в C++?
4. Что такое методы класса?
5. Какие спецификаторы доступа определены в C++?

4.Задание к выполнению

- Разработать алгоритм и программу, реализующую простой класс на основе заданной структуры данных в соответствии с индивидуальным заданием.
- Разработать набор функций – членов класса вида (get, set и show).
- Функция main должна иллюстрировать использование разработанного класса.

Варианты индивидуальных заданий

Вариант №1

Отдел кадров			
Фамилия	Инициалы	Год рожд	Оклад
Иванов	И.И.	1975	517.50
Петренко	П.П.	1956	219.10
Паниковский	М.С.	1967	300.00
Примечание: оклад установлен по состоянию на 1 января 2000 года			

Вариант №2

Ведомость деталей			
Наименование	Тип	Количество	Вес 1 детали (г)
Фланец	З	3	450
Переходник	П	8	74
Станина	О	1	117050
Примечание: принято такое кодирование типов: О - оригинальная, П - покупная, З - заимствованная			

Вариант №3

Каталог библиотеки			
Автор книги	Название	Год выпуска	Группа
Сенкевич	Потоп	1978	X

Ландау	Механика	1989	У
Дойль	Сумчатые	1990	С
Примечание: Х - художественная литература; У - учебная литература; С - справочная литература			

Вариант №4

Сельскохозяйственные культуры			
Наименование	Тип	Посевная площадь (га)	Урожайность (ц/га)
Соя	Б	13000	45
Чумиза	З	8000	17
Рис	З	25650	24
Примечание: З - зерновые, Б - бобовые			

Вариант №5

Ведомость спортивных состязаний			
Фамилия участника	Код команды	Количество балов	Место в итоге
Баландин	С	123.7	2
Шишков	Ш	79.98	3
Кравченко	Д	134.8	1
Примечание: Д - "Динамо", С - "Спартак", Ш - "Шахтер"			

Вариант №6

Ведомость общественного транспорта			
Вид транспорта	№ маршрута	Протяженность маршрута (км)	Время в дороге (мин)
Трамвай	12	27.55	75
Троллейбус	17	13.6	57
Автобус	12а	57.3	117
Примечание: Тр - трамвай, Тс - троллейбус, А - автобус			

Вариант №7

Сельскохозяйственные предприятия			
Название	Вид собственности	Площадь земли (га)	Кол. работников
Заря	Г	300	120
Росинка	К	174	27
Петренко	Ч	56	6
Вид собственности: Г - государственная, Ч - частная, К - кооперативная			

Вариант №8

Прайс-лист			
Наименование товара	Тип товара	Цена за 1 шт (руб.)	Минимальное количество в партии
Папка	К	4.75	4
Бумага	К	13.90	10
Калькулятор	О	411.00	1
Примечание: К - канцтовары, О - оргтехника			

Вариант №9

Вязкость металлов в жидком состоянии			
Вещество	Атомный номер	Температура (град.С)	Вязкость (кг/м*сек)
Алюминий	13	700	2.90
Висмут	83	304	1.65
Свинец	82	441	2.11
Примечание: данные даны для температуры плавления			

Вариант №10

Коэффициенты теплопроводности материалов			
Вещество	Тип	Влажность (%)	Коэффициент
Алюминий	М	0-100	209.3
Стекловата	Т	0-100	0.035
Глина	Д	15-20	0.73
Примечание: М - металлы, Т - термоизоляционные материалы, Д - другие материалы			

Вариант №11

Температура перехода веществ в сверхпроводниковое состояние		
Вещество	Тип	Температура
Zn	М	0.8-0.85
Pb-Au	П	2.0-7.3
NbC	С	10.1-10.5
Тип вещества: М - металл, П - сплав, С - соединение		

Вариант №12

Сплавы с высоким сопротивлением			
Сплав	Сопротивление	Темп.коэфф.сопр	Макс.температура
Константан	0.44	0.00001	500
Никелин	0.39	0.39	150
Февраль	1.1	0.0001	900
Единицы измерения: сопротивление - ом*кв.мм/м. Коэффициент сопротивления - 1/град. Температура - град.С			

Вариант №13

Элементарные частицы			
Частица	Группа	Заряд	Масса покоя
Нейтрон	Н	0	940
Ка-плюс	М	+1	494
Электрон	Л	-1	0.511
Группы частиц: Г - гипероны, Н - нуклоны, М - мезоны, Л - лептоны			

Вариант №14

Некоторые виды антилоп			
Название	Группа	Место обитания	Численность популяции
Джейран	А	Азия	30000
Гну	В	Африка	560000
Бейза	Н	Африка	2500
Группы: А - настоящие антилопы, В - коровьи антилопы, Н - лошадиные антилопы			

Вариант №15

Японские острова			
Остров	Площадь (кв.км)	Кол. малых островов	Протяженность береговой линии (км)
Хонсю	230500	192	11875
Хоккайдо	78500	75	3072
Сикоку	18800	13	2946

5.Пример программы

5.1.Условие задачи.

- Разработать алгоритм и программу, реализующую простой класс на основе заданной структуры данных в соответствии с индивидуальным заданием.
- Разработать набор функций – членов класса вида (**`get, set и show`**).
- Функция **`main`** должна иллюстрировать использование разработанного класса.

Буддийские монастыри Японии периода Нара			
Название	Школа	Количество монахов	Площадь земли (га)
Тодайдзи	Т	220	368.8
Якусидзи	С	50	54.7
Дайандзи	Д	10	12.2
Примечание: Т - Тэндай; С - Сингон; Д - Дзедзицу			

5.2. Описание структуры класса.

5.2.1. Описание полей класса.

Начнем разработку класса с выбора типа данных для его полей. В соответствии с заданием наш класс должен иметь следующие поля:

- *Название.*
- *Школа.*
- *Количество монахов.*
- *Площадь земли.*

- *Название.* Тип этого поля необходимо определить как ***string*** , так как это позволит размещать строки различной длины, не обращая внимания на возможность выхода за границы массива.
- *Школа* . Данное поле будет хранить всего лишь один символ, поэтому его тип мы определим как ***char*** .
- *Количество монахов.* Для хранения данных данного поля достаточно типа ***int*** и так эти данные, не могут принимать отрицательные значения, то для исключения ошибок тип определим как ***unsigned int***.
- *Площадь земли.* Тип данного поля определим однозначно - ***float***.

Итак, таким образом, мы имеем класс с полями:

```
class church {
        string name;    //Название
        char school;    //Школа
        unsigned int count;    //Количество монахов
        float square;    //Площадь земли
.
.
.
}
```

5.2.2. Компонентные функции.

В соответствии с условиями задания наш класс должен иметь три функции:

- **set()**
- **get()**
- **show()**

Начнем с функции **set()** . Задача этой функции - считать данные, вводимые пользователем с клавиатуры, и записать их в поля класса. Т.е. для работы с функцией необходимо определить ряд переменных, типы которых соответствуют типам полей класса.

Определим эти переменные:

```
string n;    //Название
char t;    //Школа
unsigned int s;    //Количество монахов
float h;    //Площадь земли
```

Присваивание значений полям класса производится как обычно с помощью оператора равенства =. Т.е форма записи будет выглядеть следующим образом:

поле_класса = определенная_переменная.

Далее приступим к функции **get()**. Она отвечает за считывание значений из полей класса. Для записи этих значений нам опять подойдут переменные, определенные выше. Работа функции **get()** полностью аналогична работе функции **set()** за исключением того, что в роли приёмника значений выступают переменные, а в роли передатчика - поля класса. Т.е форма записи будет выглядеть следующим образом:

определенная_переменная=поле_класса

Третья функция **show()** производит вывод на экран значений полей класса и ее реализация не составляет особого труда.

5.3.Текст программы

```cpp
#include <iostream>
#include <string>
#include<iomanip>
#include <Windows.h>
using namespace std;
#define N 3

class church
{
    string name;
    char school;
    unsigned int count;
    float square;
public:
    void set(string a, char b, unsigned int c, float d);
    void get(string a, char& b, unsigned int& c, float& d);
    void show(void);
};

void church::set(string a, char b, unsigned int c, float d) {
    name=a;
    school = b;
    count = c;
    square = d;
}

void church::get(string a, char& b, unsigned int& c, float& d)
{
    a = name;
    b = school;
    c = count;
    d = square;
}

void church::show(void)
{
    cout << setw(8)<<name << " ";
    cout << setw(4) << school << " ";
    cout << setw(12) << count << " ";
    cout << setw(17) << square << " ";
}
```

```
int main(void)
{
    string n;
    char t;
    unsigned int s;
    float h;
    short i;
    church obj[N];

    SetConsoleCP(1251);// установка кодовой страницы win-cp 1251 в поток ввода
    SetConsoleOutputCP(1251); // установка кодовой страницы win-cp 1251 в поток вывода
    cout << "Работа функции SET!\n";
    for (i = 0; i < N; i++)
    {
        cout << "Название, Школа, Количество монахов, Площадь земли: ";
        cin >> n;
        cin >> t;
        cin >> s;
        cin >> h;
        obj[i].set(n, t, s, h);
    }
    cout << "\nРабота функции SHOW!\n";
    cout << "Название, Школа, Количество монахов, Площадь земли: \n";
    for (i = 0; i < N; i++)
    {
        obj[i].show();
        cout << "\n";
    }
    cout << "\nРабота функции GET и SHOW!\n";
    cout << "Название, Школа, Количество монахов, Площадь земли: \n";
    for (i = 0; i < N; i++)
    {
        obj[i].get(n, t, s, h);
        obj[i].show();
        cout << "\n";
    }
}
```

5.4.Тестирование программы

Результат работы программы:

```
Работа функции SET!
Название, Школа, Количество монахов, Площадь земли: Тодайдзи Т 220 368.8
Название, Школа, Количество монахов, Площадь земли: Якусидзи С 50 54.7
Название, Школа, Количество монахов, Площадь земли: Дайандзи Д 10 12.2

Работа функции SHOW!
Название, Школа, Количество монахов, Площадь земли:
Тодайдзи    Т          220             368.8
Якусидзи    С           50              54.7
Дайандзи    Д           10              12.2

Работа функции GET и SHOW!
Название, Школа, Количество монахов, Площадь земли:
Тодайдзи    Т          220             368.8
Якусидзи    С           50              54.7
Дайандзи    Д           10              12.2
```

ЛАБОРАТОРНАЯ РАБОТА №27. КЛАССЫ. ПЕРЕЗАГРУЗКА

1. Краткие сведения из теории

Перегрузка операторов позволяет определить действия, которые будет выполнять оператор. Перегрузка подразумевает создание функции, название которой содержит слово **operator** и символ перегружаемого оператора. Функция оператора может быть определена как член класса, либо вне класса.

Перегрузить можно только те операторы, которые уже определены в C++. Создать новые операторы нельзя.

Если функция оператора определена как отдельная функция и не является членом класса, то количество параметров такой функции совпадает с количеством операндов оператора. Например, у функции, которая представляет унарный оператор, будет один параметр, а у функции, которая представляет бинарный оператор, - два параметра. Если оператор принимает два операнда, то первый операнд передается первому параметру функции, а второй операнд - второму параметру. При этом как минимум один из параметров должен представлять тип класса

Рассмотрим пример с классом `Counter`, который представляет секундомер и хранит количество секунд:

```
#include <iostream>
class Counter
{
public:
    Counter(int sec)
    {
        seconds = sec;
    }
    void display()
    {
       cout << seconds << " seconds" <<endl;
    }
int seconds;
};
Counter operator + (Counter c1, Counter c2)
{
    return Counter(c1.seconds + c2.seconds);
}
int main()
{
    Counter c1(20);
    Counter c2(10);
    Counter c3 = c1 + c2;
    c3.display();    // 30 seconds
    return 0;
}
```

Здесь функция оператора не является частью класса Counter и определена вне его. Данная функция перегружает оператор сложения для типа Counter. Она является бинарной, поэтому принимает два параметра. В данном случае мы складываем два объекта Counter. Возвращает функция также объект Counter, который хранит общее количество секунд. То есть по сути здесь операция сложения сводится к сложению секунд обоих объектов:

```
Counter operator + (Counter c1, Counter c2)
{
    return Counter(c1.seconds + c2.seconds);

}
```

При этом необязательно возвращать объект класса. Это может быть и объект встроенного примитивного типа. И также мы можем определять дополнительные перегруженные функции операторов:

```
int operator + (Counter c1, int s)
{
    return c1.seconds + s;

}
```

Данная версия складывает объект Counter с числом и возвращает также число. Поэтому левый операнд операции должен представлять тип Counter, а правый операнд - тип int. И, к примеру, мы можем применить данную версию оператора следующим образом:

```
Counter c1(20);
int seconds = c1 + 25;  // 45

cout << seconds <<endl;
```

Также функции операторов могут быть определены как члены классов. Если функция оператора определена как член класса, то левый операнд доступен через указатель this и представляет текущий объект, а правый операнд передается в подобную функцию в качестве единственного параметра:

```
#include <iostream>
class Counter
{
public:
    Counter(int sec)
    {
        seconds = sec;
    }
    void display()
    {
        std::cout << seconds << " seconds" << std::endl;
    }
    Counter operator + (Counter c2)
    {
        return Counter(this->seconds + c2.seconds);
    }
    int operator + (int s)
    {
        return this->seconds + s;
    }
int seconds;
};

int main()
{
    Counter c1(20);
```

```
Counter c2(10);
    Counter c3 = c1 + c2;
    c3.display();            // 30 seconds
    int seconds = c1 + 25;   // 45

    return 0;
}
```

В данном случае к левому операнду в функциях операторов мы обращаемся через указатель `this`.

Какие операторы где переопределять? Операторы присвоения, индексирования ([ ]), вызова (( )), доступа к члену класса по указателю (->) следует определять в виде функций-членов класса. Операторы, которые изменяют состояние объекта или непосредственно связаны с объектом (инкремент, декремент), обычно также определяются в виде функций-членов класса. Все остальные операторы чаще определяются как отдельные функции, а не члены класса.

2. Порядок выполнения работы

2.1 Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

2.2 Содержание отчёта

1. Номер работы и её название.
2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.
6. Результат тестирования программы.

3.Вопросы к защите

1. Что называют перегрузкой операторов?
2. Какие операторы можно перегружать?
3. Какое количество параметров должно быть у перегружаемой функции?
4. Какие операторы нужно переопределять в виде методов класса, а какие как отдельные функции?

4.Задание к выполнению

Разработать алгоритм и программу, реализующую простой класс на основе заданной структуры данных в соответствии с индивидуальным заданием. Варианты индивидуальных заданий взять из **лабораторной работы № 26**. В программе необходимо предусмотреть функцию доступа к полям класса (функцию ***get()***) и необходимо перегрузить следующие операторы:

- Сложение (в случае символьных полей сложение понимается как конкатенация);
- Соответствие.

5.Пример программы

5.1.Условие задачи.

Разработать алгоритм и программу, реализующую простой класс на основе заданной структуры данных. В программе необходимо предусмотреть функцию доступа к полям класса (функцию ***get()***) и необходимо перегрузить следующие операторы:

- Сложение (в случае символьных полей сложение понимается как конкатенация);
- Соответствие.

Буддийские монастыри Японии периода Нара			
Название	Школа	Количество монахов	Площадь земли (га)
Тодайдзи	Т	220	368.8
Якусидзи	С	50	54.7
Дайандзи	Д	10	12.2
Примечание: Т - Тэндай; С - Сингон; Д - Дзедзицу			

5.2. Разработка текста программы

Для удобства работы с программой введём переменную-константу, которая будет отвечать за количество экземпляров класса **`church.`** То есть будем работать с массивом объектов класса **`church`** размерностью ***N***.

```
#define N 3
```

Перейдём к описанию объявленных выше функций:

Отметим, что функции ***`get, show`*** взяты из предыдущей лабораторной работы и остались без изменений.

```
int operator == (church &o1);
```

Функция позволяет установить соответствие между экземплярами одного класса. Т.е. её тело включает проверку на равенство значений числовых полей и сравнение содержимого строк. В зависимости от того, тождественны ли экземпляры функцией возвращается 0 или 1.

```
church operator + (church &o1);
```

Функция ***operator+()*** возвращает объект типа ***church*** Отметим тот факт, что временный объект ***tr*** используется внутри функции ***operator+()*** для хранения результата и является возвращаемым объектом. Отметим также, что ни один из операндов не меняется.

В данной ситуации (как и в большинстве ситуаций) оператор + был перегружен способом, аналогичным своему традиционному арифметическому использованию. Поэтому и было важно, чтобы ни один из операндов не менялся. Например, когда вы складываете 10+4, результат равен 14, но ни 10, ни 4 не меняются. Таким образом, временный объект необходим для хранения результата.

5.3.Текст программы

```
#include <iostream>
#include <string>
#include<iomanip>
#include <Windows.h>
using namespace std;
using std::string;
#define N 3

class church
{
    string name;
    string school;
    unsigned int count;
    float square;
public:
    church() { square = 0; count = 0; }
    church(string a, string b, unsigned int c, float d);
    void get(string a, string b, unsigned int& c, float& d);
    void show(void);
    int    operator == (church& o1);
    church operator + (church& o1);
};
church church::operator + (church& o1) {
    church tr;
    int i, j;
    tr.name= name+o1.name;
    tr.school= school+ o1.school;
    tr.count = count + o1.count;
    tr.square = square + o1.square;
    return tr;
}

int church::operator == (church& o1) {
    if (count != o1.count) { cout << "Данные экземпляры класса не равны.";  }
    else if (ceil(square) != ceil(o1.square)) { cout << "Данные экземпляры класса не равны.";  }
    else if (name!= o1.name) { cout << "Данные экземпляры класса не равны.";  }
    else if (school!= o1.school) { cout << "Данные экземпляры класса не равны.";  }
    else cout << "Экземпляры класса равны.";
    return 0;
}
```

```
void church::get(string a, string b, unsigned int& c, float& d)
{
    a = name;
    b = school;
    c = count;
    d = square;
}

void church::show(void)
{
    cout << setw(8) << name << " ";
    cout << setw(4) << school << " ";
    cout << setw(12) << count << " ";
    cout << setw(17) << square << " ";
}
church::church(string a, string b, unsigned int c, float d) {
    name = a;
    school = b;
    count = c;
    square = d;
}
int main(void)
{
    string n;
    string t;
    unsigned int s;
    float h;
    short i, q, q1;

    church obj[N] = { church("Тодайдзи","Т",220,368.8),
              church("Якусидзи","С",50,54.7),
              church("Дайандзи","Д",10,12.2) };

    SetConsoleCP(1251);// установка кодовой страницы win-cp 1251 в поток ввода
    SetConsoleOutputCP(1251); // установка кодовой страницы win-cp 1251 в поток вывода
    cout << "\nРабота функции SHOW!\n";
    cout << "Название, Школа, Количество монахов, Площадь земли: \n";
    for (i = 0; i < N; i++)
    {
        obj[i].show();
        cout << "\n";
    }
    cout << "\nПерегрузка оператора соответствия '=='.\n";
    cout << "Введите номера экземпляров класса, которые надо сравнить>\n";
    cin >> q;
    cin >> q1;
    obj[q] == obj[q1];
    cout << "\nПерегрузка оператора суммы '+'.\n";
    cout << "Введите номера экземпляров класса, которые надо сложить>\n";
    cin >> q;
    cin >> q1;
    church temp;

    temp = obj[q] + obj[q1];
    cout << "Название, Школа, Количество монахов, Площадь земли: \n";
    temp.get(n, t, s, h);
    temp.show();
}
```

5.4.Тестирование программы

Результат работы программы может выглядеть так:

```
Название, Школа, Количество монахов, Площадь земли:
Тодайдзи    Т           220              368.8
Якусидзи    С            50               54.7
Дайандзи    Д            10               12.2

Перегрузка оператора соответствия '=='.
Введите номера экземпляров класса, которые надо сравнить>
1
1
Экземпляры класса равны.

Перегрузка оператора суммы '+'.
Введите номера экземпляров класса, которые надо сложить>
0
1
Название, Школа, Количество монахов, Площадь земли:
ТодайдзиЯкусидзи   ТС          270              423.5

Название, Школа, Количество монахов, Площадь земли:
Тодайдзи    Т           220              368.8
Якусидзи    С            50               54.7
Дайандзи    Д            10               12.2

Перегрузка оператора соответствия '=='.
Введите номера экземпляров класса, которые надо сравнить>
0
1
Данные экземпляры класса не равны.

Перегрузка оператора суммы '+'.
Введите номера экземпляров класса, которые надо сложить>
2
2
Название, Школа, Количество монахов, Площадь земли:
ДайандзиДайандзи   ДД          20               24.4
```

ЛАБОРАТОРНАЯ РАБОТА №28. НАСЛЕДОВАНИЕ. СОЗДАНИЕ КЛАССОВ, ИЕРАРХИЧЕСКИ СВЯЗАННЫХ МЕЖДУ СОБОЙ

1. Краткие сведения из теории

Наследование (`inheritance`) представляет один из ключевых аспектов объектно-ориентированного программирования, который позволяет наследовать функциональность одного класса или базового класса (`base class`) в другом - производном классе (`derived class`).

Зачем нужно наследование? Рассмотрим небольшую ситуацию, допустим, у нас есть классы, которые представляют человека и работника предприятия:

```
class Person
{
public:
    string name;        //  имя
    int age;            // возраст
    void display()
    {
     cout << "Name: " << name << "\tAge: " << age <<endl;
    }
};
class Employee
{
public:
    string name;        // имя
    int age;             // возраст
    string company;     // компания
    void display()
    {
     cout << "Name: " << name << "\tAge: " << age <<endl;
    }
};
```

В данном случае класс **`Employee`** фактически содержит функционал класса **`Person:`** свойства **`name и age`** и функцию **`display`**. И было бы не совсем правильно повторять функциональность одного класса в другом классе, тем более что по сути сотрудник предприятия в любом случае является человеком. Поэтому в этом случае лучше использовать механизм наследования. Унаследуем класс **`Employee`** от класса **`Person:`**

```
class Person
{
public:
    string name;        //  имя
    int age;                 // возраст
    void display()
    {
      cout << "Name: " << name << "\tAge: " << age << endl;
    }
};
class Employee : public Person
{public:
    string company; // компания
};
```

Для установки отношения наследования после названия класса ставится двоеточие, затем идет название класса, от которого мы хотим унаследовать функциональность. В этом отношении класс `Person` еще будет называться базовым классом, а `Employee` - производным классом.

Перед названием базового класса также можно указать спецификатор доступа, как в данном случае используется спецификатор **`public`**, который позволяет использовать в производном классе все открытые члены базового класса. Если мы не используем модификатор доступа, то класс `Employee` ничего не будет знать о переменных `name и age` и функции `display`.

После установки наследования мы можем убрать из класса Employee те переменные, которые уже определены в классе `Person`. Используем оба класса:

```
#include <iostream>
#include <string>

class Person
{
public:
    string name;        //  имя
    int age;                // возраст
    void display()
    {
      cout << "Name: " << name << "\tAge: " << age << endl;
    }
};
class Employee : public Person
{
public:
    string company;     // компания
};

int main()
{
    Person tom;
    tom.name = "Tom";
    tom.age = 23;
    tom.display();

    Employee bob;
    bob.name = "Bob";
    bob.age = 31;
    bob.company = "Microsoft";
    bob.display();

    return 0;

}
```

Таким образом, через переменную класса `Employee` мы можем обращаться ко всем открытым членам класса `Person`.

2.Порядок выполнения работы

2.1 Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

2.2 Содержание отчёта

1. Номер работы и её название.
2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.
6. Результат тестирования программы.

3.Вопросы к защите

1. Что такое наследование классов?
2. Для чего предназначено наследование классов?
3. Каков формат описания класса-наследника?
4. Какой спецификатор доступа используется при наследовании?

4.Задание к выполнению

Разработать программу, реализующую базовый класс и класс-наследник в соответствии с индивидуальным заданием.

Варианты индивидуальных заданий:

№ варианта	базовый класс	класс-наследник	поля	методы
1	Прямоугольный параллелепипед	куб	длины сторон	– вычисление длин диагоналей – вычисление площадей граней – вычисление объема – вычисление площади поверхности
2	Треугольник	Равносторонний треугольник	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление углов (в градусах) – вычисление периметра – вычисление площади
3	Прямая треугольная призма	Правильная треугольная призма	длины сторон	– вычисление площадей граней – вычисление диагоналей боковых граней – вычисление объема – вычисление полной площади поверхности

4	параллелограмм	ромб	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление длин диагоналей – вычисление углов (в градусах) – вычисление периметра – вычисление площади
5	Прямая четырехугольная призма	Правильная четырехугольная призма	длины сторон	– вычисление площадей граней – вычисление диагоналей боковых граней – вычисление объема – вычисление полной площади поверхности
6	параллелограмм	прямоугольник	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление длин диагоналей – вычисление углов (в градусах) – вычисление периметра – вычисление площади
7	Четырехугольная пирамида, у которой все боковые ребра равны.	Правильная четырехугольная пирамида	длины сторон	– вычисление площадей граней – вычисление объема – вычисление полной площади поверхности
8	трапеция	Равнобедренная трапеция	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление длин диагоналей – вычисление периметра – вычисление площади
9	Правильная треугольная пирамида	тетраэдр	длины сторон	– вычисление площадей граней – вычисление объема – вычисление полной площади поверхности
10	Выпуклый четырёхугольник	квадрат	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление длин диагоналей – вычисление углов (в градусах) – вычисление периметра – вычисление площади
11	Прямой круговой цилиндр	конус	Радиус основания, высота	– вычисление площади основания – вычисление объема – вычисление полной площади поверхности

12	параллелограмм	квадрат	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление длин диагоналей – вычисление углов (в градусах) – вычисление периметра – вычисление площади
13	трапеция	Прямоугольная трапеция	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление длин диагоналей – вычисление периметра – вычисление площади
14	треугольник	Прямоугольный треугольник	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление углов (в градусах) – вычисление периметра – вычисление площади
15	Треугольник	Равнобедренный треугольник	координаты вершин, заданные в порядке их обхода	– вычисление длин сторон – вычисление углов (в градусах) – вычисление периметра – вычисление площади

5.Пример программы
5.1.Условие задачи.

Написать программу, реализующую базовый класс «выпуклый четырехугольник» (поля типа «координаты вершин, заданные в порядке их обхода») и класс-наследник «параллелограмм». Описать для указанных фигур методы «вычисление углов» (в градусах), «вычисление диагоналей», «вычисление длин сторон», «вычисление периметра», «вычисление площади».

5.2.Текст программы

```
//Наследование
#include<iostream>
#include<math.h>
#include<locale.h>
using namespace std;
//Объявление базового класса Четырёхугольников
class FourAngle
{
protected:
    double x1, y1, x2, y2, x3, y3, x4, y4,
        a, b, c, d, d1, d2,
        alpha, beta, gamma, delta,
        P, S;
public:
    void Init(void);
    void Storony(void);
    void Diagonali(void);
    void Angles(void);
    void Perimetr(void);
    void Ploshad(void);
    void Print(void);
};
```

```
//Объявление класса Параллелограммов - наследника Четырёхугольников
class Parall : public FourAngle
{
public:
    void Storony(void);
    void Perimetr(void);
    void Ploshad(void);
};

//Описание методов класса Четырёхугольников
void FourAngle::Init(void)
{
    cout << "\n   Введите координаты вершин:\n  ";
    cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;
}
void FourAngle::Storony(void)
{
    a = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    b = sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));
    c = sqrt((x3 - x4) * (x3 - x4) + (y3 - y4) * (y3 - y4));
    d = sqrt((x1 - x4) * (x1 - x4) + (y1 - y4) * (y1 - y4));
}
void FourAngle::Diagonali(void)
{
    d1 = sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
    d2 = sqrt((x2 - x4) * (x2 - x4) + (y2 - y4) * (y2 - y4));
}
//Ugol - вспомогательная функция для метода Angles
double Ugol(double Aa, double Bb, double Cc)
{
    double Pi;
    Pi = 4 * atan(1.0);
    return acos((Aa * Aa + Bb * Bb - Cc * Cc) / 2 / Aa / Bb) / Pi * 180;
}
void FourAngle::Angles(void)
{
    alpha = Ugol(d, a, d2); beta = Ugol(a, b, d1);
    gamma = Ugol(b, c, d2); delta = Ugol(c, d, d1);
}
void FourAngle::Perimetr(void)
{
    P = a + b + c + d;
}
void FourAngle::Ploshad(void)
{
    double p1, p2;
    p1 = (a + d + d2) / 2;
    p2 = (b + c + d2) / 2;
    S = sqrt(p1 * (p1 - a) * (p1 - d) * (p1 - d2)) +
        sqrt(p2 * (p2 - b) * (p2 - c) * (p2 - d2));
}
void FourAngle::Print(void)
{
    cout << "\n   Стороны:\n  " << a << "  " << b << "  " << c << "  " << d << "\n";
    cout << "   Углы:\n  " << alpha << "  " << beta << "  "
        << gamma << "  " << delta << "\n";
    cout << "   Периметр:\n  " << P << "\n";
    cout << "   Площадь:\n  " << S << "\n";
    cout << "   Диагонали:\n  " << d1 << "  " << d2 << "\n";
}
```

```
//Описание методов класса Параллелограмм
void Parall::Storony(void)
{
    a = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    b = sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));
    c = a;
    d = b;
}
void Parall::Perimetr(void)
{
    P = 2 * (a + b);
}
void Parall::Ploshad(void)
{
    double p1;
    p1 = (a + b + d2)/2;
    S = 2 * sqrt(p1 * (p1 - a) * (p1 - b) * (p1 - d2));
}
//Основная функция
//По координатам вершин заданной фигуры
//вычисляет все его параметры
void main(void)
{
    setlocale(LC_ALL, ".1251");

    FourAngle obj; //Объект класса Четырёхугольник
    cout << "\n  Выпуклый четырёхугольник\n";
    obj.Init();
    obj.Storony();
    obj.Diagonali();
    obj.Angles();
    obj.Perimetr();
    obj.Ploshad();
    obj.Print();

    Parall obj1; //Объект класса Параллелограмм
    cout << "\n  Параллелограмм\n";
    obj1.Init();
    obj1.Storony();
    obj1.Diagonali();
    obj1.Angles();
    obj1.Perimetr();
    obj1.Ploshad();
    obj1.Print();
}
```

5.3.Тестирование программы

Результат работы программы может выглядеть так:

```
Выпуклый четырёхугольник

 Введите координаты вершин:
0 0 0 1 2 5 1 1

 Стороны:
1  4.47214  4.12311  1.41421
 Углы:
45  153.435  12.5288  149.036
 Периметр:
11.0095
 Площадь:
2.5
 Диагонали:
5.38516  1

Параллелограмм

 Введите координаты вершин:
0 0 1 1 4 1 3 0

 Стороны:
1.41421  3  1.41421  3
 Углы:
45  135  45  135
 Периметр:
8.82843
 Площадь:
3
 Диагонали:
4.12311  2.23607
```

ЛАБОРАТОРНАЯ РАБОТА №29. ПРОСТЕЙШИЕ ПРОГРАММЫ С ИСПОЛЬЗОВАНИЕМ ВИЗУАЛЬНЫХ КОМПОНЕНТОВ

1. Краткие сведения из теории

Основной частью визуального приложения является форма. Поэтому при программировании очень важно уделять внимание внешнему виду форм и функциям. Итак, форма представляет собой пустую доску, которую разработчик оснащает элементами управления, формируя интерфейс пользователя, и кодом для управления данными. Для этого Visual Studio обеспечивает интегрированную среду разработки, способствующую написанию кодов, а также расширенный набор элементов управления .NET Framework. Дополняя функциональными возможностями этих элементов управления свои коды, пользователь может легко и быстро разработать необходимое приложение.

При создании проекта в Visual C++ создается объект, называемый **формой.**
Форма – это главный контейнер, на котором размещаются компоненты среды разработки. Как и любой объект, форма имеет свойства, методы и события.

Элемент управления `TextBox`

Компонент находится в списке `All Windows Forms` палитры компонентов. Он задает многострочное или однострочное редактируемое поле, через которые вводят (выводят) строчные данные.

Текст, отображаемый в элементе управления, содержится в свойстве `Text`. По умолчанию в текстовом поле можно ввести до 2048 знаков. Свойство `Text` может быть установлено в окне «Свойства» во время разработки, программными средствами во время выполнения или в результате ввода данных пользователем во время выполнения.

Текущее содержимое текстового поля может быть получено во время выполнения путем считывания значения свойства `Text`.

В следующем примере кода текст помещается в элемент управления во время выполнения.

Пример записи текста в элемент управления `TextBox`, имеющего имя `TextBox1`.

```
textBox1->Text = "Это элемент управления TextBox.";
```

Элемент управления `Button`

Элемент управления `Windows Forms Button` (кнопка) служит для выполнения действия с помощью мыши. На элементе управления `Button` может отображаться как текст, так и рисунок. Если нажать кнопку, она выглядит так, словно она нажата и отпущена. При нажатии на кнопку вызывается обработчик событий `Click`. В обработчик событий `Click` помещается код, отвечающий за выполнение нужного действия.

Текст, отображающийся на кнопке, содержится в свойстве `Text`. Внешний вид текста управляется свойством `Font` и свойством `TextAlign`.

Чаще всего элемент управления `Button в Windows Forms` используется для выполнения какой-либо программы при нажатии кнопки.

Элемент управления Label

Элементы управления `Windows Forms Label` предназначены для отображения текста или изображений, которые пользователь не может изменить с клавиатуры. Они используются для идентификации объектов в форме, например, для описания, что произойдет с элементом управления после выполнения на нем щелчка мышью, или для отображения сведений в ответ на процесс или событие времени выполнения в приложении. Например, имеется возможность использовать надписи для добавления описательных заголовков в текстовые поля, списки,

поля со списком и т. д. Кроме того, возможно написание кода, который изменяет текст, отображаемый в надписи, в ответ на события во время выполнения.

2.Порядок выполнения работы

2.1.Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

2.2 Содержание отчёта

1. Номер работы и её название.
2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.
6. Схема компонента `Forms`
7. Результат тестирования программы.

3.Вопросы к защите

1. Как создать проект Windows Forms и его сохранение?
2. Компонент Forms и его основные свойства.
3. Компонент Label и его основные свойства.
4. Компонент TextBox и его основные свойства.
5. Компонент Button и его основные свойства.

4.Задание к выполнению

Составить алгоритм и написать программу с использованием визуальных компонентов в среде разработки Visual C++ в соответствии с индивидуальным заданием. Программа должна быть написана таким образом, что в поля редактирования пользователь может вводить только правильные данные – дробные числа.

Список индивидуальных заданий лабораторной работы № 29

1. Вычислить периметр и площадь прямоугольного треугольника по длинам *a* и *b* двух катетов.
2. Заданы координаты трех вершин треугольника (x_1, y_1), (x_2, y_2), (x_3, y_3). Найти его периметр и площадь.
3. Вычислить длину окружности и площадь круга одного и того же заданного радиуса *R*.
4. Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геометрическое модулей этих чисел.
5. Вычислить расстояние между двумя точками с данными координатами (x_1, y_1) и (x_2, y_2).
6. Даны два действительных числа *x* и *y*. Вычислить их сумму, разность, произведение и частное.
7. Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.
8. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.

9. Найти площадь кольца, внутренний радиус которого равен r, а внешний – R $(R>r)$.
10. Найти площадь треугольника, две стороны которого равны a и b, а угол между этими
а. сторонами γ.
11. Составить программу перевода радианной меры угла в градусы и минуты.
12. Вычислить высоты треугольника со сторонами a, b, c.
13. Составить программу вычисления объема цилиндра и конуса, которые имеют одинаковую высоту H и одинаковый радиус основания R.
14. Вычислить объем и площадь поверхности цилиндра с заданным радиусом основания и высотой.
15. Даны длины сторон параллелограмма и угол между ними. Найти площадь параллелограмма.

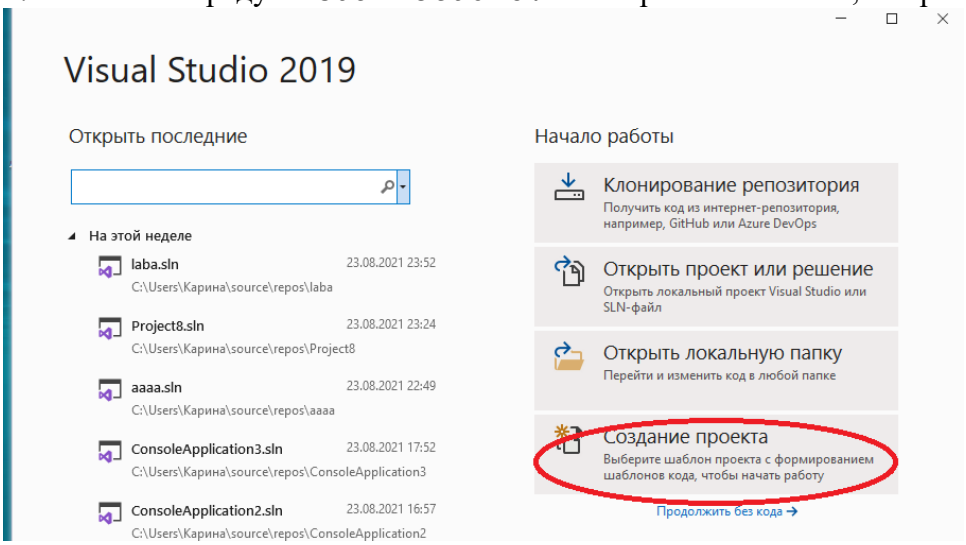
5.Пример выполнения лабораторной работы

Условие задачи.

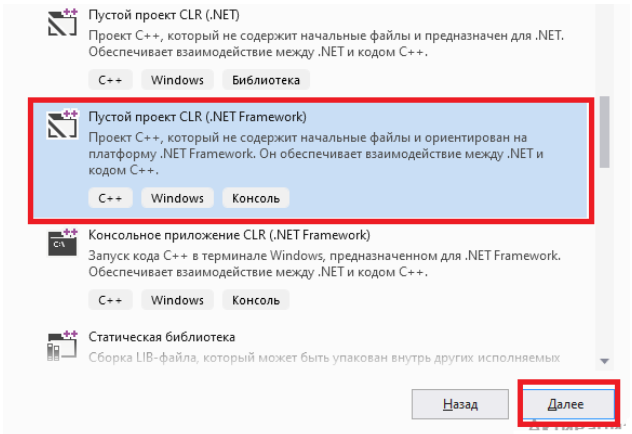
Написать программу перевода километров в мили.

Ход выполнения

1. Войти в среду `Visual Studio.` В открывшемся окне, выбрать **Создание проекта.**

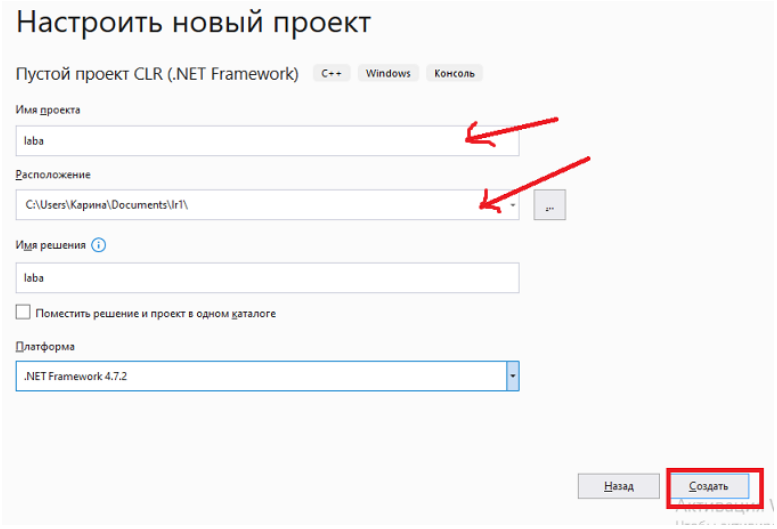


2. В окне **Создать Проект** следует выделить **Пустой проект CLR(.NET Framework)** и нажать **Далее.**

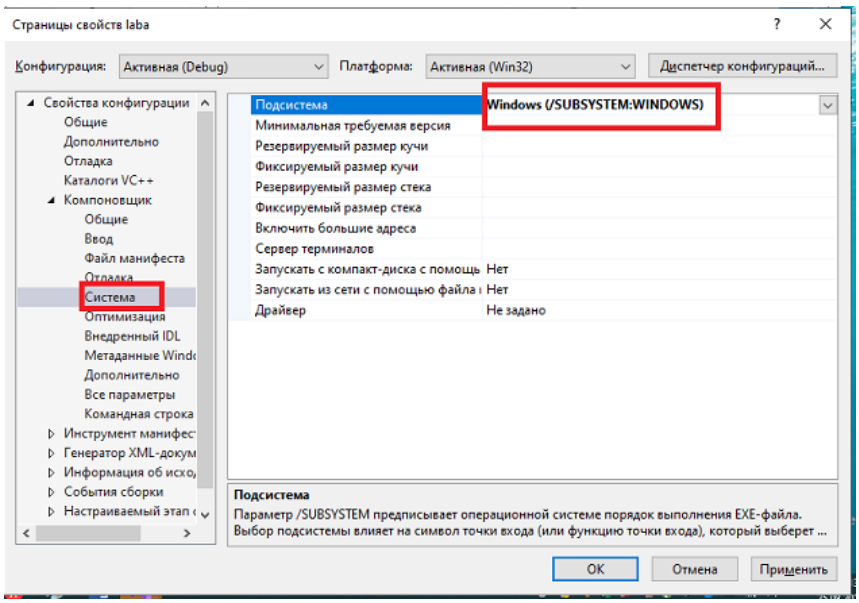


3. Затем в окне **Настроить новый проект** ввести **`Имя проекта`**, в поле **Расположение** можно указать путь размещения проекта или выбрать путь размещения проекта с помощью

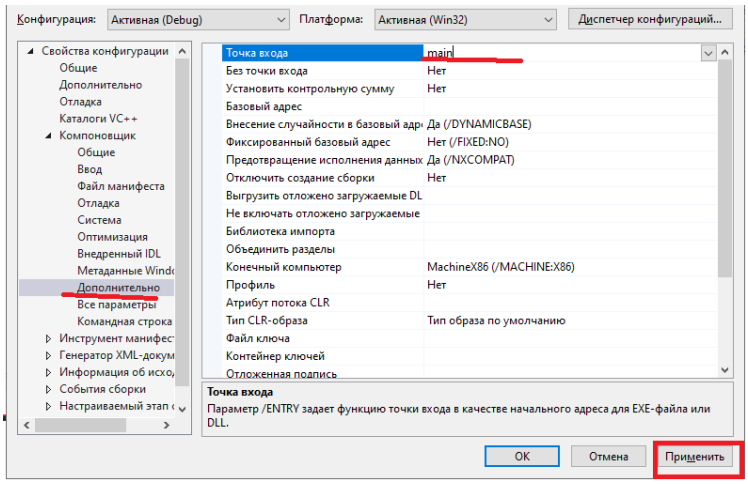
клавиши (кнопки) Обзор (…) . Нажать кнопку **Создать**. После этого создается проект Windows Forms C++/



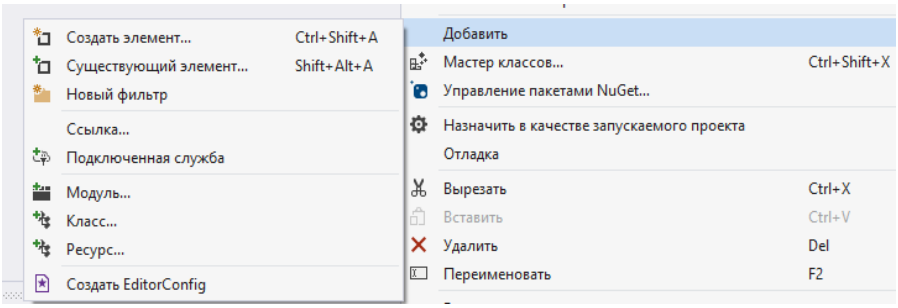
4. Для создания формы необходимо в Главном меню выбрать пункт **Проект —› Свойства —› Компоновщик —› Система —› Подсистема** выбрать Windows;



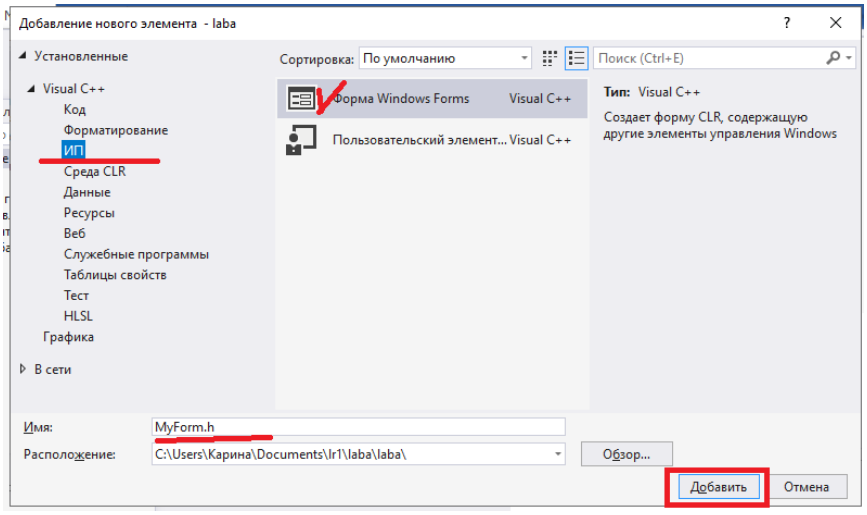
Затем, там же выбрать **Дополнительно** —› ввести точку входа **main.** Нажать **Применить** и **ОК**.



5. На панели **Обозреватель решений** щелкнуть правой кнопкой мыши. В выпадающем меню выбрать **Добавить —› Создать элемент**

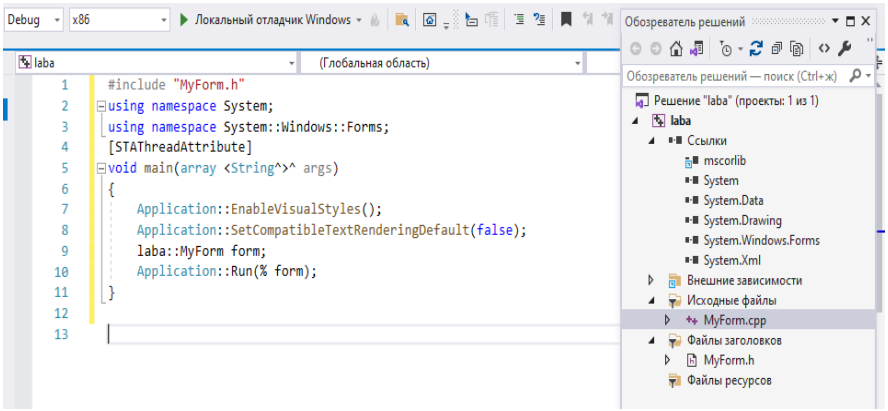


В появившемся окне **Добавление нового элемента**, щелкнуть по пункту **ИП**, выбрать **Форма —›** ввести **Имя файла формы (**по умолчанию MyForm.h**)** —›**Добавить/**



6. На панели **Обозреватель решений** щелкнуть 2 раза на файле **.cpp**. В открывшемся окне ввести код, приведенный ниже:

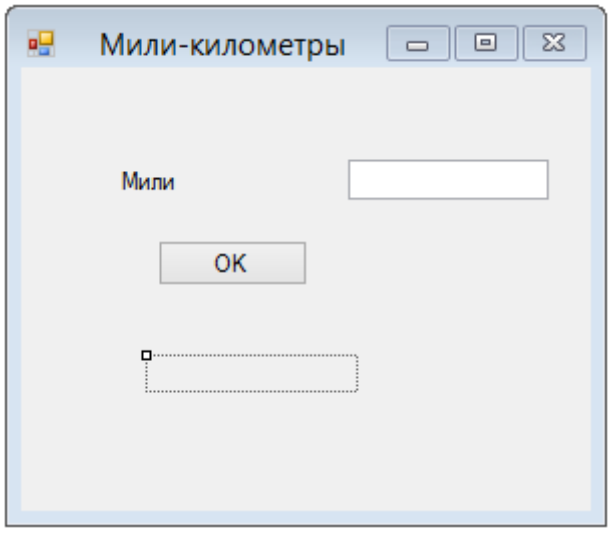
```
#include "MyForm.h"
using namespace System;
using namespace System::Windows::Forms;
[STAThreadAttribute]
void main(array <String^>^ args)
{
   Application::EnableVisualStyles();
   Application::SetCompatibleTextRenderingDefault(false);
   Имя проекта::MyForm form;
   Application::Run(% form);
}
```



Запустить код на выполнение. На экране появится форма, которую необходимо **Закрыть** и перезапустить проект заново.

7. Для формы изменить значение свойства `Text`, занеся следующие данные: «`Выполнил студент группы… Иванов И.И. Лабораторная работа 29`».

Рабочая форма программы:



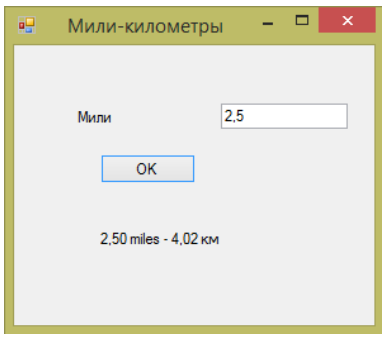
Текст программы:

```
// Щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
      double mile,km;
     // Если в поле редактирования нет данных,
     // то при попытке преобразовать пустую строку
     // в число возникает исключение
     try{
          mile=Convert::ToDouble(textBox1->Text);
          km=mile*1.609344;
          label2->Text=mile.ToString("n")+" miles - "+
          km.ToString("n")+" км";
     }
     catch(System::FormatException^ ex){
          // обработка исключения
          // - сообщение
          MessageBox::Show(
```

```
                    "Надо ввести исходные данные", "Мили-километры",
                    MessageBoxButtons::OK,
                    MessageBoxIcon::Exclamation);
                // - установить курсор в поле редактирования
                textBox1->Focus();
            }
        }

        // Нажатие клавиши в поле textBox
        private: System::Void textBox1_KeyPress(System::Object^  sender,
                     System::Windows::Forms::KeyPressEventArgs^  e) {
            // Правильными символами считаются цифры, запятая, <Enter>,
        <Backspace>.
            // Точка автоматически будет заменяться запятой.
            // Остальные символы запрещены.
            // Чтобы запрещённый символ не отображался в поле
        редактирования,
            // присвоим значение true свойству Handled параметра e.
            if((e->KeyChar>='0')&&(e->KeyChar<='9')) return; //цифра
            if(e->KeyChar=='.') e->KeyChar=',';        // точку заменим
        запятой
           if(e->KeyChar==','){
                if(textBox1->Text->IndexOf(',') != -1)
                // Запятая уже есть в поле редактирования
                e->Handled=true;
                return;
           }
           if(Char::IsControl(e->KeyChar)){
                // <Enter>, <Backspace>
                if(e->KeyChar==(char)Keys::Enter)
                // нажата клавиша <Enter>, установить "фокус" на кнопку
OK
                button1->Focus();
                return;
           }
           // остальные символы запрещены
           e->Handled=true;
}
};
}
```

Тестирование программы

Результат выполнения программы может выглядеть следующим образом:


Мили-километры
Мили
2,5
OK
2,50 miles - 4,02 км

ЛАБОРАТОРНАЯ РАБОТА №30. ПРОГРАММИРОВАНИЕ ВЕТВЛЕНИЙ С ИСПОЛЬЗОВАНИЕМ КОМПОНЕНТА RADIOBUTTON

1. Краткие сведения из теории

В языке программирования **C++** используются несколько конструкций для принятия решений:

– оператор `if`;

– оператор `switch`;

– условный оператор `?` (оператор условия).

Для прерывания программного цикла, при некотором условии применяется утверждение (оператор) `break`, для продолжения итераций цикла при выполнении некоторых условий применяется утверждение (оператор) `continue`, для выхода из функции при выполнении некоторых условий применяется оператор `return`, для перехода к заданному месту программы применяется оператор `goto`, хотя считается, что в программировании не существует ситуаций, в которых нельзя обойтись без оператора `goto`. Утверждение `break` применяется также в теле оператора `switch`.

Условный оператор `if` в языке программирования C++

Общая форма записи оператора `if`:

```
if (условие) {действие};
```

В операторе `if` используется результат вычисления условия, заключенного в круглые скобки, на основе которого принимается решение. Результат вычисления условия (действие) может быть арифметическим или логическим. Если результат выполнения условия будет истинным, то возможно выполнить несколько действий.

Элемент управления RadioButton

Элемент управления `Windows Forms RadioButton` (переключатель) обеспечивает выбор из двух или более взаимоисключающих вариантов. Функции переключателей и флажков могут показаться схожими, но между ними есть важное отличие: в случае переключателя пользователь может выбрать лишь один вариант. Однако флажков можно выбрать любое количество. Определяя группу значений переключателя, разработчик формы предлагает пользователю набор вариантов, из которых может быть задан один и только один.

При щелчке элемента управления `RadioButton`, его свойству `Checked` присваивается значение `true` и вызывается обработчик событий `Click`. При изменении значения свойства `Checked` происходит событие `CheckedChanged`. Если свойство `AutoCheck` имеет значение `true` (принимается по умолчанию), то при выборе одного значения переключателя остальные значения группы автоматически сбрасываются. Обычно этому свойству присваивают значение `false` только в тех случаях, когда в коде предусмотрена проверка допустимости выбранного варианта переключателя. Текст, связанный с этим элементом управления, задается свойством `Text`, которое также может определять клавиши быстрого доступа. Клавиша доступа позволяет пользователю щелкнуть другой элемент управления, используя сочетание клавиши `ALT` и заданной клавиши. Элемент управления `RadioButton` может выглядеть как кнопка команды, которая отображается как нажатая при выбранном значении переключателя, если свойство `Appearance` имеет значение `Button`. В переключателях можно также отображать рисунки с помощью свойств `Image` и `ImageList`.

2.Порядок выполнения работы

2.1 Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

2.2 Содержание отчёта

1. Номер работы и её название.
2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.
6. Зарисованный вид Forms программы
7. Результат тестирования программы.

3.Вопросы к защите

1.Компонент `RadioButton`, его свойства и методы

4.Задание к выполнению

Составить алгоритм и написать программу с использованием компонентов `RadioButton` в среде разработки Visual C++ в соответствии с индивидуальным заданием. В программе должен быть предусмотрен контроль правильности вводимых данных (ввести можно только число).

Список индивидуальных заданий:

1. Составить программу, осуществляющую перевод величин либо из радианной меры угла в градусную, либо из градусной в радианную.
2. Составить программу, выполняющую либо сложение, либо умножение заданных комплексных чисел в алгебраической форме.
3. Составить программу, которая по заданной стороне основания и высоте вычисляет либо объем правильной четырёхугольной пирамиды, либо площадь её поверхности.
4. Составить программу, осуществляющую перевод величины веса либо из килограммов в граммы, либо из граммов в килограммы.
5. Составить программу, выполняющую либо вычитание, либо деление заданных комплексных чисел в алгебраической форме.
6. Составить программу, которая по заданной стороне основания и высоте вычисляет объем либо правильной треугольной пирамиды, либо правильной треугольной призмы.
7. Составить программу, осуществляющую перевод величины расстояния либо из километров в метры, либо из метров в километры.
8. Составить программу, вычисляющую для двух заданных комплексных чисел z1 и z2 либо величину z1/z2, либо z2/z1.
9. Составить программу, которая по заданным основанию и высоте вычисляет либо площадь треугольника, либо площадь параллелограмма.
10. Составить программу, осуществляющую перевод величины скорости либо из метров/секунду в километры/час, либо наоборот.
11. Составить программу, выполняющую либо возведение вещественного числа в квадрат, либо извлечение из него квадратного корня. При извлечении корня нужно предусмотреть,

чтобы число не было отрицательным.

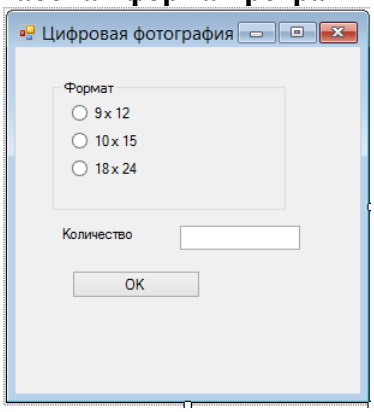
12. Составить программу, которая по заданному радиусу вычисляет либо площадь круга, либо длину окружности.
13. Составить программу, пересчитывающую цену либо из долларов в рубли, либо из рублей в доллары.
14. Составить программу, вычисляющую либо синус, либо косинус заданного угла.
15. Составить программу, которая по заданному радиусу вычисляет либо объём шара, либо площадь его поверхности.

6.Пример выполнения лабораторной работы

Условие задачи.

Написать программу, позволяющую рассчитать стоимость печати фотографий в зависимости от их размера.

Рабочая форма программы:



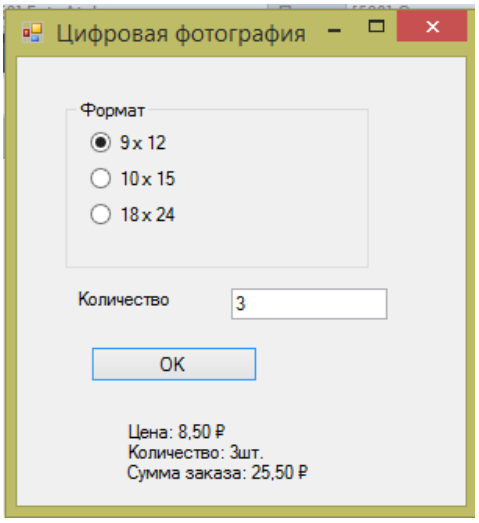
Текст программы:

```
// Щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
       double cena=0,sum; int n;
       if(radioButton1->Checked) cena=8.5;
       if(radioButton2->Checked) cena=10;
       if(radioButton3->Checked) cena=15.5;
       n=Convert::ToInt32(textBox1->Text);
       sum=n*cena;
       label2->Text="Цена: "+cena.ToString("c")+
           "\nКоличество: "+n.ToString()+"шт.\n"+
           "Сумма заказа: "+sum.ToString("C");
       }
// Изменилось содержимое поля редактирования
private: System::Void textBox1_TextChanged(System::Object^ sender,
System::EventArgs^  e) {
           if(textBox1->Text->Length==0) button1->Enabled=false;
           else  button1->Enabled=true;
           label2->Text="";
}
// Щелчок на радиокнопке
private: System::Void radioButton1_Click(System::Object^ sender,
System::EventArgs^  e) {
           label2->Text="";
```

```
                                }
// Щелчок на радиокнопке
private: System::Void radioButton1_Click(System::Object^  sender,
System::EventArgs^  e) {
            label2->Text="";
            // Установить курсор в поле Количество
            textBox1->Focus();
        }
// Чтобы в поле Количество можно было ввести только целое число
private: System::Void textBox1_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e) {
            if((e->KeyChar>='0')&&(e->KeyChar<='9')) return;
            if (Char::IsControl(e->KeyChar)){
                if(e->KeyChar==(char)Keys::Enter)
                    // нажата клавиша <Enter>
                    button1->Focus();
                return;
            }
            // остальные символы запрещены
            e->Handled=true;
        }
};
}
```

Тестирование программы

Результат выполнения программы может выглядеть следующим образом:



ЛАБОРАТОРНАЯ РАБОТА №31. ПРОГРАММИРОВАНИЕ ВЕТВЛЕНИЙ С ИСПОЛЬЗОВАНИЕМ КОМПОНЕНТА CHECKBOX

1.Краткие сведения теории

Компонент `CheckBox`

Компонент **`CheckBox`** (кнопка с независимой фиксацией – флажок Windows)  .

Компонент расположен в группе `Common Controls` палитры компонентов. `CheckBox` используется как флажок-переключатель и позволяет пользователю выбрать / отменить определенную опцию, т. е. выдает результат «включен / выключен».

Если компонент `Checkbox` находится в группе себе подобных, то, включив один `Checkbox`, можно включать и остальные, при этом ни один из них не выключится (т. е. не изменит своего состояния). Отличие компонента `Checkbox` от компонента `RadioButton` заключается в том, что одновременно может быть включено несколько `Checkbox`, а компонент `RadioButton` размещенный на форме может быть включен только один и если мы включаем другой компонент `RadioButton`, то первый отключается автоматически.

Этот компонент соответствуют математическим понятиям конъюнкции и дизъюнкции. Когда мы говорим «находится в группе», то имеется в виду, что у множества таких компонентов один родитель (например, одна панель).

2.Порядок выполнения работы

2.1 Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

2.2 Содержание отчёта

1. Номер работы и её название.
2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.
6. Зарисованный вид Forms программы
7. Результат тестирования программы.

3.Вопросы к защите

1.Компонент `CheckBox,` его свойства и методы

4.Задание к выполнению

Составить алгоритм и написать программу с использованием компонентов `CheckBox` в среде разработки Visual C++ в соответствии с индивидуальным заданием.

Список индивидуальных заданий:

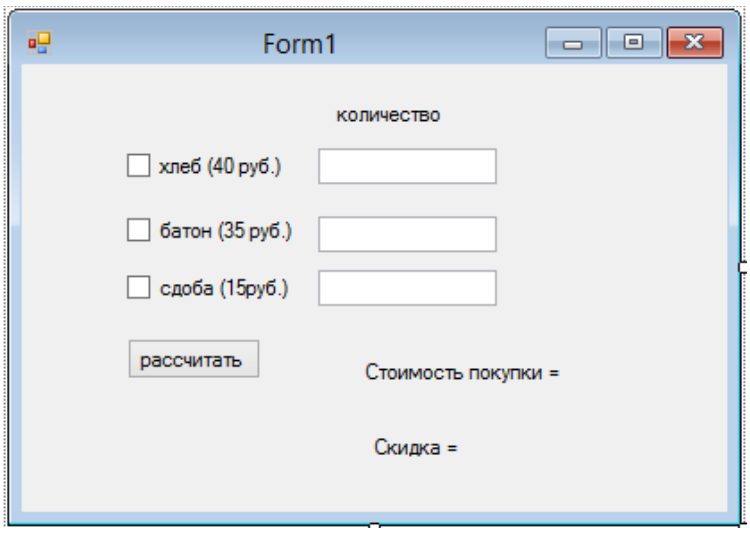
1. Напишите программу, вычисляющую стоимость покупки канцелярских товаров (ручек, карандашей, тетрадей, скрепок).
2. Напишите программу, вычисляющую расстояние в метрах. Задано расстояние может быть в километрах, метрах и сантиметрах.
3. Напишите программу, вычисляющую стоимость покупки строительных материалов (шпаклёвки, грунтовки, досок, плинтусов).
4. Напишите программу, вычисляющую вес в килограммах. Задан вес может быть в килограммах, граммах и центнерах.
5. Написать программу, вычисляющую общее количество дней в заданных месяцах.
6. Напишите программу, вычисляющую стоимость покупки набора посуды (чашка, блюдце, ложка, тарелка).
7. Напишите программу, вычисляющую угол в минутах. Угол может быть задан в минутах, градусах, радианах и секундах.
8. Напишите программу, вычисляющую стоимость покупки книг (учебник, задачник, детектив, сборник стихов).
9. Напишите программу, вычисляющую расстояние в миллиметрах. Задано расстояние может быть в миллиметрах, метрах и сантиметрах.
10. Составьте программу, вычисляющую количество дней в заданном временном интервале. Компонентами могут быть неделя, месяц, декада.
11. Составьте программу, вычисляющую сумму площадей заданных геометрических фигур: треугольника, квадрата, круга и прямоугольника.
12. Напишите программу, вычисляющую стоимость покупки ремонтных материалов (обои, клей, кисти, валик).
13. Напишите программу, вычисляющую вес в граммах. Вес может быть задан в килограммах, граммах и миллиграммах.
14. Напишите программу, вычисляющую стоимость мебели (диван, кресло, стол, шкаф).
15. Напишите программу, вычисляющую площадь земли в квадратных метрах. Задана площадь может быть в квадратных метрах, гектарах и арах.

5.Пример выполнения лабораторной работы

Условие задачи.

Написать программу расчёта покупки хлебобулочных изделий.

Рабочая форма программы:

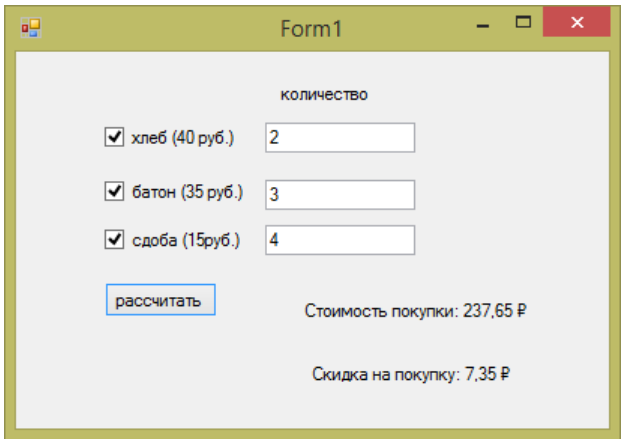


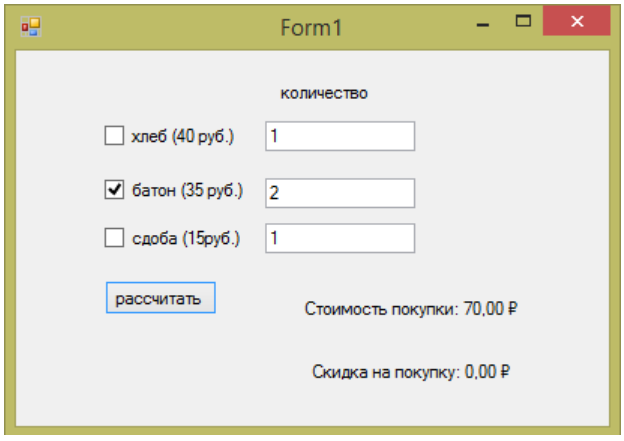
Текст программы:

```
// Щелчок на кнопке Рассчитать
private:    System::Void    button1_Click(System::Object^        sender,
System::EventArgs^  e) {
     double k,itog=0,skid=0;
     if(checkBox1->Checked){
     k=Convert::ToDouble(textBox1->Text);
     itog+=40*k;
        }
    if(checkBox2->Checked){
     k=Convert::ToDouble(textBox2->Text);
    itog+=35*k;
         }
    if(checkBox3->Checked){
     k=Convert::ToDouble(textBox3->Text);
     itog+=15*k;
     }
    if (itog>=200) skid=itog*0.03;
    itog-=skid;
    label1->Text="Стоимость покупки: "+itog.ToString("C");
    label2->Text="Скидка на покупку: "+skid.ToString("C");
     }
};
}
```

Тестирование программы

Результат выполнения программы может выглядеть следующим образом:





ЛАБОРАТОРНАЯ РАБОТА №32. ПРОГРАММИРОВАНИЕ В РЕЖИМЕ ТОЧЕЧНОЙ ГРАФИКИ

1.Краткие теоретические сведения
Графические примитивы

Отображение графики обеспечивает компонент `PictureBox` (рис.1). Подобно тому, как художник рисует на поверхности холста, программа "рисует" на графической поверхности компонента `PictureBox`.

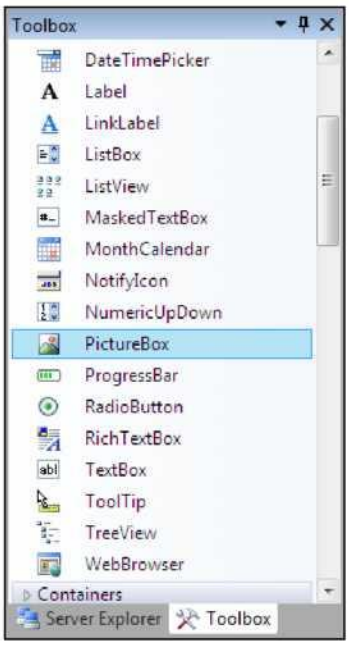


Рис.1. Компонент `PictureBox`

Графическая поверхность компонента `PictureBox` представляет собой объект `Graphics`, методы которого и обеспечивают вывод графики. Таким образом, для того чтобы на поверхности компонента `PictureBox` появилась линия, прямоугольник, окружность или загруженная из файла иллюстрация, необходимо вызвать соответствующий метод объекта `Graphics`.

Доступ к графической поверхности объекта (свойству `Graphics`) есть только у функции обработки события `Paint`. Поэтому сформировать (отобразить) графику может только она.

Создается процедура обработки события `Paint` обычным образом: сначала надо выбрать компонент `PictureBox`, затем в окне `Properties` открыть вкладку `Events` и в поле события `Paint` ввести имя функции обработки события (или сделать двойной щелчок левой кнопкой мыши).

Графическая поверхность

Графическая поверхность состоит из отдельных точек — пикселов. Положение точки на графической поверхности характеризуется горизонтальной (x) и вертикальной (y) координатами (рис.2). Координаты точек отсчитываются от левого верхнего угла и возрастают слева направо (координата x) и сверху вниз (координата y). Левая верхняя точка графической поверхности имеет координаты (0, 0). Размер графической поверхности формы соответствует размеру клиентской области (т. е. без учета высоты области заголовка и

ширины границ) формы (свойство `ClientSize`), а размер графической поверхности компонента `PictureBox` — размеру компонента.

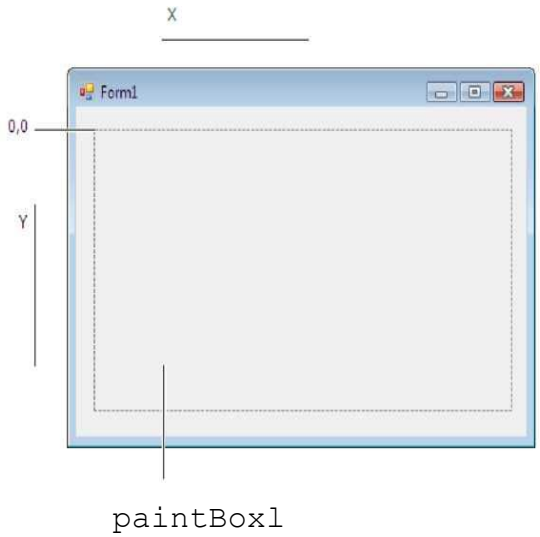


рис.2. Координаты точек графической поверхности формы

Карандаши и кисти

Методы рисования графических примитивов (например, `DrawLine` — линия, `DrawRectangle` — прямоугольник, `FillRectangle` — область) используют *карандаши и кисти*. Карандаш (объект `Pen`) определяет вид линии, кисть (объект `Brush`) — вид закраски области. Например, метод

`DrawLine(System::Drawing::Pens::Black,10,20,100,20);` - рисует из точки (`10,20`) горизонтальную линию длиной 100 пикселов, используя черный (`Black`) карандаш из стандартного набора карандашей (`Pens`), а метод

`FillRectangle(System::Drawing::Brushes::Green,5,10,20,20);` - при помощи стандартной кисти зеленого (`Green`) цвета рисует зеленый квадрат, левый верхний угол которого находится в точке (`5,10`).

Карандаши и кисти определены в пространстве имен `System::Drawing`.

Карандаш

Карандаш определяет вид линии — цвет, толщину и стиль. В распоряжении программиста есть два набора карандашей: стандартный и системный. Также программист может создать собственный карандаш.

Стандартный набор карандашей — это цветные карандаши (всего их 141), которые рисуют непрерывную линию толщиной в один пиксел. Некоторые карандаши из стандартного набора приведены в табл. 1.

Таблица 1. Некоторые карандаши из стандартного набора

Карандаш	Цвет
Pens::Red	Красный
Pens::Orange	Оранжевый
Pens::Yellow	Желтый
Pens::Green	Зеленый
Pens::LightBlue	Голубой
Pens::Blue	Синий
Pens::Purple	Пурпурный
Pens::Black	Черный
Pens::LightGray	Серый
Pens::White	Белый
Pens::Transparent	Прозрачный

Карандаш из стандартного (`Pens`) и системного (`systemPens`) наборов рисует непрерывную линию толщиной в *один пиксел*. Если надо нарисовать пунктирную линию или линию толщиной больше единицы, то следует использовать карандаш программиста.

Карандаш программиста — это объект типа `Pen`, свойства которого (табл.2) определяют вид линии, рисуемой карандашом.

Таблица 2. Свойства объекта Pen

Свойство	Описание
`Color`	Цвет линии
`Width`	Толщина линии (задается в пикселах)
`DashStyle`	Стиль линии (`DashStyle::Solid` — сплошная; `DashStyle::Dash` — пунктирная, длинные штрихи; `DashStyle::Dot` — пунктирная, короткие штрихи; `DashStyle::DashDot` — пунктирная, чередование длинного и короткого штрихов; `DashStyle::DashDotDot` — пунктирная, чередование одного длинного и двух коротких штрихов;
`DashPattern`	Длина штрихов и промежутков пунктирной линии `DashStyle::Custom`

Для того чтобы использовать карандаш программиста, его надо создать. Создает карандаш конструктор объекта `Pen`. Конструктор перегружаемый, т. е. для объекта `Pen` определено несколько конструкторов, которые различаются количеством параметров. Например, конструктор `Pen(Цвет)` создает карандаш указанного цвета толщиной в один пиксел, а `Pen(Цвет, Толщина)` — карандаш указанного цвета и толщины. В качестве параметра Цвет можно использовать константу типа `Color` (табл.3).

Таблица 3. Константы типа `Color`

Константа	Цвет
`Color::Red`	Красный
`Color::Orange`	Оранжевый
`Константа`	Цвет
`Color::Yellow`	Желтый
`Color.Green`	Зеленый
`Color::LightBlue`	Голубой
`Color::Blue`	Синий
`Color::Purple`	Пурпурный
`Color::Black`	Черный
`Color::LightGray`	Серый
`Color::White`	Белый
`Color::Transparent`	Прозрачный

Кисть

Кисти используются для закраски внутренних областей геометрических фигур. Например, инструкция

`e->Graphics->FillRectangle(Brushes::DeepSkyBlue, x, y, w, h);` рисует закрашенный прямоугольник.

В приведенном примере `Brushes::DeepSkyBlue` — это стандартная кисть темно-небесного цвета. Параметры `x, y` определяют положение прямоугольника; `w, h` — его размер.

В распоряжении программиста есть три типа кистей: стандартные, штриховые и текстурные.

Стандартная кисть закрашивает область одним цветом (сплошная закраска). Всего есть 140 кистей, некоторые из них приведены в табл.4.

Таблица 4. Некоторые кисти из стандартного набора

Кисть	Цвет
Brushes::Red	Красный
Brushes::Orange	Оранжевый
Brushes::Yellow	Желтый
Brushes::Green	Зеленый
Brushes::LightBlue	Голубой
Brushes::Blue	Синий
Brushes::Purple	Пурпурный
Brushes::Black	Черный
Brushes::LightGray	Серый
Brushes::White	Белый
Brushes::Transparent	Прозрачный

Штриховая кисть (`HatchBrush`) закрашивает область путем штриховки. Область может быть заштрихована горизонтальными, вертикальными или наклонными линиями разного стиля и толщины. В табл.5 перечислены некоторые из возможных стилей штриховки.

Таблица 5. Некоторые стили штриховки областей

Стиль	Штриховка
`HatchStyle::LightHorizontal`	Редкая горизонтальная
`HatchStyle::Horizontal`	Средняя горизонтальная
`HatchStyle::NarrowHorizontal`	Частая горизонтальная
`HatchStyle::LightVertical`	Редкая вертикальная
`HatchStyle::Vertical`	Средняя вертикальная
`HatchStyle::NarrowVertical`	Частая вертикальная
`HatchStyle::LageGrid`	Крупная сетка из горизонтальных и вертикальных линий
`HatchStyle::SmallGrid`	Мелкая сетка из горизонтальных и вертикальных линий
`HatchStyle::DottedGrid`	Сетка из горизонтальных и вертикальных линий,
`HatchStyle::ForwardDiagonal`	Диагональная штриховка
`HatchStyle::BackwardDiagonal`	Диагональная штриховка "назад"

Градиентная кисть представляет собой прямоугольную область, цвет точек которой зависит от расстояния до границы. Обычно градиентные кисти двухцветные, т.е. цвет точек по мере удаления от границы постепенно меняется с одного на другой. Цвет может меняться вдоль горизонтальной или вертикальной границы области. Возможно также изменение цвета вдоль линии, образующей угол с горизонтальной градацией.

Графические примитивы

Любая картинка, чертеж, схема представляет собой совокупность графических примитивов: точек, линий, окружностей, дуг, текста и др. Вычерчивание графических примитивов на графической поверхности (`Graphics`) выполняют соответствующие методы (табл.6).

Таблица 6. Некоторые методы вычерчивания графических примитивов

Метод	Действие
`DrawLine(Pen, x1, y1, x2, y2)`, `DrawLine(Pen, p1, p2)`	Рисует линию. Параметр `Pen` определяет цвет, толщину и стиль линии; параметры `x1, y1, x2, y2` или `p1 и p2` — координаты точек начала и конца линии
`DrawRectangle(Pen, x, y, w, h)`	Рисует контур прямоугольника. Параметр `Pen` определяет цвет, толщину и стиль границы прямоугольника: параметры `x, y` — координаты левого верхнего угла; параметры `w и h` задают размер прямоугольника

`FillRectangle(Brush,x,y,w,h)`	Рисует закрашенный прямоугольник. Параметр `Brush` определяет цвет и стиль закраски прямоугольника; параметры `x, y` — координаты левого верхнего угла; параметры `w и h` задают размер прямоугольника
`DrawEllipse(Pen, x, y, w, h)`	Рисует эллипс (контур). Параметр `Pen` определяет цвет, толщину и стиль линии эллипса; параметры `x, y, w, h` — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
`FillEllipse(Brush, x, y, w, h)`	Рисует закрашенный эллипс. Параметр `Brush` определяет цвет и стиль закраски внутренней области эллипса; параметры x, y, w, h — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
`DrawPolygon(Pen, P)`	Рисует контур многоугольника. Параметр `Pen` определяет цвет, толщину и стиль линии границы многоугольника; параметр P (массив типа `Point`) — координаты углов многоугольника
`FillPolygon(Brush, P)`	Рисует закрашенный многоугольник. Параметр `Brush` определяет цвет и стиль закраски внутренней области многоугольника; параметр P (массив типа `Point`) — координаты углов многоугольника
`DrawString(str, Font, Brush, x, y)`	Выводит на графическую поверхность строку текста. Параметр `Font` определяет шрифт; `Brush` — цвет символов; x и y — точку, от которой будет выведен текст
`DrawImage(Image, x, y)`	Выводит на графическую поверхность иллюстрацию. Параметр `Image` определяет иллюстрацию; `x и y` — координату левого верхнего угла области вывода иллюстрации

Один и тот же элемент можно нарисовать при помощи разных, но имеющих одинаковые имена.

Например, прямоугольник можно нарисовать методом `DrawRectangle`, которому в качестве параметров передаются координаты левого верхнего угла и размеры прямоугольника:

```
e->Graphics->DrawRectangle(Pens::Black, x, x, w, h)
```

Эту же задачу может решить метод `DrawRectangle`, которому в качестве параметра передается структура типа `Rectangle`, поля которой определяют прямоугольник (положение и размер):

```
Rectangle aRect = Rectangle(20,100,50,50); e->Graphics
```

```
->DrawRectangle(Pens::Blue, aRect);
```

Существование нескольких методов, выполняющих одну и ту же задачу, позволяет программисту выбрать метод, наиболее подходящий для решения конкретной задачи.

В качестве параметров методов вычерчивания графических примитивов часто используется структура `Point`. Ее поля X и Y определяют положение (координаты) точки графической поверхности. Например:

```
Point p1 = Point(10,10);
Point p2 = Point(100,10);
// рисуем линию из p1 в p2
e->Graphics->DrawLine(Pens::Green, p1, p2);
```

Пример: Составить программу, рисующую на поверхности формы итальянский флаг.

w,h – размер флага

ws – ширина полосы

x0,y0 – координаты левого верхнего угла флага

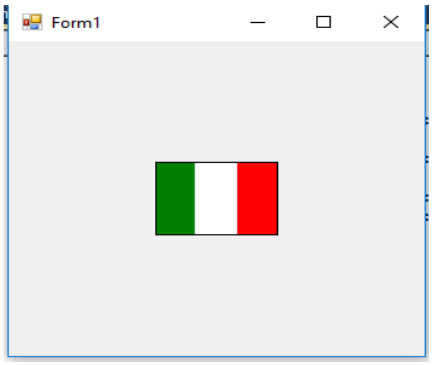
x,y – координаты левого верхнего угла полосы

Текст обработчика

// обработка события `Paint`

```
private: System::Void Form1_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e)
{
    int w,h,ws,x0,y0,x,y;
     w=83; h=60;
// разместим флаг в центре формы
// ClientSize.Width - ширина внутренней области формы
// ClientSize.Height - высота внутренней области формы
    x0=(this->ClientSize.Width-w)/2;
    y0=(this->ClientSize.Height-h)/2;
    ws=w/3;  // три вертикальные полосы
      // рисуем флаг
     x=x0; y=y0;
    // зеленая полоса
  e->Graphics->FillRectangle(System::Drawing::Brushes::Green,x,y,ws,h);
      //белая полоса
     x=x+ws+1;
  e->Graphics->FillRectangle(System::Drawing::Brushes::White,x,y,ws,h);
      //красная полоса
    x=x+ws+1;
    e->Graphics->FillRectangle(System::Drawing::Brushes::Red,x,y,ws,h);
    //окантовка
    e->Graphics->DrawRectangle(System::Drawing::Pens::Black,x0,y0,w,h);
     }
```

В результате получим



2.Порядок выполнения работы

2.1 Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

2.2 Содержание отчёта

1. Номер работы и её название.
2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.
6. Зарисованный вид Forms программы
7. Результат тестирования программы.

3.Вопросы к защите

1. Назовите основные методы рисования графических примитивов
2. Каковы свойства объекта `Pen?`
3. Назовите основные кисти графических примитивов

4.Задания к выполнению

1. Составить программу, заполняющую экран контурами прямоугольников разных цветов.

2. Составить программу, заполняющую экран закрашенными треугольниками, заполненными установленными ранее шаблонами закраски.

3. Составить программу, заполняющую экран контурами треугольников разных цветов.

4. Составить программу, заполняющую экран закрашенными пятиугольниками, заполненными установленными ранее шаблонами закраски.

5. Составить программу, заполняющую экран контурами пятиугольников разных цветов.

6. Составить программу, заполняющую экран закрашенными секторами эллипса, заполненными установленными ранее шаблонами закраски.

7. *Составить программу, заполняющую экран ромбами разных цветов.*
8. *Составить программу, заполняющую экран контурами эллипсов разных цветов.*
9. *Составить программу, заполняющую экран закрашенными кругами, заполненными установленными ранее шаблонами закраски*
10. *Составить программу, заполняющую экран дугами эллипсов разных цветов.*
11. *Составить программу, заполняющую экран закрашенными шестиугольниками, заполненными установленными ранее шаблонами закраски.*
12. *Составить программу, заполняющую экран линиями разных цветов.*
13. *Составить программу, заполняющую экран закрашенными прямоугольными треугольниками, заполненными установленными ранее шаблонами закраски.*
14. *Составить программу, заполняющую экран дугами окружностей разных цветов.*
15. *Составить программу, заполняющую экран ромбами разных цветов.*

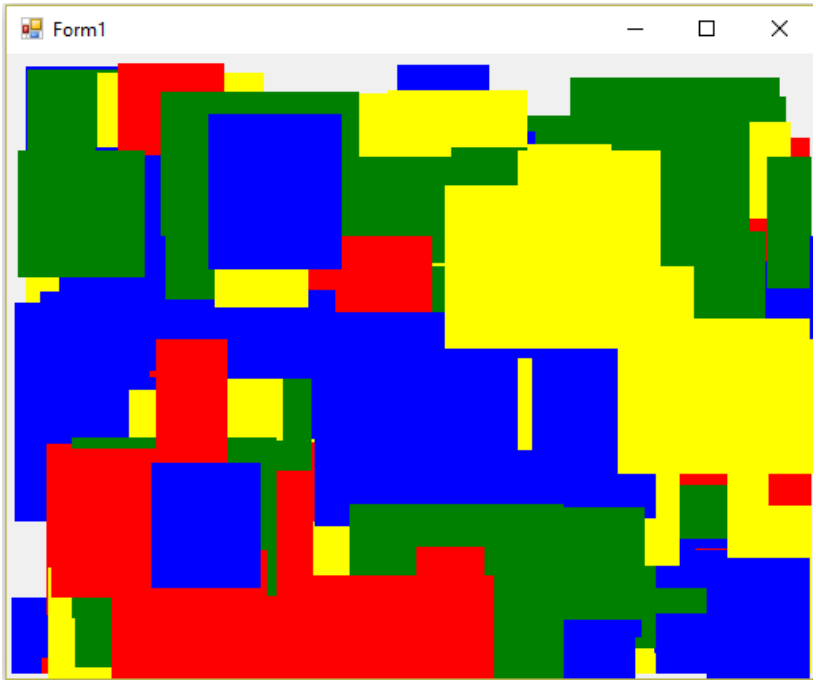
5.Пример выполнения лабораторной работы

Составить программу, заполняющую экран закрашенными прямоугольниками, заполненными установленными ранее шаблонами закраски.

Код программы

```
private: System::Void Form1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e) {
            srand(time(0));

            this->timer1->Enabled = true; // запускаем таймер
            this->timer1->Interval = 50;  // время появления нового прямоугольника
      }

      private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs^ e) {
            Graphics^ gr = this->CreateGraphics();

            int x1, y1, x2, y2, w, h;
            w = ClientSize.Width;   //  ширина формы
            h = ClientSize.Height;  // высота формы
            x1 = rand() % w; y1 = rand() % h;  // выбираем случайные координаты
                                               //левого верхнего угла прямоугольника
            x2 = rand() % 100 + 40; y2 = rand() % 100 + 40;  // Случайным образом выбираем
                                                             // длины сторон
прямоугольника

            Brush^ br1 = gcnew SolidBrush(Color::Blue);  // Создаём кисти разных цветов
            Brush^ br2 = gcnew SolidBrush(Color::Red);   // для рисования прямоугольников
            Brush^ br3 = gcnew SolidBrush(Color::Green);
            Brush^ br4 = gcnew SolidBrush(Color::Yellow);

            int i = rand() % 4; // вспомогательная величина, принимающая случайные
значения
            switch (i)
            {
            case 0: gr->FillRectangle(br1, x1, y1, x2, y2); break; // выбираем цвет
            case 1: gr->FillRectangle(br2, x1, y1, x2, y2); break; // прямоугольника
            case 2: gr->FillRectangle(br3, x1, y1, x2, y2); break; // в зависимости
            default:gr->FillRectangle(br4, x1, y1, x2, y2); break; // от случайной
                                                                   // величины i
            }
      }
      };
}
```


Form1


Form1

ЛАБОРАТОРНАЯ РАБОТА № 33. ПОСТРОЕНИЕ СЕЧЕНИЙ ГЕОМЕТРИЧЕСКИХ ТЕЛ

1. Краткие теоретические сведения

Графические примитивы

Любая картинка, чертеж, схема представляет собой совокупность графических примитивов: точек, линий, окружностей, дуг, текста и др. Вычерчивание графических примитивов на графической поверхности (`Graphics`) выполняют соответствующие методы (табл.1).

Таблица 1. Некоторые методы вычерчивания графических примитивов

Метод	Действие
`DrawLine(Pen, x1, y1, x2, y2)`, `DrawLine(Pen, p1, p2)`	Рисует линию. Параметр `Pen` определяет цвет, толщину и стиль линии; параметры `x1, y1, x2, y2` или `p1 и p2` — координаты точек начала и конца линии
`DrawRectangle(Pen, x, y, w, h)`	Рисует контур прямоугольника. Параметр `Pen` определяет цвет, толщину и стиль границы прямоугольника: параметры `x, y` — координаты левого верхнего угла; параметры `w и h` задают размер прямоугольника
`FillRectangle(Brush,x,y,w,h)`	Рисует закрашенный прямоугольник. Параметр `Brush` определяет цвет и стиль закраски прямоугольника; параметры `x, y` — координаты левого верхнего угла; параметры `w и h` задают размер прямоугольника
`DrawEllipse(Pen, x, y, w, h)`	Рисует эллипс (контур). Параметр `Pen` определяет цвет, толщину и стиль линии эллипса; параметры `x, y, w, h` — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
`FillEllipse(Brush, x, y, w, h)`	Рисует закрашенный эллипс. Параметр `Brush` определяет цвет и стиль закраски внутренней области эллипса; параметры x, y, w, h — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
`DrawPolygon(Pen, P)`	Рисует контур многоугольника. Параметр `Pen` определяет цвет, толщину и стиль линии границы многоугольника; параметр P (массив типа `Point`) — координаты углов многоугольника

`FillPolygon(Brush, P)`	Рисует закрашенный многоугольник. Параметр `Brush` определяет цвет и стиль закраски внутренней области многоугольника; параметр P (массив типа `Point`) — координаты углов многоугольника
`DrawString(str, Font, Brush, x, y)`	Выводит на графическую поверхность строку текста. Параметр `Font` определяет шрифт; `Brush` — цвет символов; x и y — точку, от которой будет выведен текст
`DrawImage(Image, x, y)`	Выводит на графическую поверхность иллюстрацию. Параметр `Image` определяет иллюстрацию; `x и y` — координату левого верхнего угла области вывода иллюстрации

Один и тот же элемент можно нарисовать при помощи разных, но имеющих одинаковые имена методов.

Например, прямоугольник можно нарисовать методом `DrawRectangle`, которому в качестве параметров передаются координаты левого верхнего угла и размеры прямоугольника:

```
e->Graphics->DrawRectangle(Pens::Black, x, x, w, h)
```

Эту же задачу может решить метод `DrawRectangle`, которому в качестве параметра передается структура типа `Rectangle`, поля которой определяют прямоугольник (положение и размер):

```
Rectangle aRect = Rectangle(20,100,50,50); e->Graphics
->DrawRectangle(Pens::Blue, aRect);
```

Существование нескольких методов, выполняющих одну и ту же задачу, позволяет программисту выбрать метод, наиболее подходящий для решения конкретной задачи.

В качестве параметров методов вычерчивания графических примитивов часто используется структура `Point`. Ее поля X и Y определяют положение (координаты) точки графической поверхности. Например:

```
Point p1 = Point(10,10);
Point p2 = Point(100,10);
// рисуем линию из p1 в p2
e->Graphics->DrawLine(Pens::Green, p1, p2);
```

2.Порядок выполнения работы

2.1 Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

2.2 Содержание отчёта

1. Номер работы и её название.
2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.
6. Зарисованный вид Forms программы
7. Результат тестирования программы.

3. **Контрольные вопросы**

1. Назовите основные методы рисования графических примитивов
2. Каковы свойства объекта `Pen`?
3. Назовите основные кисти графических примитивов

4.Задания к выполнению

1. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямой четырехугольной призме провести сечение, проходящее через диагональ верхнего основания и одну из вершин нижнего основания.
2. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямоугольном параллелепипеде провести сечение, проходящее через сторону верхнего основания и противоположную сторону нижнего основания.
3. Написать программу, выполняющую следующие действия в режиме точечной графики. В правильной шестиугольной призме построить сечение, проходящее через большую диагональ нижнего основания и одну из сторон верхнего.
4. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямоугольном параллелепипеде провести сечение, проходящее через одну из сторон нижнего основания и одну из вершин верхнего.
5. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямом цилиндре построить осевое сечение.
6. Написать программу, выполняющую следующие действия в режиме точечной графики. В правильной шестиугольной призме провести сечение, проходящее через одну из сторон нижнего основания и противоположную ей сторону верхнего основания.
7. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямой четырехугольной призме провести сечение, проходящее через диагональ нижнего основания и одну из вершин верхнего основания.
8. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямоугольном параллелепипеде провести сечение, проходящее через сторону нижнего основания и противоположную сторону верхнего основания.
9. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямом параллелепипеде провести сечение, проходящее через большую диагональ нижнего основания и середину одного из ребер, не пересекающихся с рассматриваемой диагональю.
10. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямом параллелепипеде провести сечение, проходящее через меньшую диагональ верхнего основания и одну из вершин нижнего основания.
11. Написать программу, выполняющую следующие действия в режиме точечной графики. В правильной шестиугольной пирамиде провести сечение, параллельное основанию.
12. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямом параллелепипеде провести сечение, проходящее через большую диагональ верхнего основания и одну из вершин нижнего основания.

13. Написать программу, выполняющую следующие действия в режиме точечной графики. В прямом параллелепипеде провести сечение, проходящее через меньшую диагональ нижнего основания и одну из вершин верхнего основания.

14. Написать программу, выполняющую следующие действия в режиме точечной графики. В правильной треугольной призме построить сечение, проходящее через одну из сторон верхнего основания и противолежащую вершину нижнего.

15. Написать программу, выполняющую следующие действия в режиме точечной графики. В правильной треугольной призме построить сечение, параллельное основанию

5. Пример выполнения лабораторной работы

В треугольной пирамиде построить сечение, параллельное основанию этой пирамиды.

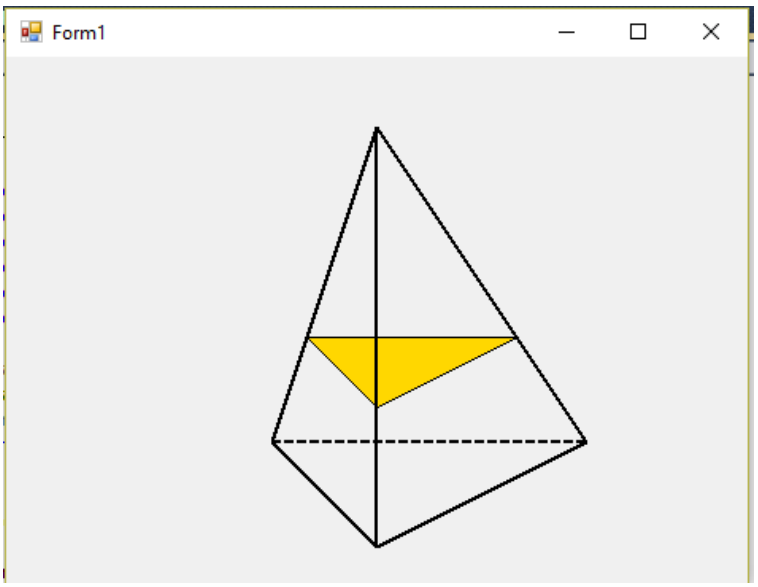
Текст обработчика

```
private:   System::Void   Form1_Paint(System::Object^      sender,
System::Windows::Forms::PaintEventArgs^  e) {

   int W=this->ClientSize.Width;      //Ширина экрана
   int H=this->ClientSize.Height;     //Высота экрана
   int d=H/15;

   Pen^ aPen = gcnew Pen( Color::Black,2 );     //Цвет и толщина
карандаша сплошных линий
   Pen^ cPen = gcnew Pen( Color::Black,2 );     //Цвет и толщина
пунктирных линий

   cPen->DashStyle  =  System::Drawing::Drawing2D::DashStyle::Dash;
//Пунктирный карандаш

   array<Point>^ p;              //Массив для координат X и Y
   p = gcnew array <Point>(4);     //Массив вершин треугольника в
сечении пирамиды
    p[0].X =W/2-2*d ;
    p[0].Y =8*d;
    p[1].X =W/2 ;
    p[1].Y =10*d;
    p[2].X =W/2+4*d ;
    p[2].Y =8*d;
    p[3].X =W/2-2*d ;
    p[3].Y =8*d;
  e->Graphics->FillPolygon(Brushes::Gold, p);   //Заливаем сечение
желтым цветом
e->Graphics->DrawPolygon(Pens::Black,p);          //Обводим  сечение
черным

e->Graphics->DrawLine(aPen,W/2,2*d,W/2-3*d,11*d);   //Боковые ребра
пирамиды
   e->Graphics->DrawLine(aPen,W/2,2*d,W/2+6*d,11*d);
   e->Graphics->DrawLine(aPen,W/2,2*d,W/2,14*d);
   e->Graphics->DrawLine(cPen,W/2-3*d,11*d,W/2+6*d,11*d);
//Основание пирамиды
   e->Graphics->DrawLine(aPen,W/2-3*d,11*d,W/2,14*d);
   e->Graphics->DrawLine(aPen,W/2+6*d,11*d,W/2,14*d);
        }
```

```
private:   System::Void   Form1_Resize(System::Object^      sender,
System::EventArgs^  e) {
      this->Refresh();   //Возможность пропорционального изменения
формы и рисунка
            }
   };
}
```

Тестирование программы

Результат выполнения программы может выглядеть следующим образом:



ЛАБОРАТОРНАЯ РАБОТА №34. ПОСТРОЕНИЕ ГРАФИКОВ ФУНКЦИЙ В ВИЗУАЛЬНОЙ СРЕДЕ НА ЯЗЫКЕ VISUAL C++

1.Краткие теоретические сведения

Элемент управления Chart

Элемент управления `Chart` позволяет создавать диаграммы для сложного статистического или финансового анализа. Этот элемент поддерживает следующие функциональные возможности:

- ряды данных, области диаграммы, оси, условные обозначения, метки и заголовки;
- привязку данных;
- операции с данными: копирование, разбиение, слияние, выравнивание, группирование, сортировку, поиск, фильтрацию и т. д.;
- статистические и финансовые формулы;
- расширенную настройку внешнего вида диаграммы: трехмерную графику, сглаживание, освещение и перспективу;
- события и индивидуальную настройку.

С помощью компонента `Chart` (чертеж) удобнее всего выполнять построение графиков функции по уравнению на промежутке.

Построение графика функции:

1) нужно добавить компонент `Chart` на форму;
2) щелкнуть левой кнопкой по пункту «Series» окна «*Свойства*» (рис. 1);

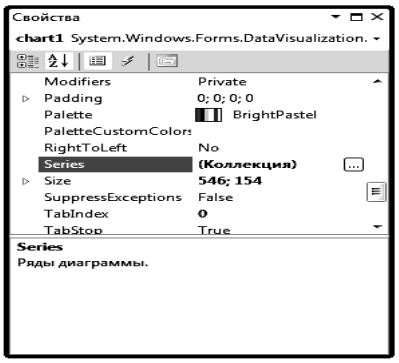


Рис.1 – Окно «Свойства»

3). В результате увидим окно редактора коллекции `Series`(рис.2);

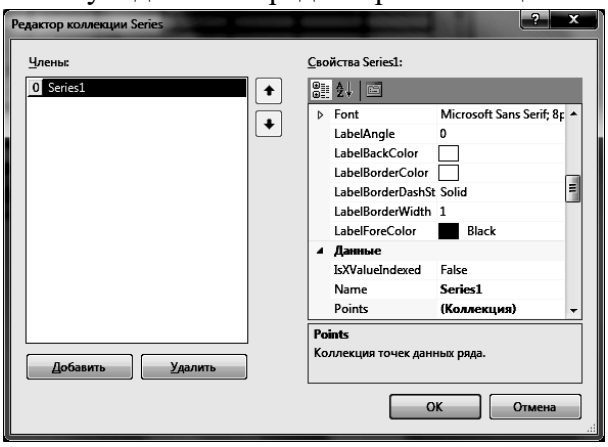


Рис.2 – Диалоговое окно «Редактор коллекции Series»

4) чтобы на одном компоненте Chart построить несколько графиков функции, нужно выбрать страницу Series (левая часть окна) и щелкнуть по кнопке *Добавить* (если нужно построить несколько графиков на одной координатной плоскости, то нужно сделать соответствующее количество **Series**);

5) в правой части окна для каждой Series можно сделать индивидуальные настройки, основные из них:

- ChartType – тип диаграммы для представления данных;
- XValueType – тип значений, хранимых на оси OX;
- YValueType – тип значений, хранимых на оси OY;
- Color – цвет точки данных;
- BorderColor – задает цвет границ;
- ShadowColor – задает цвет тени;
- Font – шрифт точки данных;
- LabelBackColor – задает цвет фона метки;
- LabelForeColor – задает цвет метки;
- IsVisibleInLegend – включает/выключает отображение легенды;

Многие из этих параметров можно также задавать программно, например:

```
Series^ plot1 = chart1->Series[0];
     if(colorDialog1->ShowDialog()==System::Windows::
     Forms::DialogResult::OK);
plot1->Color = colorDialog1->Color;
```

В данном случае изменяется цвет графика функции при выборе цвета в диалоговом окне «ColorDialog».

Если графики функций нужно распечатать, то на черно-белом принтере линии будут неразличимыми. В этом случае нужно использовать возможности страницы Series и из списка опций BorderdashStyle выбрать другую прорисовку линии (например, точками) или в опции BorderWidth выбрать большую толщину линии (по умолчанию стоит значение 1).

Пример: Построить график функции $y=sinx$

1. На форме разместить компонент Chart со свойством Series
2. Поместить кнопку Button, в свойстве Text занести текст «*Построить график*»
3. Щёлкнув два раза по компоненту Button, перейти к коду программы и после строки
   `#pragma once`
   подключить библиотеку использования математических функций:
   `#include <cmath>`

В разделе using добавить строку:

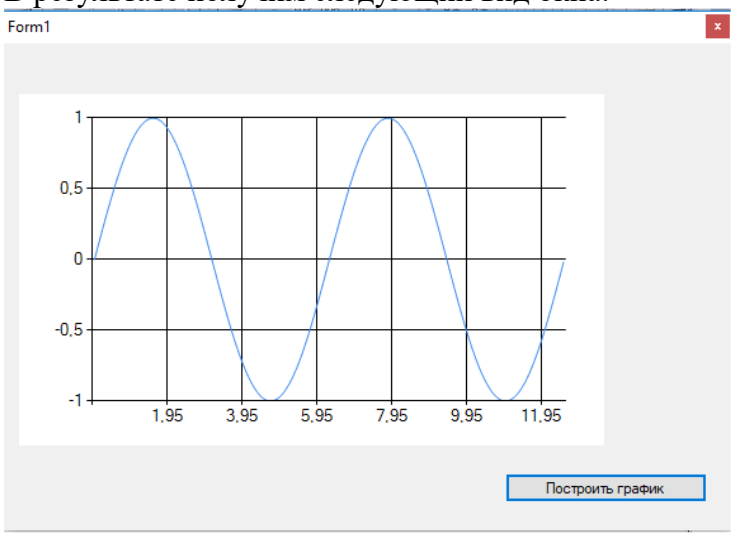
```
using namespace
System::Windows::Forms::DataVisualization::Charting;
```

Код программы:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
       double x,xn,xk,xh,y;
       Series^ plot1=chart1->Series[0];
       plot1->Points->Clear();
       xn=0; xk=12.56; xh=0.05;
       x=xn;
       while (x<=xk)
          {  y=sin(x);
             plot1->Points->AddXY(x,y);
             x=x+xh;
```

```
        }
    }
```

В результате получим следующий вид окна:



Алгоритм выполнения лабораторной работы

Создать Windows-приложение, которое предлагает пользователю ввести данные начала промежутка (xn), конца промежутка (xk) и шага изменения переменной (xh) для построения на одной координатной плоскости трех графиков $f_1(x)$, $f_2(x)$, $f_3(x)$

$$y = f(x) = \begin{cases} 2x + 2, & x \le 0 \\ \sqrt{x+3}, & 0 < x \le 5 \\ cos^2(x+2), & x > 5 \end{cases}$$

1 Войти в среду ***Visual Studio.***

2 Для формы изменим значение свойства **Text,** занеся, например, следующие данные: ***«Выполнил студент Иванов П. А. Лабораторная работа 34».***

3 Свойству **FormBorderStyle** формы (стиль рамки окна) присвоить значение ***FixedToolWindow***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

4 В верхней части окна разместить **Label1** и в свойстве **Text** занести текст «***Построение графика функции на интервале [xn;xk]***».

5 Ниже под **Label1** разместить компонент **Chart1**. Выделить его и зайти в окно свойств; выбрать свойство **Series** и нажать на троеточие. Появится диалоговое окно (рис. 3). **Нажать на кнопку Добавить и у нас появится Series1 в левой части окна (каждая серия способна строить на компоненте Chart1 новый график, поэтому, если нужно построить несколько графиков на одной координатной плоскости, то нужно сделать соответствующее количество Series).**

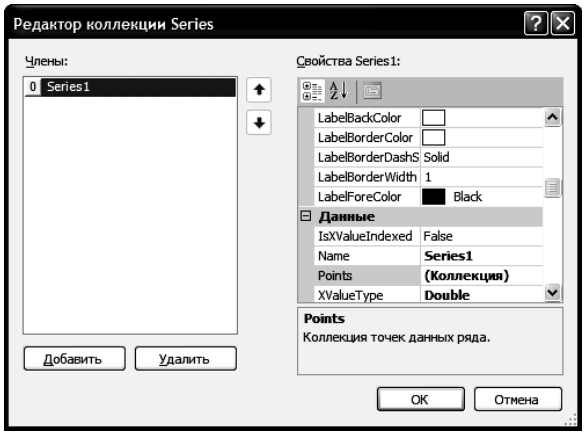


Рисунок 3 – Диалоговое окно свойства Series компонента Chart

8. В правой части этого же окна выбрать подпункт «*Данные*» свойства **Series1**, а в нем пункт **XValueType** и с помощью выпадающего списка выбрать параметр **«*Double*»**. Аналогично для пункта **YValueType** выбрать параметр **«*Double*»**.
9. Ниже в правой части этого же окна выбрать подпункт «*Диаграмма*» свойства **Series1**, а в нем – пункт **ChartType** и с помощью выпадающего списка выбрать параметр **«*Spline*»** (*это позволяет соединять точки кривой линией*).
10. И последняя настройка компонента **Chart1**: в правой части этого же диалогового окна выбрать подпункт «*Условные обозначения*» свойства **Series1**, а в нем – пункт **IsVisibleInLegend** и с помощью выпадающего списка выбрать параметр **«*False*»** (*это позволяет не отображать легенду в диаграмме*).
11. Ниже компонента **Chart1** разместить четыре компонента **Label**: **Label2**, **Label3**, **Label4**, **Label5**.
12. Выделить **Label2** и в свойстве **Text** занести текст «*Введите интервал построения графика*». Аналогично выделить **Label3** и в свойстве **Text** занести текст «*Введите начальное значение XN=*», для **Label4** – в свойстве **Text**занести текст «*Введите конечное значение XK=*» и для **Label5** в свойстве **Text**занести текст «*Введите значение шага XH=*».
13. Напротив компонента **Label3** разместить компонент **TextBox1**, напротив компонента **Label4** – компонент **TextBox2**, а напротив компонента **Label5** – компонент **TextBox3**.
14. В правом нижнем углу разместить две кнопки **Button1** и **Button2**.
15. Выделить компонент **Button1** и в свойство **Text** занести текст «***Нарисовать график***». При нажатии на эту кнопку будет в компоненте **Chart1** рисоваться график функции
16. Выделить компонент **Button2** и в свойство **Text** занести текст «***Выход***».
17. В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 4.

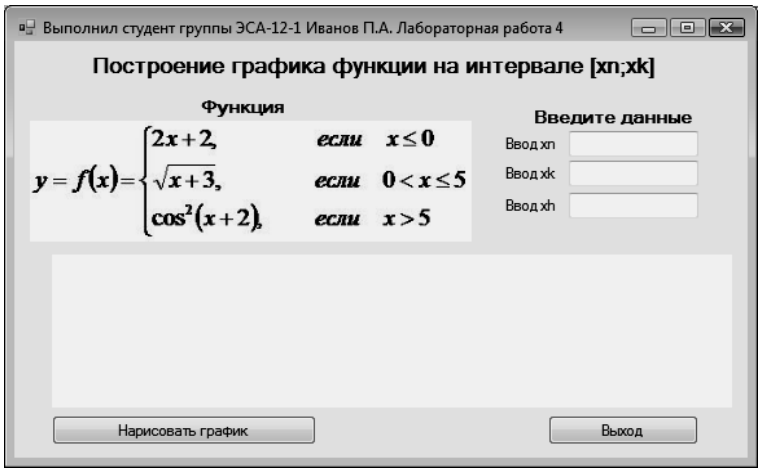


Рисунок 4 – Окно формы на этапе создания программы

18. Перейти к коду программы и после строки

```
#pragma once
```

подключить библиотеки использования математических функций, для этого вставить следующую строку:

```
#include<cmath>
```

19. В разделе ***using*** добавить строку:

```
using namespace System::Windows::Forms::DataVisualization::Charting;
```

20. Создадим событие ***Click*** для кнопки Button1 с надписью ***«Нарисовать график»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту Button1. Внесем изменения в код подпрограммы:

```
 double x,xn,xk,xh,y;
 Series^ plot1 = chart1->Series[0];
 //Очистка компонента Chart1
 plot1->Points->Clear();
 //Проверка, что введены данные xn, xk, xh и их преобразование в переменные типа
 Double
 if((textBox1->Text!="")&&(textBox2->Text!="")
 &&(textBox3->Text!=""))
 {
 xn = Convert::ToDouble(textBox1->Text); xk
 = Convert::ToDouble(textBox2->Text); xh =
 Convert::ToDouble(textBox3->Text);
 //Проверка правильности ввода данных
if ((xn>=xk)||(xh>(xk-xn)))
 MessageBox::Show( "Данные заполнены неверно", "Ошибка
 ввода данных",MessageBoxButtons::OK,
 MessageBoxIcon::Exclamation );
     }
 else
      {x=xn;
      while (x<=xk)
{if (x<=0) y=2*x+2;
 else
 if (x>0&&x<5) y=sqrt(x+3);
 else y=pow(cos(x+2),2);
```

```
//Нанесение точки с координатами X и Y в компоненте Chart1
plot1->Points->AddXY(x, y);
          x=x+xh;
        }}
}
else
{
MessageBox::Show( "Заполните, пожалуйста, данные", "Ошибка
ввода данных", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation);
}
```

21 Создадим событие ***Click***для кнопки **Button2**с надписью ***«Выход»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button2** и в полученную заготовку подпрограммы вставить код выхода из программы (уже использовался в предыдущих лабораторных работах).

22. На примере лабораторной работы 15 сделать проверку на корректность ввода данных XN, XK, XH, используя событие **Leave** в окне свойств (в код программы добавить 3 процедуры, проверяющие правильность ввода)

23. Запустить программу на выполнение, нажав на функциональную кнопку **F5**. Получим следующий вид окна (рис. 5).

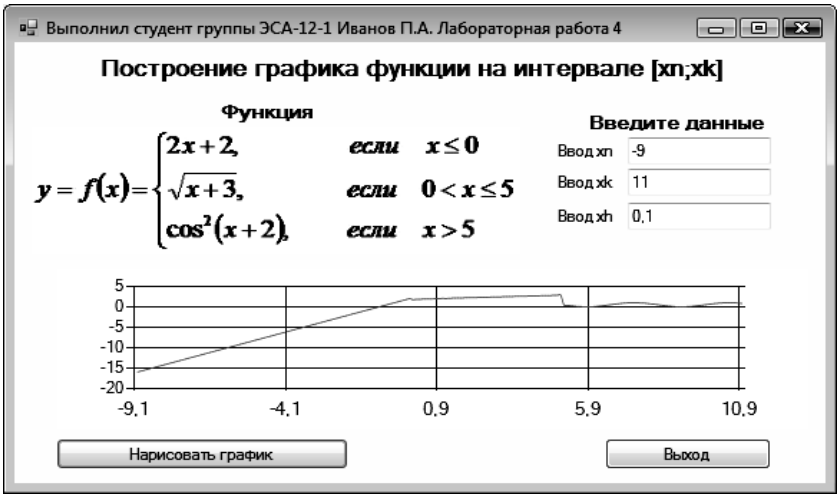


Рисунок 5 – Рабочий вид формы приложения

Индивидуальные задания

Создать Windows-приложение для построения графиков функций, которое предлагает пользователю ввести данные начала промежутка (xn), конца промежутка (xk) и шага изменения переменной (xh) (сделать проверку правильности ввода данных). На одной координатной плоскости должны быть построены три графика $f_1(x)$, $f_2(x)$, $f_3(x)$ разного цвета.

$$y = f(x) = \begin{cases} f_1(x), & x \le 0 \\ f_2(x) & , \quad 0 < x \le 5 \\ f(x), & x > 5 \end{cases}$$

Выражения для функций *f1(x)*, *f2(x)* и *f3(x)*, данные промежутка и шага выбрать из

таблицы 1 в соответствии с номером своего варианта.

Вариант задания	Функции			Начальное значение xn	Конечное значение xk	Шаг (xh)
	$f_1(x)$	$f_2(x)$	$f_3(x)$			
1	2	3	4	5	6	7
1	$\left\|\sqrt[3]{\frac{2x+5}{(x^3+2)}}\right\|$	$\frac{\sqrt{\sin(x^2+3)+4}}{x^2+2}$	$\frac{\sin(x+2)^3}{\ln\left\|x^2+3x+1\right\|}$	2,1	16,5	0,2
2	$\left\|5\right\|^{\sin(x)+2}x+\sin(x)$	$x^3+(\left\|x\right\|+1)^{0,1x}$	$\frac{\sin^3(x+2)}{\sqrt[4]{\sin^2 x+\cos^4 x}}$	–4,2	28,1	0,1
3	$\frac{\mathrm{tg}(x+3)^2}{\left\|x\right\|^{1,2}\sin 3x}$	$\frac{x^3-4x+2}{x^2+\sin(7x)-1}$	$\frac{tg(0,1\pi x^2)+x}{\cos^2(2x+3)}$	-1,7	45,3	0,3
4	$\frac{\cos(x^3-4x+4)}{x^3-\ln(\left\|x\right\|+1)}$	$\frac{\sin(x+2)^2}{\sqrt[3]{2x^2+x^4+1}}$	$\frac{\sqrt{\left\|x\right\|^3}\sin x^3}{\cos^2(x+1)}$	-2,25	34,9	0,5
5	$\frac{\sin^2(x+5)^2}{e^{-x}+\sqrt[3]{3x^2+1}}$	$\frac{x^4+2x^3-x}{\cos(x+3)^2}$	$\frac{tg(x^2+4x-1)}{2\sqrt{x^3}\sin(x^3)}$	2,45	25,2	0,1
6	$\left\|x\right\|^5 ctg(x+2)$	$\frac{5x+x^2}{(x^2+3)^3}$	$\frac{\sin^2(x+3)}{x^5-ctg(\pi x^3)}$	3,35	36,26	0,2
7	$\frac{\sin(x+3)}{x^5 ctg(2x^3)}$	$\frac{\left\|x\right\|+2}{\cos^2(x^3+2x+1)^2}$	$\frac{\sin^2(x+5)}{\sqrt[3]{\left\|x\right\|+2}-1}$	1,7	4,9	0,5
8	$\frac{e^x\ln\left\|x\right\|}{ctg(3x-1)^2}$	$\frac{5}{tg(2x+3)+1}$	$\frac{(3x-1)^2}{x^5+\sin(x+2)^3}$	–5,2	11,7	0,5
9	$e^{-2x}+\sqrt[7]{2x^4+x^2+1}$	$\frac{x^3+2x^2-4x}{x^5 ctg(2x^3)}$	$\frac{\cos^2(x+2)^3}{2\sqrt{x^3}\sin(x^3)}$	2,9	17,48	0,3
10	$\frac{tg^2(x+1)}{x^4+2x^3-x}$	$\frac{2x+2}{tg(2x-1)+1}$	$\frac{\cos(x+2)^2}{e^{-2x}+\sqrt[4]{3x^2+1}}$	–1,9	29,7	0,1
11	$\sqrt{\sin^2 x+\cos^4 x}$	$\ln^2(x)+\sqrt{x}$	$\mathrm{tg}^2(x)+\sqrt{x}$	–2,74	28,29	0,1
12	$x^3-\ln(\left\|x\right\|+1)$	$\frac{2x+2}{(\mathrm{tg}(2x-1)+1)}$	x^4-x^x	–1,25	9,39	0,4
13	x^4+2x^3-x	$e^{-x}+\sqrt[4]{x}$	$\ln(x^3+x^2)$	–1,78	11,99	0,5
14	$\frac{(3x-1)^2}{x^5}$	$\ln^2\left\|\sqrt{x+5}\right\|$	$\cos\left(\sqrt{1+x^2}\right)$	–2,46	28,8	0,6
15	$x^5\mathrm{ctg}(2x^3)$	$\frac{5}{\mathrm{tg}(2x+3)+1}$	$\mathrm{tg}(x^2+1)e^{-x}$	3,75	17,7	0,4

Контрольные вопросы

1 Методика построения графиков функций по их уравнениям (*для всех языков программирования*).

2 Можно ли использовать цикл For для построения графика функции? Если да, то как?

3 Вызов компонента Chart и его настройка.

4 Как построить нескольких графиков функций на одном компоненте Chart?

ЛАБОРАТОРНАЯ РАБОТА №35. ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ В ВИЗУАЛЬНОЙ СРЕДЕ НА ЯЗЫКЕ VISUAL C++

1.Краткие теоретические сведения

В языке программирования **C++** заложены средства для задания последовательностей упорядоченных данных. Такие последовательности называются *массивами*. *Массив* – это упорядоченная совокупность пронумерованных однотипных данных. В массивах должны быть упорядочены данные одного и того же типа. Номер элемента характеризуется целыми числами, называемыми *индексами.* **Массив характеризуется именем, размерностью и размером**.

Одномерный массив – это вектор (список связанных однотипных переменных). В частности, для векторов в приведенной записи индекс означает количество элементов. Для названия массива может быть использована переменная, состоящая из букв (буквы), букв с цифрами, букв с цифрами и знаком подчеркивания и т. д. в соответствии с правилами объявления переменных, принятых в языке **C++**. *Имя массива* образуется по общему правилу образования имен, т. е. представляет собой идентификатор, например, `A`, `B1`, `C8` и т. д. Однако оно не должно совпадать с именем ни одной простой переменной, используемой в той же программе.

Если размерность массива меньше, чем требуется, то компилятор не выдаст сообщения об ошибке. Выход за границы массивов должен следить только сам программист.

Общая форма записи одномерного массива:

`тип имя_массива[размер];`

В приведенной записи элемент *тип* объявляет базовый тип массива. Количество элементов, которые будут храниться в массиве с именем *имя_массива*, определяется элементом *размер*.

В языке **C++** индексация массива начинается с нуля. Например, если размер массива определен величиной 9, то в массиве можно хранить 10 элементов с индексацией 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Доступ к отдельному элементу массива осуществляется с помощью индекса. Индекс описывает позицию элемента внутри массива.

Индексы определяют положение элемента в массиве. *Число индексов определяет размерность массива*, т. е. форму его компоновки: одномерный, двумерный и т. д. Одномерный массив соответствует линейной таблице. Его элемент обозначается переменной с одним индексом: `A[0]`, `A[i]`, соответственно, первый и *i*-й элементы одномерного массива A.

Все массивы занимают смежные ячейки памяти, т. е. элементы массива в памяти расположены последовательно друг за другом. Ячейка памяти с наименьшим адресом относится к первому элементу массива, а с наибольшим – к последнему.

Селективная обработка массива – это выделение из массивов элементов, удовлетворяющих условию, и обработка выделенных фрагментов. Часто из выделенных фрагментов формируют новый (рабочий) массив, который далее и обрабатывают.

Наиболее часто встречаются такие условия обработки элементов массива:

```
- четные                          A[i] %2 ==0
- нечетные                        A[i] %2 !=0
- кратные  k                      A[i] % k ==0
- некратные k                     A[i] % k !=0
- стоящие на четных местах        (i+1) % 2==0
- стоящие на нечетных местах      (i+1) % 2!=0
```

```
- положительные                    A[i] >0
- отрицательные                    A[i] <0
- в интервале (x1,x2)              ((A[i]>x1)&& (A[i]<x2)).
```

Задание: Создать Windows-приложение, которое предлагает пользователю задать размер линейного массива, заполняет автоматически этот массив случайными целыми числами в диапазоне от –20 до 20, выводит элементы массива и далее выполняет обработку массива в соответствии с индивидуальным заданием. Индивидуальные задания взять из **лабораторной работы № 15.**

Пример выполнения лабораторной работы

Условие задачи.

Создать Windows-приложение, которое предлагает пользователю задать размер линейного массива, заполняет автоматически этот массив случайными целыми числами в диапазоне от –10 до 10, выводит элементы массива, затем по выбору пользователя определяет сумму четных элементов массива и количество положительных элементов массива

Ход выполнения

1 Войти в среду ***Visual Studio.***

2 Для формы изменить значение свойства **Text,** занеся следующие данные: ***«Выполнил студент Иванов П. А. Лабораторная работа 35».***

3 Свойству **FormBorderStyle** формы (стиль рамки окна) присвоить значение ***FixedToolWindow***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

4 В верхней части окна разместить контейнер **GroupBox1** и в свойстве **Text** занести текст «***Ввод элементов массива***».

5 На этом же контейнере **GroupBox1** разместить пять компонентов: **Label1**, **Label2**, **TextBox1**, **TextBox2** и **Button1**. Для первого компонента **Label1** в свойство **Text** занести текст «***Введите число элементов массива***». Для второго компонента **Label2** в свойство **Text** занести текст «***Исходный массив***». Для компонента **TextBox2** в свойстве **ReadOnly** (только чтение) присвоить значение ***true***, запрещающее пользователю заносить в компонент какие-либо данные. Для компонента **TextBox1** данные будут вводиться с клавиатуры с помощью компонента **TextBox**. В свойства **Text** этого компонента ввести какое-либо значение – значение по умолчанию. Это значение будет показываться при запуске приложения на выполнение. При выполнении приложения, его можно будет заменить другим.

6 Выделить компонент **Button1** и в свойство **Text** занести текст «***Создать массив***». При нажатии на эту кнопку будет автоматически создаваться массив с помощью генератора случайных чисел, элементы которого будут в указанном диапазоне [–10; 10] выведены в компоненте **TextBox2**. Получим следующую форму (рис. 1).

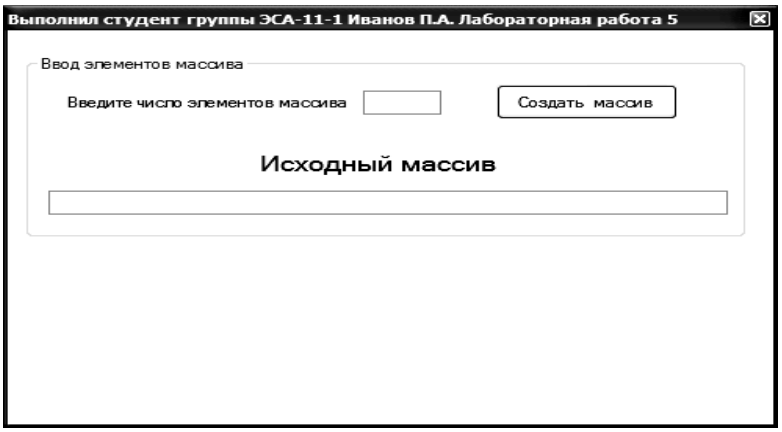


Рисунок 1 – Окно формы на этапе создания программы

8. В нижней части окна разместить контейнер **GroupBox2** и в свойстве **Text**занести текст «***Найти***».
9. На этом же контейнере **GroupBox2** разместить шесть компонентов: **TextBox3**, **TextBox4**, **checkBox1**, **checkBox2**, **Button2** и **Button3**.
10. Выделить компонент **checkBox1** и в свойство **Text** занести текст «***сумму четных элементов массива***». Аналогично для компонента **checkBox2** в свойство **Text** занести текст «***количество положительных элементов массива***».
11. . Выделить компонент **Button2** и в свойство **Text** занести текст «***Вычислить***». Аналогично для компонента **Button3** в свойство **Text** занести текст «***Выход***».
12. Компоненты **TextBox3** и **TextBox4** будут использованы для вывода результатов нахождения задач, поэтому необходимо запретить ввод в них данных пользователем. Для этого свойству **ReadOnly**(только чтение) присвоить значение ***true***, запрещающее пользователю заносить в компонент какие-либо данные.
13. В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 2.

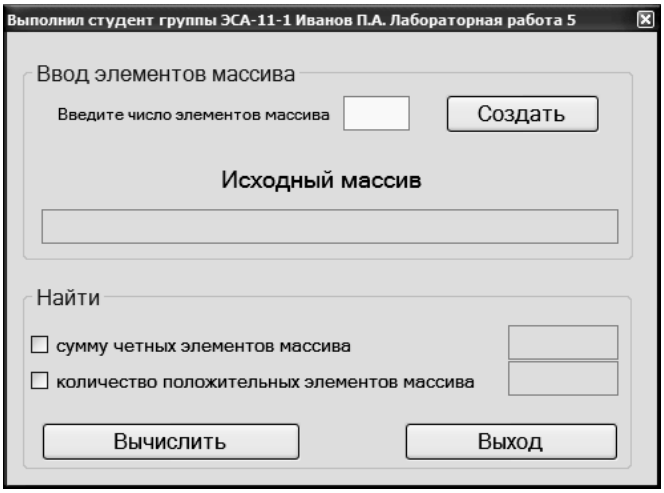


Рис.2 – Окно формы программы

14. Перейти к коду программы и после строки

```
#pragma once
```

подключить библиотеки использования математических функций, описание переменных и размера массива, для этого вставить следующие строки:

```
#include<cmath>
#include <stdlib.h>
#define m 15
int i,n;
```

```
int A[m];
```

15. Создать событие ***Click*** для кнопки **Button1** с надписью ***«Создать»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button1**. Внести изменения в код подпрограммы:

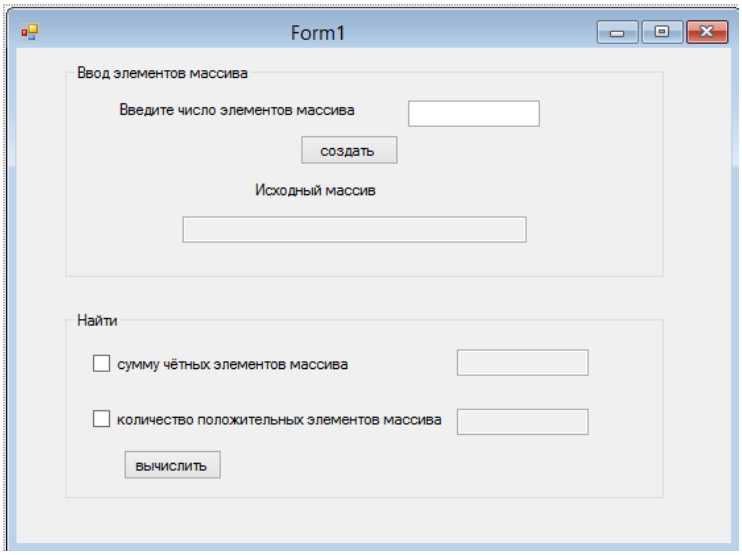
```
//Проверка, что не пустой компонент textBox1
      if(textBox1->Text!="")
      {n=Convert::ToInt32(textBox1->Text); }
      Else {MessageBox::Show( "Заполните, пожалуйста, данные",
      "Ошибка ввода данных", MessageBoxButtons::OK,
      MessageBoxIcon::Exclamation );}
//очистка компонента textBox2
{textBox2->Text = "";
//Процесс создания массива и заполнение компонента textBox2 случайными числами
из диапазона [-10;10]
for (i = 0; i < n; ++i)
{
A[i] = rand( ) % 21-10;
this->textBox2->AppendText(A[i]+"  ");
}
}
```

15. Создать событие ***Click*** для кнопки **Button2** с надписью ***«Вычислить»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button2** и в полученную заготовку подпрограммы вставить код для расчетов:

```
//Обнуляем переменные kol и sum
int kol=0; int sum=0;
     for (i = 0; i < n; ++i)
{
//Проверяем условие и находим сумму
if (A[i]%2==0){sum=sum+A[i];}
//Проверяем условие и находим количество
if (A[i]>0){kol=kol+1;}
}
//Если компоненты checkBox включены, то выводим результаты заданий
в компоненты textBox
if (checkBox1->Checked==true)
{this->textBox3-> Text=Convert::ToString (sum);}
if (checkBox2->Checked==true)
{this->textBox4-> Text=Convert::ToString (kol);}
```

16. Создать событие ***Click***для кнопки **Button3**с надписью ***«Выход»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button3** и в полученную заготовку подпрограммы вставить код выхода из программы (уже использовался в предыдущих лабораторных работах).
17. Запустить программу на выполнение, нажав на функциональную кнопку **F5**.

Рабочая форма программы:



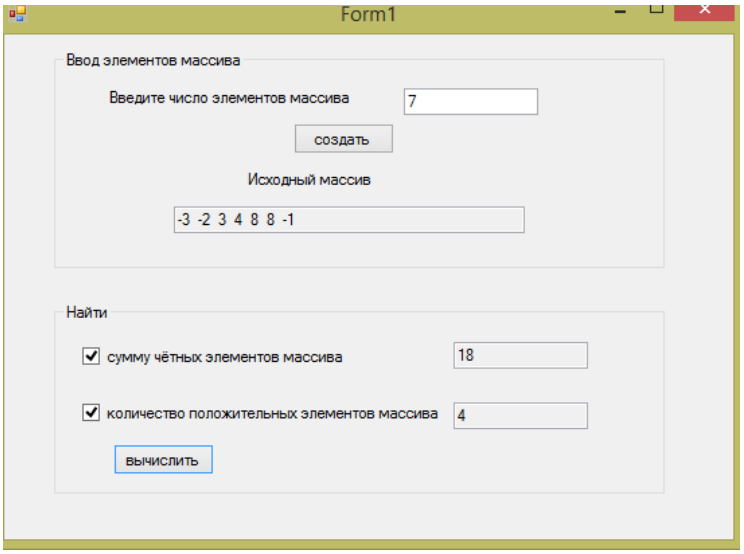
Текст программы:

```
#include<stdlib.h>
#define m 15
int i,n;
int A[m];

private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
    //Проверка, что не пустой компонент  textBox1
     if(textBox1->Text!="")
    { n=Convert::ToInt32(textBox1->Text);
    }
     else
    { MessageBox::Show("Заполните, пожалуйста данные",
                "Ошибка ввода данных", MessageBoxButtons::OK,
                MessageBoxIcon::Exclamation);
      }
    // очистка компонента textBox2
    { textBox2->Text="";
    //Процесс создания массива и заполнение компонента  textBox2
    //случайными числами из диапазона[-10; 10]
    for(i=0;i<n;i++)
    {   A[i]=rand()%21-10;
        this->textBox2->AppendText(A[i]+"  ");
```

```
    }
  }
}
private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e) {
// Обнуляем переменные kol и sum
    int kol=0,sum=0;
    for(i=0;i<n;i++)
    {
        //Проверяем условие и находим сумму
        if(A[i]%2==0) sum+=A[i];
        //Проверяем условие и находим количество
        if(A[i]>0) kol++;
    }
//Если компоненты checkBox включены, то выводим результаты заданий в
компоненты textBox
      if(checkBox1->Checked==true)
      this->textBox3->Text=Convert::ToString(sum);
      if(checkBox2->Checked==true)
      this->textBox4->Text=Convert::ToString(kol);
}
};
}
```

Тестирование программы

Результат выполнения программы может выглядеть следующим образом:



Контрольные вопросы

1. Понятие одномерного числового массива в языке **C++**?
2. Как организуется индексирование числовых массивов в языке **C++**?
3. Для чего применяется начальная инициализация числовых массивов при дальнейшем их использовании?
4. Как использовать генератор случайных чисел для заполнения массива элементами?
5. Условия селективной обработки элементов массива.

6. Нахождение минимального и максимального элементов массива.
7. С помощью каких визуальных компонентов можно сделать ввод элементов массива в ручном режиме?

ЛАБОРАТОРНАЯ РАБОТА №36. РАБОТА С ДВУМЕРНЫМИ МАССИВАМИ В ВИЗУАЛЬНОЙ СРЕДЕ НА ЯЗЫКЕ VISUAL C+ +

1.Краткие теоретические сведения

Двумерный массив (матрица) — это упорядоченная совокупность пронумерованных однотипных данных. **Матрица характеризуется именем, размерностью и размером**.

Для *имени* матрицы может быть использована переменная, состоящая из букв (буквы), букв с цифрами, букв с цифрами и знаком подчеркивания и т. д. в соответствии с правилами объявления переменных, принятых в языке **C++**. *Имя матрицы* образуется по общему правилу образования имен, т. е. представляет собой идентификатор, например, `A`, `Bl`, `C_8` и т. д. Однако оно не должно совпадать с именем ни одной простой переменной, используемой в той же программе.

В матрице позиция любого элемента определяется двумя целыми числами, называемыми *индексами.* Индексы каждого из размеров массива начинаются с 0 (с нуля). Число индексов определяет *размерность* массива *(у матрицы два индекса: первый показывает номер строки, а второй номер столбца, где элемент находится).*

Двухмерный массив (*матрица*) представляет собой список одномерных массивов.

Общая форма записи двухмерного массива:

```
тип имя_массива[размер1] [размер2];
```

В приведенной записи *размер1* означает количество строк двухмерного массива, а *размер2* – количество столбцов.

Место хранения для всех элементов матрицы определяется во время компиляции. Память, выделенная для хранения массива, используется в течение всего времени существования массива.

Для двухмерных массивов общий размер массива в байтах вычисляется по формуле:

```
всего байт = число строк * число столбцов * размер типа в байтах.
```

Для определения размера типа в байтах применяется функция `sizeof()`, которая возвращает целое число. Например, `sizeof (float)`.

Инициализация матрицы

При инициализации матрицы для улучшения наглядности элементы инициализации каждого измерения можно заключать в фигурные скобки.

Пример инициализации двухмерного массива:

```
int  MN[3][4] = {
{1, 2, 3, 4},
{5, 6, 7, 8},
{9, 10, 11, 12}
};
```

Массив MN[3][4] – это матрица, у которой 3 строки и 4 столбца.

В языке **C++** нельзя присвоить одну матрицу другой. Для передачи элементов одной матрицы другой необходимо выполнить присвоение поэлементно.

Для доступа к элементу матрицы следует указать имя матрицы с последующим числом (индексом), заключенным в квадратные скобки.

Элементы массива можно использовать в любом выражении точно также, как и

значение константы или переменной.

Например:

```
a[0][0]=11.2;
a[1][2]=10.2;
a[3][1]=22.1;
a[4][2]=1.1;
Y = 2*a[0][1] – a[1][0];
```

Селективная обработка элементов матрицы

Селективная обработка матрицы – это выделение из матрицы элементов, удовлетворяющих условию, и обработка выделенных фрагментов. Часто из выделенных фрагментов формируют новый (рабочий) массив, который далее и обрабатывают.

Наиболее часто встречаются такие условия обработки элементов массива:

– четные	A[i] %2 ==0
– нечетные	A[i] %2 !=0
– кратные k	A[i] % k ==0
не кратные k	A[i] % k !=0
– положительные	A[i] >0
– отрицательные	A[i] <0
– в интервале [x1,x2]	((A[i]>=x1)&&(A[i]<=x2))

При обработке матриц часто приходится выделять элементы:

– k–й строки	A[i][j], где i==k-1, j=0,…, M-1
– k–го столбца	A[i][j], где i=0, …, N-1; j==k-1

а для квадратных матриц (M=N) также:

– главной диагонали	i==j,
– побочной диагонали	j=N-1-i,
– наддиагональные	j>i,
– поддиагональные	j<i,

где i – номер строки, j – номер столбца.

Например, для нахождения условия «найти элементы 3 столбца 2 строки», укажем условие с помощью условного `IF`:

```
If (i==1&&j==2) {действие};
```

Использование элемента управления `DataGridView` для работы с матрицами

`DataGridView` – это элемент управления, который может отображать буквально любой вид данных в ячейках прямоугольной сетки.

Элемент управления `DataGridView` позволяет отображать и изменять прямоугольный массив данных из множества различных источников данных. Его можно использовать также для отображения практически любых данных, созданных непосредственно в программе. По своей сущности это сложный элемент управления, который обеспечивает огромную гибкость при его применении, и преимуществами множества его функциональных возможностей можно воспользоваться через множество свойств, функций и событий. В то же время применение элемента управления `DataGridView` может быть поразительно простым. Можно не обращать внимания

на внутреннюю сложность и использовать его посредством инструмента Form Design (Конструктор форм), которое берет на себя заботу обо всех основных характеристиках.

Данные элемента управления DataGridView отображаются в прямоугольном массиве ячеек, которые можно представить в виде коллекции строк или столбцов. Каждый столбец ячеек имеет в верхней части ячейку заголовка, которая, как правило, содержит идентифицирующий этот столбец текст, а в начале каждой строки расположена ячейка заголовка строки, как показано на рис. 1. Обращение к строкам и столбцам ячеек выполняется через свойства объекта DataGridView. Свойство Rows возвращает значение типа DataGridRowCollection, представляющее коллекцию всех строк, а обращение к конкретной строке выполняется с помощью индекса, что можно видеть на рис. 1

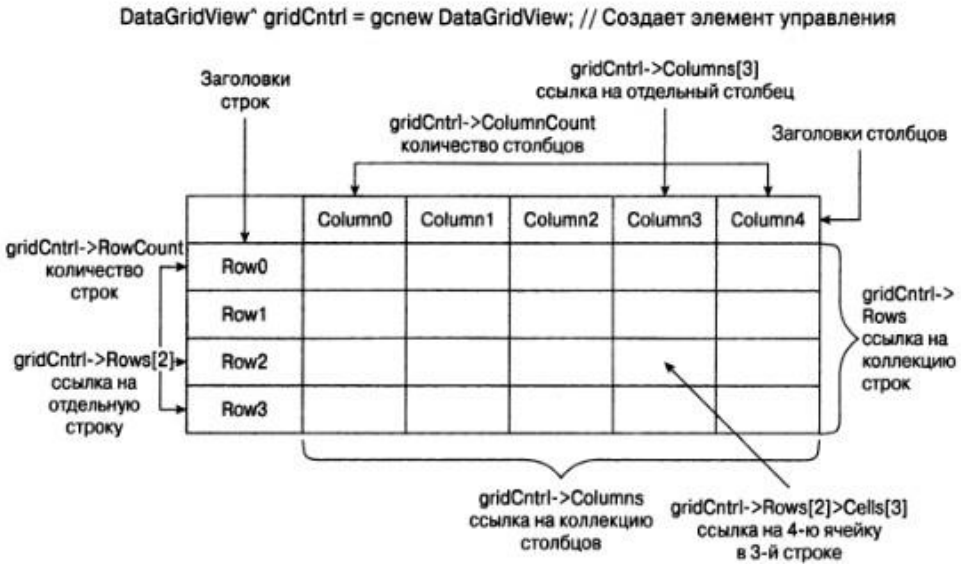


Рис.1 – Компонент DataGridWiew

Аналогично свойство Columns элемента управления возвращает значение типа DataGridViewColumnCollection, которое также можно индексировать для обращения к конкретному столбцу. Индексация строк и столбцов осуществляется, начиная с нуля. Свойство Cells объекта Data- GridRowCollection представляет коллекцию, содержащую ячейки строки, и это свойство можно индексировать для получения доступа к конкретной ячейке в строке. Пример ссылки на четвертую ячейку в третьей строке приведен на рис. 1.

Количество строк доступно как значение свойства RowCount, а свойство ColumnCount возвращает количество столбцов. Вначале, когда элемент управления еще нс связан с источником данных, он не будет содержать ни строк, ни столбцов. Количество столбцов и / или строк можно определить, устанавливая значения свойств элемента управления, но при его использовании для отображения данных из источника данных это действие выполняется автоматически.

Элемент управления DataGridView применяется в трех различных режимах. Мы будет применять «*несвязанный режим*». В несвязанном режиме передача данных элементу управления выполняется вручную, как правило, с помощью функции Add () применительно к свойству Rows элемента управления. Этот режим следует использовать для отображения сравнительно небольших объемов данных.

В несвязанном режиме элемент управления DataGridView можно использовать для отображения в приложении любых данных, которые могут быть отображены в табличном виде. Потому, этот инструмент очень удобен для отображения данных во множестве разнообразных приложений.

Использование элемента управления DataGridView в несвязанном режиме.

Данные в элементе управления DataGridView хранятся в прямоугольной структуре, определяемой свойствами Rows и Columns элемента управления. В несвязанном режиме добавление данных в элемент управления выполняется с помощью функции Add () применительно к свойству Rows. Но прежде чем в элемент управления можно будет добавлять строки, потребуется определить столбцы – задать количество элементов в строке. Программная установка свойства ColumnCount элемента управления определяет количество столбцов и указывает, что элемент управления будет работать в несвязанном режиме. Следующие операторы создают элемент управления, обращение к которому выполняется посредством дескриптора DataGridView, а затем устанавливают количество столбцов, равное 3:

```
DataGridView^ dataGridView = dcnew DataGridView; DataGridView-
>ColumnCount = 3; // Устанавливает количество столбцов.
```

При желании столбцы в элементе управления можно пометить путем установки свойства Name для каждого столбца, указывая заголовки, идентифицирующие данные в каждом из них.

Это можно было бы выполнить так:

```
dataGridView->Columns[0]->Name="Name";
dataGrldView->Columns[l]->Name="Phone Number";
dataGridView->Columns[2]->Name="Address";
```

Свойство Columns элемента управления – индексированное свойство, поэтому доступ к отдельным столбцам можно получить с помощью значений индексов, начинающихся с 0. Таким образом, эти три оператора снабжают метками три столбца в элементе управления DataGridView. Заголовки столбцов можно определить также в окне Properties (Свойства) элемента управления.

Свойство Rows возвращает значение, представляющее собой коллекцию типа **DataGridViewRowCollection**, который определен в пространстве имен System:: Windows: :Forms.

Свойство Count упомянутой коллекции возвращает количество строк. Кроме того, коллекция имеет также свойство начальной индексации, возвращающее строку в позиции данного индекса. Коллекция строк обладает множеством функций, наиболее полезных, предназначенных для добавления и удаления строк

2.Порядок выполнения работы

2.1 Последовательность выполнения лабораторных работ

1. Получение у преподавателя варианта задания для предварительной проработки материала дома.
2. Предварительная проработка материала (подготовка необходимых формул и изучение рекомендованного теоретического материала).
3. Написание программы и ее отладка на ПК.
4. Тестирование программы.
5. Составление отчёта по итогам лабораторной работы.
6. Защита лабораторной работы.

3.2 Содержание отчёта

1. Номер работы и её название.
2. Задание для выполнения.
3. Код программы на языке Visual C++.
4. Описание входных и выходных данных.
5. Блок-схема программы.

6. Зарисованный вид Forms программы
7. Результат тестирования программы.

4. Пример выполнения работы

Создать Windows-приложение, которое предлагает пользователю ввести количество строк и столбцов для создания таблицы, в которую будут заноситься элементы матрицы, затем по выбору пользователя определяет: произведение отрицательных элементов матрицы и количество четных элементов матрицы.

Ход выполнения

1. Войти в среду ***Visual Studio.***
2. Для формы изменить значение свойства **Text**, занеся, например, следующие данные: ***«Выполнил студент Иванов П. А. Лабораторная работа 36».***
3. Свойству **FormBorderStyle** формы (стиль рамки окна) присвоить значение ***FixedToolWindow***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.
4. В верхней части окна разместить контейнер **GroupBox1** и в свойстве **Text** занести текст «***Ввод данных матрицы***».
5. На этом же контейнере **GroupBox1** разместить семь компонентов: **Label1**, **Label2**, **Label3**, **Label4**, **TextBox1**, **TextBox2** и **DataGridWiew1**. Для первого компонента **Label1** в свойство **Text** занести текст «***Задайте размер матрицы***». Для второго компонента **Label2** в свойство **Text** занести текст «***N-строк***». Для третьего компонента **Label3** в свойство **Text** занести текст «***M-столбцов***».

   Для четвертого компонента **Label4** в свойство **Text** занестти текст «***Матрица A[n,m]***».
6. Выделить компонент **DataGridView1** и для свойств **RowHeadersVisible** (отображение заголовка строк) и **ColumnHeadersVisible** (отображение заголовка столбцов) выбрать параметр ***False***.
7. Ниже контейнера **GroupBox1** разместить контейнер **GroupBox2** и в свойстве **Text** занести текст «***Найти***».
8. Выделить компонент **DataGridView2** и на него разместить два компонента **checkBox1** и **checkBox2**.
9. Выделить первый компонент **checkBox1** и в свойство **Text** занести текст «***произведение отрицательных элементов матрицы***». Аналогично для компонента **checkBox2** в свойство **Text** занести текст «***количество четных элементов матрицы***». Напротив этих компонентов **checkBox** разместить два компонента: **TextBox3** и **TextBox4**.
10. Ниже компонента **GroupBox2** разместить три компонента: **Button1**, **Button2** и **Button3**. На первом компоненте **Button1** в свойство **Text** занести текст «***Создать таблицу***». Для второго компонента **Button2** в свойство **Text** занести текст «***Перенести данные из таблицы в массив и решить задание***». Для третьего компонента **Button3** в свойство **Text** занести текст «***Выход***».
11. В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 2.
12. Перейти к коду программы и после строки

    ```
    #pragma once
    ```

    подключить библиотеки использования математических функций, описание переменных и размера массива, для этого вставить следующие строки:

    ```
    #include<math.h>
    ```

```
int ,n,kol,kol2,p;
int A[50][50];
```

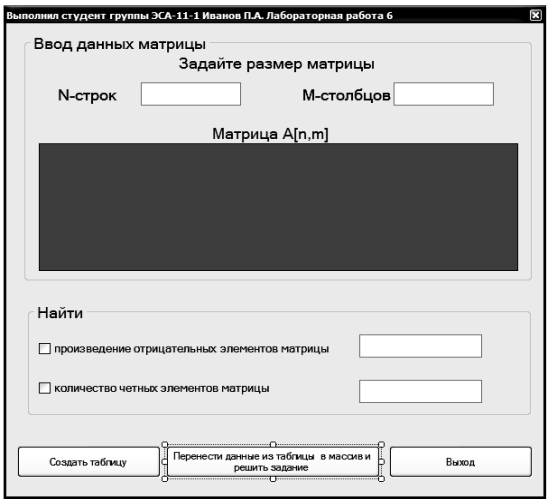


Рис.2 – Окно формы программы

13. Создать событие ***Click*** для кнопки **Button1** с надписью ***«Создать таблицу»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button1**. Внести изменения в код подпрограммы:

```
    //Проверка, что не пустые компоненты textBox1 и textBox2
if ((textBox1->Text!="")&&(textBox2->Text!=""))
{m = Convert::ToInt32(textBox1->Text);
n = Convert::ToInt32(textBox2->Text);
    //Чистка столбцов компонента DataGridView, если они не пусты
    dataGridView1->Columns->Clear();
    //Заполнение компонента DataGridView столбцами
        dataGridView1->ColumnCount = n;
    //Заполнение компонента DataGridView строками
 dataGridView1->RowCount =m;}
 else
 {MessageBox::Show( "Заполните, пожалуйста, данные",
 "Ошибка ввода данных", MessageBoxButtons::OK,
  MessageBoxIcon::Exclamation );}
```

14. Создать событие ***Click*** для кнопки **Button2** с надписью ***«Перенести данные из таблицы в массив и решить задание»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button2** и в полученную заготовку подпрограммы вставить код для расчетов:

```
    //переменную kol и kol2 обнуляем, а переменную p присваиваем единице
    kol=0;kol2=0;p=1;
    //Производим считывание из ячеек таблицы и вносим данные в массив
for (int i = 0; i < m; i++)
for (int j = 0; j < n; j++)
{
A[i][j]=Convert::ToSingle(this->dataGridView1->
Rows[i]->Cells[j]->Value);
if (A[i][j]<0)    {p=p*A[i][j];kol2++;}
if(A[i][j]%2==0) {kol++;}
```

```
//Вывод данных нахождения произведения отрицательных элементов матрицы
if ((checkBox1->Checked==true)&&(kol2!=0))
 {this->textBox3->Text=Convert::ToString (p);}
else
if (checkBox1->Checked==true) {this->textBox3-
>Text=Convert::ToString ("нет элементов");}
//Вывод данных нахождения количество четных элементов матрицы
if ((checkBox2->Checked==true)&&(kol!=0)) {this->
textBox4->Text=Convert::ToString (kol);}
else
if (checkBox2->Checked==true) {this->
textBox4->Text=Convert::ToString ("нет элементов");}
 return;
```

15. Создать событие ***Click***для кнопки **Button3**с надписью ***«Выход»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button3** и в полученную заготовку подпрограммы вставить код выхода из программы (уже использовался в предыдущих лабораторных работах).
16. Запустить программу на выполнение, нажав на функциональную кнопку **F5**. Получим следующий вид окна (рис. 3).

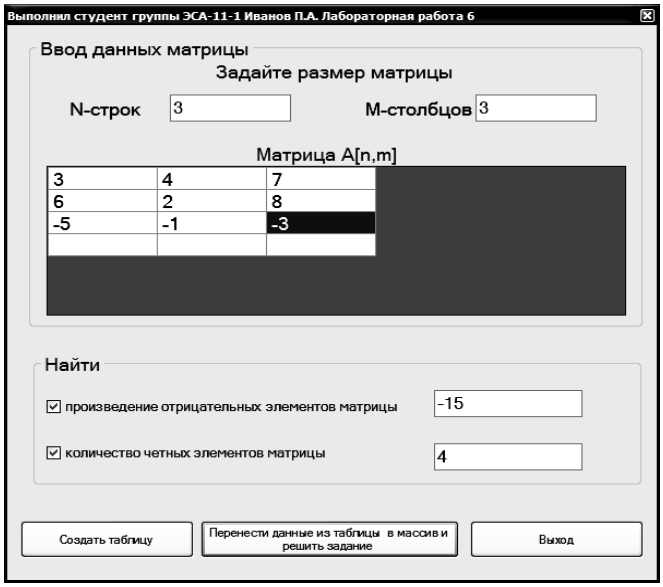


Рис.3 – Рабочий вид формы приложения

5. Индивидуальные задания

Создать Windows-приложение, которое предлагает пользователю задать количество строк и столбцов матрицы, при нажатии на кнопку автоматически создается таблица, в ее ячейки автоматически вводятся элементы матрицы с помощью генератора случайных чисел.

Данные взять из таблицы 1.

Таблица 1

Вариант	Задание	Условие задания
1	***а***	Найти количество элементов, больших заданного числа С (ввод числа С сделать с клавиатуры)
	б	Найти сумму элементов, расположенных по периметру
	в	В матрице А(4;4) найти сумму произведения четных чисел 1-ой строки и произведения положительных чисел 3-го столбца
2	***а***	Найти минимальный по модулю элемент и номер строки и столбца, где он находится
	б	Найти сумму наибольшего положительного и наименьшего четного
	в	Подсчитать количество кратных 3 чисел 2-ой строки и количество четных чисел 1-го столбца матрицы А(6; 6)
3	***а***	Найти произведение элементов, меньших заданного числа Т (ввод числа Т сделать с клавиатуры)
	б	Найти произведение элементов, находящихся на главной диагонали
	в	Найти разность произведения нечетных чисел 3-ей строки и произведения отрицательных чисел 1-го столбца матрицы А(4; 4)
4	***а***	Найти сумму положительных кратных 5 элементов
	б	Найти произведение элементов, находящихся на побочной диагонали
	в	В матрице А(8; 8) найти разность произведения нечетных чисел 3-ей строки и суммы положительных чисел 6-го столбца
5	***а***	Найти произведение отрицательных четных элементов.
	б	Найти сумму элементов, находящихся на главной диагонали.
	в	В матрице А(5; 5) найти сумму количества четных чисел 2-ой строки и количества отрицательных чисел 4-го столбца
6	***а***	Найти количество положительных нечетных элементов
	б	Найти сумму элементов, находящихся на побочной диагонали
	в	В матрице А(3;3) найти произведение количества нечетных чисел 1-ой строки и количества положительных чисел 3-го столбца
7	***а***	Найти количество элементов, меньших числа 5
	б	Найти количество отрицательных элементов, находящихся на главной диагонали
	в	Найти максимальный элемент 3-го столбца и сумму нечетных элементов 1-ой строки матрицы А(5; 5)
8	***а***	Найти произведение положительных кратных 3 элементов
	б	Найти количество отрицательных элементов, находящихся на побочной диагонали
	в	В матрице А(6; 6) найти произведение суммы четных чисел в 3-ей строке и суммы отрицательных чисел 1-го столбца

9	***а***	Найти сумму отрицательных некратных 5 элементов
	б	Найти количество четных элементов, находящихся на главной диагонали
	в	Найти произведение суммы положительных чисел в 4-ом столбце на количество четных чисел 2-ой строки матрицы A(6; 6)
10	***а***	Найти квадрат максимального элемента и номер строки и столбца, где он находится
	б	Найти количество четных элементов, находящихся на побочной диагонали
	в	В матрице A(7; 7) найти разность количества положительных чисел 1-ой строки и количества четных чисел 3-го столбца
11	***а***	Найти сумму четных элементов из интервала [–10; 10] матрицы A(4; 4)
	б	Найти количество нечетных элементов, находящихся на главной диагонали
	в	Найти произведение количества четных элементов 3 строки на сумму нечетных элементов 2 столбца матрицы A(4; 4)
12	***а***	Найти количество кратных 3 элементов из интервала [–6; 8] матрицы A(5; 5)
	б	Найти количество нечетных элементов, находящихся на побочной диагонали
	в	В матрице A(5; 5) найти произведение количества четных чисел 2-ой строки и количества отрицательных чисел 4-го столбца
13	***а***	Найти произведение отрицательных нечетных элементов матрицы A(5; 5)
	б	Найти сумму четных элементов, находящихся на главной диагонали
	в	В матрице A(5; 5) найти произведение количества нечетных чисел 3-го столбца и количества отрицательных чисел 3 строки
14	***а***	Найти количество положительных элементов из интервала [–5; 6] матрицы A(6; 6)
	б	Найти сумму четных элементов, находящихся на побочной диагонали
	в	В матрице A(6; 6) найти произведение суммы кратных 3 чисел 2-ей строки и суммы отрицательных чисел 2-го столбца
15	***а***	Найти максимальный по модулю элемент и номер строки и столбца, где он находится
	б	Найти сумму элементов, кратных 3, находящихся на главной диагонали
	в	Найти произведение количества четных чисел во 2-ом столбце на количество нечетных чисел 2-ой строки матрицы A(4; 4)