Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

ФИЛИПОВСКИЙ ВЛАДИМИР МИХАЙЛОВИЧ

Теория и Технология Программирования. Рекомендации по выполнению практических заданий

Санкт-Петербург
2022

УДК 519.6

Филиповский В.М. Теория и Технология Программирования. Рекомендации по выполнению практических заданий: Учебное пособие.- СПб.: СПбПУ, 2022.- 141 с., ил.

Пособие соответствует ФГОС ВО по направлению подготовки 27.03.04 «Управление в технических системах» (уровень бакалавриата).

Рассматриваются последовательность действий и основные подходы к разработке программ при решении задач различных типов. На всевозможных примерах показываются особенности написания программ для широкого класса задач с использованием массивов, структур, списков, деревьев, а также циклов и рекурсий. Особенное внимание уделяется тестированию разработанных программ с применением трассировочных таблиц (контрольных печатей). Ряд задач посвящено созданию задач с использованием классов, шаблонов, перегрузок, а также разработке оконных приложений в среде MS Visual Studio.

Пособие содержит множество примеров написания программ, а также образцы оформления отчетов по отдельным задачам. В Приложении приведены наборы задач по различным темам.

Изложение материала опирается на традиционные методы и накопленный богатый опыт разработки грамотных алгоритмов и программ с использованием стандартных (типовых) схем.

Учебное пособие предназначено студентам Высшей Школы Киберфизических Систем и Управления Института Компьютерных Наук и Технологий (программы «Управление в технических системах») в качестве пособия для изучения лекционных курсов и выполнения практических работ по курсам "Теория и технология программирования", "Практикум по программированию", "Программирование и Алгоритмизация" и других, связанных с разработкой программ для ЭВМ.

Ключевые слова: Алгоритм, оператор, переменная, метод пошаговой детализации, функции, параметры, циклы, рекурсии, структуры, массивы, списки, деревья, тестирование, трассировочные таблицы, базы данных, шаблоны, перегрузка, оконный интерфейс.

Оглавление

Введение

Программирование и алгоритмизация, теория и технология программирования (ТТП), практикум по программированию, - основные начальные курсы, с которыми встречаются студенты, поступившие учиться в Высшую школу киберфизических систем и управления института компьютерных наук и технологий. Эти курсы, наряду с математикой и физикой, являются **базовыми**, на которых строится весь процесс обучения. Без умения разрабатывать (и отлаживать) грамотные программы невозможно успешное обучение в институте.

Написание программ, - это навык, который приобретается и совершенствуется на протяжении долгого времени. Даже самый опытный программист когда-то был новичком. Сам язык программирования для получения этого навыка существенного значения не имеет. Изучение структур программ и логики преобразования в них информации являются наиболее важными, эти знания и навыки можно приобрести при написании программ на любом языке.

Разработка каждой программы начинается с формулировки ответов на три основных вопроса:

- **что дано?**
- **что надо получить?**
- **каков должен быть результат?**

И только потом переходят к разработке алгоритма, отвечающего на вопрос:

- **как получить результат?**

Построение большинства алгоритмов предполагает использование **метода декомпозиции** (расчленение задачи на ряд более простых) с применением **типовых структур алгоритма**.

В процесс кодирования алгоритма происходит постоянное изучение языка. При этом под рукой у студента всегда должен быть соответствующий учебник, вполне допустимо искать подсказки в интернете, смотреть, как другие люди решают подобные проблемы. Как правило, необходимо понимать (и стремиться к этому), почему определенные программы и их фрагменты "ведут себя" определенным образом.

Существенную роль играет прямое общение с преподавателем или иным специалистом в области программирования.

В повышении квалификации программиста большую роль играют изучение математики и других наук с компонентами логики, так как эти предметы часто требуют владения программированием на высоком уровне.

При написании программы следует быть терпеливым. Не всегда результат получается с первого раза. В большинстве случаев - это длительный процесс,

состоящий из последовательности небольших этапов, неоднократно повторяемый, переделываемый.

По некоторым оценкам, человек должен заниматься программированием около 15 000 часов, прежде чем он станет высококвалифицированным специалистом. Годы постоянной практики необходимы для овладения мастерством программирования.

Программистом становится только тот, кто программирует!

1. Правила по оформлению отчетов

При выполнении практических работ по ТТП студенты должны результаты своих трудов представить не только в виде работающей программы, но изложить их в специальном документе, называемом отчетом. Все отчеты выполняются по единому образцу, определенном в Университете.

Отчет пишется на листах формата А4. Допускается оформление отчета "от руки". Титульный лист выполняется на отдельном листе в строгом соответствии с образцом, приведенным в Приложении П 1. Оборотная сторона титульного листа остается чистой. Все остальные страницы отчета могут быть заполнены информацией с обеих сторон.

Структура отчета обычно стандартная и включает в себя:

- задание;
- описание последовательности его выполнения;
- результаты, полученные при выполнении работы;
- выводы (заключение).

В силу того, что практические работы заключаются в основном в написании программ, то в отчете в обязательном порядке содержатся:

- описание главной программы;
- описание отдельных функций по мере их использования в программе.

Описание любого программного модуля должно включать:

- схема алгоритма;
- описание переменных, используемых в нем;
- исходный текст данного модуля (функции).

В Выводах пишется Заключение о ходе выполнения работы, соответствии полученных результатов ожидаемым, дается характеристика результатов задачи в целом и отдельных ее этапов, высказывается собственное мнение студента о работе и используемых методиках, функциях и прочее.

Весь графический материал, вставляемый в отчет, а именно: схемы алгоритмов, таблицы, графики, - выполнять только черным цветом. Если схемы алгоритмов либо графики изображаются вручную, то возможно использовать для этого простой карандаш.

Пример готового отчета приведен в Приложении П 2.

Отчет в обязательном порядке подписывается студентом (на титульном листе), где указывается дата выполнения работы. Преподаватель после проверки выполненной работы и отчета по ней ставит на титульном листе соответствующую отметку.

Рекомендуется титульный лист с отметкой преподавателя о том, что работа выполнена и принята, отсканировать и выложить в Личный Кабинет Студента. Сам отчет остается у преподавателя. В отдельных случаях уже принятый отчет по электронной почте высылается преподавателю.

2. Типы рассматриваемых задач

2.1. Тестирование программы

На этапе верификации (проверки) программного обеспечения возникают задачи, когда требуется понять, как работает тот или иной фрагмент программы, каков будет результат, возможны ли (и какие) ошибки в рассматриваемой части программы.

Предварительно составляется трассировочная таблица, где отслеживается изменение всех переменных на каждом шаге выполнения программы. Для этой цели можно использовать различные электронные таблицы, например, Эксель, либо производить вычисления вручную.

Затем интересующий фрагмент оформляется в виде функции с формированием необходимого заголовка, формальных параметров, локальных переменных и результата. В главной программе осуществляется проверка работы этой функции с присваиванием параметрам необходимых значений (лучше организовать ввод этих параметров с клавиатуры) и вывода результатов функции на экран.

Результат выполнения функции сравнивается с соответствующим значением, полученным в трассировочной таблице. Полное совпадение результатов при различных значениях входных параметров позволяет сделать вывод как о правильности написанной вспомогательной функции, так и понять смысл исследуемого фрагмента программы.

В качестве примера рассматривается следующая задача:

Задание

Дан фрагмента кода функций, описанный на языке программирования C++. При этом необходимо:

1) Проанализировать фрагмент, переменные и их "смысл".
2) Оформить фрагмент в виде функции (заголовок, формальные параметры, результат, локальные переменные).
3) Проверить работу функций, протестировать программы и подтвердить гипотезы.

Исходный код:

```
for (n=a, s=0; n!=0; n=n/10)
{k=n%10; s=s+k;}
```

Данный фрагмент принимает в качестве входного параметра некоторое значение переменной a. Пусть a = 13533.

Составим трассировочную таблицу:

a	n	n!=0	n=n/10	k=n%10	s	s=s+k
13533	13533	ИСТИНА	1353	3	0	3
	1353	ИСТИНА	135	3	3	6
	135	ИСТИНА	13	5	6	11
	13	ИСТИНА	1	3	11	14
	1	ИСТИНА	0	1	14	15
	0	ЛОЖЬ	0	0	15	15

```
Result: 15
```

Вывод по фрагменту функции:
Функция вычисляет сумму цифр числа.
Входные параметры функции: число a,введенное пользователем.
Функция выводит: `s` - сумма цифр числа.

Напишем, как необходимо по заданию, функцию.
Структура данных:

Имя переменной	Назначение	Тип	Вид	Примечание
a	Число, для которого нужно вычислить сумму цифр	Целое	Входная	Ввод с клавиатуры
n	Число, без последней цифры	Целое	Локальная	Сначала равна входному параметру, затем: = n/10
k	Выделение последней цифры	Целое	Локальная	= n%10
s	Сумма цифр числа	Целое	Локальная, возвращаемая	Сначала равна 0, затем: = s+k

Код функции в разработанной программе:
//функция вычисляет сумму всех цифр числа

```
int Sum (int a) //вычисление суммы цифр числа
{
     int n, s, k;
     clog << "Start Sum\n";
     clog << "a" <<"\t"<< "n" << "\t" << "k" << "\t" <<"s" << \t"<<endl;
     clog << a <<"\t";
     for (n = a, s = 0; n != 0; n = n / 10)
     {
          k = n % 10;
          clog << n << "\t";
          clog << k << "\t";
          s = s + k;
          clog << s  << "\t";
          cout << endl<< "\t";
     }
     cout << endl;
     return s;
}
```

Имя функции “Sum”. Тип возвращаемого значения – `int`. Входной аргумент – число *a* типа `int`.

Результаты работы программы:

Вывод: Результат выполнения программы сходится с результатом таблицы тестирования.

Весь ход выполнения задачи вместе с фрагментами отчета приведен в Приложении П 3.

Примечание. Печать, организуемая для проверки работы функции, называется **отладочной (контрольной) печатью.**

Рекомендация. Для большего понимания работы программы имеет смысл дополнительно вставить в программу операторы печати, соответствующие результатам проверки условий, например:

```
if(n != 0)
     cout << " (n != 0)?  - > true"<< endl
     else
     cout << " (n != 0)?  - > false"<< endl;
```

Задание 1

Даны 3 фрагмента кода функций, описанные на языке программирования C++. При этом необходимо:

1) Проанализировать фрагменты, переменные и сформулировать назначение переменных и фрагментов в целом.
2) Оформить фрагменты в виде функции (заголовок, формальные параметры, результат, локальные переменные).
3) Проверить работу функций, протестировать программы и подтвердить гипотезы.
4) Сделать соответствующие выводы.
5) Оформить отчет по выполненной работе.

Примечание. Рекомендуется для ввода и вывода информации использовать операторы форматного ввода/вывода `scanf` и `printf` соответственно.

Исследуемые фрагменты функций приведены в Приложении П 4. Так, если студент по списку имеет номер N, то для него определены следующие варианты:
a) N; b) 62-N; c) 24+N.

2.2. Ввод/вывод информации на языке С

В процессе выполнения программы на любом языке одной из важнейших задач является ввод информации из файла (с клавиатуры) с целью присвоения переменным программы конкретных значений. Так, на С++ для этой цели используются оператор `cin`, а на С - операторы `getchar, gets, fgets`, а также операторы форматного ввода `scanf` либо `fscanf`.

С целью понимания работы этих операторов студентам необходимо самостоятельно прочитать о них в соответствующей литературе по языку Си, а также выполнить следующие задачи **(Задание 2)**:

1. посимвольное чтение с клавиатуры всех символов (введенных до клавиши "Enter");
2. посимвольное чтение с клавиатуры заданного количества символов (предполагается, что введено может быть значительно большее их число);
3. чтение всех символов с клавиатуры в массив типа `char`;
4. чтение заданного количества символов с клавиатуры в массив типа `char`;
5. чтение всех символов первой строки из файла (возможно, что в файле несколько строк);
6. чтение заданного количества символов из файла:
   а) необходимое число символов находится в одной строке;
   б) необходимое число символов находится в нескольких строках;
7. чтение символов из файла: из первой строки - в первый массив типа char, из второй - во второй массив:
   а) всех символов из соответствующих строк;
   б) заданного количества символов из каждой строки;

Аналогично выполнить подобные операции чтения чисел (целых либо вещественных):

1. чтение заданного количества чисел с клавиатуры;
2. чтение заданного количества чисел из файла;
3. чтение всех чисел из первой строки файла в один массив, а из следующей строки - в другой.

Рекомендация 1. При чтении символов в режиме "до конца строки" следует проверять каждый прочитанный символ на равенство '\n' либо '\0'.

Рекомендация 2. С целью проверки правильности чтения символов организовать контрольные печати, используя операторы вывода: `puts, fputs, putchar` и `fputchar`, а также операторы форматного вывода `printf` и `fprintf`. Последние операторы использовать с выводом контрольных символов, например, "!" и "?":

```
printf("!%c? %d\n", c, c);
```

- здесь организована печать символа и его кода.

Рекомендация 3. При чтении чисел в режиме "до конца строки" с целью определения конца строки рекомендуется вслед за числом читать и символ с последующей проверкой его на равенство '\n' либо '\0'.

Рекомендация 4. Перед выполнением каждой отдельной задачи будет правильнее создать алгоритм в виде блок схемы, показав его преподавателю.

Результаты работы показать преподавателю, создав для этой цели необходимое количество программ. Отчет по данной работе заключается в написании небольшого ЭССЕ на нескольких страницах (**Задание 2**) с указанием особенностей чтения в режиме "до конца строки" и чтения заданного количества значений (символов и чисел). В качестве иллюстраций к отчету приложить фрагменты созданных программ.

Примечание: Будет проведена проверка работ на антиплагиат.

2.3. Строковые переменные. Указатели

Один из распространенных в С (С++) типов данных - массив, состоящий из элементов символьного типа. Во многих версиях этого языка данный тип обозначается как string (строковая переменная). Но допустимо, и даже более правильно, использовать явное объявление переменных этого типа, например:

```
char  mass[12];
```

К любому элемента массива mass можно обратиться, используя переменную целого типа для указания индекса (номера) этого элемента в массиве, например:

```
i = 2;
a = mass[i];
mass[i+2] = 'a';
```

В С и (С++) принято, что нумерация элементов в массиве начинается с 0. Таким образом, первый элемент массива `mass` есть `mass[0]`, а последний - `mass[11].`

Кроме того, для обозначения какого-либо элемента массива используется переменная, называемая указателем.

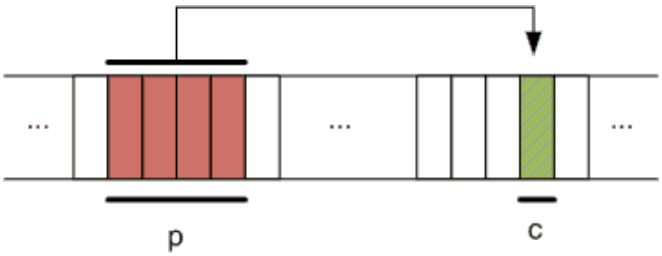
Указатель - переменная, значение которой является адресом переменной соответствующего типа. То есть указатель - переменная, содержащая адрес объекта. Указатель не несет информации о содержимом объекта, а содержит сведения о том, где размещен объект.

Тип указателя— это тип переменной, адрес которой он содержит. Для работы с указателями в Си определены две операции:

- операция * (звездочка) — позволяет получить значение объекта по его адресу — определяет значение переменной, которое содержится по адресу, содержащемуся в указателе;
- операция & (амперсанд) — позволяет определить адрес переменной.

Например,

```
char c;  // переменная
char *p; // указатель на переменную типа char
p = &c;  // p = адрес c
```



Применительно к рассматриваемому типу данных (массиву символов) указатель объявляется как:

```
char  *k; // k  есть указатель на переменную типа char
```

Тогда, если объявлен массив `mass` типа `char`, можно переменной `k` присвоить адрес первого элемента этого массива:

```
k = &mass[0];
```

Синонимом этого оператора в случае одномерного массива будет оператор:

```
k = mass; // Здесь полагается, что mass является адресом данного массива
```

Использование указателя `k` для работы со строковыми массивами понятно из следующих операторов:

```
char b;    // объявлена переменная b символьного типа

b = *k; // переменной b присвоено значение переменной, расположенной по
        // адресу k

b  =  *(mass+i) //переменной b присвоено значение элемента массива
                //mass[i], переменная i является смещением относительно
               //начала массива mass, то есть его первого элемента:
              //mass[0]. Иными словами mass[i] = &(mass+i). Здесь
             // знак "=" обозначает равно (равенство)

*(mass+i) = *k;  // значение переменной, расположенной по адресу k,
          //присваивается элементу mass[i] (записывается по адресу mass+i)
```

Передача в функцию строковых массивов как параметров с использованием указателей записывается следующим образом:

```
c = sss(k);  либо  c = sss(mass);  либо  c = sss(&mass[0]);
```

Причем функция `sss` может быть объявлена как:

```
char  sss(char *m);
```

Более подробно об указателях и их использовании смотрите в соответствующей литературе по C (C++) и материале лекций по курсу ТТП.

Задание 3

Написать функцию, определенную соответствующим Заданием (см. Приложение П 5). Прототип функции должен выглядеть, как определено в Задании.

Также написать вспомогательные функции ввода строковых массивов из файла и с клавиатуры, функции записи результатов основной функции в файл или на консоль.

В главной программе предусмотреть по запросу чтение исходных данных с клавиатуры либо из файла, также как и вывод полученных результатов на консоль либо в файл.

Все функции написать в двух вариантах: а) с использованием индексов и б) с использованием указателей.

Протестировать написанные функции с различными исходными данными и каналами ввода и вывода.

Сделать соответствующие выводы.

По результатам выполнения Задания составить отчет.

Рекомендации

- Вначале должны быть написаны функции с использованием индексов. Второй вариант является чисто формальным преобразованием выражений с индексами в выражения с указателями.
- Схемы алгоритмов функций, даже если они реализованы с использованием указателем, остаются такими же, что и с применением индексов, что делает эти алгоритмы более понятными.

2.4. О параметрах функции

При разработке функций основные проблемы возникают с передачей в нее значений и получения результатов. Эти задачи решаются с использованием параметров функции и специального их объявления.

Например, каким образом можно изменить переменную внутри функции и передать новое значение в программу? Существует два варианта:

1. В левом примере функция принимает указатель, например типа int (в зависимости от типа возвращаемой переменной). В главной программе в функцию передается адрес этой переменной.
2. В правом примере параметры (аргументы функции) объявляются как МЕСТОДЕРЖАТЕЛИ. Передача таких аргументов производится по ссылке.

```
#include <conio.h>
#include <stdio.h>

void change(int *a)
{
    *a = 100;
    printf("%d\n", *a);
}


void main()
{
    int d = 200;
    printf("%d\n", d);
    change(&d);
    printf("%d", d);
    getch();
}
```

```
#include <iostream>
using namespace std;
void minimax(int a, int b, int c, int &min,int &max)
{
  min = a;
  max = a;
  if (b < min) min = b;
  else max = b;
  if (c < min) min = c;
  if(c > max) max = c;
  return;
}
void main()
{
  int min, max;
  minimax(3, 8, 4, min, max);
  cout << "min = " << min << endl;
  cout << "max = " << max << endl;
  cin.get();
  }
```

По мнению автора, **использование указателей** (адресов переменных) является более понятным и потому - **предпочтительней**.

Существенную сложность вызывает ситуация, когда параметром функции является массив.

Если массив одномерный, имя этого массива подменяется на указатель, поэтому передача одномерного массива эквивалентна передаче указателя.

Пример: функция получает массив и его размер и выводит на печать. Ниже представлены два варианта функции:

```
void printArray(int *arr, unsigned size)
{    unsigned i;
    for (i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
}
```

```
void printArray(int arr[], unsigned size)
{    unsigned i;
    for (i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
}
```

Вызов обеих функций одинаков:

```
void main()
{
    int x[10] = {1, 2, 3, 4, 5};
    printArray(x, 10);
    getch();
}
```

Опять же, по мнению автора, использование указателя для передачи массива в функцию более понятно, и потому предпочтительно!

Как передавать двумерный массив? Существует несколько способов решения этой задачи.
Например:

```
void printArray(int arr[][5], unsigned size)
{
    unsigned i, j;
    for (i = 0; i < size; i++) {
        for (j = 0; j < 5; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

```
void main()
{
    int x[][5] = {
        { 1, 2, 3, 4, 5},
        { 6, 7, 8, 9, 10}};
    printArray(x, 2);
    getch();
}
```

Если двумерный массив создан динамически, то можно передавать указатель на указатель. Например при создании функции, которая получает массив слов и возвращает массив целых, равных длине каждого слова (левый вариант).

Можно вместо того, чтобы возвращать указатель на массив, передавать массив, который необходимо заполнить (правый вариант):

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define SIZE 10
unsigned* getLengths(const char **words,
unsigned size)
{
    unsigned *lengths = NULL;
    unsigned i;
    lengths = (unsigned*) malloc(size *
                    sizeof (unsigned));
    for (i = 0; i < size; i++) {
        lengths[i] = strlen(words[i]);
    }
    return lengths;
}
 void main() {
    char **words = NULL;
    char buffer[128];
```

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define SIZE 10
void getLengths(const char **words, un-
signed size, unsigned *out)
{
    unsigned i;
    for (i = 0; i < size; i++) {
        out[i] = strlen(words[i]);
    }
}

void main() {
    char **words = NULL;
    char buffer[128];
    unsigned i;
    unsigned *len = NULL;
    words = (char**) malloc(SIZE *
```

```
    unsigned i;
    unsigned *len = NULL;
    words = (char**) malloc(SIZE *
sizeof(char*));

    for (i = 0; i < SIZE; i++) {
        printf("%d. ", i);
        scanf("%127s", buffer);
        words[i] = (char*) malloc(128);
        strcpy(words[i], buffer);
    }

    len = getLengths(words, SIZE);
    for (i = 0; i < SIZE; i++) {
        printf("%d ", len[i]);
        free(words[i]);
    }
    free(words);
    free(len);
    getch();
}
```

```
sizeof(char*));

    for (i = 0; i < SIZE; i++) {
        printf("%d. ", i);
        scanf("%127s", buffer);
        words[i] = (char*) malloc(128);
        strcpy(words[i], buffer);
    }

    len = (unsigned*) malloc(SIZE *
sizeof(unsigned));
    getLengths(words, SIZE, len);
    for (i = 0; i < SIZE; i++) {
        printf("%d ", len[i]);
        free(words[i]);
    }
    free(words);
    free(len);
    getch();
}
```

Но можно воспользоваться, как будет показано в следующем пункте, представлением в функции двумерного массива как одномерного, с использованием в качестве параметра адреса этого массива. Данный вариант и проще и опять же понятнее!

2.5. Двумерные массивы

Двумерные массивы как типы данных в языках программирования появились из-за необходимости решать алгебраические задачи с матрицами. Поэтому синтаксис операторов, с помощью которых производятся действия с элементами массивов, близок к написаний выражений, что используются в высшей алгебре.

Так, `A[i][j]` в С (С++) соответствует a_{ij} - элементу матрицы `A`, находящемся на пересечении `i`-й строки и `j`-го столбца. Следует лишь учитывать, что в С нумерация индексов начинается с 0.

Объявление двумерных массивов понятно из следующих примеров:

```
int  A[10][5];
float B[12][3];
char   Mass[3][12];
```

Использование двумерных массивов в программах полностью аналогично работе с одномерными массивами. Также для указания места расположения элемента в матрице вводятся специальные переменные целого типа:

```
int   i, j, c;
i = 1;   j = 2;
c = A[i][j] + A[i-1][j+2];
```

Введение в С двумерных массивов позволяет различные математические операции над матрицами записывать более компактно, используя циклы. Например, вычисление суммы элементов третьего столбца матрицы А может быть организовано с помощью следующих операторов:

```
int i, s;
for (i = 0, s = 0; i < 10; i++)
     {s = s + A[i][2];
     }
```

Как и в случае с одномерным массивом, для работы с матрицами удобно использовать указатели. Только для их применения следует помнить важный момент: в оперативной памяти компьютера двумерный массив располагается построчно. То есть если элемент a_{00} находится в самом начале выделенной памяти, то a_{04} - сдвинут относительно его уже на 4 позиции (находится на пятом месте). А элемент a_{10} смещен относительно начала массива на 5 (1 * 5 + 0) позиций. В общем случае элемент a_{ij} смещен по отношению к началу выделенного под массив места на (i * 5 + j) позиций (ячеек памяти). Здесь 5 - объявленный размер массива по ширине (количество столбцов в матрице).

Таким образом, используя указатели, как и в предыдущей задаче с одномерным массивом, можно записать вычисление суммы элементов третьего столбца матрицы как:

```
int i, s, k;
int   *mas;
mas = &A[0][0];
for (i = 0, s = 0; i < 10; i++)
     {k = i * 5 + 2;
     s = s + *(mas + k);
     }
```

Следовательно, теперь в функции, обрабатывающие двумерные массивы, можно передавать в качестве параметров адрес двумерного массива и объявленные размеры, что позволяет одну и ту же функцию использовать для работы с массивами произвольной длины!

В самой функции в цикле вычисляется необходимое смещение k исходя из координат i и j элемента и заявленных размерностей dim1 и dim2:

```
int   fff (int *mas, int dim1, int dim2)
{int i, s, k;
for (i = 0, s = 0; i < dim1; i++)
     {k = i * dim2 + 2;
     s = s + *(mas + k);
     }
return s;
}
```

Задание

Написать функцию, определенную соответствующим Заданием (см. Приложение П 6). Прототип функции (в большинстве Заданий) должен выглядеть следующим образом:

```
int Func( int * Array, int Dim1, int Dim2, ...).
```

Также написать вспомогательные функции ввода двумерных массивов из файла и с клавиатуры, функции записи результатов основной функции в файл или на консоль.

В главной программе предусмотреть по запросу чтение исходных данных с клавиатуры либо из файла, также как и вывод полученных результатов на консоль либо в файл.

Все функции написать в двух вариантах: а) с использованием индексов и б) с использованием указателей.

Протестировать написанные функции с различными исходными данными и каналами ввода и вывода.

Сделать соответствующие выводы.

По результатам выполнения Задания составить отчет.

Рекомендации

- Вначале должны быть написаны функции с использованием индексов. Второй вариант является чисто формальным преобразованием выражений с индексами в выражения с указателями.
- При неполном заполнении массива, когда реальные размеры этого массива меньше объявленных, величину смещения относительно начала памяти, выделенного под массив, следует вычислять с учетом объявленных координат массива!!!!
- Реальные и объявленные размеры массива передаются в функцию через параметры, например, `dim1 и dim2.`
- Схемы алгоритмов функций, даже если они реализованы с использованием указателем, остаются такими же, что и с применением индексов, что делает эти алгоритмы более понятными.

2.6. Преобразование двумерного массива

Задание 5

Составить функцию преобразования квадратной матрицы A размерности M по правилу, определенному Заданием (см. Приложение П 7). Вспомогательными массивами, рекурсиями и другими средствами пользоваться лишь с разрешения преподавателя. Функцию написать с использованием указателей.

Полученную функцию использовать трижды:

1) для преобразования квадратной матрицы B, определенной Заданием;
2) для преобразования верхнего левого угла матрицы B размерности в два раза меньшей, чем заданная размерность матрицы B.
3) для преобразования "маленькой" матрицы C, размерность которой совпадает с размерностью угла матрицы B (при втором использовании функции).

Все матрицы перед каждым преобразованием формируются (заполняются) по одному и тому же правилу. Матрицы печатать полностью, как объявлены, до и после каждого преобразования. Функции формирования матрицы и ее печати также пишутся с использованием указателей.

По выполнению работы представить отчет.

Пояснение к выполнению Задания 5.

Итак, требуется преобразовать квадратную матрицу A размерности m по правилу, написанному в задании. Под преобразованием понимается замена всех элементов исходной матрицы. В задании определено правило преобразования элемента матрицы, расположенного в i-й строке и j-м столбце. Преобразовать требуется все элементы матрицы! Вспомогательными массивами и рекурсиями не пользоваться.

Главная сложность преобразования заключается в определении такого порядка (очередности) обновления, при котором ранее измененные элементы матрицы не учитывались бы при обновлении следующих элементов. В некоторых задачах преобразование выполняется построчно (сверху вниз либо снизу вверх), внутри строки - справа налево либо слева направо. Такой порядок при реализации алгоритма обычно сложностей не вызывает. В ряде задач приходится выполнять преобразование подиагонально, переходя по этим диагоналям из одного угла матрицы в другой. В этом случае можно порекомендовать следующую последовательность действий:

1. Диагонали пронумеровать.
2. Затем в цикле от первой до последней диагонали:
   - сначала определить координаты верхнего и нижнего элементов конкретной (текущей) диагонали.
   - далее, двигаясь от одного крайнего элемента диагонали до другого, выполняется поочередное преобразование всех элементов данной диагонали.

Для лучшего понимания (и удобства) в схеме алгоритма операторы (действия) описывать с использованием индексов! Разработанный алгоритм закодировать как функцию. При написании функции учесть, что реальная матрица, подвергающаяся преобразованию, может иметь меньший размер, чем объявлена (например, при 2-м обращении к функции).

Параметрами функции являются: адрес (указатель) массива, объявленная и реальная его размерности:

```
void Func( int * Array, int Dim1, int Dim2)
```

Так как в соответствии с заданием преобразуется квадратная матрица (число строк равно количеству столбцов), то здесь Dim1 - объявленная размерность этой матрицы, а Dim2 - реальная.

Вспомогательные функции формирования матрицы и ее печати (вывод на консоль либо в файл - по согласованию с преподавателем) пишутся для работы со всей объявленной матрицей соответствующей размерности, следовательно имеют только два параметра:

```
void FuncP( int * Array, int Dim)
```

2.7. Решение итерационных задач с помощью циклов

В практике программирования встречаются задачи вычисления значений различных сумм, например, отрезков степенных рядов. С основными приемами решения таких задач познакомимся на следующем примере.

Для значений x, изменяющихся в заданном диапазоне с шагом h, построить таблицу значений функции $y = f(x)$:

$$y = \log x = \frac{1}{\ln 10} \sum_{n=1}^{\infty} \frac{(x-1)^n}{2 \cdot x^n}, \qquad x \in [x_0, x_k].$$

Суммирование выполнять до тех пор, пока соблюдается условие $\frac{a_n(x)}{S_n(x)} > eps$, формула действительна для $x \geq 0.5$.

Здесь $a_n(x)$ - общий член ряда;

$S_n(x)$ - частная сумма, отвечающая суммированию членов ряда до $a_n(x)$ включительно $S_n(x) = a_o(x) + a_1(x) + \ldots + a_n(x)$,

eps наперед заданное малое число - погрешность вычисления.

Определить, при каком значении n достигается заданная точность.

Здесь сформулирована задача с двойным циклом: внешним - по количеству значений аргумента функции $y(x)$ (цикл по x с известным числом повторений) и внутренним по переменной n (цикл по n с неизвестным числом повторений).

С внешним циклом все понятно: количество точек по переменной x, то есть число строк в таблице, определим по формуле $i\max = \frac{x_k - x_0}{h} + 1$.

Замечание. Число повторений внешнего цикла заранее определять необязательно. Цикл может быть явно организован по вещественной переменной x от x_0 до x_k с шагом h.

Внутренний цикл.

Обычно вместо внутреннего цикла организуют вызов функции, в которой и вычисляют сумму ряда.

Построим математическую модель вычисления частичной суммы $S_n(x)$. С целью создания ясного, надежного и эффективного алгоритма выразим следующий член ряда $a_{n+1}(x)$ через очередной член $a_n(x)$. Для этого найдем их отношение: $f(n,x) = \frac{a_{n+1}(x)}{a_n(x)} \Rightarrow f(n,x) = \frac{(x-1) \cdot n}{x \cdot (n+1)}$.

Отношение $a_{n+1}(x) = f(n,x) \cdot a_n(x)$ называется рекуррентным, для его реализации необходимо предварительно вычислить самый первый член ряда, в данном случае это $a_1(x) = \frac{x-1}{x}$ (при $n = 1$).

Так как частная сумма $S_n(x)$ может быть нулевой, условие окончания суммирования удобнее записать "в линию": $a_n(x) \le eps \cdot S_n(x)$, что только усилит надежность алгоритма.

При составлении алгоритма необходимо:

1. Определить объекты (переменные), с которыми будем работать.
2. Классифицировать их на входные (исходные), локальные (текущие, промежуточные) и выходные (искомые).
3. Исходным переменным присвоить начальные значения.
4. Указать порядок действий, ведущий к искомому результату.

Последовательность действий решения основной задачи

Структура входных данных:

`x0, xK` — диапазон изменения x (вещественные), x>0

`h` — шаг изменения x

`n_max` — ограничение числа итераций (целое)

`eps` — погрешность вычисления функции

Исходные данные:

`x0, xK , h, eps, n max` — вводятся с клавиатуры в интерактивном режиме.

Алгоритм главной программы

1. Начало
2. Запрос на ввод `x0, xK, h, eps, n_max`
3. Ввод `x0, xK, h, eps, n_max`
4. Вывод заголовка таблицы
5. Шаг по x. Если `i_max<=1  h = 0` иначе `h = (xk-x0)/(i_max-1)`
6. Цикл для расчета результатов по переменной `x` от `x0` до `xk` с шагом `h`
   - 6.1. `y = f(x,eps,n_max,n)`
   - 6.2. `d = y - log(x)`
   - 6.3. вывод `x,n,y,d`
7. конец

Проверка алгоритма

Выполняется при конкретных значениях входных переменных с использованием трассировочной таблицы.

Проектирование функции вычисления суммы ряда:

Исходные данные для функции (входные параметры):

аргумент - `x` вещественное;

погрешность - `eps` - вещественное;

ограничение на количество членов ряда - `n_max` - целое.

Результаты работы функции (выходные параметры, значения):

значение `y` - вещественное, возвращаемая переменная;

число членов ряда: `n` - целое, выходной параметр.

Алгоритм расчета логарифма:
1. начало
2. `n = 1` — инициализация счетчика
3. `a = (x-1)/x` — первый член ряда
4. `s = 0` — инициализация суммы ряда
5. начало цикла
    5.1. `f = n*(x+1)/((n+1)*x)`
    5.2. `a = a*f`
    5.3. `s = s+a`
    5.4. `delta = abs(a) /abs(s)`
    5.5. `n = n+1`
    5.5. если `n<n_max` и `delta> eps` повторить цикл 5
6. вернуть `s / log(10)`

Замечание. В рассмотренном примере удобнее по отдельности вычислять числитель и знаменатель очередного члена с последующим делением их друг на друга для получения значения a. Но в этом случае возможно переполнение разрядной сетки при вычислении этих величин. С целью устранения данного недостатка рекомендуется в цикле контролировать значения, например, знаменателя. И как только его значение превысит какое-либо **большое число**, поделить и числитель и знаменатель на это **большое число**.

Проверка алгоритма

Для удобства проверки алгоритма составляется трассировочная таблица в Excel

x	n	a	f=((x-1)*n)/ ((n+1)*x)	a*f	s	|a| / s	delta < e
0,9	1	-0,1111111	-0,05556	0,0061724	-0,1111111	1	ЛОЖЬ
0,9	2	0,00617284	-0,07407	-0,000455	-0,1049382	0,006630087	ЛОЖЬ
0,9	3	-0,0004572	-0,08333	3,810E-05	-0,1053955	0,000495351	ЛОЖЬ
0,9	4	3,8103E-05	-0,08889	-3,38E-06	-0,1053574	4,1491E-05	ЛОЖЬ
0,9	5	-3,387E-06	-0,09259	3,136E-07	-0,105360	3,70063E-06	ЛОЖЬ
0,9	6	3,1361E-07	-0,09524	-2,98E-08	-0,1053604	3,43481E-07	ЛОЖЬ
0,9	7	-2,986E-08	-0,09722	2,902E-09	-0,1053605	3,27717E-08	ИСТИНА

Результаты трассировочной таблицы подтверждают работоспособность и правильность алгоритма.

На основании разработанных алгоритмов записываются исходные тексты внутренней функции вычисления суммы ряда и главной программы (здесь не приводятся в силу их очевидности).

Написанная программа тестируется при различных значениях входных параметров. На этапе проверки ее работоспособности рекомендуется вставить в функцию необходимую отладочную печать.

Вывод

Программа работает правильно. Результаты тестирования совпадают с плановыми результатами. При x<0.5 ряд становится расходящимся, а цикл – бесконечным. Необходимо защищать подобные алгоритмы от зацикливания, ограничивая цикл максимальным числом повторений.

Задание 6

Получить таблицу значений функции *y(x)* на произвольно выбранном отрезке с самостоятельно заданным шагом *h*. Выражение для вычисления функции взять из Приложения П 8. Сумму ряда вычислять с точностью до относительной погрешности *eps*.

Функцию вычисления суммы ряда написать в двух вариантах:

- с использованием цикла `while`;
- с использованием цикла `do while`

Проанализировать количество повторений цикла функции *y(x)* при конкретном значении аргумента *x* в зависимости от заданного *eps*. Отдельно показать, что при выбранном слишком малом значении погрешности происходит зацикливание функции, что оправдывает введение переменной *n_max* - максимального числа повторений операторов цикла.

Проверить работу функции *y(x)* для значений аргумента *x* вне допустимого диапазона.

Сделать соответствующие выводы.

Результаты работы отразить в отчете.

2.8. Решение итерационных задач с помощью рекурсий

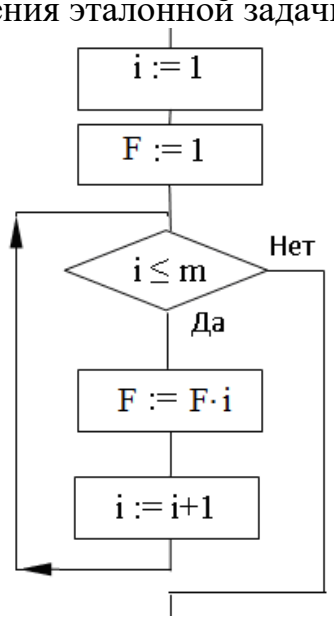
В программировании под рекурсией понимают такую реализацию, в которой функция использует в своем теле вызов самой себя. Такие вызовы называют рекурсивными.

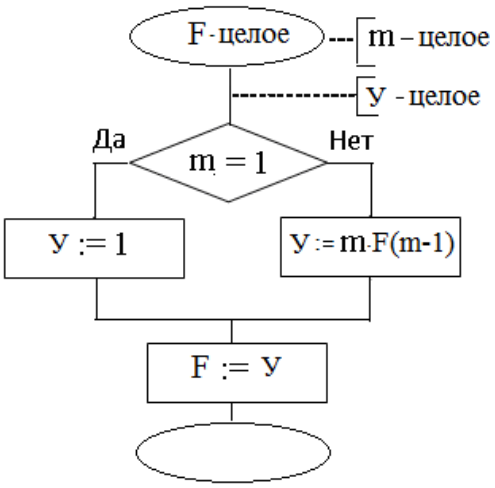
Рекурсивные алгоритмы сложно отлаживать, но порой они позволяют очень гибко и красиво решить задачу. Правда, во многих случаях за внешней красотой может скрываться более длительный вычислительный процесс.

При реализации рекурсивных алгоритмов требуется уверенность, что алгоритм является конечным, то есть выполнение рекурсивной функции должно когда-нибудь завершиться. Конечность рекурсивного алгоритма обеспечивается с помощью специального (терминального, граничного) условия, реализуемого оператором ЕСЛИ. Естественно, помимо конечности алгоритм должен быть правильным.

Любой рекурсивный алгоритм можно заменить нерекурсивным (циклическим), несмотря на то, что на его реализацию может потребоваться большее время. Во многих случаях проектирование циклического алгоритма позволяет проще и быстрее решить поставленную задачу.

Ниже приведены Циклический (слева) и Рекурсивный (справа) алгоритмы решения эталонной задачи - вычисления функционала: m!





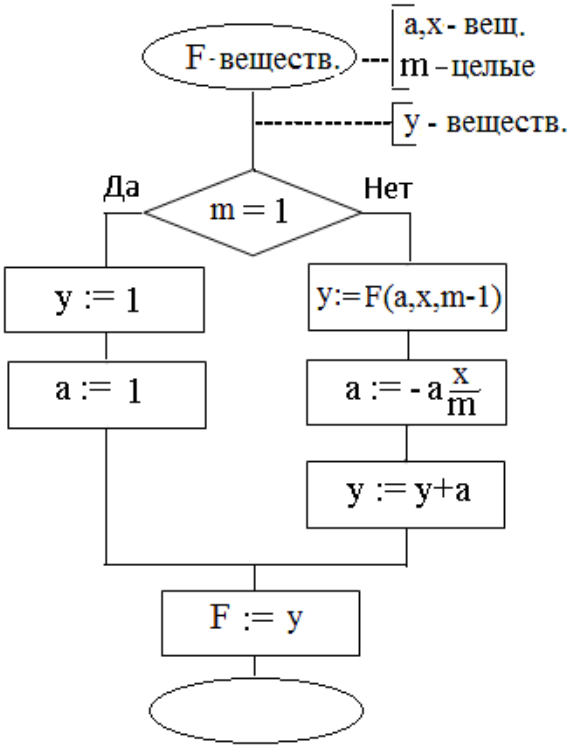
В главной программе вызов рекурсивной функции будет иметь вид: `Y = F(m).`

Рекурсия может быть использована и для других циклических задач, например, для вычисления суммы ряда. Ниже приведены два рекурсивных алгоритма решения задачи:

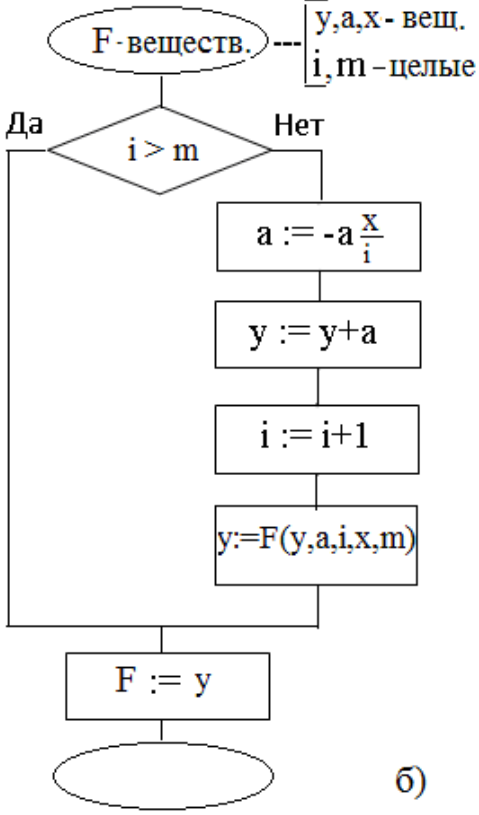
$$y(x) = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots + \frac{x^m}{m!}.$$

Оба алгоритма существенно отличаются друг от друга. Функции, построенные по ним, имеют разный набор параметров. Различными будут и вызовы

этих функций. В главной программе вызвать левую функцию можно так: `Y = F(a, x, m)`.
Правая же функция должна вызываться при следующих значениях параметров: `Y = F(1., 1., 1, x, m)` (желательно определить соответствующие переменные для передачи необходимых значений в эту функцию).



а)



б)

Алгоритм б) при несущественной доработке может быть использован и для вычисления суммы ряда с заданной точностью. Для этого необходимо изменить всего лишь условие в операторе ЕСЛИ. Дополнительно увеличивается число параметров функции, - к уже имеющимся добавляется требуемая точность.

Следует заметить, что левый алгоритм (Рис. а)) называется НИСХОДЯЩЕЙ рекурсией, а правый (Рис. б)) - ВОСХОДЯЩЕЙ.

В нисходящей рекурсии основные вычисления (вычисление очередного члена ряда и добавление его к переменной, в которой накапливается сумма ряда) выполняются после рекурсивного обращения к функции F. Кроме того, один из параметров функции, а именно a, является выходным параметром. Функция F, кроме уточняемой (накапливаемой) суммы ряда, возвращает и значение предыдущего члена ряда.

При восходящей рекурсии вычислительный процесс начинается сначала, как в циклическом алгоритме. Номер шага постоянно возрастает: `i:= i+1`. В качестве терминальной (граничной) ситуации выбирается момент достижения условий окончания процесса `i > m`. Начальные условия задаются в качестве значений аргументов при вызове рекурсивной функции. В восходящей рекурсии параметры, характеризующие состояние процесса, вычисляются на каждой

стадии рекурсии в процессе выполнения прямого хода. Результат строится постепенно и окончательное значение приобретается в момент достижения терминальной ситуации. Таким образом, список аргументов функции, проектируемой по схеме восходящей рекурсии, содержит две группы параметров: одна - это исходные данные и результат, вторая - промежуточные значения переменных процесса.

Достоинством восходящей рекурсии является экономия памяти - стека, куда помещаются при рекурсивном вызове функции значения переменных и параметров, а также еще невыполненные операторы, что расположены после обращения к функции. Как следствие - восходящая рекурсия более эффективна по быстродействию. Поэтому во всех случаях, когда это возможно, рекомендуется переходить от нисходящей рекурсии к восходящей.

Задание 7

Вычислить несколько значений функции *y(x)* при произвольно заданных значениях аргумента с помощью рекурсий: восходящей и нисходящей. Выражение для вычисления функции взять из Приложения П 8. Показать, в каком случае возможно вычисление суммы ряда с заданной точностью, а в каком - это невозможно.

С помощью контрольных печатей (трассировочной таблицы) проанализировать изменение все переменных (параметров) функции как на входе, так и на выходе из нее.

Получить таблицы значений *y(x)* для обоих вариантов функций на произвольно выбранном отрезке с самостоятельно заданным шагом *h*.

Показать, что при выбранном слишком малом значении погрешности *eps* при очень большом значении *n_max* (максимальном числе повторений операторов цикла) происходит аварийное завершение работы одной из функций. Объ-яснить причину этого прерывания.

Сделать соответствующие выводы.

Результаты работы отразить в отчете.

2.9. Структуры

Структура - это совокупность переменных, объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации. Объявление структуры приводит к образованию шаблона, используемого для создания объектов структуры. Переменные, образующие структуру, называются членами структуры. В качестве членов структуры могут выступать переменные, массивы, указатели, а также другие структуры. Члены структуры также называются элементами или полями.

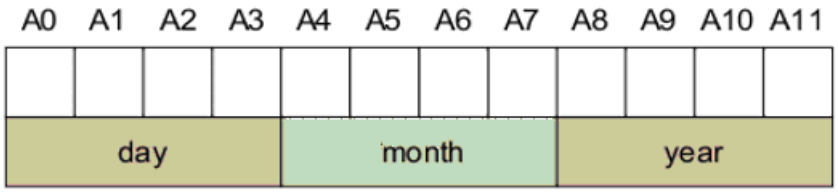
Обычно все члены структуры связаны друг с другом. Например, информация о дне, месяце и годе удобно представить в виде структуры. Следующий фрагмент кода объявляет шаблон структуры, определяющий дату. Ключевое слово struct сообщает компилятору об объявлении структуры.

```
struct date
{
  int day;          // 4 байта
  int month;        // 4 байта
  int year;         // 4 байта
};
```

Объявление завершается точкой с запятой, поскольку объявление структуры - это оператор. Для объявления переменной, соответствующей данной структуре, следует написать:

```
struct date date_info;
```

В данной строке происходит объявление переменной `date_info` типа `struct date`. Когда объявлена структурная переменная, компилятор автоматически выделяет необходимый участок памяти для размещения всех ее членов. Рис. показывает размещение `date_info` в памяти. Поля структуры располагаются в памяти в том порядке, в котором они объявлены:



В указанном примере структура `struct date` занимает в памяти 12 байт.

Для упрощения объявления переменных типа структура используется слово `typedef`. С помощью этого слова создается синоним объявленной структуры:

```
typedef struct date  sdate;
sdate a, *p, b[5];
```

То есть выполняется замена `struct date` синонимом `sdate`, что значительно облегчает понимание кода.

В предыдущем примере объявлена переменная a типа структура `sdate`, `p` - указатель на структуру, а также массив `b`, состоящий из пяти структур.

Большинство программистов используют `typedef` сразу при объявлении структуры, что также уменьшает код:

```
typedef struct date
{
  int day;          // 4 байта
  char *month;      // 4 байта
  int year;         // 4 байта
}sdate;
```

Инициализация полей структуры может осуществляться двумя способами:

- присвоение значений элементам структуры в процессе объявления переменной, относящейся к типу структуры;
- присвоение начальных значений элементам структуры с использованием функций ввода-вывода (например, `scanf()`), либо с помощью операторов присвоения.

В первом способе инициализация осуществляется по следующей форме:

```
struct date bd={8, 03, 1978};
```

Второй способ инициализации объектов языка понятен из следующих примеров:

```
scanf("%d %d %d",bd.day, bd.month, bd.year);
bd.day = 8;   bd.month = 03;  bd.year = 1978;
```

Как следует из последнего примера, имя элемента структуры является составным. Для обращения к элементу структуры указывается имя структуры и имя самого элемента. Они разделяются точкой.

Структура может быть передана в функцию в качестве параметра:

```
void   fufu (sdate  bb, int k),   либо
void   fufu (struct date  bb, int k)
```

Задание 8

Выполнить задание из Приложения П 9, используя массив структур в качестве хранилища информации. Файлы с необходимой информацией создать самостоятельно.

По результатам выполнения задания написать отчет.

Рекомендация. Предлагается следующая последовательность действий, решающих поставленную задачу:

1. Объявите тип данных СТРУКТУРА с полями, соответствующими набору информации из Вашего задания.
2. Объявите массив элементов типа СТРУКТУРА.

3. Научитесь заполнять элемент типа СТРУКТУРА информацией.
4. Научитесь "смотреть" содержимое конкретного элемента типа СТРУКТУРА. Необходимый элемент определяется соответствующим номером массива.
5. Напишите функцию чтения информации из файла и присвоения значений полям элемента. Адрес файла и элемент массива (Структура) - параметры функции.
6. Напишите функцию чтения и печати информации из полей СТРУКТУРЫ. Элемент массива (Структура) - параметр функции.
7. **ВАЖНО!** Для целей сортировки массива с информацией написать функцию сравнения двух элементов массива (двух структур целиком), где будет организовано поочередное (последовательное) сравнение значений полей этих структур. Параметры функции - два элемента массива (две Структуры), либо соответствующие поля двух СТРУКТУР (определяется конкретным заданием).
8. Некоторым студентам необходимо написать функцию, определяющую место символа в алфавите Кириллица. Параметр функции - символ. Данная функция может быть использована в функции сравнения (см. п.7 Рекомендации).

2.10. Списки

Списки являются второй по частоте использования структурой данных после массивов. Они являются достаточно простой реализацией динамических структур данных, использующие указатели (pointers).

Понимание работы указателей является необходимым условием, чтобы понять связанные списки. Для понимания списков кроме того требуется также понимание структур и динамического выделения памяти. Обо всем этом было рассказано в предыдущих разделах.

Связный (связанный) список — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и один (или два) указателя («связки») на следующий (и предыдущий) узел списка. Принципиальным преимуществом списка перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

Связные списки имеют несколько основных преимуществ:

- элементы могут быть добавлены или удалены из любого места списка;
- нет необходимости объявления размера списка при его инициализации.

Но у списков имеются и недостатки:

- связанные списки не имеют возможности свободного доступа к любому элементу - то есть нет возможности получить элемент, находящийся внутри списка, без того что бы пройтись по всем элементам до него;
- для работы списков требуется динамическое выделение памяти и указатели, что усложняет код и может привести к утечкам памяти (потери отдельных ее участков);
- связанные списки требуют больше ресурсов операционной системы, так как их элементы выделяются динамически и каждый элемент должен хранить дополнительный указатель.

Списки бывают односвязные и двусвязные.

Каждый элемент (узел) односвязного (однонаправленного) списка содержит одно поле указателя на следующий узел:

Узел двусвязного (двунаправленного) списка содержит два указателя на следующий и предыдущий узлы:

У крайних элементов списка один из указателей имеет нулевое значение (указывает на NULL). У единственного элемента двусвязного списка оба указателя - нулевые.

Ниже показаны основные операции (действия), выполняемые в односвязном списке:

- Декларация списка
- Инициализация списка
- Добавление элемента в список
- Удаление элемента из списка
- Вывод информации элемента списка
- Перестановка двух элементов списка

Декларация списка заключается в объявлении структуры:

```
typedef struct list
{
 int field;            // поле данных
 struct list *ptr;     // указатель на следующий элемент
}slist;                // объявили псевдоним структуры
```

Инициализация списка заключается прежде всего в определении указателя начала списка:

```
slist * head;    // объявление начала списка
head = NULL;     //пока список пустой
```

Затем создается элемент списка:

```
slist * lst;          //  объявляется указатель на элемент списка
lst = (slist *)malloc(sizeof(slist));  //  выделяется место под
                                       //  элемент списка
lst -> ptr = NULL;    // предполагается, что это последний элемент списка
lst -> field = 12;    // элемент списка заполняется информацией
head = lst;           // первый элемент вставляется в список
```

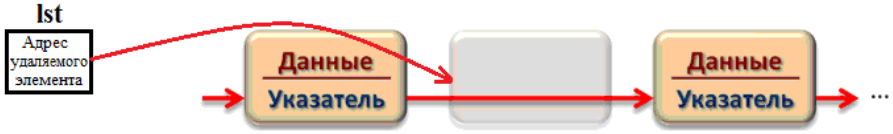
Добавление элемента в односвязный список включает в себя следующие действия:

- создание добавляемого узла и заполнение его поля данных;
- переустановка указателя узла, предшествующего добавляемому, на добавляемый узел;
- установка указателя добавляемого узла на следующий узел (тот, на который указывал предшествующий узел).

Операция добавления элемента в список понятна из следующего рисунка:


```
slist *temp, *p;          // объявление указателей на элемент списка
temp = (slist *)malloc(sizeof(list));// выделение места под новый
                                     // элемент списка
temp -> field = 144;  // заполнение информацией добавляемого элемента
p = lst -> ptr;           // сохранение указателя на следующий элемент
lst -> ptr = temp;      // предыдущий элемент указывает на добавляемый
temp -> ptr = p;        // добавляемый элемент указывает на следующий
```

Удаление элемента из списка может быть представлено схемой:



Удаление включает в себя следующие этапы:

- установка указателя предыдущего узла на узел, следующий за удаляемым;
- освобождение памяти удаляемого узла.

Операторы, реализующие удаление элемента списка, адрес которого содержится в переменной lst:

```
slist *temp;              // объявляем вспомогательный указатель
temp = head;              //  ставим указатель на начальный элемент списка
while (temp -> ptr != lst) // просматриваем список начиная с корня, пока
{                                  // не найдем узел, предшествующий lst
  temp = temp -> ptr;        // сдвигаем указатель на следующий элемент
 }
 temp -> ptr = lst -> ptr; // переставляем указатель найденного элемента
 free(lst);                       // освобождаем память удаляемого узла
```

Вывод содержимого списка выполняется с помощью операторов:

```
temp = head;              // Временно указываем на адрес первого элемента
 while (temp != NULL)   // Пока не встретим пустое значение (конец списка)
 {
   cout << temp -> field << " "; // Выводим значение очередного
```

```
                                        // элемента на экран
    temp = temp -> Next;   // Смена адреса на адрес следующего элемента
   }
   cout << "\n";
```

Объявление и инициализация двусвязного списка выполняется операторами:

```
typedef struct Node       //Структура, являющаяся элементом списка
{
 int x;                   //Значение x будет храниться в элементе списка
  struct Node *Next, *Prev; //Указатели на адреса следующего и
                            // предыдущего элементов
} sNode;                  //Объявление псевдонима (синонима) структуры

sNode *head, *tail;      //Декларация указателей на начало и конец списка
head = NULL;   tail = NULL;  // Инициализация указателей head и tail
                             // как пустые
```

Все основные действия с двусвязным списком предлагается изучить самостоятельно в процессе выполнения Задания.

Задание 9

Выполнить задание из Приложения П 9, используя двусвязные списки в качестве хранилища информации. Файлы с необходимой информацией создать самостоятельно.

По результатам выполнения задания написать отчет.

Рекомендация 1. Для целей сортировки списка с информацией написать функцию сравнения двух элементов списка (двух структур), где будет организовано поочередное последовательное сравнение значений полей структур.

Рекомендация 2.

Пусть AAA - тип структуры (элемента списка),

AAA *C, *СПИСОК, *Д - указатели на элемент списка

Предварительно создать (разработать, написать) следующие вспомогательные функции:

1) Создание элемента:

   AAA ***Созд**();

2) Заполнение элемента С информацией:

   Заполн(AAA *C); или **Заполн**(AAA *C, информация);

3) Добавление элемента С к списку СПИСОК:

   AAA ***Доб**(AAA *СПИСОК, AAA *C);

4) Просмотр (контрольная печать) содержимого списка (от начала до конца и в обратном порядке - от конца до начала списка):
ПечатьК(ААА *Список);

5) Печать содержимого списка (от начала до конца)
Печать(ААА *СПИСОК);

6) Удаление из Списка элемента Д:
ААА ***Удал**(ААА *СПИСОК, ААА *Д);

7) Вставка элемента Д в Список после С:
ВставкаП(ААА *Д, ААА *С);

8) Вставка Элемента Д в Список перед С:
ААА ***ВставкаД**(ААА *Д, ААА *С);

Далее, используя уже написанные функции, создать еще две:

9) Перенос информации из файла в список (с одновременным его созданием):
ААА ***ЧтениеФ**("Имя файла");

10) Перестановка местами двух элементов списка:
ААА ***Перест**(ААА *СПИСОК, ААА *С, ААА *Д).

После написания (создания) всех этих вспомогательных функций можно переходить к решению задачи Задания.

Примечание 1. **Недопустим** перенос информации из одного элемента списка в другой! Изменяются только значения указателей!!!!

Примечание 2. Вспомогательные функции, предложенные к созданию, сформулированы исходя из действий (задач), требующих выполнения в процессе работы со списками. Кроме того, при их формулировке (определении списка параметров и типа функций) учтен важный принцип создания любой функции, а именно **определены понятия**:

- **ЧТО ДАНО?**
- **ЧТО ТРЕБУЕТСЯ ПОЛУЧИТЬ?**
- **КАКОВ БУДЕТ РЕЗУЛЬТАТ?**

(попробуйте самостоятельно ответить на эти вопросы для каждой из функций)

Студенту остается только решить задачу: **КАК ПОЛУЧИТЬ РЕЗУЛЬТАТ**?

2.11. Деревья

Наряду со списками в программировании используются и другие динамические структуры, например, деревья. Фактически, связные списки являются подвидом деревьев, называемым вырожденным деревом. Дерево, как тип структуры данных, обычно строится на использовании рекурсии.

В общем смысле, дерево - это иерархическая структура, хранящая коллекцию объектов. Самым близким примером является файловая система, представляющая собой иерархическую структуру из файлов и каталогов. Дерево - это непустой набор вершин (узлов) и ребер, удовлетворяющих определенным требованиям. Вершина - это объект, называемый также узлом по аналогии со списками, который может иметь имя и содержать другую информацию. Ключевое свойство дерева - между любыми двумя узлами существует только один путь, соединяющий их. Ребро - это путь, соединяющий два соседних узла.

Для определения узлов существуют следующие сложившиеся термины:

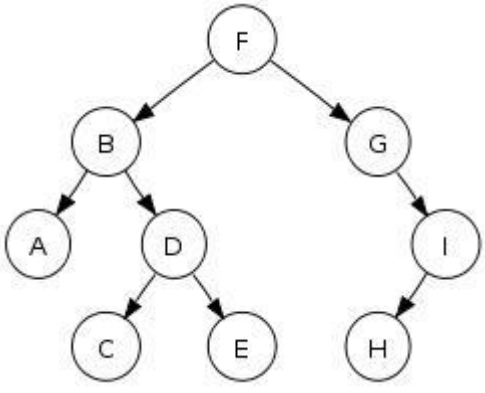
- **root** - узел, расположенный в корне дерева (корневой узел);
- **siblings** - узлы, имеющие одного и того же родителя;
- **ancestor** или **parent** – родительский узел;
- **descendant** – дочерний узел;
- **size** – размер узла, равный количеству дочерних узлов плюс один.
- **path** - путь, или последовательность узлов, которые нужно пройти, чтобы из корневого узла попасть в какой-либо дочерний узел. Длина пути равна числу узлов минус один;
- **height** - высота дерева, это путь от корня до наиболее удаленного узла.

Деревья можно разделить на следующие категории:

- деревья с корнем (в дереве только один корень);
- деревья без корня (любая вершина (узел) дерева может служить корнем);
- упорядоченные деревья;
- м-арные и бинарные деревья.

К м-арному относится дерево, соответствующее оглавлению книги. В корневом узле имеются поддеревья, соответствующие количества главам, каждый дочерний узел имеет свое количество поддеревьев, определяемых числом параграфов данной главы.

В бинарном дереве каждый родительский узел имеет не более двух дочерних узлов. Пример бинарного дерева приведен на следующем рисунке:



Как следует из рисунка, информация слева от узла меньше, чем в самом узле, а справа от узла превосходит информацию узла.

Возможны различные обходы уже существующего дерева:

- **preorder** – **сверху-вниз-слева-направо**: F, B, A, D, C, E, G, I, H;
- **inorder** – **слева-вверх-вниз-вправо**: A, B, C, D, E, F, G, H, I;
- **postorder** – **обратное движение от детей к родителям**: A, C, E, D, B, H, I, G, F.

Обход **inorder** позволяет получить из дерева упорядоченную (по возрастанию) информацию, что удобно и освобождает пользователя от ее сортировки.

Несмотря на указанное преимущество, в древовидных структурах, по сравнению со связными списками, сложнее выполняются операции вставка и удаление узлов. Например, при удалении узла из бинарного дерева необходимо решить проблему наличия двух дочерних и одного родительского узла.

Каждая вершина (узел) дерева представляет собой структуру, полями которой являются структура с информацией, а также указатели на дочерние узлы. Так, для бинарного дерева его декларация имеет вид:

```
typedef struct tree
{
    int data;
    struct tree * left, * right;
}stree;
```

Ниже приведены псевдокоды трех указанных выше способов обхода бинарного дерева, основанные на использовании рекурсии:

```
preorder(node)
    print node.data
    if node. ≠ null then preorder(node.left)
    if node. ≠ null then preorder(node.right)

inorder(node)
    if node.left  ≠ null then inorder(node.left)
    print node.data
    if node.right ≠ null then inorder(node.right)

postorder(node)
    if node.left  ≠ null then postorder(node.left)
    if node.right ≠ null then postorder(node.right)
    print node.data
```

Написать соответствующие функции по указанным псевдокодам не представляет проблемы.

Вставка узлов в дерево может быть выполнена с помощью функции (смотри ниже):

```
stree* insert(stree *root, stree *node) // root - корень дерева,
                                        //node - вставляемый элемент
{
 if (root == NULL)            // Если дерево пустое, то формируем корень
 { root = node  }
  else                        // иначе
  if (root->data > node ->field)//Если элемент node меньше корневого,
                                // уходим влево
    root ->left = insert(root->left, node); // Рекурсивно добавляем
                                            // элемент
   else                                     // иначе уходим вправо
   root ->right = insert(root->right, node);//Рекурсивно добавляем
                                            // элемент
  return(root);
}
```

С помощью следующей процедуры можно удалить дерево и освободить память:

```
void freemem(stree *root)
{
 if (root != NULL)        // если дерево не пустое
 {
  freemem(root->left);    // рекурсивно удаляем левую ветвь
  freemem(root->right);   // рекурсивно удаляем правую ветвь
  delete root;            // удаляем корень
 }
}
```

Задание 10

Выполнить задание из Приложения П 9, используя бинарное дерево в качестве хранилища информации. Файлы с необходимой информацией создать самостоятельно.

Функцию Вставки информации в дерево сопроводить контрольными печатями для обеспечения "видимости хождения по дереву".

По результатам выполнения задания написать отчет.

Рекомендация 1. В некоторых вариантах для более эффективного решения задачи удобно создать вспомогательный список. Но это в незначительной степени усложняет все задание.

Рекомендация 2. Предлагается следующая последовательность действий, решающих поставленную задачу:

1. Объявите тип данных СТРУКТУРА с полями, соответствующими набору информации из Вашего задания.

2. Научитесь выделять место под элемент типа СТРУКТУРА и заполнять его информацией.
3. Научитесь "смотреть" содержимое конкретного элемента типа СТРУКТУРА. Необходимый элемент определяется соответствующим указателем на него.
4. Напишите функцию для чтения информации из файла и присвоения ее полям элемента. Адрес файла и адрес элемента - параметры функции.
5. Напишите функцию чтения и печати информации из полей СТРУКТУРЫ. Адрес элемента - параметр функции.
6. **Важно!** Для сравнения информации двух узлов дерева написать функцию сравнения информации двух структур, где будет организовано поочередное последовательное сравнение значений полей структур. Параметры функции - адреса элементов СТРУКТУР либо соответствующие поля двух СТРУКТУР (определяется конкретным заданием).
7. Некоторым студентам необходимо написать функцию, определяющую место символа в алфавите Кириллица. Параметр функции - символ. Данная функция может быть использована в функции сравнения (см. п.6 Рекомендаций 2).
8. Напишите функцию добавления элемента к списку (если такое необходимо). Параметры функции - адреса списка и вставляемого элемента.
9. Напишите функцию чтения списка (для проверки правильности работы функции п. 8). Параметр функции - адрес списка. В данной функции использовать функцию п. 5.
10. Напишите функцию выбора соответствующего элемента из списка (если такое необходимо по заданию). Параметр функции - адрес списка и маска (критерий) выбора. При нахождении элемента, возможно, необходимо организовать его "выем", удаление из списка. Для этой цели можно написать также соответствующую функцию.
11. Напишите две заключительные функции:

- Добавления элемента в дерево.
- Чтение содержимого дерева.

Обе последние функции сопроводить соответствующими контрольными печатями для обеспечения "видимости хождения по дереву" с выводом адресов родительского и левого и правого элементов с необходимой информацией, например: "Подошел к дереву", "Сравниваюсь с элементом", "Пошел направо (налево)", "Элемент добавлен к дереву", "Возвращаюсь", "Прохожу мимо элемента..." и тому подобное.

Рекомендация 4. Для лучшего наблюдения за ходом выполнения программы наряду с контрольными печатями вставьте в функции оператор "ожидания" Вашей команды на продолжение, например, оператор чтения.

2.12. Создание оконных приложений

Практически в любой операционной системе, в том числе и Windows, возможны 2 типа структур программ:

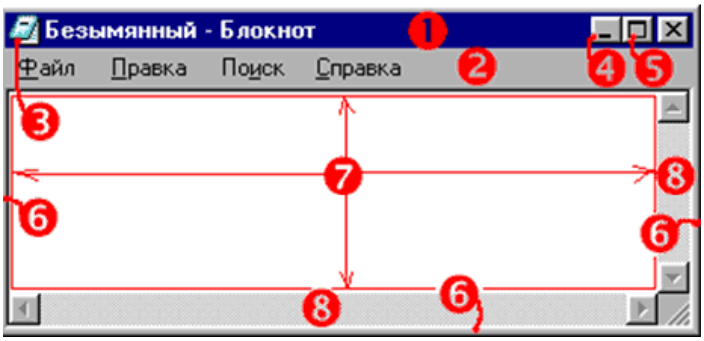
- консольная, или безоконная структура,
- классическая (оконная, каркасная) структура.

Безоконные (**консольные**) приложения представляет собой программу, работающую в текстовом режиме. Работа консольного приложения напоминает работу программы в MS DOS. Но это лишь внешнее впечатление. Консольное приложение обеспечивается специальными функциями операционной системы. Консольные приложения представляют собой систему средств взаимодействия пользователя с компьютером, основанную на использовании текстового (буквенно-цифрового) режима дисплея или аналогичных (командная строка MS DOS, Far). Консольные приложения очень компактны не только в откомпилированном виде, но и в текстовом варианте, и имеют такие же возможности обращаться к ресурсам операционной системы посредством API-функций, как и оконные приложения.

Оконные (каркасные) приложения строятся на базе специального набора функций API (Application Programming Interfaces — интерфейс программного приложения). Все взаимодействия с внешними устройствами и ресурсами операционной системы происходят посредством таких функций. Windows API в настоящее время поддерживает свыше тысячи вызовов функций, которые можно использовать в приложениях.

Любая программа для Windows имеет окно — прямоугольную область на экране, в котором приложение отображает информацию и получает реакцию от пользователя. Окно идентифицируется заголовком. Большинство функций программы запускается посредством меню. Слишком большой для экрана объем информации может быть просмотрен с помощью полос прокрутки. Некоторые пункты меню вызывают появление окон диалога, в которые пользователь вводит дополнительную информацию.

Окно может содержать элементы управления: кнопки, списки, окна редактирования и др.



Основными элементами окна являются:

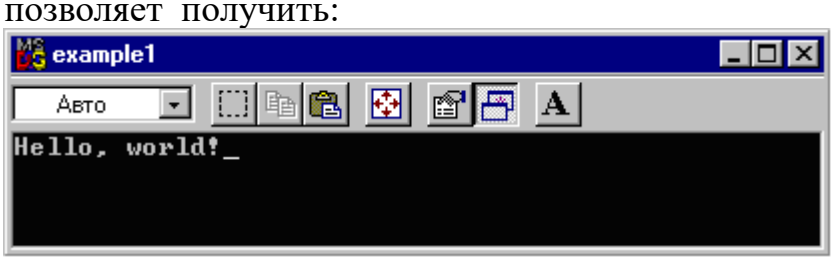
1 — строка заголовка **title bar**
2 — строка меню **menu bar**
3 — системное меню **system menu**
4 — кнопка сворачивания окна **minimize box**

5 — кнопка разворачивания окна **maximize box**
6 — рамка изменения размеров **sizing border**
7 — клиентская область **client area**
8 — горизонтальная и вертикальная полосы прокрутки **scroll bars**

Консольное приложение начинается с функции int **main**(), называемой стартовой. Например, программа, имеющая код:

```
#include <stdio.h>
int main() {
 printf("Hello, world!");
 getc(stdin);
 return 0;
}
```

после запуска позволяет получить:



Для работы с оконными приложениями используется другая стартовая функция:

```
int WINAPI WinMain (HINSTANCE hInst, HINSTANCE hpi, LPSTR cmdline, int ss)
```

- `hInst` - дескриптор для данного экземпляра программы,
- `hpi` - в Win32 не используется (всегда NULL),
- `cmdline` - командная строка,
- `ss` - код состояния главного окна.

Пример простейшей программы, иллюстрирующей работу с оконным приложением, имеет вид:

```
#include <windows.h>
int WINAPI WinMain(HINSTANCE,HINSTANCE,LPSTR,int)
{
 MessageBox(NULL,"Hello, World!","Test",MB_OK);
 return 0;
}
```

Оконные приложения и организующие их работу функции могут быть созданы как с помощью специальных мастеров, встроенных в среду программирования, так и с помощью MFC (Microsoft Foundation Classes - Базовые Классы Microsoft). MFC - это базовый набор (библиотека) классов, написанных на языке C++ и предназначенных для упрощения и ускорения процесса программирования для Windows. Библиотека содержит многоуровневую иерархию классов, насчитывающую около 200 членов. Они дают возможность создавать Windows-приложения на базе объектно-ориентированного подхода.

Структура оконного приложения

Оконные приложения строятся по принципам событийно-управляемого программирования (event-driven programming) — стиля программирования, при котором поведение компонента системы определяется набором возможных внешних событий и ответных реакций компонента на них. Такими компонентами в Windows являются окна. С каждым окном связана определенная функция обработки событий – оконная функция. События для окон называются сообщениями. Сообщение относится к тому или иному типу, идентифицируемому определенным кодом (32-битным целым числом), и сопровождается парой 32-битных параметров (WPARAM и LPARAM), интерпретация которых зависит от типа сообщения.

Задача любого оконного приложения — создать главное окно и сообщить Windows функцию обработки событий для этого окна. Самое важное для такого приложения происходит именно в функции обработки событий главного окна.

В соответствии с этим, классическое оконное приложение, как правило, состоит по крайней мере из двух функций:

- стартовая функция, создающая главное окно WinMain();
- функция обработки сообщений окна (оконная функция).

Стартовая функция WinMain создает главное окно.

Если в консольной программе на C точкой входа (с этого места программа начинает выполняться) является функция ***main***(), то точкой входа программы для Windows является функция ***WinMain***().

```
int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                         PSTR szCmdLine,
                    int iCmdShow)
                {...}
```

Эта функция использует последовательность вызовов API и при завершении возвращает операционной системе целое число.

Аргументы функции:

hInstance – дескриптор процесса (instance handle) – число, идентифицирующее программу, когда она работает под Windows. Если одновременно работают несколько копий одной программы, каждая копия имеет свое значение hInstance.

hPrevInstance – предыдущий дескриптор процесса (previous instance) — в настоящее время всегда равен NULL.
szCmdLine — указатель на оканчивающуюся нулем строку, в которой содержатся параметры, переданные в программу из командной строки. Можно запустить программу с параметром командной строки, вставив этот параметр после имени программы в командной строке.
iCmdShow — целое константное значение, показывающее, каким должно быть выведено на экран окно в начальный момент. Задается при запуске программы другой программой. В большинстве случаев число равно 1.

Цикл обработки сообщений

В Windows программа пассивна. После запуска она ждет, когда ей уделит внимание операционная система. Операционная система делает это посылкой сообщений, которые могут быть разного типа; они функционируют в системе достаточно хаотично, и приложение не знает, какого типа сообщение придет следующим. Логика построения Windows-приложения должна обеспечивать корректную и предсказуемую работу при поступлении сообщений любого типа.

ОС Windows поддерживает «очередь сообщений» для каждой программы, работающей в данный момент в системе Windows. Когда происходит ввод информации, Windows преобразует ее в «сообщение», которое помещается в очередь сообщений программы. Программа извлекает сообщения из очереди сообщений, выполняя блок команд, известный как «цикл обработки сообщений».

Для обработки сообщений окна предназначена **Оконная функция**. Эта функция организована по принципу ветвления, состоящего из последовательной проверки типа сообщения. При совпадении типа сообщения, переданного в структуре Message с соответствующей веткой, осуществляется его обработка.

Элементы управления окна

Главным элементом программы в среде Windows является окно. Окно может содержать элементы управления: кнопки, списки, окна редактирования и др. Эти элементы также являются окнами, но обладающими особым свойством: события, происходящие с этими элементами (и самим окном), приводят к приходу сообщений в процедуру окна.

Системный класс	Предназначение
BUTTON	Кнопка.
COMBOBOX	Комбинированное окно (окно со списком и поля выбора).
EDIT	Окно редактирования текста.
LISTBOX	Окно со списком
SCROLLBAR	Полоса прокрутки
STATIC	Статический элемент (текст)

В любом окне может быть создано меню.

Меню (menu) - это перечень пунктов, которые устанавливают параметры или группы параметров (подменю) для прикладной программы, выполняемой в

окне. Щелчок по пункту меню открывает подменю или заставляет приложение выполнить команду.

Меню выстраивается иерархически. На верхнем уровне иерархии находится строка меню (***menu bar***), содержащая перечень меню, которые, в свою очередь, могут содержать подменю (***submenus***). Пункт меню может или выполнить команду или открыть подменю (выпадающее меню - ***drop-down menu***). Для получения информации о строке меню используется функция GetMenuBarInfo.

Окно может содержать меню окна (системное меню). Системное меню содержит команды для изменения размера или позиции окна, а также его закрытия.

Система также предусматривает меню команд (***shortcut menus***), называемое также контекстным меню. Меню команд не соединено со строкой меню; оно может появиться в любом месте экрана. Контекстное меню остается скрытым до тех пор, пока пользователь не активизирует его, обычно щелкая правой кнопкой мыши по выбранной позиции, инструментальной панели или кнопке панели задач. Меню обычно отображается в позиции курсора мыши.

Меню может быть создано с помощью шаблона меню либо специальных функций. Шаблоны меню обычно определяются как ресурсы. Ресурсы шаблона меню могут быть загружены явно или назначены как заданное по умолчанию меню для класса окна.

Программирование элементов управления окна

После создания окна (формы) переходят к написанию требуемых функций. Функции элементов управления окна и пунктов меню программируются в специальном режиме "Code", где и записываются исходные тексты соответствующих функций. Эти, так называемые "событийные" функции, система выполняет по мере наступления события, связанному с активизацией конкретного элемента окна: кнопки либо пункта меню.

На этом проектирование окна можно считать законченным. Далее осуществляется выполнение полученного приложение и его отладка (в MS Visual Studio - с помощью клавиши F5).

Задание 11

Создать программный продукт, позволяющий решать поставленную задачу (см. Приложение П 10).

Обеспечить удобный интерфейс для ввода исходных данных, наглядное представление получаемых результатов.

Предусмотреть:

- возможность чтения исходных данных с клавиатуры и из файла (имя файла запрашивается у пользователя);
- создание, хранение и коррекцию исходный данных;
- произвольную размерность решаемой задачи.

Программу, реализующую конкретный вычислительный метод, написать на языке ”СИ”. Интерфейс создать в среде MS Visual Studio.

Рекомендация. Сначала выполняется Задание в консольном режиме, и только убедившись, что программа работает грамотно с любыми входными значениями, переходят к созданию оконного приложения.

2.13. Перегрузка функций и операторов

C++ является так называемым объектно-ориентированным языком. Объектно-ориентированное программирование (ООП), в отличие от процедурого программирования, прежде всего является иным подходом к разработке программ, популярной парадигмой создания программного обеспечения.

Программа, написанная с помощью процедурного подхода к программированию — это монолитная программа, содержащая определенное количество инструкций (операторов), необходимых программисту, а также, функций (подпрограмм). Все эти операторы и функции пишутся и выполняются последовательно, в соответствии с заложенным при создании программы алгоритмом.

ООП при разработке программ использует шаблоны, перегрузки, классы,... обладает некоторыми особенными, иными свойствами, а также понятиями: полиморфизм, инкапсуляция, наследование...

Изучению и использованию ООП для решения различных задач посвящен дальнейший материал.

Перегрузка функций

Перегрузка функций — это особенность в C++, которая позволяет определять несколько функций с одним и тем же именем, но с разными параметрами. Перегрузка функций относится к одному из способов реализации полиморфизма в C++.

Перегрузка может быть выполнена с помощью функции-шаблона.

Функция-шаблон определяет общий набор операций, который будет применен к данным различных типов **при неизменном их количестве**. При помощи функции-шаблона определяется алгоритм действий безотносительно к типу данных. То есть, при создании функции- шаблона, создается функция, которая может автоматически перегружать сама себя.

Функции-шаблоны создаются с использованием ключевого слова template (шаблон). Шаблон используется для создания каркаса функции, оставляя компилятору реализацию подробностей (в зависимости от типа параметров).

```
#include <iostream.h>
// шаблон функции
template <class X>    void swap(X &a, X &b)
{
 X temp;
 temp = a;    a = b;    b = temp;
}
```

Строка

```
template <class X> void swap (X &a, X &b)
```

указывает компилятору, что создается шаблон. Здесь X — шаблон типа, используемый в качестве параметра-типа. Далее следует объявление функции

swap() с использованием типа данных X для тех параметров, которые будут обмениваться значениями.

Применение функции-шаблона в программе понятно из следующих операторов:

```
swap(i, j);  // обмен целых
swap(x, y);  // обмен вещественных значений
swap(a, b);  // обмен символов
```

Поскольку функция swap() является функцией-шаблоном, то компилятор автоматически создаст три разные версии функции swap() — одну для работы с целыми числами, другую для работы с числами с плавающей запятой и, наконец, третью для работы с переменными символьного типа.

Можно определить несколько типов-шаблонов данных в инструкции template, используя список с запятыми в качестве разделителя:

```
template <class type1, class type2> void myfunc(type1 x, type2 y)
{
cout << X << ' ' << y << endl;
}
```

Итак, функция-шаблон перегружает себя по мере необходимости.

Кроме того, можно любую функцию **перегружать явным образом**:

```
void swap(int &a, int &b)
{
 int temp;
 temp = a;   a = b;     b = temp;
}

void swap(float &a, float &b)
{
 float temp;
 temp = a;    a = b;     b = temp;
}
```

Примечания:

- Для перегруженных функций можно выполнять различные действия в теле каждой функции с разным числом параметров.
- Для функции-шаблона необходимо выполнять одни и те же общие действия, и только тип данных может быть различным (при неизменном их количестве).
- Символ "&" при описании параметров, определяющий "местодержатель" этого параметра, было бы грамотнее заменить на символ "*", обозначающий указатель на объект данного типа.

Перегрузка операторов

Оператор в C++ - это некоторое действие или функция, обозначенная специальным символом. Перегрузка операторов в C++ служит для того, что бы распространить эти действия на новые типы данных, сохраняя при этом естественный синтаксис.

При перегрузке (переопределении) операторов следует помнить, что:

- Операторы "." и "a?b:c"(тернарный оператор) переопределить нельзя.
- Операторы "a->", "[]", "()", "=" и "(type)" можно переопределить только как методы класса.
- При переопределении операторов "," "&&" "||" теряются их "ленивые" свойства.

  Ленивые вычисления — применяемая в некоторых языках программирования стратегия вычисления, согласно которой вычисления следует откладывать до тех пор, пока не понадобится их результат. Ленивые вычисления относятся к нестрогим вычислениям.
- Перегрузка операторов выполняется прежде всего на объекты вновь создаваемого класса.

Рекомендация. Прежде чем выполнять задание по перегрузке операторов, следует **изучить**:

- что такое класс,
- что такое объект класса,
- что такое методы класса, включая
  - конструктор и
  - деструктор,

а затем создать класс, конструкторы, деструктор, объекты этого класса, а также методы, включая методы вводы и вывода.

Перегрузка операторов выполняется разными способами в зависимости от места их определения (внутри объявления класса, либо после его объявления).

Кроме того, по разному определяются бинарные и унарные операторы.

Отдельно необходимо обратить внимание на перегрузку оператора "=" - "присвоить", и отличие этого оператора от оператора "+=" - "добавить".

Обо всем этом при выполнении задания студенту предлагается прочитать **самостоятельно**.

Задание 12

Выполнить задание из Приложения П 14. При перегрузке функций использовать (по возможности) шаблон функции. Перегрузку операторов выполнить с учетом рекомендаций, описанных выше.

Сделать соответствующие выводы.

По результатам выполнения составить отчет.

Примечание. Для каждой части Задания написать отдельную программу.

2.14. Создание базы данных в Access

Microsoft Access — реляционная система управления базами данных (СУБД) корпорации Microsoft. MS Access имеет широкий спектр функций, включая запросы на добавление, сортировку по разным полям, связь с внешними таблицами и базами данных. Создаваемые формы, запросы и отчеты позволяют быстро и эффективно обновлять данные, получать ответы на вопросы, осуществлять поиск нужной информации, анализировать ее, печатать отчеты.

Создание любой базы данных начинается с изучения и описания предметной области.

По результатам этой работы составляется структура информационной системы.

Эта структура после необходимого этапа нормализации отношений может быть представлена следующей схемой данной:

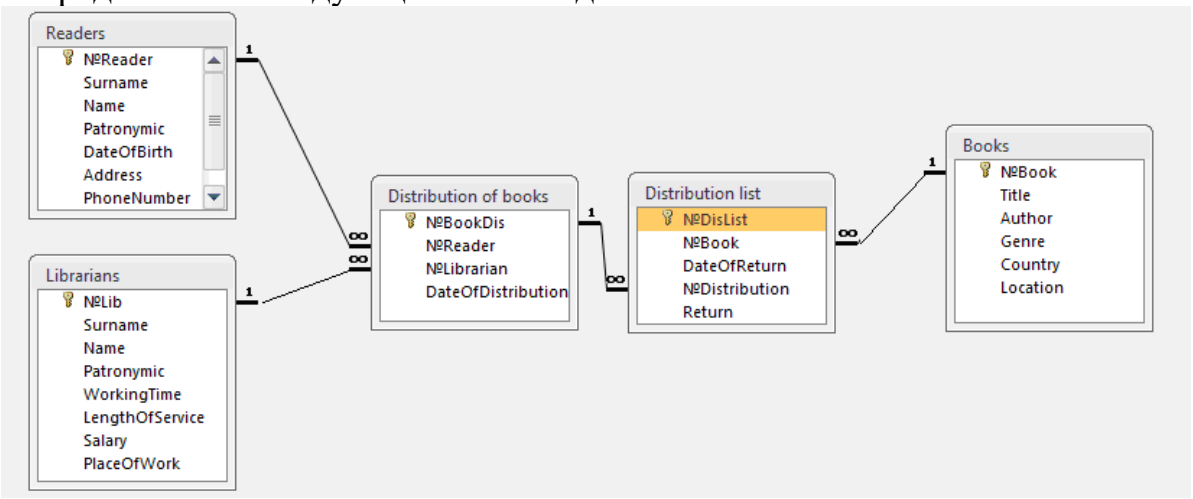


Схема данных прежде всего определяет состав таблиц, в которых предполагается хранение информации, а также связи между таблицами.

В большинстве баз данных эти связи имеют характер "один-ко-многим". Если используется связь "один-к-одному", то это признак того, что информация какой-то одной таблицы разбита на две, когда значения одних полей конкретных сущностей размещены в одной таблице, а оставшихся полей - в другой.

Рекомендация 1. Связь "многие-ко-многим" следует исключить, преобразовав схему данных, добавив вспомогательную таблицу с двумя связями "один-ко-многим".

Нормализация отношений подразумевает такое разбиение всей информации по таблицам, когда каждое значение сохраняется только в одном экземпляре, без повторов.

На втором этапе происходит создание таблиц. Создание таблиц в MS Access следует выполнять в режиме конструктора. Результатом этого этапа работы будет необходимое количество таблиц, строго с разработанной ранее схемой данных.

Рекомендация 2. При именовании таблиц и их полей лучше пользоваться латиницей. Все названия писать без пробелов. В крайнем случае можно использовать знак "_" - подчеркивание.

Ниже приведены скрины трех таблиц, открытых в конструкторе:

1. Readers

Имя поля	Тип данных
№Reader	Счетчик
Surname	Текстовый
Name	Текстовый
Patronymic	Текстовый
DateOfBirth	Текстовый
Address	Текстовый
PhoneNumber	Текстовый
№LibraryCard	Числовой

2. Distribution_of _books

Имя поля	Тип данных
№BookDis	Счетчик
№Reader	Числовой
№Librarian	Числовой
DateOfDistribution	Дата/время

3. Distribution_list

Имя поля	Тип данных
№DisList	Счетчик
№Book	Числовой
DateOfReturn	Дата/время
№Distribution	Числовой
Return	Дата/время

При создании таблиц в левом столбце указаны наименования полей, соответствующие, характеристикам (атрибутам) информационных сущностей, а в правом - типы данных. Некоторые поля обозначены "ключами". С помощью этих ключей организуются в дальнейшем горизонтальные связи между таблицами. Столбец с таким символом служит для описания родительского ключа. Дочерний ключ в соответствующей таблице будет определен непосредственно в схеме данных после создания всех таблиц. Пока же (при конструировании таблиц), это поле должно иметь числовой тип (при условии, что поле "Родительский ключ" есть Счетчик.

Рекомендация 3. При создании таблиц никакими подстановками не пользоваться и таковых не создавать! Все необходимые связи и подстановки будут организованы при создании соответствующих форм.

Рекомендация 4. После конструирования таблиц запрещается вводить в них информацию вручную! Для этого, как будет показано ниже, создаются специальные формы.

На третьем этапе организуются горизонтальные связи между таблицами. Данная задача решается в режиме "СХЕМА ДАННЫХ", куда добавляются все созданные таблицы, и мышкой "протягиваются" необходимые связи от одной таблицы к другой. В свойствах созданных связей указать, что рассматриваемая связь имеет тип "одна-ко-многим", причем должны быть реализованы каскадные удаления и изменения полей.

Четвертый этап является по сложности и по значению самым важным и ответственным. Это этап создания форм. Формы служат как для ввода информации в таблицы, так и для управления всей базой данных. Сначала организуется возможность ввода информации через создаваемые формы в соответствующие таблицы, а затем формы могут подвергнуться необходимому редактирова-

нию для удобного управления базой данных. При этом на формы добавляются дополнительные кнопки, флажки и прочие органы управления. Возможно создание отдельных форм, откуда могут быть вызваны ранее созданные.

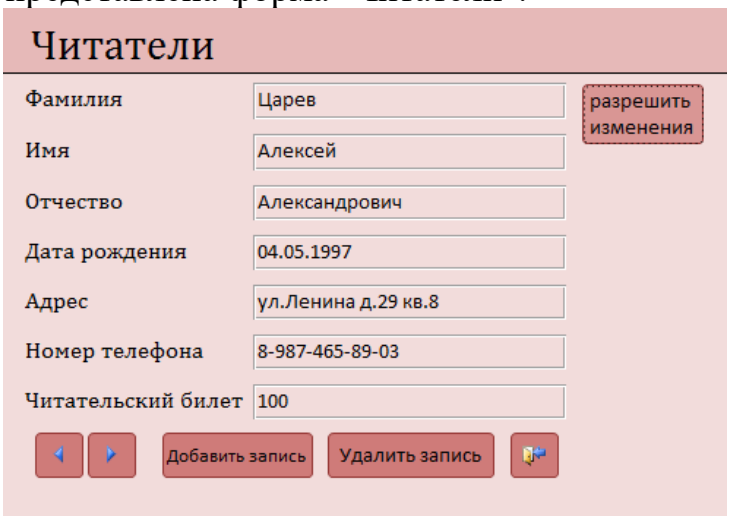
Рекомендуется следующий порядок создания форм. Лучше начать с простых форм, таблицы которых имеют только родительские ключи. Формы ввода удобнее всего создавать с помощью мастера форм, сразу указав, в какую таблицу будет сохраняться информация.

Рекомендация 5. В формах не следует отображать поля таблиц, значения которых заполняются автоматически (счетчики).

Рекомендация 6. При разработке большинства форм можно использовать формы типы "в столбец", а когда заранее известно, что информация в таблице уместится в несколько строк, лучше использовать "ленточную форму".

Далее, с целью редактирования формы, выполняют переход в режим конструктора. В режиме редактирования форм можно вставить соответствующие кнопки для управления формой и информацией в форме (переходы по записям). Поменять внешний вид, убрать лишние элементы формы.

На рисунке представлена форма "Читатели".



В ней будет отображаться и вводиться информация о читателях библиотеки. В форме организованы кнопки перехода по записям, кнопка "Выход", а также "Добавить запись" и "Удалить запись". При нажатии на кнопку "Добавить запись" открывается аналогичная форма с пустыми полями (в режиме добавления) для добавления нового читателя.

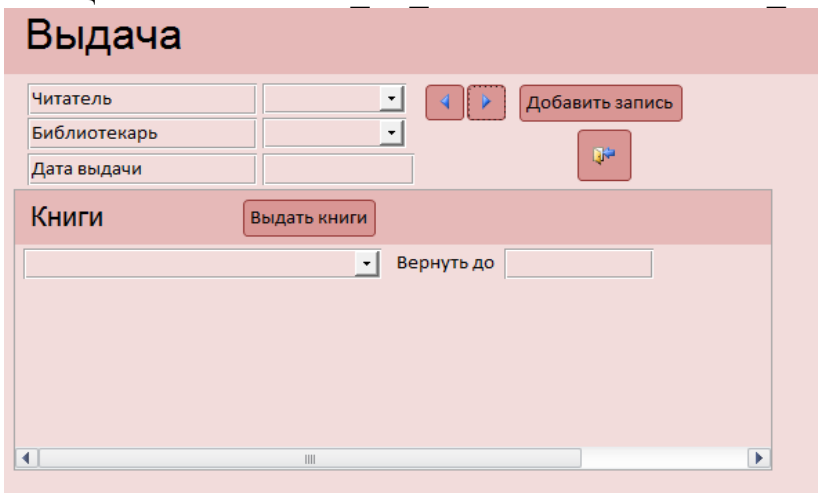
Для защиты информации от случайного удаления либо изменения предусмотрена специальная защита. Она организована с помощью ряда процедур (приведены ниже) и двух кнопок: "Разрешить изменения" и "Запретить изменения". Как следует из текста процедур, при Открытии формы (по умолчанию) запрещается вносить изменения в записи. При нажатии кнопки "Разрешить изменения" пользователю разрешается изменения записи, и вместо этой кнопки появляется (становится видимой) кнопка "Запретить изменения". Чтобы снова запретить изменения нужно нажать на кнопку "Запретить изменения", вместо нее снова появится кнопка "Разрешить изменения". Обе указанные кнопки

первоначально создаются рядом друг с другом; затем, после написания процедуры, их можно совместить (поставить в форме на одно место).

```
Option Compare Database
Private Sub form_open(Cancel As Integer
Me.Кнопка15.Visible = True
Me.Кнопка15.SetFocus
Me.Кнопка16.Visible = False
Form.AllowEdits = False
End Sub
Private Sub Кнопка15_Click()
Me.Кнопка16.Visible = True
Me.Кнопка16.SetFocus
Me.Кнопка15.Visible = False
Form.AllowEdits = False
End Sub
Private Sub Кнопка16_Click()
Me.Кнопка15.Visible = True
Me.Кнопка15.SetFocus
Me.Кнопка16.Visible = False
Form.AllowEdits = False
End Sub
```

Кроме того, отредактирована и сама форма. Удалены полосы прокрутки, область выделения, а также некоторые кнопки управления окном. Все эти действия выполнены в режиме конструктора формы, путем изменения соответствующих свойств в макете формы.

Некоторые формы удобно создавать в режиме Подчиненные формы. Для этого при создании формы следует указать сразу две связные между собой таблицы и выбрать режим "подчиненные формы". Например, форма "Выдача" построена по таблицам " Distribution_of _books" и "Distribution_list".



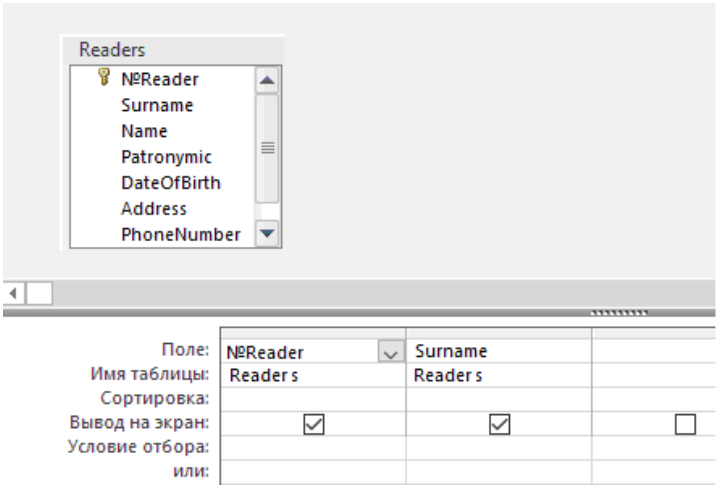
Из рисунка видно, что в главной форме использован способ представления информации "В столбец", в подчиненной форме режим - "Ленточная форма". Для отображения в поле формы "подставляемой" информации, соответствующей некоторому родительскому ключу, данное поле предварительно преобразуется в поле со списком, и с помощью соответствующего запроса, в данном поле организуется необходимое отображение информации из родительской таблицы. В рассматриваемой форме эти преобразования выполнены над полями Читатель и Библиотекарь главной формы, а также Книги Подчиненной формы.

Пример запроса, позволяющего в указанных полях окна (формы) отобразить правильную информацию, будет приведен ниже.

На пятом этапе переходят к созданию запросов. Следует заметить, что некоторые запросы создаются уже при редактировании форм.

Запрос для поля со списком.

Поле со списком необходимо в форме для выбора нужного элемента. Для этого с помощью конструктора запросов пишется соответствующий запрос. (). Например, в форме "Выдача" для поля "Читатель" (в режиме конструктора форм) после преобразования этого Поля в Поле со списком, в Свойствах поля для Источника строк строится запрос (предварительно следует нажать мышкой на ... - три точки).



Далее в свойствах запроса указывается, что Присоединенный столбец - 1-й (именно из этого столбца информация переносится в таблицу), всего столбцов - 2, и для вывода информации следует дополнительно указать ширину столбцов, например: 0; 3 см (значение первого столбца не показывается, а на второе поле выделено 3 см). Кроме того для удобства пользователя имеет смысл в поле "Surname" запроса назначить режим сортировки информации по алфавиту (в рассматриваемом примере это упущено).

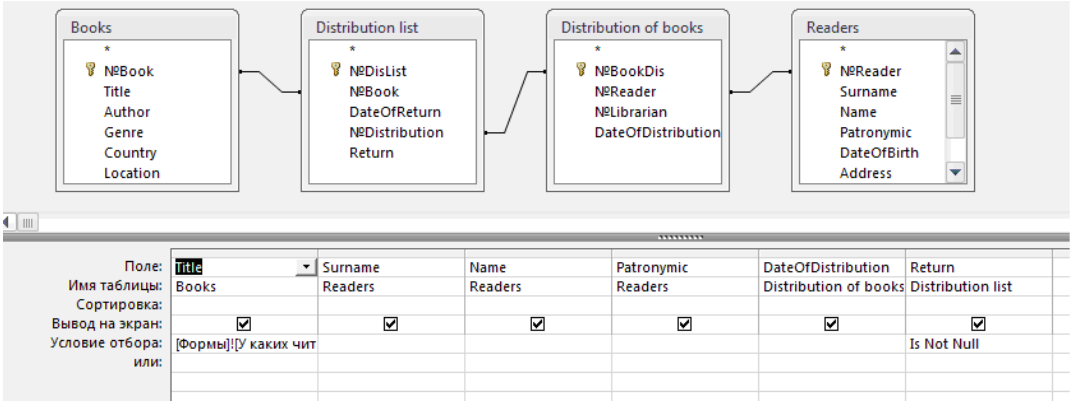
При создании формы "Выдача" следует для поля "Книги" в соответствующий запрос добавить условия отбора: Показывать только те книги, которые есть в библиотеке. Кроме того, необходимо создать запрос На изменение информации, изменяющий у этих книг значение поля Location на "У читателя". Данный запрос запускается при нажатии на кнопку "Выдать книги".

Запрос на выборку

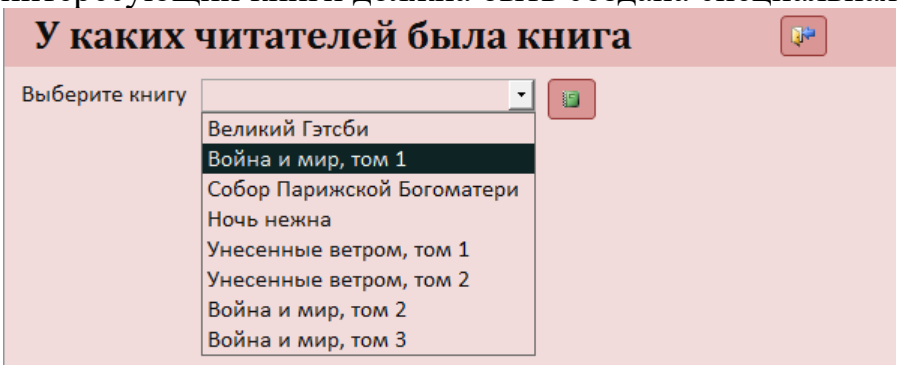
Данный запрос создан для того, чтобы выбрать Читателей, когда-либо читавших книгу, которая в данный момент находится в библиотеке.

Запрос создается в режиме Конструктора запросов сразу по четырем таблицам.

При создании запроса имеет смысл организовать сортировку выбранной инфомации, указав нужный режим в строке Сортировка (в соответствующем поле).



Для выбора интересующий книги должна быть создана специальная форма:



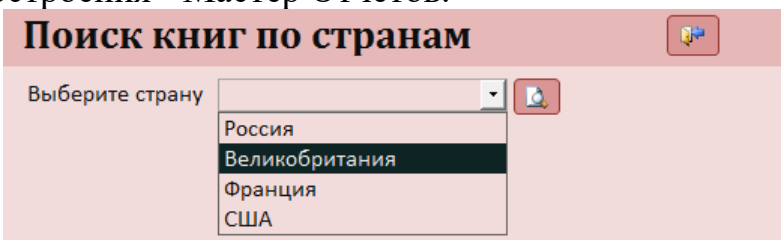
На этой форме рядом с выбранной книгой находится кнопка построения отчета по информации, которая будет отобрана с помощью этого запроса на выборку (но об этом - ниже).

Рекомендация 7. Следует помнить, что Запрос - это прежде всего **средство**, с помощью которого решается задача **отбора** необходимой информации. Или для отображения в форме, или для построения соответствующего отчета.

Кроме запросов на выборку создаются (по мере необходимости) запросы на добавление информации в таблицу или запросы на изменение информации. Иногда удобнее создавать запросы на языке SQL. Об этом рекомендуется прочитать самостоятельно. Можно поискать нужную информацию в интернете, на соответствующих форумах.

На шестом этапе выполняется Создание отчетов.

Например, для получения информации о книгах, авторы которых являются (являлись) гражданами какой-либо страны, создается вспомогательная форма, где выбирается интересующая страна, строится запрос на выборку, в котором отбираются книги этой страны, и уже по данному запросу строится нужный отчет. Режим построения - Мастер Отчетов.



Вид отчета:

Книги определенной страны

17 декабря 2018 г.
1:06:54

Название	Автор	Жанр	Страна
Собор Парижской Богоматери	Виктор Гюго	Роман	Франция

Кол-во книг: 1

Страница 1 из 1

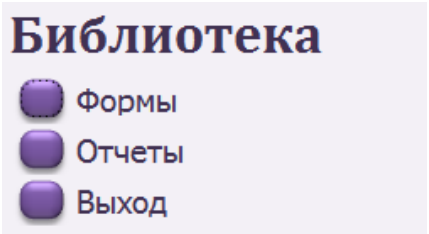
В форме "Поиск книг по странам" рядом с полем, где отображается выбранная страна, расположена кнопка, при нажатии на которую будет построен требуемый отчет.

Рекомендация 8. Желательно данный отчет отредактировать в режиме "Конструктор отчетов", чтобы вставить название выбранной страны непосредственно в заголовок отчета, убрав ее из табличной части.

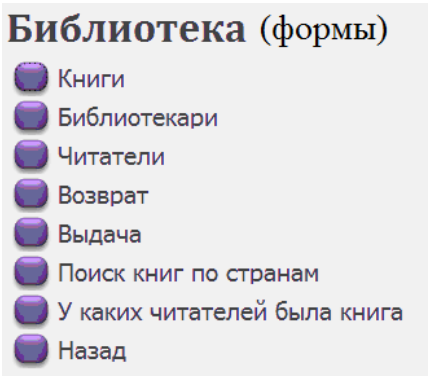
В ряде отчетов на этапе их построения может быть использована Группировка информации по какому-либо признаку. Также, если это необходимо, возможно добавление промежуточных либо общего итога по отчету.

Седьмой этап. Создание интерфейса

На этом этапе создается форма, откуда можно управлять работой данной базой данных. Из неё будет осуществляться переход на другие формы и формироваться необходимые отчеты. Один из возможных вариантов такого интерфейса изображен на рисунке:

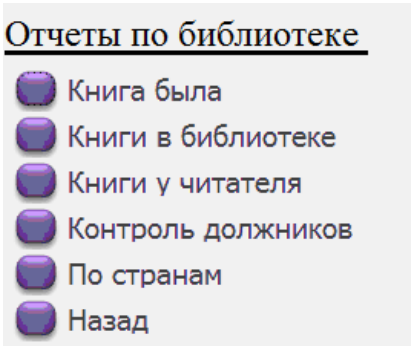


При нажатии на кнопку «Формы» открывается список форм с кнопками для их открытия, также есть кнопка «назад» для возврата на главную кнопочную форму.



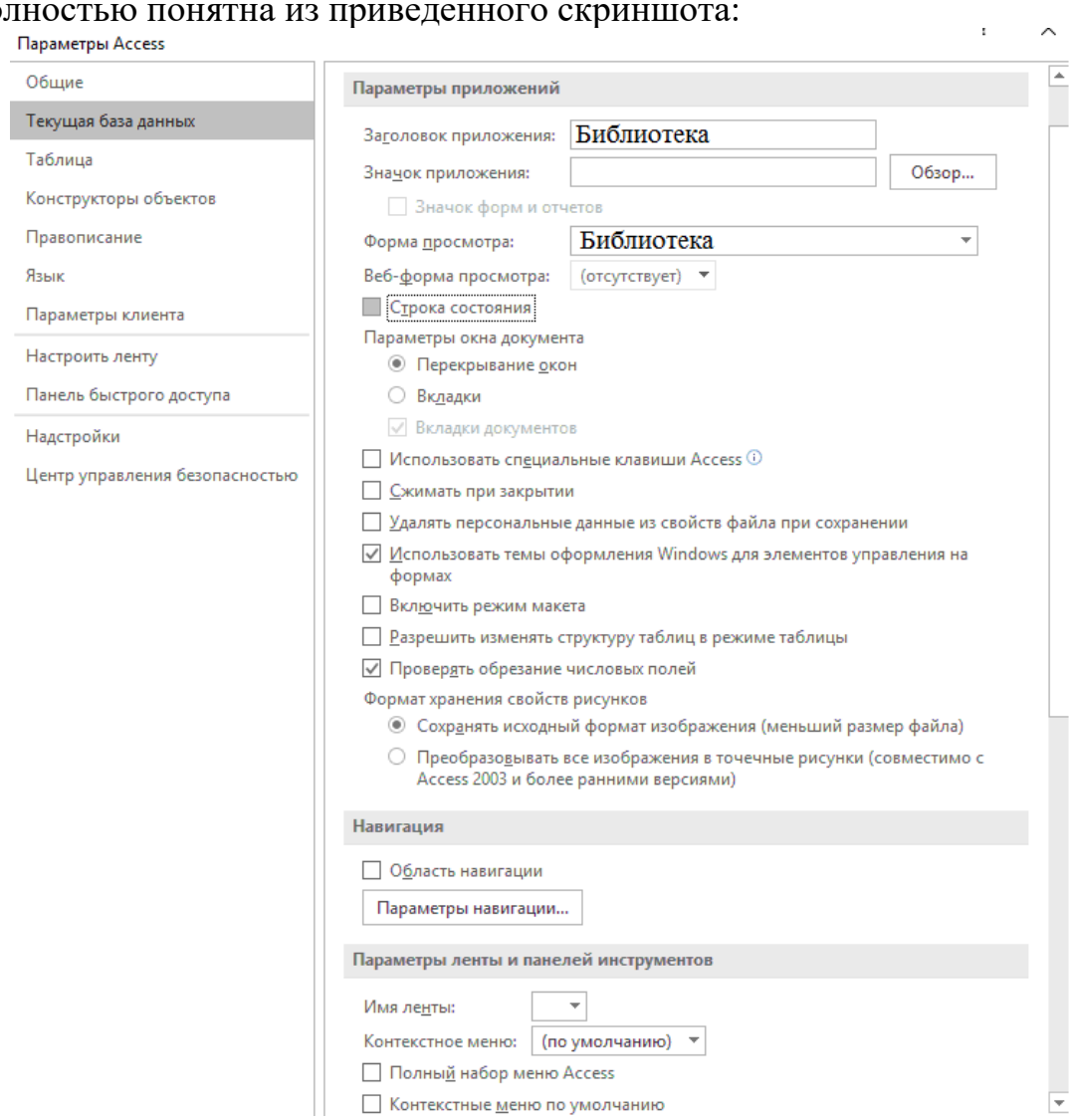
Форма «Выдачи» по умолчанию открывается для добавления записи, остальные формы - в режиме просмотра информации.

При нажатии на кнопку «Отчеты» открывается список отчетов с кнопками для их открытия, также есть кнопка «назад» для возврата на главную кнопочную форму.



Форму "Библиотека" следует указать в настройках, чтоб она выводилась на экран при открытии разработанной базы данных.

Заключительный этап. Настройка и защита базы данных. Данная функция полностью понятна из приведенного скриншота:



На этом создание базы Данных в MS ACCESS можно считать завершенным.

Задание 13

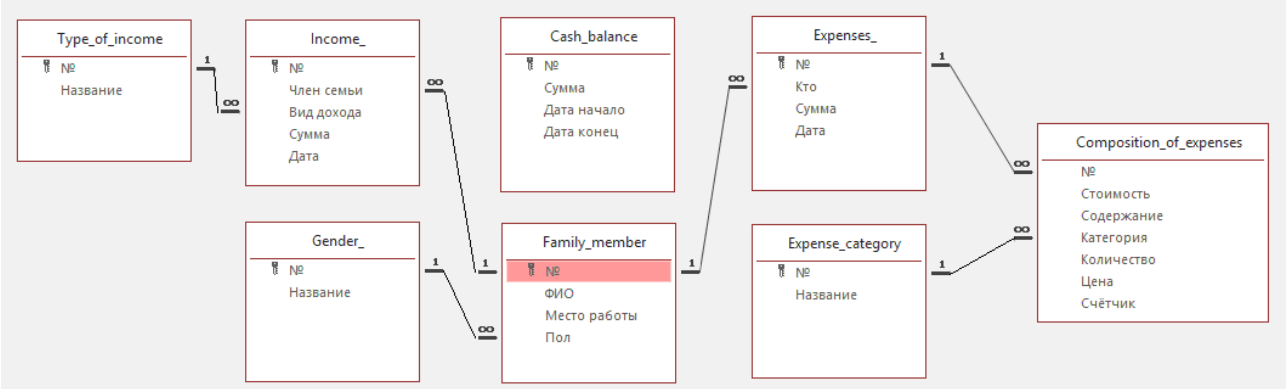
Создать базу данных в среде MS Access. Вариант задания взять из Приложения П 15. По результатам работы написать отчет.

2.15. Создание базы данных на C++ в среде MS Visual Studio

Задание 14

Создать базу данных (Приложение П 15) на C++ с разработкой удобного оконного интерфейса в среде MS Visual Studio.

В предыдущей работе при создании реляционной базы в среде MS Access была разработана схема данных информационной системы:



Настоящим заданием требуется повторить эту базу на C++ в среде разработки приложений MS Visual Studio. То есть необходимо создать приложение с оконным интерфейсом, идентичным визуальной части базы Access.

Вместо таблиц используются связные списки.

Рекомендация 1. В силу того, что обычно для поиска необходимой информации таблицы всегда просматриваются только сверху вниз, вставка новых строк в любой реляционной базе осуществляется только в конец таблицы, удаление информации из таблиц практически не происходит, **допустимо** использование **односвязных списков** (что не обязательно).

На первом этапе для создания списков прежде всего разрабатывается шаблон класса, например, следующего вида:

```
//шаблон для создания классов (списков)
template <typename T,typename V>
class table
{
public:
     T data;
     V *next;
     V *prev;
     table();
     V *push(V *, V *);
     . . . .
     V *request_object_equal_id(V *,int);
};
```

Шаблон класса кроме информационного поля **data** типа **T** содержит указатели на следующий и предыдущий элемент списка (для случая двусвязного списка).

Кроме упомянутых компонентов в шаблон класса включаются все основные методы работы со списком и его элементами:

- создание нового элемента списка;
- вставка элемента в список;
- поиск адреса элемента в списке по его id;
- поэлементная печать содержимого списка;
- загрузка в список информации из файла;
- выгрузка информации из списка в файл;
- и другие полезные функции.

Например: `push` - метод добавления нового элемента в список:

```
//шаблон метода добавления нового элемента в начало списка
template <typename T, typename V>
inline V *table<T, V>::push(V *a, V *b)
{
     a->next = b;
     b->prev = a;
return a;
}
```

`request_object_equal_id` - метод поиска адреса элемента в списке по его Id.

Рекомендация 2. Для того, чтоб данная функция была написана как шаблонная, необходимо при объявлении соответствующих структур именовать все идентификаторы одинаковым именем Id.

Рекомендация 3. Прежде чем создавать шаблоны классов и шаблоны методов (функций), почитайте об этом в соответствующей литературе по C++, освежите в памяти работу (см. п. 2.13 настоящего пособия) по перегрузке функций с помощью шаблонов функций.

Далее можно переходить к объявлению классов, соответствующих содержимому (структуре информации) каждой из таблиц (каждого создаваемого списка). Для этого требуется создать классы, содержащие структуры с полями (как в таблицах Access) и методы обработки этой структуры.

Например, для работы с информацией о сотрудниках

а) создается структура:

```
typedef struct Income_
{ int id;
  Fam_member *id_fam_member;
  Type_of_income *id_type_of_income;
}INCOME;
```

б) и объявляется класс `Income`:

```
class Income : public table<INCOME,  Income>
{
public:
```

```
    Income();
    Income load_information_in_box(const char * );
    Income add_new(Fam_member, Type of income, const char*)
    void add_into_file(const char * , Income *);
    void Show(Income * , vector<int>&, Fam_member *,
    Type_of_income *);
    void Show(Income * , Fam_member *, Type_of_income *);
     .  .   .
  };
```

Рекомендация 4. При объявлении структуры данных в классах, соответствующих таблицам (спискам), где есть поля с дочерними ключами, в этой структуре требуется явно объявлять данное поле как указатель на элемент из родительского списка. В последующем это поле заполняется соответствующим значением с помощью шаблонной функции `request_object_equal_id.`

Например, в приведенной структуре **`Income_`**, в отличие от описания соответствующей таблицы базы Access, два поля являются адресами элементов из родительских списков (в таблицах эти поля были объявлены числовыми). Поэтому при формировании списка в результате чтения информации из файла (или другим способом) по значениям `id_type_of_income` и `id_fam_member` находятся адреса элементов в списках `Type_of_income` и `Family_member` с соответствующими значениями идентификаторов. Эти адреса и вставляются в качестве значений полей структуры (класса) рассматриваемого элемента списка Сотрудники. То есть в полях структуры элемента списка кроме необходимой информации хранятся не Id (идентификатор) элемента родительской таблицы, а его адрес в соответствующем списке, в отличие от содержимого таблиц Access.

Именно этим фактом обуславливается необходимость разработки в шаблоне класса метода `request_object_equal_id` – метода поиска адреса элемента в списке по его Id.

При переносе информации из списка в файл по данным указателям (адресам) находятся идентификаторы соответствующих строк родительского списка, которые и записываются в этот файл в качестве конкретных значений (здесь как в Access!!!).

В данный класс входят методы: чтение строки из файла, перенос информации в соответствующую структуру, запись информации элемента списка в файл, чтение информации в список с поля окна формы, перенос информации из элемента списка в поле формы. Возможно создание других методов работы со списком конкретного класса.

В приведенном выше примере класса `Income` методы объявлены условно. Их количество, параметры и наименование будут определяться конкретными задачами, решаемыми при создании базы данных, а именно этим списком.

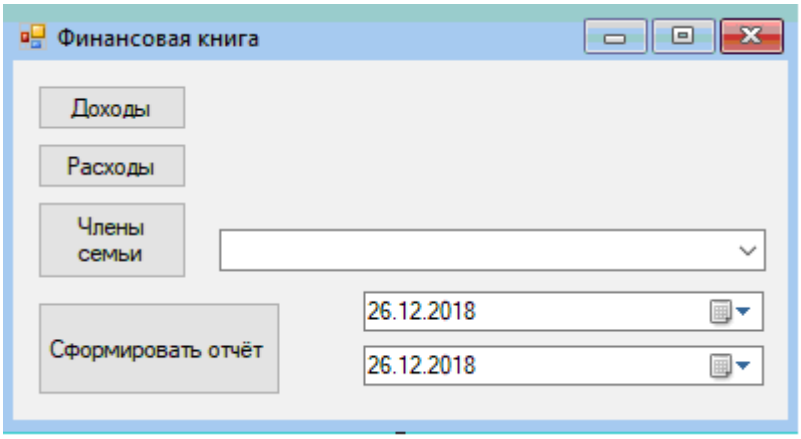
Часть методов класса (для работы с конкретными данными) могут быть использованы как в консольном приложении, так и оконном. Отдельные мето-

ды применяются только в оконном приложении, - для загрузки информации в соответствующее окно формы (и выгрузке из него в список).

Рекомендация 5. При создании методов класса удобнее выполнить их перегрузку, используя одинаковые имена для однотипной работы со структурой информации разных классов. Кроме того, располагая методы в одном файле, можно "увидеть" их схожесть, и вместо одинаковых методов для разных классов написать ***один шаблонный метод***.

Рекомендация 6. Информация из файла читается всегда только в список с последующим (если необходимо) отражением ее в окне. Выгрузка в файл - в обратном порядке, только из списка. Окна служат только для отображения и ввода информации. Поэтому написанную на C++ базу данных проверяют сначала в консольном режиме. И только убедившись, что все функции базы работают нормально, переходят к созданию оконного приложения, переписав методы ввода информации с консоли в список на ввод с окна.

Создание оконного приложения лучше начинать с разработки главной формы:



откуда по нажатию кнопок будут выполняться соответствующие функции, в том числе открывающие и "дочерние" формы (окна).

Внимание! Про создании соответствующего файла main главной формы не забыть для корректной работы приложения включить визуальные стили и выполнить необходимые начальные установки:

```
#include "Form1.h"
using namespace System;
using namespace System::Windows::Forms;
[STAThread]
void main(array<String^>^ args) {
    Application::EnableVisualStyles();
    Applica-
tion::SetCompatibleTextRenderingDefault(false);
    DATABASEFAMILY::Form1 form;
    Application::Run(%form);
```

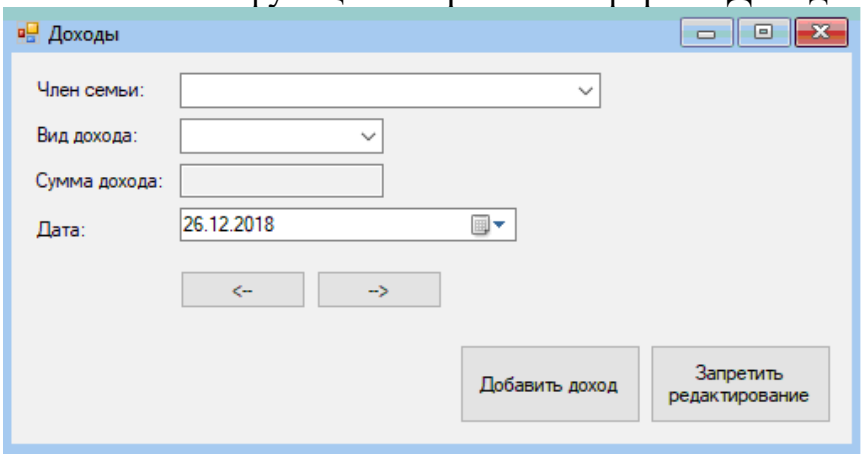
```
}
```

Из панели инструментов в форму вставляются все требуемые для работы компоненты. Затем записываются по соответствующим событиям необходимые функции.

Например, при нажатии на кнопку "Доходы" создается событие, на которое пишется функция:

```
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    Incomeform ^income_form = gcnew Incomeform(tt, gg, ff, ii, ee);
    income_form->Show();
}
```

При выполнении этой функции открывается форма "Доходы":



При нажатии на другие кнопки по написанным функциям обработки событий будут открываться соответствующие формы.

В открывшихся окнах можно просматривать информацию о доходах, расходах, людях и прочее. Для добавления информации в какой-либо список нажимается кнопка "Добавить…", в появившимся окне заполняются соответствующие поля с последующим переносом этой информации в список (при нажатии на кнопку "Добавить", например:

```
private: System::Void button7_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    newincome ^expense_form = gcnew newincome(tt, gg, ff, ii, ee);
    expense_form->ShowDialog();
}
```

Во всех формах, где показывается информация, имеется кнопка, которая либо разрешает, либо запрещает редактирование информации (по умолчанию: редактирование запрещено). Функция, выполняемая при нажатии на эту кнопку, аналогична соответствующей процедуре, написанной для формы MS Access, и имеет вид:

```
private: System::Void button4_Click(System::Object^  sender,
```

```
                                                System::EventArgs^  e)
{
     this->textBox1->ReadOnly = false;
     this->textBox2->ReadOnly = false;
     this->button5->Visible = true;
     this->button4->Visible = false;
}
private: System::Void button5_Click(System::Object^  sender,
                                                System::EventArgs^  e)
{
     this->textBox1->ReadOnly = true;
     this->textBox2->ReadOnly = true;
     this->button5->Visible = false;
     this->button4->Visible = true;
}
```

При запуске программы (открытии главной формы) происходит загрузка данных из файлов в списки. Фрагмент функции `MyForm` имеет вид:

```
MyForm(void)
{
     tt = NULL;
     .   .   .   .
     ff->load_information_in_box(PEOPLE);
     .   .   .   .
     InitializeComponent();
     Fam_member *a = ff;
     while (a != NULL)
     {
     this->comboBox1->Items->Add(gcnew String(a->data.Surname)
      + " " + gcnew String(a->data.Name) + " " +
                                gcnew String(a->data.Patronomic));
     a = a->next;
     }
}
```

Во всех остальных формах данные (на примере формы «Доходы») отражаются в окнах с помощью функции следующего вида:

```
Incomeform(Type_of_income *t, Gender *g, Fam_member *f,
                                                Income *i, Expense *e)
{
 this->tt = t; this->gg = g; this->ff = f; this->ii = i; this->ee = e;
     InitializeComponent();
     //load data into combobox1
     Fam_member *a = ff;
```

```
    while (a != NULL)
    {
        this->comboBox1->Items->Add(gcnew String(a-
    >data.Surname)+" "+
        gcnew  String(a->data.Name)+ " " +
                              gcnew String(a->data.Patronomic));
        a = a->next;
    }
    //load data into combobox2
    Type_of_income *c = tt;
    while (c != NULL)
    {
      this->comboBox2->Items->Add(gcnew String(c->data.name));
      c = c->next;
    }
    //show current information
    Fam_member *d = ff;
    Type_of_income *qw = tt;
    ii->data.id_fam_member =
                    ff->request_object_equal_id(d, c->data.Id);
    ii->data.id_type_of_income =
                   tt->request_object_equal_id(qw, c->data.Id);
    this->comboBox1->Text = gcnew String(gcnew String(
             d->data.Surname) + " " + gcnew String(d->data.Name)
                      + " " + gcnew String(d->data.Patronomic));
    this->comboBox2->Text = gcnew String(qw->data.name);
    this->textBox1->Text =
             System::Convert::ToString(ii->data.cash) + " p.";
    this->dateTimePicker1->Value = DateTime::Parse(
        System::Convert::ToString(ii->data.date.dd) + "." +
        System::Convert::ToString(ii->data.date.mm) + "." +
        System::Convert::ToString(ii->data.date.yy));
}
```

Для перемещения между элементами списка используются кнопки "←" и "→", по которым запускаются функции, перемещающие указатель на текущий элемент списка "вперёд" или "назад".

При нажатии на кнопку "Сформировать отчёт" выполняется следующая функция:

```
private: System::Void button4_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    Date one; Date two;
    one.dd = this->dateTimePicker1->Value.Day;
    one.mm = this->dateTimePicker1->Value.Month;
```

```
        one.yy = this->dateTimePicker1->Value.Year;
        two.dd = this->dateTimePicker2->Value.Day;
        two.mm = this->dateTimePicker2->Value.Month;
        two.yy = this->dateTimePicker2->Value.Year;
        Fam_member *b = new Fam_member; b = ff;
        Income *rep = new Income;
        ii->Report_request(rep, ii, b, one, two);
        ii->Report(REPORT, rep, ff, tt);
}
```

по выполнении которой формируется отчет:

Report.txt — Блокнот

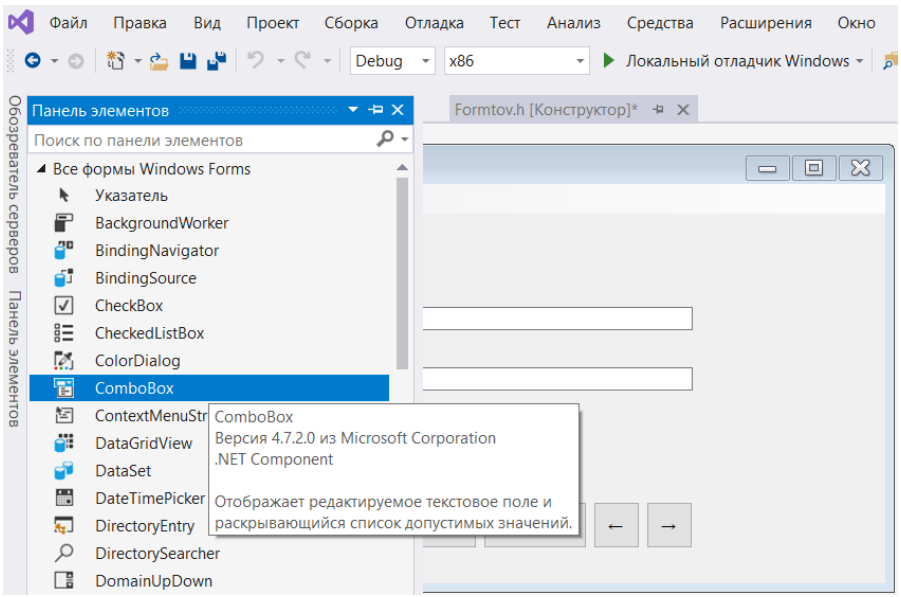
Файл Правка Формат Вид Справка

```
                Отчёт о доходах
Член семьи:  Иванчев Юрий Иванчев

1.      Стипендия  3000        22.01.2018
2.      Стипендия  3500        12.11.2016
3.      Стипендия  9000        21.01.2018
```

Будьте внимательны! В предыдущей функции по формированию отчета часть операторов может быть (и должна) написана по другому, в соответствии с конкретными средой программирования и разрабатываемым отчетом (требованиям к нему и содержащейся в нем информации).

В некоторых формах отдельные поля предполагают выбор информации из соответствующего списка. Процесс создания такого поля понятен из следующего рисунка.



Для заполнения информацией этого объекта пишется программный код по следующему образцу:

```
while (hh != NULL){
namec = gcnew String(hh->data.Name);
```

```
comboBox1->Items->Add(namec);
hh = hh->next;
}
```

Предполагается, что перед циклом и после него указателю hh должен быть присвоен адрес начала списка.

Создание остальных событий и соответствующих им функций происходит аналогично рассмотренным. При необходимости всегда можно найти примеры на сайте Microsoft.

Кроме того, в C++ имеется возможность вызова какого-либо редактора либо иной программы операционной системы с помощью функции CreateProcess (обращения из программы к командной строке), что позволяет автоматизировать работу создаваемого приложения. Об этом также предлагается ознакомиться самостоятельно.

Рекомендация 7. При создании форм следите за тем, чтоб каждая соответствующая функция имела заголовочный файл, содержащий информацию обо всех объявленных пользовательских типах данных (шаблонах, классах с методами и структурах).

Заключение

Программирование - важнейший процесс создания последовательности команд, организующий работу любого компьютер. Разработке программ различного класса и сложности, изучению принципов их построения посвящено настоящее пособие.

В пособии рассмотрена последовательность действий при разработке программ для решения различного класса задач с использованием типовых структур алгоритмов.

Продемонстрировано применение метода декомпозиции для написания сложных программ.

На всевозможных примерах показаны особенности написания программ с использованием массивов, структур, списков, деревьев, а также циклов и рекурсий. Особенное внимание уделено тестированию разработанных программ с применением трассировочных таблиц (контрольных печатей).

Отдельно рассмотрены задания по разработке программного обеспечения с оконным интерфейсом.

Пособие содержит множество примеров написания программ для задач различного типа и разной сложности, а также образцы оформления отчетов по отдельным задачам. В Приложении приведены наборы задач по различным темам.

Список рекомендуемой литературы

1. Н. Вирт "Алгоритмы и структуры данных. Новая версия для Оберона.- М.: ДМК Пресс, 2010. - 272 с.
2. Кнут Д. Э. Искусство программирования. Том 1. Основные алгоритмы = The Art of Computer Programming. Volume 1. Fundamental Algorithms / под ред. С. Г. Тригуб (гл. 1), Ю. Г. Гордиенко (гл. 2) и И. В. Красикова (разд. 2.5 и 2.6).- 3.- Москва: Вильямс, 2002.- Т.1.- 720 с.
3. Герберт Шилдт. Полный справочник по C. — Москва: Вильямс, 2009. — 704 с.
4. Герберт Шилдт. C: Полное руководство. Классическое издание. - Москва: Вильямс, 2017.- 704 с.
5. Герберт Шилдт. Полный справочник по C++. — Москва: Вильямс, 2016. — 800 с.
6. Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с. — ISBN 978-5-8459-1975-5
7. Бьерн Страуструп. Язык программирования C++. — Москва: Бином, 2018. — 1136 с.
8. Visual C++ на примерах / Галина Довбуш, Анатолий Хомоненко; под ред. Хомоненко А. Д. - Санкт-Петербург: БХВ-Петербург, 2007. - 512 с.
9. Филиповский В.М. Основы программирования и алгоритмизации. Ч. 1. Технология создания программ: Учебное пособие.- СПб.: СПбПУ, 2018.- 38 с., ил.
10. Филиповский В.М. Основы программирования и алгоритмизации. Ч. 2. Практикум по алгоритмизации: Учебное пособие.- СПб.: СПбПУ, 2019.- 57 с., ил.
11. Ундаров С.А., Филиповский В.М. Основы технологии программирования: Учебное пособие.- СПб.: СПбГТУ, 1998.- 31 с., ил.
12. Худяков А.А., Филиповский В.М. Создание базы данных на языке C++ В среде Microsoft Visual Studio. // Современные технологии в теории и практике программирования. Сборник материалов конференции. Санкт-Петербургский политехнический университет Петра Великого; Dell Technologies; EPAM Systems. Санкт-Петербург, 2020. - с. 78-80.

Приложения

П 1. Образец титульного листа отчета

Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа киберфизических систем и управления

Отчет по лабораторной работе №1
По курсу: Теория и технология программирования
Тема: Функции для строк. Вариант №3

Работа выполнена 12.09.2021
Группа 3532704/10001
Студент: _____ К. С. Кириллов

Работа принята
Доцент _____ В. М. Филиповский

Санкт-Петербург
2021

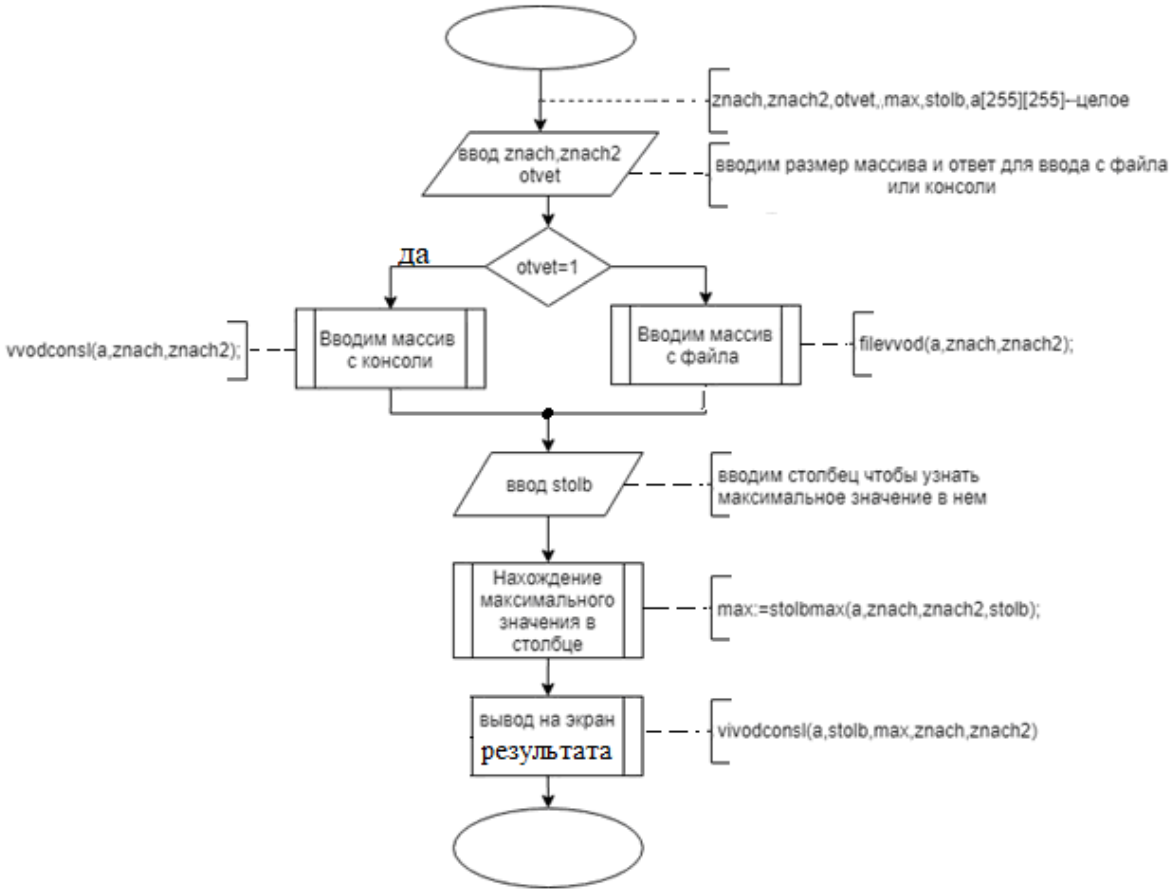
П 2. Образец готового отчета

Задание

Написать функцию нахождения максимального элемента в заданном столбце двумерного массива. Результатом функции должно быть целое число, являющиеся максимальным элементом исходного массива указанного столбца.
В главной программе предусмотреть возможность чтения массива (по запросу) из файла либо с консоли.

Разработка главной программы (main):

Блок-схема главной программы :



Структура данных:

Имя	назначение	Тип	Вид	Ограничение
a[255][255]	Исходный массив	Целое	Локальная	Не превышает размер 255*255
otvet	Определяет условие ввода с консоли или из файла	Целое	Локальная	Ввод только 1 или 0
znach	Длина массива	Целое	Локальная	-
znach2	Ширина массива	Целое	Локальная	-
max	Максимум в заданном столбце массива	Целое	Локальная	-
stolb	Хранит номер столбца	Целое	Локальная	-

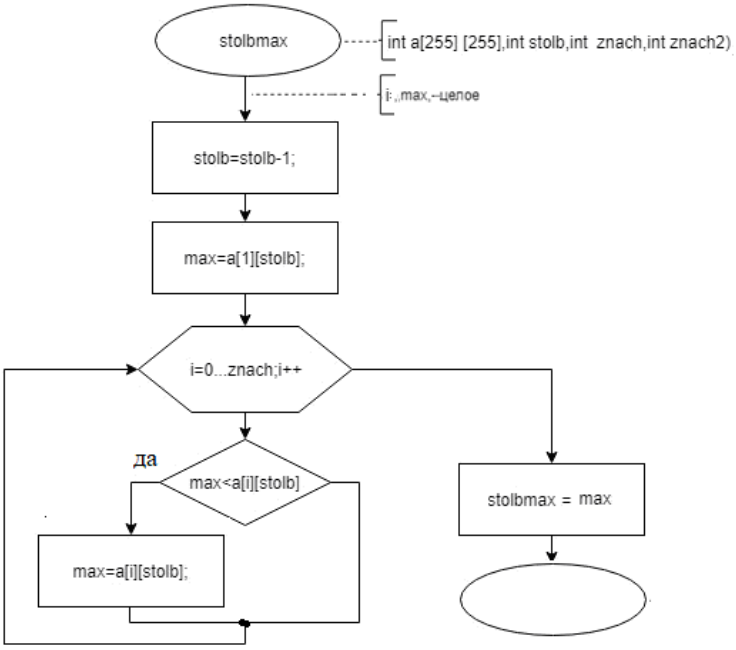
Код главной программы на языке С:

```
#include <stdio.h>
#include<string.h>
int main(void)
{int znach,stolb,max,otvet,znach2,otvet2;
 int a[255][255];
 printf("size massiv:\n");
 printf("vvedite kol-vo strok\n");
      scanf("%d,",&znach);   //Ввод длины массива
      getchar();
      printf("vvedite kol-vo stolbcov \n");
      scanf("%d",&znach2);   //Ввод ширины массива
      getchar();
printf ("vvedite 1 if s konsoli or 0 if s file: ");
      scanf("%d",&otvet); //Вводим для определения ввода массива с консоли или файла
      getchar();
if (otvet==1)
      {vvodconsl(a,znach,znach2);} // Вводим с консоли
      else
      {filevvod(a,znach,znach2);} //Вводим с файла
      printf("\n");
printf("vvedite nomer stolbca: ");
      scanf("%d",&stolb); //Вводим номер столбца
      getchar();
max=stolbmax(a,stolb,znach,znach2); // нахождение максимального элемента
      printf("\n");
vivodconsl(a,znach,znach2,stolb,max);} //Вывод результата на консоль
return 0;}
```

Разработка функции “stolbmax” – нахождения максимального элемента в столбце

```
int  stolbmax(int a[255][255],int stolb,int znach, int znach2)
```

Блок-схема функции “stolbmax”:



Структура данных:

Имя	назначение	Тип	Вид	Ограничение
a[255][255]	Массив с целыми значениями	Массив целое	Входная	Не превосходит размер 255*255
stolb	Хранит номер столбца	целое	Входная	-
znach	Длина массива	Целое	Входная	-
znach2	Ширина массива	Целое	Входная	-
i	Счетчик для перехода на следующий элемент в столбце	целое	Локальная	-
max	Максимум в заданном столбце массива	Целое	Локальная, Возвращаемая	-

Код функции на языке С:

```
int stolbmax(int a[255][255],int stolb,int znach, int znach2)
{int max, i;
stolb=stolb-1;
max=a[1][stolb];
for (i=0;i<(znach);i++)
     {if (max<a[i][stolb])
     {max=a[i][stolb];}
     }
return max;}

. . . . . .
```

Аналогично описываются все остальные функции программы. Алгоритмы вспомогательных функций могут быть описаны словесно.

Пример выполнения программы

```
size massiv:
vvedite kol-vo strok
2
vvedite kol-vo stolbcov
4
vvedite 1 if s konsoli or 0 if s file: 1
24 38 65 98
12 43 46 33
vvedite nomer stolbca: 4
massiv:
24      38      65      98
12      43      46      33
maximalnoe znachenie v stobce <4> yavlyaetsa <98>
```

Заключение

В ходе выполнения задания было написана программа нахождения максимального элемента в заданном столбце. Программа представляет собой совокупность 4 вспомогательных функций и главную программу вызова этих функций:

1. Функция ввода с консоли (`vvodconsl`);
2. Функция ввода из файла (`filevvod`);
3. Функция нахождения максимального элемента в столбце массива(`stolbmax`);
4. Функция вывода результата на консоль (`vivodconsl`).

Пример выполнения программы подтверждает правильность ее работы.

П 3. Пример тестирования фрагмента программы

Задание

Дан фрагмента кода функции, написанный на языке программирования С++. При этом необходимо:

4) Проанализировать фрагмент, переменные и их "смысл".
5) Оформить фрагмент в виде функции (заголовок, формальные параметры, результат, локальные переменные).
6) Проверить работу функций, протестировать программы и подтвердить гипотезы.

Исходный код:

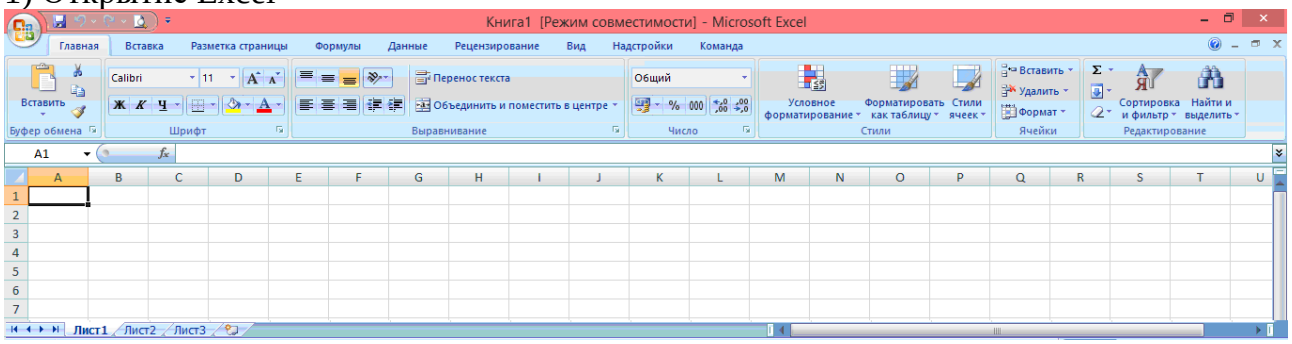
```
for (n=a, s=0; n!=0; n=n/10)
{k=n%10; s=s+k;}
```

Данный фрагмент принимает в качестве входного параметра некоторое значение переменной a. Пусть a = 13533.

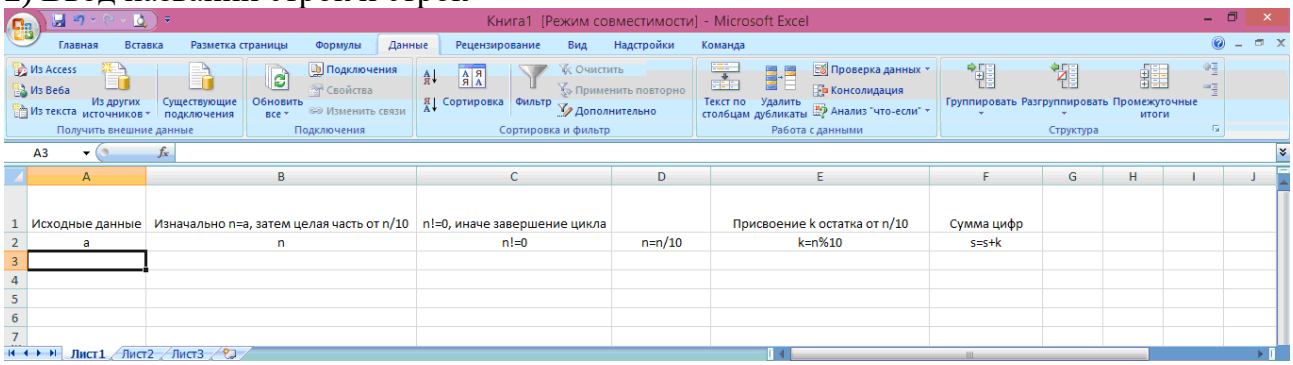
Часть 1. СОЗДАНИЕ ТРАССИРОВОЧНОЙ ТАБЛИЦЫ

Протокол тестирования кода

1) Открытие Excel

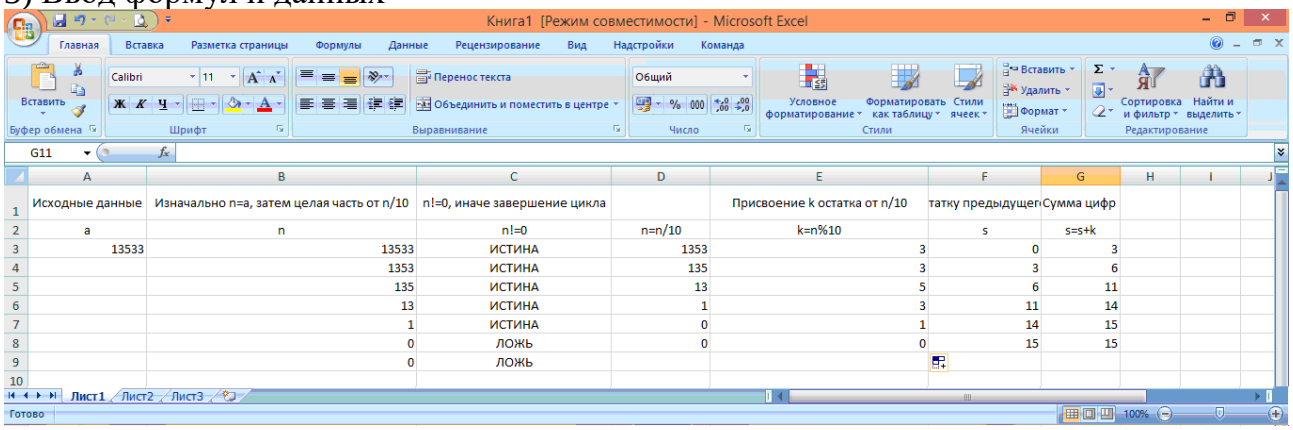


2) Ввод названий строк и строк



	A	B	C	D	E	F
1	Исходные данные	Изначально n=a, затем целая часть от n/10	n!=0, иначе завершение цикла		Присвоение k остатка от n/10	Сумма цифр
2	a	n	n!=0	n=n/10	k=n%10	s=s+k

3) Ввод формул и данных



	A	B	C	D	E	F	G
1	Исходные данные	Изначально n=a, затем целая часть от n/10	n!=0, иначе завершение цикла		Присвоение k остатка от n/10	татку предыдущег	Сумма цифр
2	a	n	n!=0	n=n/10	k=n%10	s	s=s+k
3	13533	13533	ИСТИНА	1353	3	0	3
4		1353	ИСТИНА	135	3	3	6
5		135	ИСТИНА	13	5	6	11
6		13	ИСТИНА	1	3	11	14
7		1	ИСТИНА	0	1	14	15
8		0	ЛОЖЬ	0	0	15	15
9		0	ЛОЖЬ				

Вывод: Научились анализировать код в Эксель (с помощью трассировочной таблицы).

Часть 2. ОФОРМЛЕНИЕ ФРАГМЕНТА ПРОГРАММЫ В ВИДЕ ФУНКЦИИ

Фрагмент кода функции:

```
for (n=a, s=0; n!=0; n=n/10)
{k=n%10; s=s+k;}
```

Структура данных:

Имя переменной	Назначение	Тип	Вид	Примечание
a	Число, для которого нужно вычислить сумму цифр	Целое	Входная	Ввод с клавиатуры
n	Число, без последней цифры	Целое	Локальная	Изначально равен входному параметру, затем: =n/10
k	Выделение последней цифры	Целое	Локальная	=n%10
s	Сумма цифр числа	Целое	Локальная, возвращаемая	Изначально равна 0, затем: =s+k

Код функции в разработанной программе:

//функция вычисляет сумму всех цифр числа

```
int Sum (int a) //вычисление суммы цифр числа
{
	int n, s, k;
	clog << "Start Sum\n";
	clog << "a" <<"\t"<< "n"  <<  "\t" << "k" << "\t" <<"s" << "\t"<<endl;
	clog << a <<"\t";
	for (n = a, s = 0; n != 0; n = n / 10)
	{
		k = n % 10;
		clog << n << "\t";
		clog << k << "\t";
		s = s + k;
		clog << s  << "\t"<<"\n";
		cout << endl<< "\t";
	}
	cout << endl;
	return s;
}
```

Функция называется “Sum”. Тип возвращаемого значения – int. Входной аргумент – число a типа int.

Дополнительно оформляются следующие программные модули (Приложения):

Int main

```
#include "stdafx.h"
#include "iostream"
#include "cstdlib"
#include "Sum.h"
using namespace std;

int main(int a)
{
	cin >> a;
```

```
      cout << Sum(a)<< endl;
      system ("pause");
    return 0;
}
```

Sum.h

```
int Sum (int a);
```

В главной программе (main) содержатся операторы ввода значения переменной a, вызов функции Sum и печать результата.

В приложении Sum.h содержится исходный текст функции Sum.

Как следует из приведенного текста, в функцию Sum вставлены операторы вывода, организующие контрольную (отладочную) печать, демонстрирующую работу исследуемого фрагмента.

Часть 3. РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

Вывод: Результат программы сходится с результатами таблицы тестирования.

Замечание: Необходимо проверить "работу" фрагмента программы на другом числе. Результаты трассировочной таблицы и работы функции привести в отчет!

ЗАКЛЮЧЕНИЕ: Исследуемый фрагмент кода функции служит для нахождения суммы цифр числа.

П 4. Задания на тестирование программы

№ варианта	Фрагмент программы	№ варианта	Фрагмент программы

1	int n=2; while(a%n!=0){
	n++;
	if (n==a) return 1;
	} return 0;
		2	for (int i=0; i<n; i++)
			if (A[i]<0) break;
			if (i==n) return 1;
			return 0;

3	for (int i=0; i<n; i++)
	if (A[i]<0) break;
	if (i==n) return 1;
	return 0;
		4	for (int i=0; i<n; i++)
			if (A[i]<0) break;
			if (i==n) return 1;
			return 0;

5	for (int s=0,i=0; i<n; i++)
	if (A[i]<0) { s=1; break; }
	return s;
		6	int i=0; while(A[i]>0){
			i++;
			if (i==n) return 1;
			}
			return 0;

7	for (n=a, s=0; n!=0; n=n/10) { k=n%10; s=s+k;}	8	for (n=a, s=0; n!=0; n=n/10) { k=n%10; if (k>s) s=k;}

9	for (n=a, s=0; n!=0; n=n/10)
	{ k=n%10; s=s*10+k;}
		10	for (i=0, n=a; n!=0; i++, n=n/10);
			for (A[i--]=-1, n=a; n!=0; i--, n=n/10)
			A[i]=n % 10;

11	for (j=0,a=10; a<30000; a++){
	for (n=a, s=0; n!=0; n=n/10)
	{ k=n%10; s=s+k;}
	if (a==s*s*s) A[j++]=a;
	}
		12	for (j=0,a=10; a<v; a++){
			for (n=a, s=0; n!=0; n=n/10)
			{ k=n%10; s=s*10+k;}
			if (a==s) A[j++]=a;
			}

13	for (i=0; i<n; i++){
	if (A[i]<0){
	for (j=i; j<n-1;j++)
	A[j]=A[j+1];
	n--;
	i--;
	}}
		14	for (i=1,k=0; i<n; i++)
			if (A[i]>A[k]) k=i;
			for (j=k; j<n-1;j++) A[j]=A[j+1];
			n--;

15	for (i=0,a=2; a<v; a++){
	for (n=2; n<a; n++)
	{ if (a%n==0) break; }
	if (n==a) A[i++]=a;
	}
	A[i]=0;
		16	for (i=0,a=2; a<v; a++){
			for (s=0,n=2; n<a; n++)
			if (a%n==0) { s=1; break; }
			if (s==0) A[i++]=a;
			}
			A[i]=0;

17	for (j=0,i=0; i<n ; i++){
	for (m=2; m<A[i]; m++)
	{ if (A[i]%m==0) break; }
	if (m==A[i]) B[j++]=A[i];
	}
	B[j]=0;
		18	for (j=0,i=0; i<n ; i++){
			for (s=0,m=2; m<A[i]; m++)
			if (A[i]%m==0) { s=1; break; }
			if (s==0) B[j++]=A[i];
			}
			B[j]=0;

19	for (i=0; i<n ; i++){
	for (m=2; m<A[i]; m++)
	{ if (A[i]%m==0) break; }
	if (m==A[i]) {
	for (j=i; j<n-1;j++)
	A[j]=A[j+1];
	n--;
	i--;
	}}
		20	for (j=0,i=0; i<n ; i++){
			for (s=0,m=2; m<A[i]; m++)
			if (A[i]%m==0) { s=1; break; }
			if (s==0) {
			for (j=i; j<n-1;j++)
			A[j]=A[j+1];
			n--;
			i--;
			}}

21	for (i=0; i<n-1 && val !=1; i++){
	for (m=2; val % m !=0; m++);
	val /= m; A[i] = m;
	}
	A[i] = 0;
		22	v=A[0]+1;
			do { v--;
			for (i=0,s=0;i<n;i++)
			if (A[i]%v!=0) { s=1; break; }
			} while(s==1);

23	for (i=0; i<n; i++)
	if (A[i]%v!=0) { v--; i=-1; }
			24	for (i=0,a=2; a<v && i<m-1 ; a++){
				for (s=0,j=0; j<i; j++)
				if (a%A[j]==0) { s=1; break; }
				if (s==0) A[i++]=a;
				}
				A[i]=0;

25	for (i=0; i<n-1; i++)
	for (j=i+1; j<n; j++)
	if (c[i]==c[j]) return i;
	return -1;
			26	for (s=0,i=0; i<n; i++){
				for (k=0,j=0; j<n; j++)
				if (c[i]==c[j]) k++;
				if (k>s) s=k,b=i;
				}

27	for (s=0,i=0; i<n-1; i++)
	if (A[i]==A[i+1]){
	for (k=2; i+k<n &&
	A[i]==A[i+k]; k++);
	if (k>s) s=k,b=i;
	}
			28	for (k=0, m=1; m <= n; k++, m = m * 2);
				return k-1; }

29	for (i=0,j=n-1; i < j; i++,j--)
	{ k = c[i]; c[i] = c[j]; c[j] = k; }
			30	for (i=0; i<n; i++){
				for (j=k1=k2=0; j<n; j++)
				if (c[i] != c[j])
				{ if (c[i] < c[j]) k1++; else k2++; }
				if (k1 == k2) return i;
				}
				return -1;

31	for (s=0, i=0; i<n-1; i++){
	for (j=i+1, m=0; j<n; j++)
	if (c[i]==c[j]) m++;
	if (m > s) s=m, b=i;
	}
			32	for (i=k=m=0; i<n-1; i++)
				if (c[i]==c[i+1]) k++;
				else {
				if (k > m) m=k, b=i-k-1;
				k=0;
				}

33	for (s=0,i=0; i<n; i++){
	if (A[i]<0) continue;
	if (A[i]==0) break;
	s=s+A[i];
	}
			34	for (s=0,i=0; i<n && A[i]>0; i++)
				s=s+A[i];

35	for (k=0,s=0,i=0; i<n && k==0; i++){
	if (A[i]<0) k=1;
	s=s+A[i];
	}
			36	for (s1=0,s2=0,i=0,j=n-1; i<=j;){
				if (s1<s2) s1+=A[i],i++;
				else s2+=A[j],j--;
				}
				return i;

37	for (s=0,i=0; i<n; i++){
	if (i%2==0) s=s+A[i];
	else s=s-A[i];
	}
			38	for(j=0;n!=0;j++){
				for (k=0,i=1; i<n; i++)
				if (A[i]<A[k]) k=i;
				B[j]=A[k];
				for (;k<n-1;k++) A[k]=A[k+1];
				n--;
				}

39	for(j=0,max=A[0];j<n;j++)
	if (A[j]>max) max=A[j];
	for(j=0;j<n;j++){
	for (k=0,i=1; i<n; i++)
	if (A[i]<A[k]) k=i;
	B[j]=A[k];
	A[k]=max+1;
	}
			40	while(n!=0){
				for (k=0,i=1; i<n; i++)
				if (A[i]<A[k]) k=i;
				c=A[k]; A[k]=A[n-1]; A[n-1]=c;
				n--;
				}

41	for (j=0,a=10; a<v; a++){
	for (s=0,n=a, s=0; n!=0; n=n/10){
	k=n%10;
	if (k!=0 && a%k!=0) { s=1;
	break; }
	}
	if (s==0) A[j++]=a; }
			42	for (i=0; i<n-1; i++)
				if (A[i]==A[i+1]){
				for (j=i; j<n-2;j++) A[j]=A[j+2];
				n=n-2;
				i--;
				}

43	for (i=0,k=-1; i<10; i++){
	if (A[i]<0) continue;
	if (k==-1) k=i;
	else
	if (A[i]<A[k]) k=i;
	}
			44	for (i=0,s=0,k=0; i<10; i++)
				if (A[i]<0) k=1;
				else {
				if (k==1) s++;
				k=0;
				}

45	for (i=0,s=0; i<10; i++) if (A[i]>s) s=A[i];		46	for (i=1,k=0; i<10; i++) if (A[i]>A[k]) k=i;

47	for (i=0,k=-1; i<10; i++)
	{ if (A[i]<0) continue;
	if (k==-1) k=i;
	else
	if (A[i]<A[k]) k=i;
	}
			48	for (i=0,k=-1; i<10; i++)
				{ if (A[i]<0) continue;
				if (k==-1 || A[i]<A[k]) k=i;
				}

49	for (i=0,s=0; i<10; i++) if (A[i]>0) s++;		50	for (i=1,s=0; i<10; i++) if (A[i]>0 && A[i-1]<0) s++;

51	for (i=1,s=0,k=0; i<10; i++){
	if (A[i-1]<A[i]) k++;
	else{
	if (k>s) s=k;
	k=0;
	}}
			52	for (i=0,s=0,k=0; i<10; i++)
				if (A[i]<0) k=1;
				else
				{ if (k==1) s++; k=0; }

53	for (s=1, i=1; i<=n; i++) s = s * i;		54	for (s=1, i=0; i<=n; i++) s = s * 2;
55	for (s=0, i=0; i<n; i++) s = s + A[i];		56	for (s=1, i=0; i<10; i++) s = s * A[i];

57	for (n=2; n<a; n++)
	{ if (a%n==0) break; }
	if (n==a) puts("Good");
			58	for (s=0,n=2; n<a; n++)
				{ if (a%n==0) s++; }
				if (s==0) puts("Good");

59	for (i=0; i<n; i++)
	if (A[i]<0) break;
	if (i==n) puts("Good");
			60	for (i=2; i<a; i++)
				if (a%i==0) break;
				if (i==a) puts("Good");

61	for (s=A[0], i=1; i < 10; i++) A[i-1] = A[i]; A[9] = s;			

П 5. Задания на обработку символьных массивов

1. `char* strcat(char* string1, char* string2)`
   Добавляет строку string2 в конец строки string1 и возвращает указатель на сцепленную строку.

2. `char* strncat(char* string1, char* string2, unsigned int n)`
   Добавляет строку string2 в конец строки string1 и возвращает указатель на сцепленную строку. Количество добавляемых символов ограничено значением n.

3. `int strcmp(char* string1, char* string2)`
   Сравнивает лексикографически строки string1 и string2 и возвращает значение, определяющее их соотношение:
   < 0, если string1 < string2,
   = 0, если string1 == string2 ,
   > 0, если string1 > string2.

4. `int strncmp(char* string1, char* string2, unsigned int n)`
   То же, что и strcmp, только сравнивается не более n первых символов.

5. `char* strcpy(char* string1, char* string2)`
   Копирует строку string2 вместе с завершающим нулевым символом на место строки string1, и возвращает указатель на строку string1.

6. `char* strncpy(char* string1, char* string2, unsigned int n)`
   Копирует не более n символов строки string2 на место строки string1, и возвращает указатель на строку string1. Если значение n меньше или равно длине строки string2, нулевой символ ('\0') не добавляется автоматически в новую строку.

7. `int strlen(char* string)`
   Возвращает значение длины строки.

8. `char* strchr(char* string, int chr)`
   Возвращает указатель на первое местонахождение символа chr в строке string. chr может быть нулевым символом ('\0'). Функция возвращает значение NULL, если символ не найден.

9. `char* strrchr(char* string, int chr)`
   Возвращает указатель на последнее вхождение символа chr в строке string.

10. `char* strpbrk(char* string1, char* string2)`

Находит первое вхождение в строке string1 любого символа из строки string2 и возвращает указатель на этот символ. В случае неудачи возвращает значение NULL (0).

11. `int strspn(char* string1, char* string2)`
Возвращает индекс символа в строке string1, который не принадлежит набору символов, задаваемому строкой string2. Завершающий нулевой символ не учитывается при поиске. Индексация, как это принято в C, начинается с нуля.

12. `int strcspn(char* string1, char* string2)`
Возвращает индекс символа в строке string1, который принадлежит набору символов, задаваемому строкой string2. Завершающий нулевой символ не учитывается при поиске. Индексация, как это принято в C, начинается с нуля.

13. `char* strtok(char* string1, char* string2)`
Строка string1 состоит из слов, разделенных разделителями, набор которых задан в строке string2. При первом обращении функция возвращает указатель на начало первого слова (начало string1). При последующих обращениях возвращаются указатели на начала следующих слов. Значение NULL (0) возвращается, когда нет больше слов. Обе строки завершаются нулевыми символами.

14. `char* strdup(char* string)`
Выделяет область памяти для копирования строки string, копирует туда данную строку и возвращает указатель на эту память.

15. `char* strset(char* string, int chr)`
Присваивает всем символам строки string новое значение, которое задается параметром chr. Возвращает указатель на измененную строку string.

16. `char* strnset(char* string, int chr, unsigned int n)`
Присваивает не более чем n первым символам строки string новое значение, которое задается параметром chr. Возвращает указатель на измененную строку string.

17. `char* strrew(char* string)`
Реверсирует строку и возвращает на нее указатель.

18. `char* strstr(char* string1, char* string2)`
Возвращает указатель на первое вхождение подстроки string2 в строке string1. В случае неудачи возвращается значение NULL (0).

19. `long strtol(char* string, char** endptr, int base)`
Возвращает значение длинного целого, заданного символьной строкой string. Эта функция прекращает чтение строки числа с первого символа, который не

может быть частью числа (это может быть и нулевой символ). Этим последним символом может быть также символ-цифра, код которого больше или равен коду '0'+base. `Endptr` - выходная переменная. Если значение `endptr` не NULL, то он указывает на указатель, значение которого является адресом символа, на котором был прекращен разбор строки.

20. `char* ltrim(char* string)`
Удаляет в строке лидирующие пробелы. Возвращает указатель на преобразованную строку.

21. `char* rtrim(char* string)`
Удаляет в строке концевые пробелы. Возвращает указатель на преобразованную строку.

22. `char* alltrim(char* string)`
Удаляет в строке лидирующие и концевые пробелы. Возвращает указатель на преобразованную строку.

23. `char* stuff(char* string, int begin, int length, char * string1)`
Заменяет в строке `string length` символов, начиная с символа с индексом `begin`, строкой `string1`. Возвращает указатель на преобразованную строку.

24. `char* replicate(char* string, int number, char * string1)`
Создает строку `string number`-кратным повторением строки `string1`. Возвращает указатель на преобразованную строку.

25. `char* strtran(char* string1, char * string2, char string3)`
В строке `string1` осуществляет замену подстрок `string2` строками `string3`. Возвращает указатель на преобразованную строку.

26. `char* substr(char* string1, int begin, int length, char* string2)`
В строке string1 выделяет подстроку, начиная с индекса begin, длиной length и записывает ее в string2. Возвращает указатель на сформированную строку.

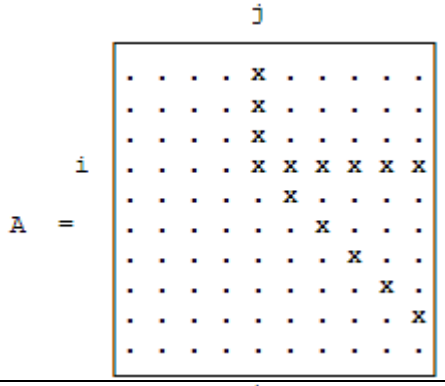
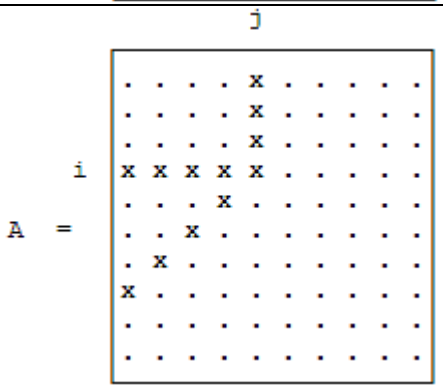
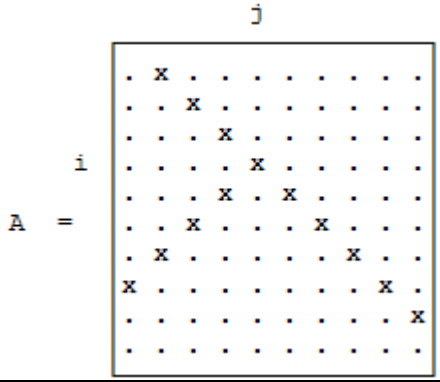
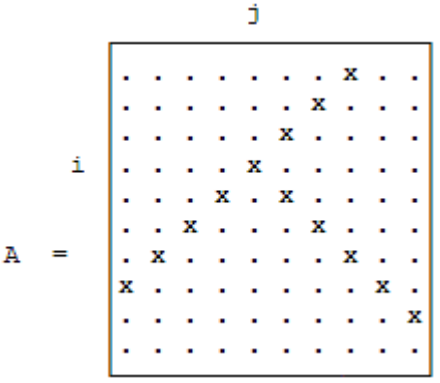
27. `char* left(char* string1, int length, char* string2)`
В строке string1 выделяет левую часть длиной length и записывает ее в string2. Возвращает указатель на сформированную строку.

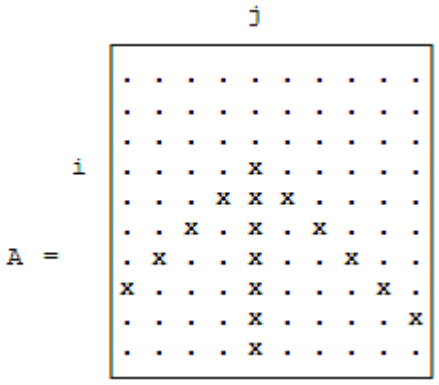
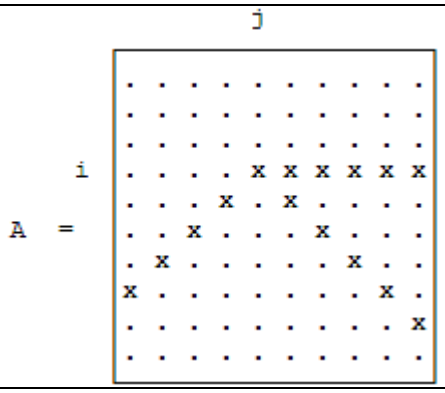
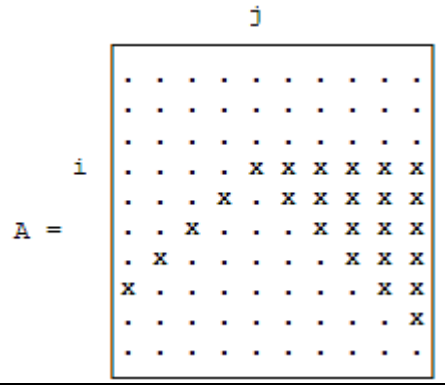
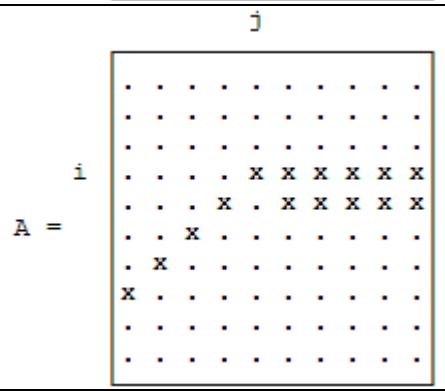
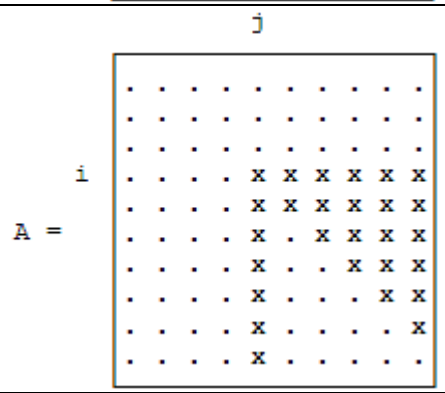
28. `char* right(char* string1, int length, char* string2)`
В строке string1 выделяет правую часть длиной length и записывает ее в string2. Возвращает указатель на сформированную строку.

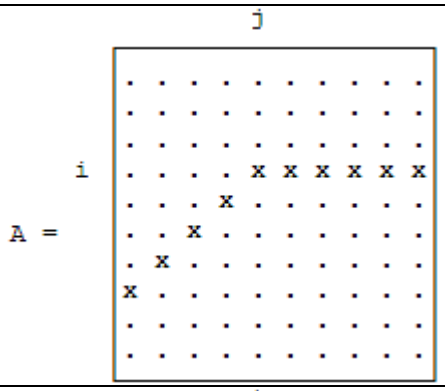
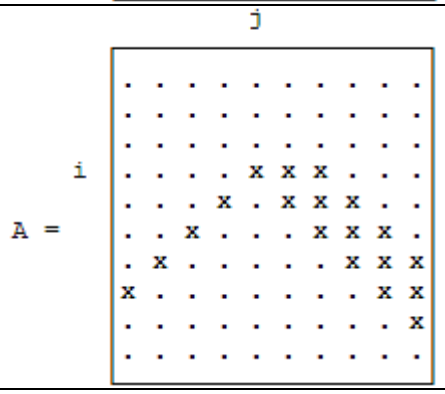
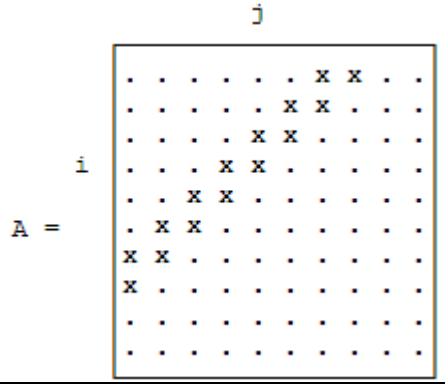
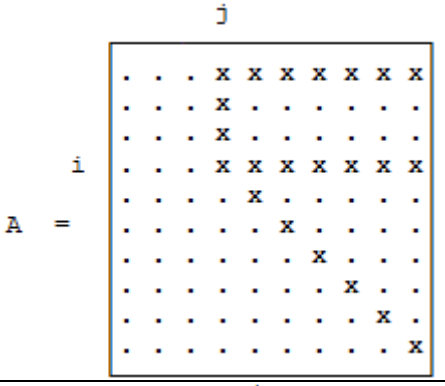
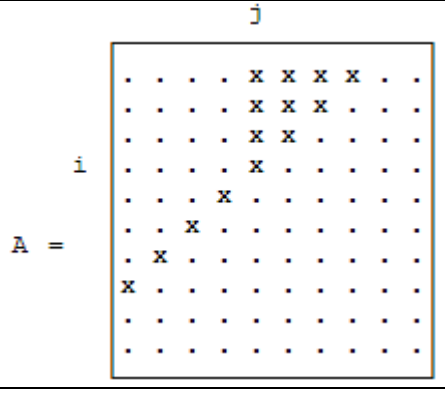
П 6. Простые задания на матрицы

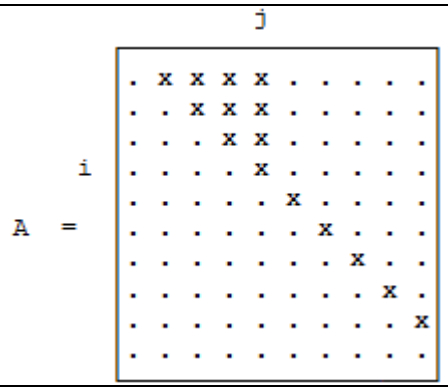
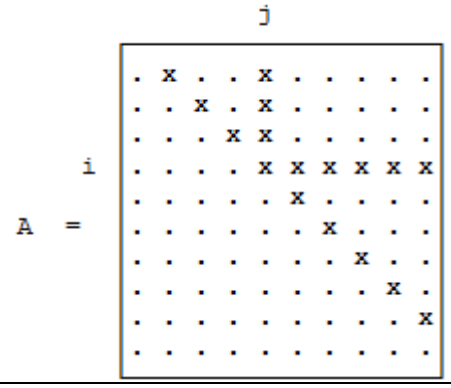
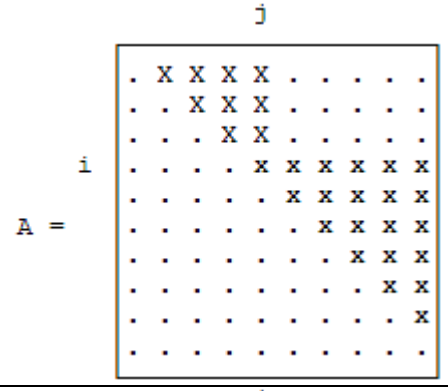
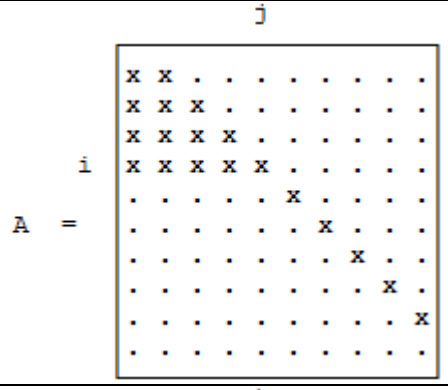
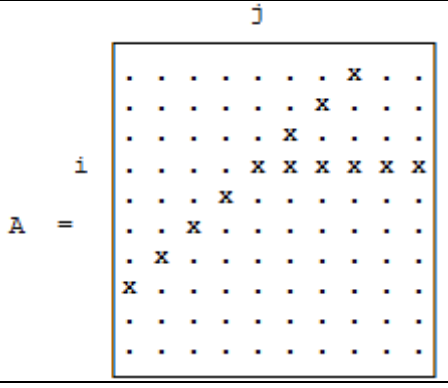
1. Вычислить сумму всех диагональных элементов
2. Вычислить сумму элементов заданной строки
3. Вычислить сумму элементов заданного столбца
4. Найти наибольший диагональный элемент
5. Найти наибольший элемент в заданной строке
6. Найти первый отрицательный элемент в заданной строке
7. Найти наибольший элемент в заданном столбце
8. Найти первый отрицательный элемент в заданном столбце
9. Переставить заданные строки
10. Переставить заданные столбцы
11. Найти количество положительных диагональных элементов
12. Определить количество нулевых диагональных элементов
13. Определить количество отрицательных элементов в заданной строке
14. Определить количество элементов, превышающих некоторое значение, в заданном столбце
15. Определить количество положительных диагональных элементов, превышающих заданное значение
16. Найти сумму абсолютных значений диагональных элементов
17. Найти сумму элементов заданных столбцов
18. Найти сумму элементов заданных строк
19. Найти сумму матриц
20. Найти произведение матриц
21. Найти произведение матрицы на вектор
22. Найти разность матриц (число элементов)
23. Найти произведение элементов заданной строки
24. Найти произведение элементов заданного столбца
25. Найти произведение диагональных элементов
26. Обнулить заданную строку
27. Обнулить заданный столбец
28. Найти среднее значение элементов в заданном столбце
29. Найти среднее значение элементов в заданной строке
30. Найти среднее значение диагональных элементов

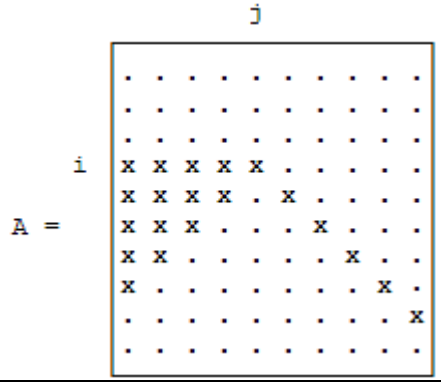
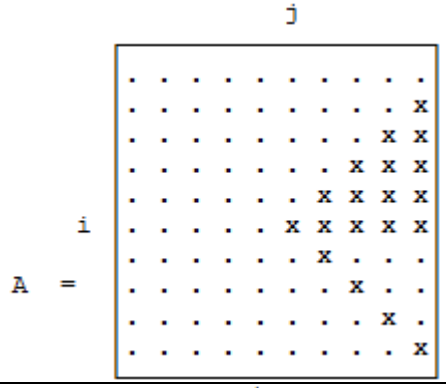
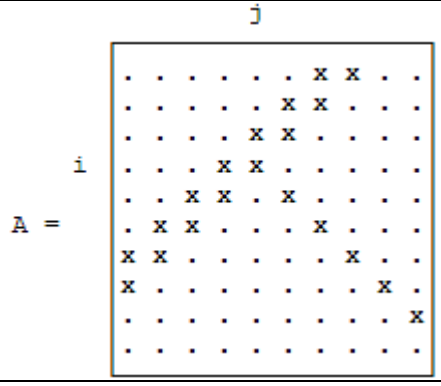
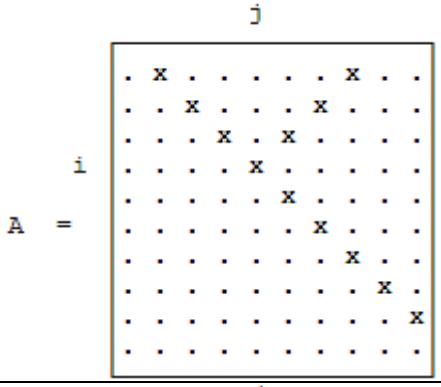
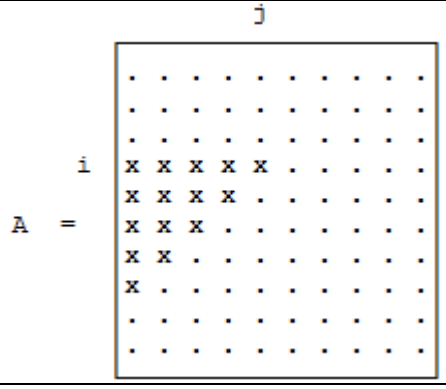
П 7. Задания на преобразование матрицы

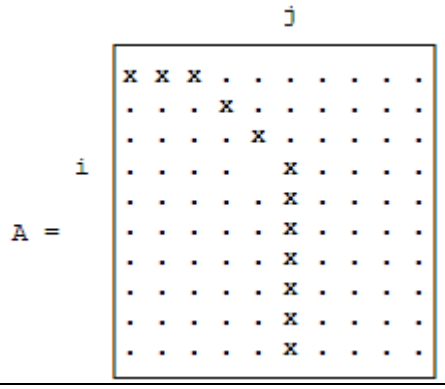
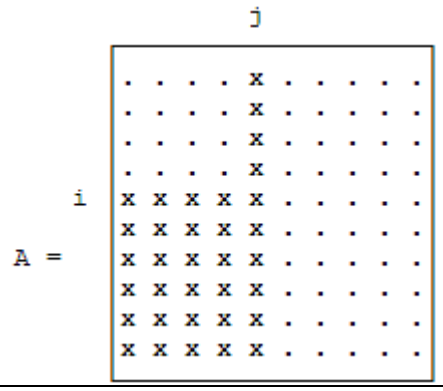
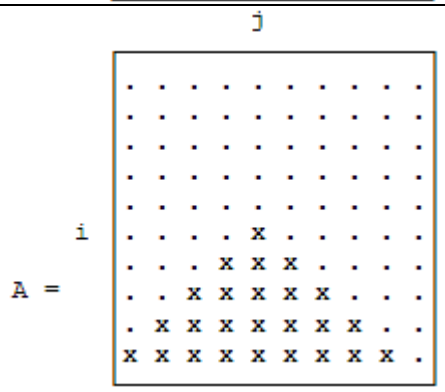
1	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 11, B(i,j) = i - j.	
2	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 10, B(i,j) = Abs(2i-3j).	
3	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = Abs(i-2j).	
4	Элемент A(i,j) обновленной матрицы есть РАЗНОСТЬ минимального и максимального из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 11, B(i,j) = i-2*j+i*j.	

5	Элемент A(i,j) обновленной матрицы есть СУММА минимального и максимального из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = Abs(2*i-j-i*j).	
6	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 12, B(i,j) = 2*i-3*j.	
7	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 10, B(i,j) = 2*i-2*j.	
8	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 10, B(i,j) = i+j.	
9	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 12, B(i,j) = 3*i-j.	

10	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = i-j.	
11	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = i-6*j+i*i.	
12	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = 5i+4j.	
13	Элемент A(i,j) обновленной матрицы есть МАКСИМАЛЬНЫЙ ИЗ элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 11, B(i,j) = i-j.	
14	Элемент A(i,j) обновленной матрицы есть РАЗНОСТЬ минимального и максимального из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 11, B(i,j) = i-2*j.	

15	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = 2*i-j.	
16	Элемент A(i,j) обновленной матрицы есть МИНИМАЛЬНЫЙ из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = 2*i+j.	
17	Элемент A(i,j) обновленной матрицы есть ПОЛУСУММА минимального и максимального элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 12, B(i,j) = i+2j.	
18	Элемент A(i,j) обновленной матрицы есть МИНИМАЛЬНЫЙ из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = i+4j.	
19	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 12, B(i,j) = 2i-3j.	

20	Элемент A(i,j) обновленной матрицы есть МИНИМАЛЬНЫЙ из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = i-j+i*i.	
21	Элемент A(i,j) обновленной матрицы есть МИНИМАЛЬНЫЙ из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 8, B(i,j) = 2i-j+i*i.	
22	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 10, B(i,j) = 2i-j.	
23	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 10, B(i,j) = i+4j.	
24	Элемент A(i,j) обновленной матрицы есть МАКСИМАЛЬНЫЙ из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 10, B(i,j) = i-j.	

25	Элемент A(i,j) обновленной матрицы есть МАКСИМАЛЬНЫЙ из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 8, B(i,j) = 2*i-3*j.	
26	Элемент A(i,j) обновленной матрицы есть СУММА элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 7, B(i,j) = Abs(3i-j).	
27	Элемент A(i,j) обновленной матрицы есть ПОЛУСУММА минимального и максимального из элементов исходной матрицы, помеченных символами "X" (смотри рисунок справа). Исходная матрица задается в главной программе как квадратная матрица B размерности 9, B(i,j) = Abs(i-2*j).	

П 8. Задания на вычисление суммы ряда

1	$y=\sum_{n=0}^{\infty}\frac{(x+a)^n}{n!}$	e^{x+a} $-\infty<x<\infty$	14	$y=\sum_{n=1}^{\infty}\frac{(-1)^{n-1}2^{2n-1}x^{2n}}{(2n)!}$	$-1\le x\le 1$
2	$y=\sum_{n=1}^{\infty}nx^{n-1}$	$(1-x)^{-2}$ $-1<x<1$	15	$y=\sum_{n=1}^{\infty}\frac{(-1)^{n+1}x^n}{n}$	$\ln(1+x)$ $-1<x<1$
3	$y=\sum_{n=1}^{\infty}\frac{x^n}{n}$	$-\ln(1-x)$ $-1<x<1$	16	$y=\sum_{n=1}^{\infty}\frac{a(a-1)\ldots(a-n+1)x^n}{n!}$	$(1+x)^a$ $-1\le x\le 1,$ если $a\ge 0$
4	$y=\sum_{n=0}^{\infty}\frac{x^n}{n+1}$	$-x^{-1}\ln(1-x)$ $-1<x<1$	17	$y=\sum_{n=0}^{\infty}(-1)^n x^{2n}$	$(1+x^2)^{-1}$ $-1<x<1$
5	$y=\sum_{n=0}^{\infty}\frac{(-1)^n(x-2)^n}{n^2}$	$1\le x\le 3$	18	$y=\sum_{n=0}^{\infty}\frac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}}$	$0<x\le 1.$
6	$y=\sum_{n=1}^{\infty}\frac{(-1)^{n-1}x^n}{n}$	$\ln(1+x)$ $-1<x<1$	10	$y=\sum_{n=1}^{\infty}\frac{(x-1)^n}{nx^n}$	$\ln(x)$ $0.5<x$
7	$y=\sum_{n=0}^{\infty}\frac{5^n x^n}{(n+1)^2}$	$-0.2\le x\le 0.2$	20	$y=\sum_{n=1}^{\infty}\frac{(-1)^{n-1}x^n}{n\cdot 2^n}$	$\ln(1+0.5x)$ $-2<x\le 2$
8	$y=\sum_{n=0}^{\infty}\frac{x^n}{2^n}$	$2(2-x)^{-1}$ $-2<x<2$	21	$y=\sum_{n=1}^{\infty}\frac{(-1)^{n+1}(x-3)^{n-1}}{3^n}$	x^{-1} $0<x<6$
9	$y=\sum_{n=0}^{\infty}\frac{x^n}{n!}$	e^x $-\infty<x<\infty$	22	$y=\sum_{n=1}^{\infty}\frac{(-1)^n x^n(\ln x)^n}{n!}$	$0<x\le 1.$
10	$y=\sum_{n=1}^{\infty}\frac{x^{2n}}{4^n}$	$2(2-x^2)^{-1}$ $-2<x<2$	23	$y=\sum_{n=0}^{\infty}\frac{(-1)^{n+1}}{(2n+1)x^{2n+1}}$	$-arctg(x^{-1})$ $1<\lvert x\rvert$
11	$y=\sum_{n=0}^{\infty}(-1)^n x^n$	$(1+x)^{-1}$ $-1<x<1$	24	$y=\sum_{n=2}^{\infty}\frac{(-1)^{n-1}2^{n-1}x^n}{(n-1)!}$	$-1\le x\le 1.$
12	$y=\sum_{n=1}^{\infty}\frac{(-1)^{n+1}x^{2n-1}}{(2n-1)!}$	$\sin(x)$ $-\infty<x<\infty$	25	$y=\sum_{n=1}^{\infty}\frac{(-1)^{n+1}x^{2n-1}}{2^{2n-1}(2n-1)!}$	$\sin(0.5x)$ $-\infty<x<\infty$
13	$y=\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(2n)!}$	$\cos(x)$ $-\infty<x<\infty$	26	$y=\sum_{n=1}^{\infty}\frac{(-1)^n 2^{2n}x^{2n}}{(2n)!}$	$\cos(2x)$ $-\infty<x<\infty$
			27	$y=\sum_{n=1}^{\infty}\frac{(-1)^{n-1}x^{2n-1}}{(2n-1)}$	$arctg(x)$ $-1\le x\le 1$

П 9. Задания на списки

1. Сведения об автомобиле состоят из фамилии и инициалов владельца (на русском языке), марки автомобиля, номерного знака и года выпуска автомобиля. Даны два файла, содержащие сведения об автомобилях (возможны пересечения).
   Выделить владельцев, имеющих более одного автомобиля. Полученные сведения сгруппировать в алфавитном порядке фамилий владельцев в отдельный файл.

2. Сведения об автомобиле состоят из фамилии и инициалов владельца (на русском языке), марки автомобиля, номерного знака и года выпуска автомобиля. Даны два файла, содержащие сведения об автомобилях (возможны пересечения).
   Выделить автомобили, принадлежащие нескольким владельцам. Полученные сведения сгруппировать в отдельный файл в алфавитном порядке марок автомобилей и их номерных знаков.

3. Сведения об автомобиле состоят из фамилии и инициалов владельца (на русском языке), марки автомобиля, номерного знака и года выпуска автомобиля. Даны два файла, содержащие сведения об автомобилях (возможны пересечения).
   Получить сведения об автомобилях, номерной знак которых содержит заданную последовательность символов. Сгруппировать найденные автомобили по маркам.

4. Сведения об автомобиле состоят из фамилии и инициалов владельца (на русском языке), марки автомобиля, номерного знака и года выпуска автомобиля. Даны два файла, содержащие сведения об автомобилях (возможны пересечения).
   Выделить автомобили, номерной знак которых содержит заданную последовательность символов. Сгруппировать найденные автомобили в алфавитном порядке владельцев.

5. Сведения об автомобиле состоят из фамилии и инициалов владельца (на русском языке), марки автомобиля, номерного знака и года выпуска автомобиля. Даны два файла, содержащие сведения об автомобилях (возможны пересечения).
   Выделить автомобили, фамилии владельцев которых содержат заданную последовательность букв. Сгруппировать найденные автомобили по годам выпуска, а внутри года - в алфавитном порядке владельцев.

6. Сведения об автомобиле состоят из фамилии и инициалов владельца (на русском языке), марки автомобиля, номерного знака и года выпуска автомоби-

ля. Даны два файла, содержащие сведения об автомобилях (возможны пересечения).
Выделить автомобили, фамилии владельцев которых содержат заданную последовательность букв. Сгруппировать их по маркам, внутри марки - по году выпуска, а внутри года - в алфавитном порядке владельцев.

7. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
Получить список некоторого (по запросу) класса в алфавитном порядке с указанием только первой буквы имени ученика и его среднего балла. После буквы имени поставить точку.

8. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
Выделить однофамильцев из общего списка. Полученные сведения сгруппировать по годам обучения, а внутри каждого года - в алфавитном порядке фамилий и имен учеников. В случае, если однофамильцы разного возраста, отнести их в группу, соответствующую году обучения старшего однофамильца.

9. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
Сгруппировать сведения по годам обучения, а внутри года - в алфавитном порядке, с указанием среднего балла каждого из учеников.

10. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
Сгруппировать в файл отличников, хорошистов, троечников либо двоечников (по запросу). Рассортировать их по годам обучения, а внутри года - в алфавитном порядке.

11. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
Выделить учеников с одинаковыми именами в параллельных (по запросу) классах. Рассортировать в алфавитном порядке.

12. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
    Найти всех учеников по нескольким первым буквам имени. Рассортировать их по успеваемости и в алфавитном порядке.

13. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
    Выяснить, имеются ли в школе ученики, у которых совпадают имена и фамилии. Рассортировать их в алфавитном порядке.

14. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы),оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
    Получить список отличников и хорошистов, фамилии которых содержат заданную последовательность букв. Напечатать их в алфавитном порядке.

15. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
    Выяснить среднюю успеваемость каждого класса. Напечатать список учеников школы по классам в порядке убывания этого показателя. Внутри класса упорядочить по алфавиту.

16. Сведения об ученике состоят из его фамилии, имени, названия класса (года обучения и буквы), оценок обучения за последнюю четверть (возможно отсутствие некоторых оценок). Дан файл, содержащий сведения об учениках школы.
    Получить список отличников и хорошистов, фамилии которых начинаются на заданную последовательность букв. Упорядочить его в соответствии со средней успеваемостью класса, в котором они учатся. При совпадении данного показателя упорядочить по алфавиту.

17. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
    Выбрать студентов, фамилии которых начинаются на заданную последовательность букв. Отсортировать по группам и внутри группы в алфавитном порядке.

18. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
Сгруппировать сведения по годам обучения, а внутри года - в алфавитном порядке.

19. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
Сгруппировать в файл студентов какой-либо группы (по запросу). Рассортировать их по возрасту (с точностью до года), а внутри года - в алфавитном порядке.

20. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
Выделить студентов с одинаковыми именами в выбранном направлении обучения (по запросу). Рассортировать в алфавитном порядке.

21. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
Найти всех студентов по нескольким первым буквам имени. Рассортировать их по группам и в алфавитном порядке (внутри каждой группы).

22. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
Выяснить, имеются ли в институте студенты, у которых совпадают имена и фамилии. Рассортировать их в алфавитном порядке.

23. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
Получить список бакалавров, магистров или специалистов (по запросу), фамилии которых содержат заданную последовательность букв. Напечатать их в алфавитном порядке.

24. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
Выяснить количество студентов в каждой учебной группе. Напечатать список студентов института по группам в порядке убывания этого показателя. Внутри группы упорядочить по алфавиту.

25. Сведения о студенте состоят из его фамилии, имени, даты рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
    Получить список магистров, фамилии которых начинаются на заданную последовательность букв. Упорядочить его в соответствии с номером группы. При совпадении данного показателя упорядочить по алфавиту.

26. Сведения о студенте состоят из его фамилии, имени, дате рождения, наименования группы (по правилам обозначения групп в СПбПУ). Дан файл, содержащий сведения о студентах института.
    Выбрать студентов определенного (по запросу) года обучения. Отсортировать их по группам, а внутри группы – по возрасту.

П 10. Задания на курсовую работу

Предлагаемые варианты задач (указана ссылка на ”Основы вычислительной математики” Б.П.Демидовича и И.А.Марона):

1. Решение системы алгебраических уравнений методом Гаусса. Применить уточнение корней.
   Глава 8. Параграф 3,4.

2. Вычисление обратной матрицы методом Гаусса.
   Глава 8. Параграф 7.

3. Решение системы алгебраических уравнений методом главных элементов.
   Глава 8. Параграф 5.

4. Решение произвольной системы линейных алгебраических уравнений методом Зейделя.
   Глава 8. Параграф 12,13.

5. Вычисление обратной матрицы с использованием тождества Гамильтона-Кели.
   Глава 11. Параграф 2.
   Глава 12. Параграф 8,16.

6. Решение системы уравнений по схеме Халецкого.
   Глава 8. Параграф 9.

7. Нахождение первого собственного значения действительной матрицы.
   Глава 12. Параграф 12.

8. Нахождение собственных элементов положительно определенной симметрической матрицы.
   Глава 12. Параграф 15.

9. Нахождение всех корней алгебраического полинома. (Метод Ньютона; деление полиномов)
   Глава 4. Параграф 5,11.

10. Нахождение всех корней алгебраического полинома. Метод Лобачевского-Греффе.
    Глава 5.

11. Решение системы линейных алгебраических уравнений методом ортогонализации.
    Глава 10. Параграф 8.

12. Нахождение корней алгебраического полинома методом Бернулли. Использовать деление полиномов.
    Глава 5. Параграф 13.

13. Решение системы линейных уравнений методом скорейшего спуска.
Глава 13. Параграф 13.

14. Получение характеристического полинома матрицы. Метод Леверье.
Глава 12. Параграф 8.

15. Получение характеристического полинома матрицы. Метод А.Н. Крылова.
Глава 12. Параграф 6. Глава 8. Параграф 3.

Далее ссылка на книгу А.А. Самарского «Введение в численные методы». М. Наука.-1982.

16. Решение системы линейных алгебраических уравнений методом верхней релаксации.
Глава 3. Параграф 3.

17. Решение системы линейных алгебраических уравнений методом минимальных невязок.
Глава 3. Параграф 6.

Для выполнения последующих заданий необходимо изучить материал, приведенный ниже.

18. Решение системы уравнений методом Некрасова.

19. Вычисление определителя матрицы

20. Определение факта устойчивости дифференциального уравнения

21. Исследование управляемости системы автоматического управления

22. Исследование наблюдаемости динамической системы

23. Решение матричного уравнения Сильвестра

24. Решение матричного уравнения Ляпунова

Вычисление определителя матрицы

Вычислить определитель произвольной матрицы путем приведения ее к треугольной.

Треугольная матрица – это такая квадратная матрица, у которой ниже главной диагонали все элементы нулевые.

Свойства определителя:

- если преобразовывать столбцы или строки незначительными действиями (умножение элементов строки на число, прибавление к элементам строки матрицы другой строки), то это не влияет на значение определителя;
- если поменять местами строки и столбцы, то знак поменяется на противоположный;
- определитель треугольной матрицы представляет собой произведение элементов, которые расположены на главной диагонали.

Привести матрицу к треугольному виду можно прямым ходом метода Гаусса Глава 8. Параграф 3,4. (Демидович и Марон)

Метод Некрасова решения системы линейных алгебраических уравнений

Пусть СЛАУ задана в виде

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \end{aligned} \qquad (1)$$

Для получения решения каждое уравнение системы (1) разрешим относительно соответствующей переменной.

$$\begin{aligned} x_1 &= -\frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 - \ldots - \frac{a_{1n}}{a_{11}}x_n + \frac{b_1}{a_{11}} \\ x_2 &= -\frac{a_{21}}{a_{22}}x_1 - \frac{a_{23}}{a_{22}}x_3 - \ldots - \frac{a_{2n}}{a_{22}}x_n + \frac{b_2}{a_{22}} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ x_n &= -\frac{a_{n1}}{a_{nn}}x_1 - \frac{a_{n2}}{a_{nn}}x_2 - \ldots - \frac{a_{nn-1}}{a_{nn}}x_{n-1} + \frac{b_n}{a_{nn}} \end{aligned} \qquad (2)$$

Систему (2) можно записать компактно

$$x_i = -\sum_{j=1,\, j\neq i}^{n} \frac{a_{ij}}{a_{ii}}x_j + \frac{b_i}{a_{ii}},\ i = \overline{1,n}. \qquad (3)$$

Во-вторых, систему (3) будем решать стационарным методом Зейделя по формулам:

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}}x_j^{k+1} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}}x_j^{k} + \frac{b_i}{a_{ii}},\ i = \overline{1,n},\ k = 0,1,2,\ldots. \qquad (4)$$

Алгоритм численной реализации метода Ньютона для решения системы (1) по формулам (4) может быть таким.

1. Выберем $X^0 = (x_1^0, x_2^0, \ldots, x_n^0)$, например, $x_i^0 = \frac{b_i}{a_{ii}},\ i = \overline{1,n}$
2. Положим $k = 0$.
3. Для всех $i = \overline{1,n}$ вычислим $x_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}}x_j^{k+1} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}}x_j^{k} + \frac{b_i}{a_{ii}}$.
4. Для всех $i = \overline{1,n}$ проверим условия $\left|x_i^{k+1} - x_i^k\right| \le \varepsilon$.

5. Если все условия в п.4 будут выполнены, то за приближенное решение системы (1) выберем либо $X^{\kappa} = (x_1^{\kappa}, x_2^{\kappa}, ..., x_n^{\kappa})$, либо $X^{\kappa+1} = (x_1^{\kappa+1}, x_2^{\kappa+1}, ..., x_n^{\kappa+1})$ и закончим вычисления. Если хотя бы одно условие в п.4 не будет выполнено, перейдем к п.6.

6. Положим $k = k + 1$и перейдем к п.3.

Достаточным условием сходимости метода Некрасова является требование, чтобы матрица А, элементами которой являются коэффициенты при неизвестных в системе (1), была симметричной и положительно определенной.

Определение устойчивости решения дифференциального уравнения по характеристическому полиному

Алгебраический критерий устойчивости Рауса. 1877г.

Раус выразил его в форме таблицы. Элементами первой строки являются четные коэффициенты характеристического уравнения (полинома), начиная с a_0. Элементы второй - нечетные коэффициенты, начиная с a_1. Элементы последующих строк вычисляются по приведенным формулам.

Итак, характеристический полином $D(p) = a_0 p^n + a_1 p^{n-1} + ... + a_{n-1} p + a_n$, где $a_0 > 0$.

	$c_{11} = a_0$	$c_{12} = a_2$	$c_{13} = a_4$	$c_{14} = a_6$
	$c_{21} = a_1$	$c_{22} = a_3$	$c_{23} = a_5$	$c_{24} = a_7$
$\lambda_3 = \frac{c_{11}}{c_{21}}$	$c_{31} = c_{12} - c_{22}\lambda_3$	$c_{32} = c_{13} - c_{23}\lambda_3$	$c_{33} = c_{14} - c_{24}\lambda_3$	c_{34}=…
$\lambda_4 = \frac{c_{21}}{c_{31}}$	$c_{41} = c_{22} - c_{32}\lambda_4$	$c_{42} = c_{23} - c_{33}\lambda_4$	c_{43}=…	c_{44}=…
$\lambda_5 = \frac{c_{31}}{c_{41}}$	$c_{51} = c_{32} - c_{42}\lambda_4$	c_{52}=…	c_{53}=…	c_{54}=…
$\lambda_6 = \frac{c_{41}}{c_{51}}$	и так далее	…	…	…

В данной таблице должна быть n+1 строка. Элементы, находящиеся в правой части матрицы, заполняются нулями. Причем, чем ниже находится строка матрицы по отношению к ее началу, тем больше в ней нулевых элементов.

Ниже приведены формулы, используемые при заполнении таблицы.

$$\lambda_i = C_{i-2,1} / c_{i-1,1}; \quad c_{i,k} = c_{i-2,k+1} - c_{i-1,k+1}\lambda_i; \quad k \in [1, \ n/2];$$

. . . . . .

$$\lambda_{n+1} = C_{n-1,1} / c_{n,1}; \quad c_{n+1,1} = c_{n-1,2} - c_{n,2}\lambda_{n+1};$$

Если все элементы первого (выделенного) столбца таблицы Рауса положительны (одного знака), то решение дифференциального уравнения (диффе-

ренциальное уравнение) устойчиво. Если хотя бы один элемент отрицателен, то дифференциальное уравнение неустойчиво. При этом число перемен знака равно числу правых корней характеристического уравнения.

Если один из элементов первого столбца равен нулю, то решение находится на границе устойчивости, а характеристическое уравнение имеет пару мнимых корней.

В случае, когда последний элемент равен нулю, то корень уравнения – нулевой вещественный. При нескольких нулевых последних элементах первого столбца таблицы имеется соответствующее количество нулевых корней характеристического уравнения.

Исследование управляемости системы автоматического управления

Математическая модель динамической системы в пространстве состояний описывается матричными уравнениями:

$$\begin{cases} \dot{X} = AX + BU, \\ Y = CX. \end{cases}$$

Здесь матрица A - квадратная, B и C = прямоугольные.
X - вектор состояния системы, U - вектор входных (управляющих) воздействий, Y - вектор выходных координат.

1. Понятие управляемости связано с возможностью приведения системы в заданное состояние с помощью входных или управляющих воздействий.
2. Управляемость определяет возможность управления со стороны входа всеми компонентами вектора состояния динамической системы.
3. Под **управляемостью** понимают существование управления *u(t)*, достаточного для перевода системы за конечное время $t_к - t_o$ из состояния X_o в $X_к$, где X_o и $X_к$ – две произвольные точки пространства состояний размерности *n*.

Определение 1. Система называется **управляемой (**вполне управляемой), если выбором управляющего воздействия $u(t)$ на интервале времени $[t_0, t_k]$ можно перевести ее из любого начального состояния $X(t_0)$ в произвольное заранее заданное конечное состояние $X(t_k)$.

Управляемость системы - важное свойство самой системы. Очевидно, что управляемость определяется матрицами *A* и *B*.

Определение 2. Система будет управляемой тогда и только тогда, когда матрица управляемости $P = [\ B \mid AB \mid \ldots \mid A^{n-1}B\ ]$ имеет ранг n, равный размерности пространства состояний ($rankP$ = n), или числу компонентов вектора X, или размерности квадратной матрицы A.

Рангом матрицы называется максимальное число линейно независимых строк (столбцов).

Определить ранг можно в ходе прямого метода Гаусса.
Глава 8. Параграф 3,4. (Демидович и Марон)

Исследование наблюдаемости системы автоматического управления

Математическая модель динамической системы в пространстве состояний описывается матричными уравнениями:

$$\begin{cases} \dot{X} = AX + BU, \\ Y = CX. \end{cases}$$

Здесь матрица A - квадратная, B и C = прямоугольные.
X - вектор состояния системы, U - вектор входных (управляющих) воздействий, Y - вектор выходных координат.

Наблюдаемость - это свойство динамической системы.

Понятие наблюдаемости связано с возможностью определения переменных состояния по результатам измерения выходных переменных.

Определение 1. Система называется **наблюдаемой** (вполне наблюдаемой), если по реакции $y(t)$ на выходе системы на интервале времени $[t_0, t_k]$ при заданном управляющем воздействии $u(t)$ можно определить начальное состояние $X(t_0)$.

Определение 2. Система будет наблюдаема тогда и только тогда, когда матрица наблюдаемости N имеет ранг n, равный размерности пространства состояния состояний ($rankP$ = n), или числу компонентов вектора X, или размерности квадратной матрицы A. В этом случае говорят, что пара (A,C) – наблюдаема.

$$N = \begin{pmatrix} C \\ CA \\ \cdots \\ CA^{n-1} \end{pmatrix}$$

Очевидно, что наблюдаемость определяется свойствами матриц A и C.

Рангом матрицы называется максимальное число линейно независимых строк (столбцов).

Определить ранг можно в ходе прямого метода Гаусса.

Глава 8. Параграф 3,4. (Демидович и Марон)

Решение матричного уравнения Сильвестра

Матричное уравнение вида:

$$AX - XB = C,$$

определяемое квадратными матрицами A, B, C и X размерности n, называется уравнением Сильвестра, разрешаемое относительно неизвестной матрицы X.

Решение этого уравнения имеет вид:

$$X = -\left(C_n + p_1 C_{n-1} + \ldots + p_{n-1} C_1\right) \cdot \left(P_A(B)\right)^{-1},$$

где $P_A(\lambda)$ - характеристический полином матрицы A:

$$P_A(\lambda) = \lambda^n + p_1 \lambda^{n-1} + \ldots + p_{n-1}\lambda + p_n,$$

а матрицы C_n определяются по формуле:

$$C_n = \sum_{k=1}^{n} A^{n-k} C B^{k-1}.$$

Решение матричного уравнения Ляпунова

Матричное уравнение вида:

$$AX + XA^T = -Q,$$

определяемое квадратными матрицами A и Q размерности n, причем матрица Q симметричная, называется уравнением Ляпунова, разрешаемое относительно неизвестной симметричной матрицы X.

Уравнение Ляпунова является частным случаем уравнения Сильвестра $AX - XB = C$ (где $B = -A^T$, $C = -Q$), следовательно, для его решения можно воспользоваться соответствующими формулами (см. выше).

То есть решение уравнения Ляпунова будет иметь вид:

$$X = \left(C_n + p_1 C_{n-1} + \ldots + p_{n-1} C_1\right) \cdot \left(P_A\left(-A^T\right)\right)^{-1},$$

где $P_A(\lambda)$ - характеристический полином матрицы A:

$$P_A(\lambda) = \lambda^n + p_1 \lambda^{n-1} + \ldots + p_{n-1}\lambda + p_n,$$

а матрицы C_n определяются по формуле:

$$C_n = \sum_{k=1}^{n} A^{n-k} Q\left(-A^T\right)^{k-1}.$$

П 11. Создание оконных приложений с помощью Windows API

Практически в любой операционной системе возможны 2 типа структур программ:

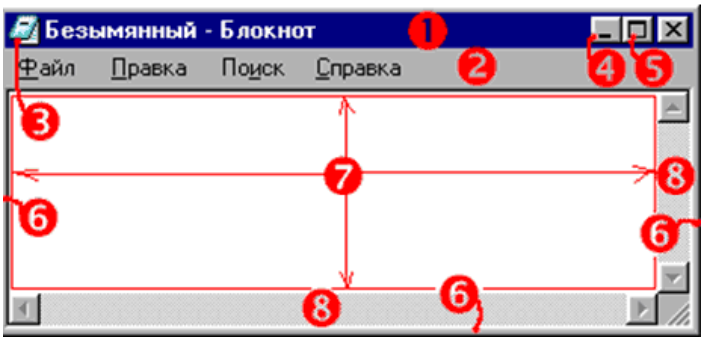
- консольная, или безоконная структура,
- оконная (каркасная) структура.

Безоконные (**консольные**) приложения представляет собой программу, работающую в текстовом режиме, и обеспечивается специальными функциями операционной системы. Консольные приложения представляют собой систему средств взаимодействия пользователя с компьютером, основанную на использовании текстового (буквенно-цифрового) режима дисплея. Они очень компактны не только в откомпилированном виде, но и в текстовом варианте.

Оконные (каркасные) приложения строятся на базе специального набора функций API (Application Programming Interfaces — интерфейс программного приложения). Все взаимодействие с внешними устройствами и ресурсами операционной системы происходит посредством таких функций. Windows API в настоящее время поддерживает свыше тысячи вызовов функций, которые можно использовать в приложениях.

Любая программа для Windows имеет окно — прямоугольную область на экране, в котором приложение отображает информацию и получает реакцию от пользователя. Окно идентифицируется заголовком. Большинство функций программы запускается посредством меню. Слишком большой для экрана объем информации может быть просмотрен с помощью полос прокрутки. Некоторые пункты меню вызывают появление окон диалога, в которые пользователь вводит дополнительную информацию.

Окно содержит элементы управления: кнопки, списки, окна редактирования и др.



Основными элементами окна являются:

1 — строка заголовка **title bar**
2 — строка меню **menu bar**
3 — системное меню **system menu**
4 — кнопка сворачивания окна **minimize box**
5 — кнопка разворачивания окна **maximize box**
6 — рамка изменения размеров **sizing border**
7 — клиентская область **client area**
8 — горизонтальная и вертикальная полосы прокрутки **scroll bars**

Для работы с оконными приложениями используется стартовая функция:

```
int WINAPI WinMain (HINSTANCE hInst, HINSTANCE hpi, LPSTR cmdline, int ss)
```

- `hInst` - дескриптор для данного экземпляра программы,
- `hpi` - в Win32 не используется (всегда NULL),
- `cmdline` - командная строка,
- `ss` - код состояния главного окна.

Структура оконного приложения

Оконные приложения строятся по принципам событийно-управляемого программирования (event-driven programming) — стиля программирования, при котором поведение компонента системы определяется набором возможных внешних событий и ответных реакций компонента на них. Такими компонентами в Windows являются окна. С каждым окном связана определенная функция обработки событий – оконная функция. События для окон называются сообщениями. Сообщение относится к тому или иному типу, идентифицируемому определенным кодом (32-битным целым числом), и сопровождается парой 32-битных параметров (WPARAM и LPARAM), интерпретация которых зависит от типа сообщения.

Задача любого оконного приложения — создать главное окно и сообщить Windows функцию обработки событий для этого окна. Самое важное для любого приложения происходит именно в функции обработки событий главного окна.

В Windows программа пассивна. После запуска она ждет, когда ей уделит внимание операционная система. Операционная система делает это посылкой сообщений, которые могут быть разного типа; они функционируют в системе достаточно хаотично, и приложение не знает, какого типа сообщение придет следующим. Логика построения Windows-приложения должна обеспечивать корректную и предсказуемую работу при поступлении сообщений любого типа.

Классическое оконное приложение, как правило, состоит по крайней мере из двух функций:

- стартовая функция, создающая главное окно WinMain();
- функция обработки сообщений окна (оконная функция).

Оконные приложения и организующие их работу функции могут быть созданы как с помощью специальных мастеров, встроенных в среду программирования, так и

Стартовая функция WinMain создает главное окно.

Если в консольной программе на С точкой входа (с этого места программа начинает выполняться) является функция ***main***(), то точкой входа программы для Windows является функция ***WinMain***().

```
int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    PSTR szCmdLine,
                    int iCmdShow)
        {...}
```

Эта функция использует последовательность вызовов API и при завершении возвращает операционной системе целое число.

Аргументы функции:
hInstance – дескриптор процесса (instance handle) – число, идентифицирующее программу, когда она работает под Windows. Если одновременно работают несколько копий одной программы, каждая копия имеет свое значение hInstance.
hPrevInstance – предыдущий дескриптор процесса (previous instance) — в настоящее время устарел, всегда равен NULL.
szCmdLine — указатель на оканчивающуюся нулем строку, в которой содержатся параметры, переданные в программу из командной строки. Можно запустить программу с параметром командной строки, вставив этот параметр после имени программы в командной строке.
iCmdShow — целое константное значение, показывающее, каким должно быть выведено на экран окно в начальный момент. Задается при запуске программы другой программой. В большинстве случаев число равно 1 (SW_SHOWNRMAL). В таблице ниже указаны возможные значения этого параметра:

Имя константы	Значение	Описание
SW_HIDE	0	Скрывает окно и делает активным другое окно
SW_SHOWNORMAL	1	Отображает и делает активным окно в его первоначальном размере и положении.
SW_SHOWMINIMIZED	2	Активизирует окно и отображает его в свернутом виде
SW_SHOWMAXIMIZED	3	Активизирует окно и отображает его в полноэкранном виде
SW_SHOWNOACTIVATE	4	Отображает окно аналогично SW_SHOWNORMAL, но не активизирует его
SW_SHOW	5	Отображает и делает активным окно с текущим размером и положением.
SW_MINIMIZE	6	Сворачивает текущее окно и делает активным следующее окно в порядке очереди.
SW_SHOWMINNOACTIVE	7	Сворачивает окно аналогично SW_SHOWMINIMIZED, но не активизирует его.
SW_SHOWNA	8	Отображает окно в текущей позиции аналогично SW_SHOW, но не активизирует его.
SW_RESTORE	9	Отображает и активизирует окно. Если окно было свернуто или развернуто во весь экран, оно отображается в своем первоначальном положении и размере.
SW_SHOWDEFAULT	10	Отображает окно способом, заданным по умолчанию.
SW_FORCEMINIMIZE	11	Применяется для минимизации окон, связанных с различными потоками.

В структуре стартовой функции Windows выделяются следующие операции, образующие «скелет» программы:

- регистрация класса окна;
- создание главного окна;
- отображение и перерисовка главного окна;
- цикл обработки очереди сообщений.

Регистрация класса окна осуществляется функцией:

```
ATOM WINAPI RegisterClass(_In_ const WNDCLASS *lpWndClass);
```

Прототип функции находится в файле библиотеки user32.dll. Единственным аргументом ее является указатель на структуру:

```
typedef struct _WNDCLASS {
        UINT     style;
        WNDPROC  lpfnWndProc;
        int      cbClsExtra;
        int      cbWndExtra;
        HINSTANCE hInstance;
        HICON    hIcon;
        HCURSOR  hCursor;
        HBRUSH   hbrBackground;
        LPCTSTR  lpszMenuName;
        LPCTSTR  lpszClassName; } WNDCLASS;
```

Члены структуры:

style — устанавливает стиль(и) класса. Этот член структуры может быть любой комбинацией стилей класса:

Имя стиля класса	Значение	Описание
CS_VREDRAW	0x01	Вертикальная перерисовка: осуществлять перерисовку окна при перемещении или изменении высоты окна.
CS_HREDRAW	0x02	Горизонтальная перерисовка: осуществлять перерисовку окна при перемещении или изменении ширины окна.
CS_KEYCVTWINDOW	0x04	В окне будет выполняться преобразование виртуальных клавиш.
CS_DBLCLKS	0x08	Окну будут посылаться сообщения о двойном щелчке кнопки мыши.
CS_OWNDC	0x20	Каждому экземпляру окна присваивается собственный контекст изображения.
CS_CLASSDC	0x40	Классу окна присваивается собственный контекст изображения, который можно разделить между копиями.
CS_PARENTDC	0x80	Классу окна передается контекст изображения родительского окна.
CS_NOKEYCVT	0x100	Отключается преобразование виртуальных клавиш.
CS_NOCLOSE	0x200	Незакрываемое окно: в системном меню блокируется выбор пункта закрытия окна.
CS_SAVEBITS	0x800	Часть изображения на экране, закрытая окном, сохраняется.
CS_BYTEALIGNCLIENT	0x1000	Выравнивание клиентской области окна: использование границы по байту по оси x.
CS_BYTEALIGNWINDOW	0x2000	Выравнивание окна: использование границы по байту по оси x.
CS_PUBLICCLASS CS_GLOBALCLASS	0x4000	Определяется глобальный класс окон.

lpfnWndProc — указатель на оконную процедуру.

cbClsExtra — устанавливает число дополнительных байт, которые размещаются вслед за структурой класса окна. Система инициализирует эти байты нулями, в большинстве случаев равен 0.

cbWndExtra— устанавливает число дополнительных байтов, которые размещаются вслед за экземпляром окна. Система инициализирует байты нулями.

hInstance — дескриптор экземпляра, который содержит оконную процедуру для класса.

hIcon — дескриптор значка класса, дескриптор ресурса значка. Если этот член структуры — NULL, система предоставляет заданный по умолчанию значок.

hCursor — дескриптор курсора класса, дескриптор ресурса курсора. Если этот член структуры — NULL, приложение устанавливает форму курсора всякий раз, когда мышь перемещается в окно прикладной программы.

hbrBackground — дескриптор кисти фона класса, дескриптор физической кисти, которая используется, чтобы красить цветом фона, или код цвета, преобразованный к типу HBRUSH.

lpszMenuName — указатель на символьную строку с символом конца строки (‘\0’), которая устанавливает имя ресурса меню класса. Можно использовать целое число, чтобы идентифицировать меню с помощью макроса MAKEINTRESOURCE(int). Если этот член структуры — NULL, окна, принадлежащие этому классу, не имеют заданного по умолчанию меню.

lpszClassName — указатель на символьную строку с именем класса, оканчивающуюся ‘\0’.

Создание окна осуществляется функцией:

```
HWND WINAPI CreateWindow(
                    _In_opt_  LPCTSTR lpClassName,
                    _In_opt_  LPCTSTR lpWindowName,
                    _In_      DWORD dwStyle,
                    _In_      int x,
                    _In_      int y,
                    _In_      int nWidth,
                    _In_      int nHeight,
                    _In_opt_  HWND hWndParent,
                    _In_opt_  HMENU hMenu,
                    _In_opt_  HINSTANCE hInstance,
                    _In_opt_  LPVOID lpParam );
```

Возвращаемое значение – дескриптор создаваемого окна. В случае невозможности создать окно возвращается NULL.

Аргументы функции:

lpClassName – указывает на строку с ‘\0’ в конце, которая определяет имя класса окна. Имя класса может быть зарегистрированным функцией RegisterClass или любым из предопределенных имен класса элементов управления.

lpWindowName — указывает на строку с ‘\0’ в конце, которая определяет имя окна.

dwStyle — определяет стиль создаваемого окна:

Имя стиля окна	Значение	Описание
WS_BORDER	0x00800000	Окно имеет тонкую границу в виде линии.
WS_CAPTION	0x00C00000	Окно имеет строку заголовка.
WS_CHILD	0x40000000	Окно является дочерним.
WS_DISABLED	0x08000000	Окно является изначально неактивным.
WS_GROUP	0x00020000	Окно группирует другие управляющие элементы.

Имя стиля окна	Значение	Описание
WS_HSCROLL	0x00100000	Окно содержит горизонтальную полосу прокрутки.
WS_MAXIMIZE	0x01000000	Исходный размер окна – во весь экран.
WS_MINIMIZE	0x20000000	Исходно окно свернуто.
WS_OVERLAPPED	0x00000000	Окно может быть перекрыто другими окнами.
WS_POPUP	0x80000000	Всплывающее окно.
WS_SYSMENU	0x00080000	Окно имеет системное меню в строке заголовка.
WS_VISIBLE	0x10000000	Окно изначально видимое.
WS_VSCROLL	0x00200000	Окно имеет вертикальную полосу прокрутки.

x — определяет координату левой стороны окна относительно левой стороны экрана. Измеряется в единицах измерения устройства, чаще всего в точках (pt). Для дочернего окна определяет координату левой стороны относительно начальной координаты родительского окна. Если установлен как CW_USEDEFAULT, Windows выбирает заданную по умолчанию позицию окна.

y – определяет координату верхней стороны окна относительно верхней стороны экрана. Измеряется в единицах измерения устройства, чаще всего в точках (pt). Для дочернего окна определяет координату верхней стороны относительно начальной координаты родительского окна.

nWidth – определяет ширину окна в единицах измерения устройства. Если параметр соответствует CW_USEDEFAULT, Windows выбирает заданную по умолчанию ширину и высоту для окна.

nHeight – определяет высоту окна в единицах измерения устройства.

hWndParent – дескриптор родительского окна.

hMenu – идентифицирует меню, которое будет использоваться окном. Этот параметр может быть NULL, если меню класса будет использовано.

hInstance — идентифицирует экземпляр модуля, который будет связан с окном.

lpParam — указывает на значение, переданное окну при создании.

Отображение окна осуществляется функцией

```
BOOL WINAPI ShowWindow(
  _In_  HWND hWnd,
  _In_  int nCmdShow);
```

Возвращаемое значение: 1 – успешное отображение окна, 0 – ошибка.

Аргументы функции:
hWnd – дескриптор отображаемого окна.

nCmdShow – константа, определяющая, как будет отображаться окно согласно таблице, приведенной выше.

Перерисовка окна осуществляется функцией

```
BOOL UpdateWindow(_In_  HWND hWnd);
```

Возвращаемое значение: 1 – успешная перерисовка окна, 0 – ошибка.
Аргумент функции ***hWnd*** – дескриптор окна.

Цикл обработки сообщений

После вызова функции UpdateWindow, окно окончательно выведено на экран. Теперь программа должна подготовить себя для получения информации от пользователя через клавиатуру и мышь. Windows поддерживает «очередь сообщений» (message queue) для каждой программы, работающей в данный момент в системе Windows. Когда происходит ввод информации, Windows преобразует ее в «сообщение», которое помещается в очередь сообщений программы. Программа извлекает сообщения из очереди сообщений, выполняя блок команд, известный как «цикл обработки сообщений» (message loop):

```
while(GetMessage(&msg,NULL,0,0))
      {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
      }
```

Для получения сообщения из очереди используется функция:

```
BOOL WINAPI GetMessage(
                        _Out_     LPMSG lpMsg,
                        _In_opt_  HWND hWnd,
                        _In_      UINT wMsgFilterMin,
                        _In_      UINT wMsgFilterMax);
```

В случае получения из очереди сообщения, отличного от WM_QUIT, возвращает ненулевое значение.

Аргументы функции:
lpMsg — указатель на структуру сообщения:

```
typedef struct MSG {
            HWND hwnd; // дескриптор окна, очередь сообщений которого
            просматривается
            UINT message; // идентификатор сообщения
            WPARAM wParam; // дополнительная информация о сообщении,
            LPARAM lParam; // зависит от идентификатора сообщения
            DWORD time; // время помещения сообщения в очередь
            POINT pt; // структура, содержащая координаты курсора в момент
            помещения сообщения в очередь
                } MSG;
```

Структура POINT имеет вид

```
typedef struct POINT{
            LONG x; // координата x
            LONG y; // координата y
                    } POINT;
```

hWnd — дескриптор окна, очередь для которого просматривается.

wMsgFilterMin — нижняя граница фильтра идентификаторов сообщений.

wMsgFilterMax — верхняя граница фильтра идентификаторов сообщений.

Функция
BOOL WINAPI TranslateMessage(_In_ const MSG *lpMsg);

передает аргумент — структуру msg обратно в Windows для преобразования какого-либо сообщения с клавиатуры. Возвращает ненулевое значение в случае успешной расшифровки сообщения, 0 – ошибка.

Функция
LRESULT WINAPI DispatchMessage(_In_ const MSG *lpmsg);

передает аргумент — структуру msg обратно в Windows. Windows отправляет сообщение для его обработки соответствующей оконной процедуре — таким образом, Windows вызывает соответствующую оконную функцию, указанную при регистрации класса окна.

После того, как оконная функция обработает сообщение, оно возвращается в Windows, которая все еще обслуживает вызов функции DispatchMessage. Когда Windows возвращает управление в стартовую функцию WinMain() к следующему за вызовом DispatchMessage коду, цикл обработки сообщений в очередной раз возобновляет работу, вызывая GetMessage, которая возвращает значение, определяемое оконной функцией.

Пример стартовой функции, создающей и выводящей окно размером 500x300 точек:

```
#include <windows.h>
LONG WINAPI WndProc(HWND, UINT, WPARAM,LPARAM);
int WINAPI WinMain (HINSTANCE hInstance,      HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,      int nCmdShow)
{
  HWND hwnd; // дескриптор окна
  MSG msg; // структура сообщения
  WNDCLASS w; // структура класса окна
  // Регистрация класса окна
  memset(&w,0,sizeof(WNDCLASS));
  w.style = CS_HREDRAW | CS_VREDRAW;
  w.lpfnWndProc = WndProc; // имя оконной функции
  w.hInstance = hInstance;
  w.hbrBackground = (HBRUSH)(WHITE_BRUSH);
  w.lpszClassName = "My Class";
  RegisterClass(&w);
  // Создание окна
  hwnd = CreateWindow("My Class",
 "Окно пользователя",
  WS_OVERLAPPEDWINDOW, 500, 300, 500, 380, NULL, NULL, hInstance, NULL);
  ShowWindow(hwnd,nCmdShow); // отображение
  UpdateWindow(hwnd);          // перерисовка
  // Цикл обработки сообщений
  while(GetMessage(&msg,NULL,0,0))
  {
```

```
  TranslateMessage(&msg);
  DispatchMessage(&msg);
 }
 return msg.wParam;
}
```

Примечание: Для корректной сборки приложения используется многобайтовая кодировка.

Для обработки сообщений окна предназначена **Оконная функция**. Эта функция организована по принципу ветвления, состоящего из последовательной проверки типа сообщения. При совпадении типа сообщения, переданного в структуре Message с соответствующей веткой, осуществляется его обработка.

Минимальный вид оконной функции представлен ниже:

```
LONG WINAPI WndProc(HWND hwnd, UINT Message, WPARAM wparam, LPARAM lparam)
        {
         switch (Message)
         {
          case WM_DESTROY:
          PostQuitMessage(0);
          break;
          default:
          return DefWindowProc(hwnd, Message, wparam, lparam);
         }
         return 0;
        }
```

Четыре аргумента оконной функции идентичны первым четырем полям структуры сообщения MSG. В примере обрабатывается только один тип сообщения WM_DESTROY, которое передается оконной функции при закрытии окна.

Вызов функции DefWindowProc() обрабатывает по умолчанию все сообщения, которые не обрабатывает оконная процедура.

Функция PostQuitMessage() сообщает Windows, что данный поток запрашивает завершение. Аргументом является целочисленное значение, которое функция вернет операционной системе.

Элементы управления окна

Главным элементом программы в среде Windows является окно. Окно может содержать элементы управления: кнопки, списки, окна редактирования и др. Эти элементы также являются окнами, но обладающими особым свойством: события, происходящие с этими элементами (и самим окном), приводят к приходу сообщений в процедуру окна.

Системный класс	Предназначение
BUTTON	Кнопка.
COMBOBOX	Комбинированное окно (окно со списком и поля выбора).
EDIT	Окно редактирования текста.
LISTBOX	Окно со списком

Системный класс	Предназначение
SCROLLBAR	Полоса прокрутки
STATIC	Статический элемент (текст)

Создание элементов управления окна осуществляется функцией

```
HWND WINAPI CreateWindow(
                    _In_opt_ LPCTSTR lpClassName, // имя предопределенного класса
                    _In_opt_ LPCTSTR lpWindowName, // текст
                    _In_ DWORD dwStyle, // стиль
                    _In_ int x, // координата x
                    _In_ int y, // координата y
                    _In_ int nWidth, // ширина
                    _In_ int nHeight, // высота
                    _In_opt_ HWND hWndParent, // дескриптор родительского окна
                    _In_opt_ HMENU hMenu, // номер пункта меню
                    _In_opt_ HINSTANCE hInstance, // дескриптор приложения
                    _In_opt_ LPVOID lpParam ); // NULL
```

Указанная функция возвращает дескриптор элемента управления окна, который может быть впоследствии использован для анализа элемента управления, с которым связано обрабатываемое событие. Дескриптор кнопки, например, передается в оконную функцию в качестве параметра lparam.

Таблицу стилей элементов управления окна можно устанавливать в параметре dwStyle как и для создания родительского окна (см. выше). При этом обязательно указывается, что создаваемое окно является дочерним — WS_CHILD.

Кнопка — маленькое прямоугольное дочернее окно, которое представляет собой кнопку, по которой пользователь может щелкать мышью, чтобы включить или выключить ее. Кнопки управления могут использоваться самостоятельно или в группах, и они могут или быть подписаны или появляться без текста. Кнопки управления обычно изменяют свой вид, когда пользователь щелкает мышью по ним.

При нажатии кнопки операционная система генерирует сообщение WM_COMMAND с параметром lParam, соответствующим дескриптору кнопки.

Обработка нажатия кнопки:

```
LONG WINAPI WndProc(HWND hwnd, UINT Message, WPARAM wparam, LPARAM lparam)
{
 ...
 switch (Message) {
  case WM_COMMAND:
   if(lparam == (LPARAM)hBtn) {
    //обработка нажатия кнопки
   }
   break;
  ...
```

```
  }
}
```

Поле редактирования — прямоугольное дочернее окно, внутри которого пользователь может напечатать с клавиатуры текст. Пользователь выбирает орган управления и дает ему фокус клавиатуры, щелкая по нему мышью или перемещая в него, каретку путем нажатия клавиши ТАБУЛЯЦИИ (TAB). Пользователь может вводить текст, когда окно редактирования текста отображает мигающую каретку.

Для считывания информации из поля редактирования используется функция:

```
int WINAPI GetWindowText(
        _In_   HWND hWnd,   // дескриптор поля
        _Out_  LPTSTR lpString,   // указатель на текстовую строку
        _In_   int nMaxCount );       // максимальное количество символов
```

Возвращаемое значение – длина считанной текстовой строки.

Для установки текста в поле редактирования используется функция

```
BOOL WINAPI SetWindowText(
        _In_      HWND hWnd,   // дескриптор поля
        _In_opt_  LPCTSTR lpString ); // указатель на текстовую строку
```

В случае успешного завершения функция возвращает ненулевое значение.

Статический текст — текстовое поле, окно или прямоугольник, используемый для надписей, не подлежащих редактированию.

Для установки статического текста используется та же функция SetWindowText.

В любом окне может быть создано меню.

Меню (menu) - это перечень пунктов, которые устанавливают параметры или группы параметров (подменю) для прикладной программы, выполняемой в окне. Щелчок по пункту меню открывает подменю или заставляет приложение выполнить команду.

Меню выстраивается иерархически. На верхнем уровне иерархии находится строка меню (***menu bar***), содержащая перечень меню, которые, в свою очередь, могут содержать подменю (***submenus***). Пункт меню может или выполнить команду или открыть подменю (выпадающее меню - ***drop-down menu***). Для получения информации о строке меню используется функция GetMenuBarInfo.

Окно может содержать меню окна (системное меню). Системное меню содержит команды для изменения размера или позиции окна, а также его закрытия.

Система также предусматривает меню команд (***shortcut menus***), называемое также контекстным меню. Меню команд не соединено со строкой меню; оно может появиться в любом месте экрана. Контекстное меню остается скрытым до тех пор, пока пользователь не активизирует его, обычно щелкая правой кнопкой мыши по выбранной позиции, инструментальной панели или кнопке панели задач. Меню обычно отображается в позиции курсора мыши.

Меню может быть создано с помощью шаблона меню либо специальных функций. Шаблоны меню обычно определяются как ресурсы. Ресурсы шаблона меню могут быть загружены явно или назначены как заданное по умолчанию меню для класса окна. Большинство прикладных программ создает меню, используя ресурсы шаблона меню. После создания ресурса шаблона меню и добавления его к исполняемому (.EXE) файлу приложения, можно с помощью функции LoadMenu загрузить этот ресурс в память.

Меню может быть создано и с помощью специальных функций. Функция CreateMenu создает пустую строку меню, функция CreatePopupMenu - для создания пустого меню. Добавить пункты к меню можно с помощью InsertMenuItem.

После создания меню его следует связать с окном. Это действие выполняется при помощи определения дескриптора меню как параметра hMenu функции CreateWindow или CreateWindowEx или путем вызова функции SetMenu.

Для создания и управления меню используются следующие Функции:

Функция	Описание
AppendMenu	Функция AppendMenu добавляет в конец заданной строки меню, "выпадающего" меню, подменю или контекстного меню новый пункт. Вы можете использовать эту функцию, чтобы определить содержание, внешний вид и поведение пункта меню.
CheckMenuItem	Функция CheckMenuItem устанавливает атрибут "галочки" заданного пункта меню в выбранное или не выбранное состояние.
CheckMenuRadioItem	Функция CheckMenuRadioItem отмечает "кружочком" заданный пункт меню и делает его пунктом с “радиокнопкой”. Одновременно, функция снимает отметку "кружочком " во всех других пунктах меню в связанной группе и очищает эти пункты от флажков пункта с “радиокнопкой”.
CreateMenu	Функция CreateMenu создает меню. Меню вначале пустое, но оно может быть заполнено пунктами меню при помощи использования функций InsertMenuItem, AppendMenu и InsertMenu.
CreatePopupMenu	Функция CreatePopupMenu создает “выпадающее” меню, подменю или контекстное меню. Меню вначале пустое. Вы можете вставлять или добавлять в конец пункты меню при помощи использования функции InsertMenuItem. Вы можете также использовать и функцию InsertMenu, чтобы вставлять пункты меню, а функцию AppendMenu, чтобы добавлять в конец пункты меню.
DeleteMenu	Функция DeleteMenu удаляет пункт из заданного меню. Если пункт меню открывает меню или подменю, эта функция уничтожает дескриптор меню или подменю и освобождает память, использованную меню или подменю.
DestroyMenu	Функция DestroyMenu уничтожает заданное меню и освобождает любую память, которую меню занимает.
DrawMenuBar	Функция DrawMenuBar перерисовывает строку меню заданного окна. Если строка меню изменяется после того, как система создала окно, эта функция должна быть вызвана, чтобы нарисовать

	измененную строку меню.
EnableMenuItem	Функция EnableMenuItem включает, отключает или окрашивает в серый цвет (делает недоступным) заданный пункт меню.
EndMenu	Функция EndMenu завершает работу активного меню вызывающего потока.
GetMenu	Функция GetMenu извлекает дескриптор меню, связанного с заданным окном.
GetMenuBarInfo	Функция GetMenuBarInfo извлекает информацию об заданной строке меню.
GetMenuCheckMarkDimensions	Функция GetMenuCheckMarkDimensions возвращает размеры заданного по умолчанию точечного рисунка "галочки ". Система показывает на экране этот точечный рисунок рядом с выбранными пунктами меню. Перед вызовом функции SetMenuItemBitmaps, чтобы заменить заданный по умолчанию рисунок "галочки " для пункта меню, приложение должно выяснить правильный размер точечного рисунка при помощи вызова GetMenuCheckMarkDimensions.
GetMenuDefaultItem	Функция GetMenuDefaultItem выясняет заданный по умолчанию пункт меню в заданном меню.
GetMenuInfo	Функция GetMenuInfo получает информацию об заданном меню.
GetMenuItemCount	Функция GetMenuItemCount выявляет число пунктов в заданном меню.
GetMenuItemID	Функция GetMenuItemID извлекает идентификатор пункта меню размещенного в заданной позиции в меню.
GetMenuItemInfo	Функция GetMenuItemInfo извлекает информацию о пункте меню.
GetMenuItemRect	Функция GetMenuItemRect извлекает рабочий (ограничивающий) прямоугольник для заданного пункта меню.
GetMenuState	Функция GetMenuState извлекает флажки меню, связанные с заданным пунктом меню. Если пункт меню открывает подменю, эта функция к тому же возвращает число пунктов в подменю.
GetMenuString	Функция GetMenuString копирует текстовую строку заданного пункта меню в определяемый буфер.
GetSubMenu	Функция GetSubMenu извлекает дескриптор "выпадающего" меню или подменю, активизируемого заданным пунктом меню.
GetSystemMenu	Функция GetSystemMenu дает возможность прикладной программе обратиться к меню окна (также известному как *системное меню (system menu)* или *меню окна (control menu)*) для копирования и модификации.
HiliteMenuItem	Функция HiliteMenuItem выделяет или удаляет выделение пункта в строке меню.
InsertMenu	Функция InsertMenu вставляет новый пункт в меню, перемещая другие пункты вниз меню. Обратите внимание! на то, что функция InsertMenu была заменена функцией InsertMenuItem. Вы можете все еще использовать InsertMenu, в том случае, если нет необходимости в каком-либо из дополнительных свойств функции InsertMenuItem.
InsertMenuItem	Функция InsertMenuItem вставляет новый пункт меню в заданной позиции в меню.
IsMenu	Функция IsMenu выясняет, является ли дескриптор дескриптором меню.
LoadMenu	Функция LoadMenu загружает заданный ресурс меню из исполняемого (.exe) файла программы, связанного с экземпляром приложения.
LoadMenuIndirect	Функция LoadMenuIndirect загружает заданный шаблон меню в

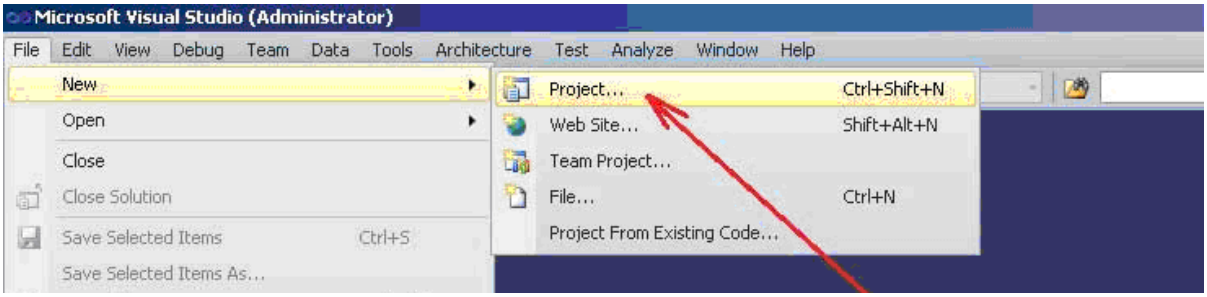
	памяти.
MenuItemFromPoint	Функция MenuItemFromPoint выясняет, какой пункт меню, если таковые вообще имеются, находится в заданном месте.
ModifyMenu	Функция ModifyMenu изменяет существующий пункт меню. Эта функция используется для, установки содержания, внешнего вида, и поведения пункта меню.
RemoveMenu	Функция RemoveMenu удаляет пункт меню или отключает подменю от заданного меню. Если пункт меню открывает "выпадающее" меню, или подменю, RemoveMenu не уничтожает меню или его дескриптор, разрешая ему многократно использоваться. Прежде, чем эта функция вызывается, функция GetSubMenu должна извлечь дескриптор "выпадающего" меню или подменю.
SetMenu	Функция SetMenu назначает новое меню для заданного окна.
SetMenuDefaultItem	Функция SetMenuDefaultItem устанавливает определенный по умолчанию пункт для заданного меню.
SetMenuInfo	Функция SetMenuInfo устанавливает информацию для заданного меню.
SetMenuItemBitmaps	Функция SetMenuItemBitmaps связывает заданный точечный рисунок (значок) с пунктом меню. Выбран ли пункт меню или нет, система показывает на экране соответствующий значок рядом с пунктом меню.
SetMenuItemInfo	Функция SetMenuItemInfo изменяет информацию о пункте меню.
TrackPopupMenu	Функция TrackPopupMenu показывает на экране контекстное меню в заданном месте и устанавливает подбор пунктов меню. Контекстное меню может появиться в любом месте экрана.
TrackPopupMenuEx	Функция TrackPopupMenuEx показывает на экране контекстное меню в заданном месте и устанавливает подбор пунктов меню. Контекстное меню может появиться в любом месте экрана.

П 12. Создание оконного приложения в MS Visual Studio

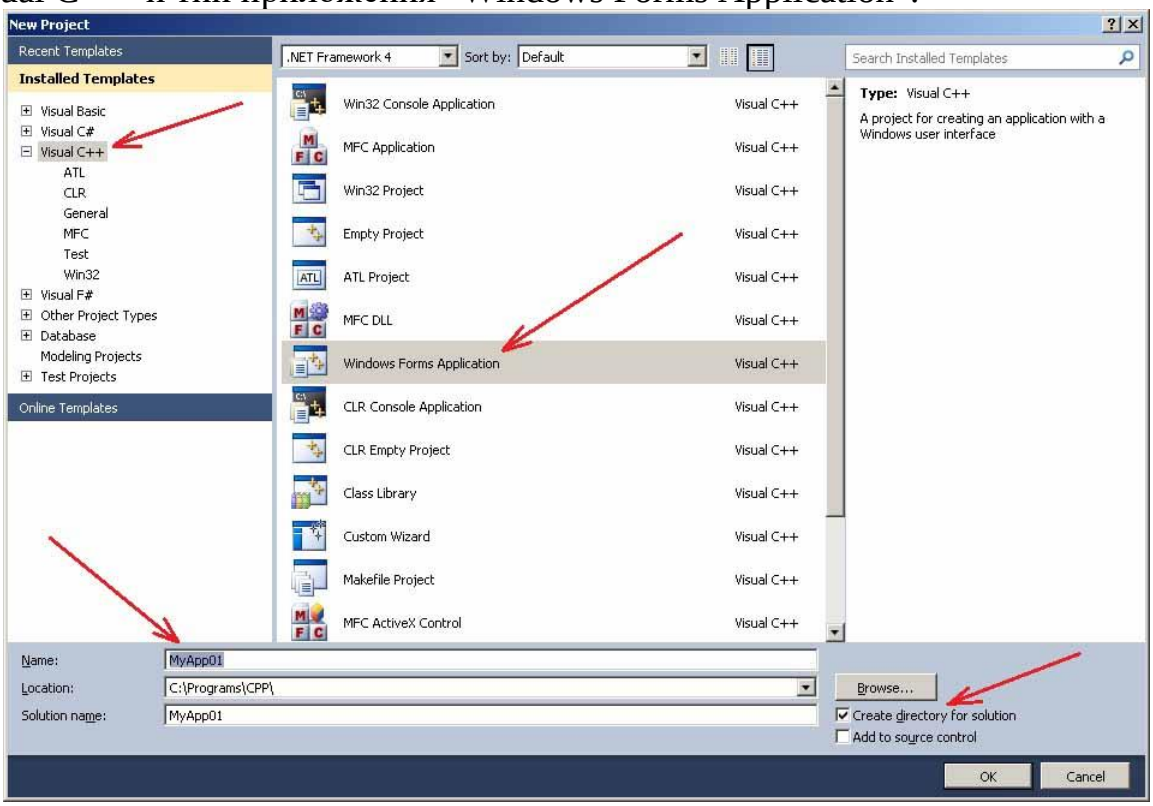
Система Microsoft Visual Studio предлагает разные виды шаблонов приложения для программирования на языке C++ . Следует заметить, что разные версии Ms Visual Studio могут в незначительной мере отличаться друг от друга.

Для создания нового проекта (решения) на языке C++ выбирается последовательность команд:

```
File -> New Project...
```



В результате откроется окно "New Project", в котором следует выбрать шаблон "Visual C++" и тип приложения "Windows Forms Application".



В поле "Location:" задается путь к папке, в которой будет сохранен проект. Например:

```
C:\Programs\CPP
```

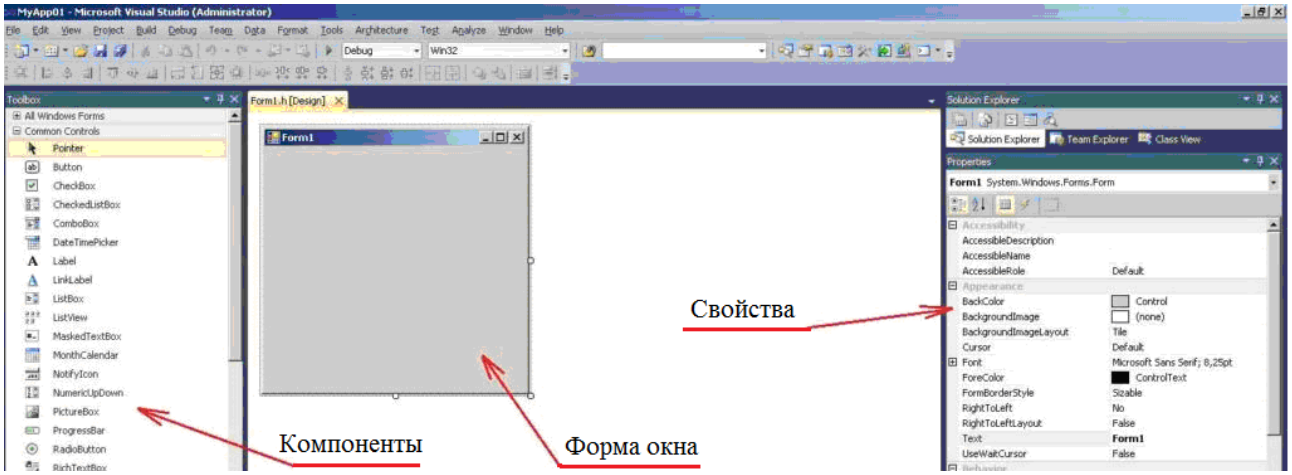
В поле "Name:" задается имя приложения. Пусть это будет "MyApp01". Если включена опция "Create directory for solution" (создать директорию для решения), то проект будет сохранен в папке:

```
C:\Programs\CPP\MyApp01
```

В поле "Solution name:" задается имя решения. Решение (solution) может объединять в себе несколько проектов. В рассматриваемом примере имя решения остается таким же, как и имя проекта.

После выбора "ОК" в предыдущем окне "New Project", система Microsoft Visual Studio создаст весь необходимый код для работы приложения типа Windows Forms Application.

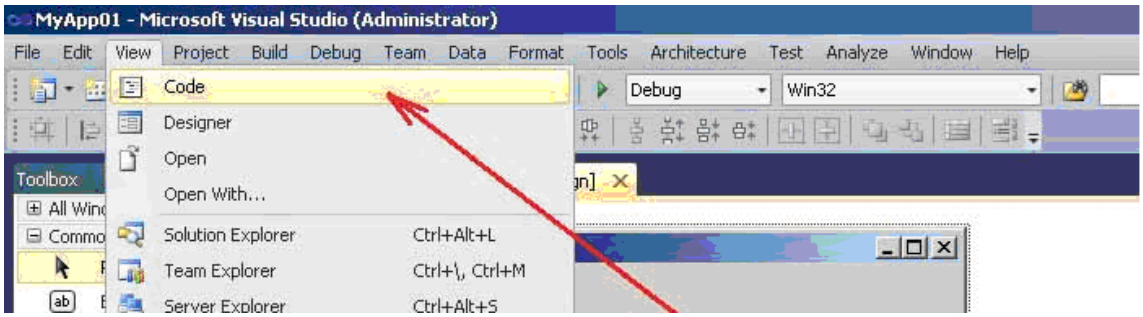
В результате окно Microsoft Visual Studio примет вид:



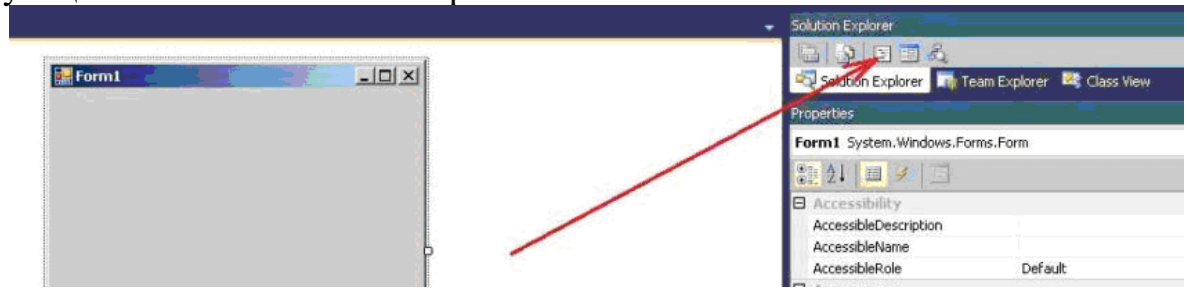
В окне, в центре отображается основная форма приложения. На этой форме можно размещать компоненты. Компоненты выбираются из панели Toolbox (левая часть экрана). Свойства формы или компонента отображаются в утилите Solution Explorer (правая часть экрана). Изменяя эти свойства, можно влиять на вид формы, поведение формы, реализовывать обработчики событий формы и прочее. Кроме того, свойства окна (и отдельных его компонентов) могут быть доступны в соответствующем контекстном меню, вызываемом правой кнопкой мышки.

Из режима проектирования формы можно перейти в режим набора текста программы, выбрав команду "Code" в меню View:

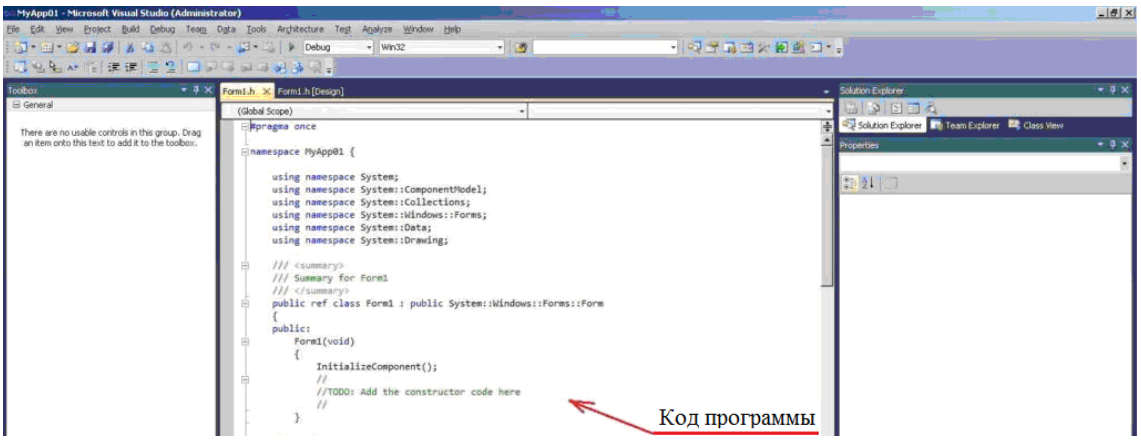
```
View -> Code
```



Другой вариант перехода в режим набора текста, это клик на соответствующей кнопке в Solution Explorer:



В результате отобразится текст программы, который при необходимости можно редактировать:



При создании проекта система Microsoft Visual Studio генерирует программный код, который сохраняется в различных файлах.

Главным файлом, в котором программист создает собственный код программы, есть файл "Form1.h". Этот файл соответствует главной форме приложения. На главной форме размещаются различные компоненты. С помощью этих компонент реализуется решение конкретной задачи. При создании проекта создается пустая форма. Кроме главной формы можно создавать другие формы и добавлять их к проекту.

Также, согласно правилам языка C++, создается файл реализации "MyApp01.cpp". Этот файл представляет главную функцию main(). В нем содержится код, отображающий главную форму.

Как пример, он может содержать следующие операторы:

```
#include "Form1.h"
using namespace System;
using namespace System::Windows::Forms;
[STAThread]
void main(array<String^>^ args) {
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);
	MyApp01::Form1 form;
	Application::Run(%form);
}
```

Первым оператором в main включены визуальные стили для приложения. Визуальные стили — это цвета, шрифты и другие визуальные элементы, которые формируют тему операционной системы. Элементы управления будут рисоваться с визуальными стилями, если элемент управления и операционная система его поддерживают.

Второй `Application` задает значения по умолчанию во всем приложении для свойства `UseCompatibleTextRendering`, определенного в конкретных (новых) элементах управления.

Основная (главная) форма может быть создана и с помощью кода:

```
using namespace Project1;
int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);
```

```
        Application::Run(gcnew MyForm);
        return 0;
}
```

Следует заметить, что если в Приложении будет создано несколько форм, то включать визуальные стили необходимо только в главном окне (форме).

Файл "MyApp01.vcxproj". В нем содержится информация о версии Visual C++, в которой сгенерирован файл, информация о платформах, настройках и особенностях (характеристиках) проекта выбранных с помощью мастера приложений (Application Wizard).

Файл "MyApp01.vcxproj.filters". В нем содержится информация об ассоциации между файлами в созданном проекте и фильтрами.

Файл "AssemblyInfo.cpp". Содержит пользовательские атрибуты для модификации данных сборки.

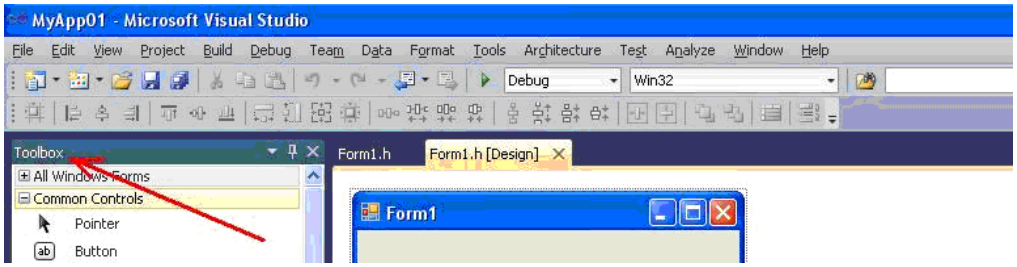
Файлы "StdAfx.h" и "StdAfx.cpp". Они используются для создания файла скомпилированных заголовков (PCH) с именем "MyApp01.pch" и скомпилированных типов названных "StdAfx.obj".

Для возврата в режим проектирования (Design), следует нажать на кнопку View "Designer" в Solution Explorer (рядом с кнопкой View "Code").

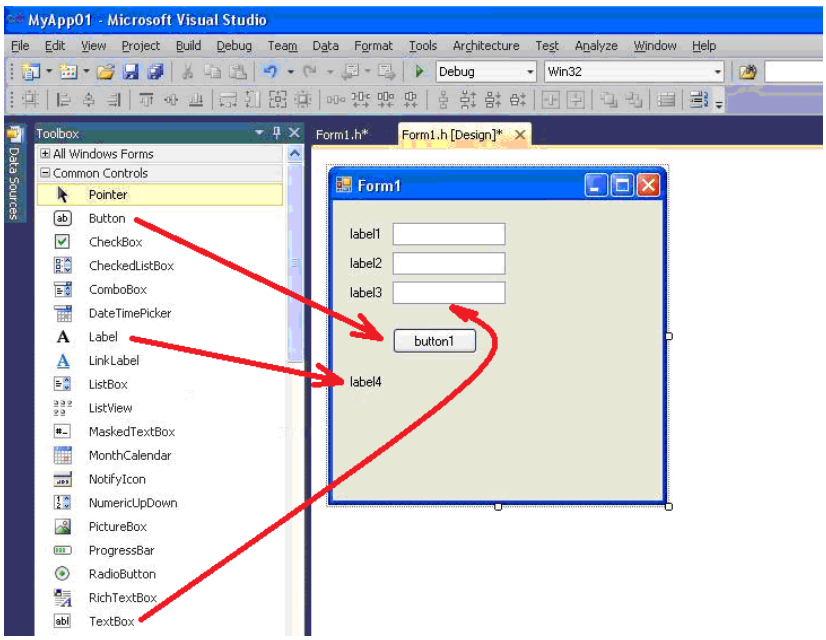
Практически любое окно содержит следующие основные компоненты:

- Label - статическая надпись на форме, определяемая именем Name и текстом Text;
- DataGridView - таблица, параметры которой RowCount, ColumnCount, Rows (список из строк таблицы);
- NumericUpDown - сочетание текстового поля и пары кнопок со стрелками для корректировки значения пользователем$
- Textbox - поле для ввода текста;
- Button - кнопка, ...

Для организации работы программы в созданном окне следует в режиме проектирования окна (Design) разместить соответствующие компоненты на форме. Все необходимые компоненты расположены на панели инструментов Toolbox:



Например, на форме размещаются следующие компоненты:



- три компонента типа Label (метка). Предназначены для вывода информационных сообщений "a = ", "b = ", "c = ". В результате создается три экземпляра (объекта) с именами label1, label2, label3;
- три компонента типа TextBox (строка ввода) – для ввода значений переменных *a*, *b*, *c*. Создается три объекта с именами textBox1, textBox2, textBox3;
- один компонент типа Button (кнопка). Предназначен для реализации команды вычисления площади и вызова соответствующего обработчика события. Создается экземпляр с именем button1;
- один компонент типа Label. Предназначен для вывода результата. Создается объект с именем label4.

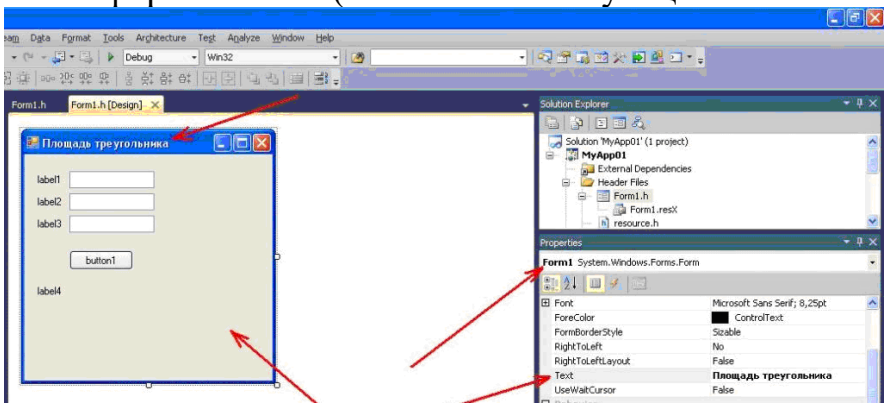
Чтобы форма имела более наглядный для решения задачи вид, нужно настроить ее свойства а также свойства компонент размещенных на ней.

Для настройки свойств формы и компонент используется окно "Properties". Это окно вызывается командой

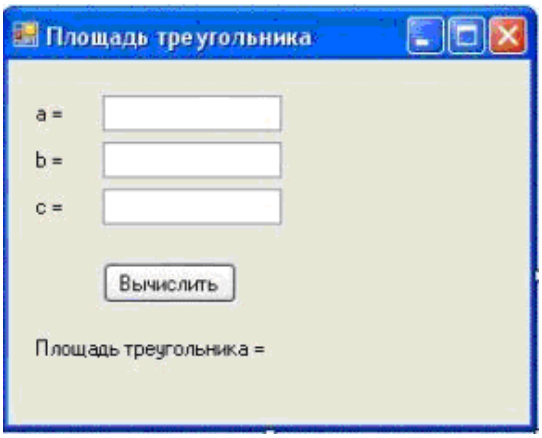
`View->Properties Window`

или клавишей F4.

Для изменения названий формы (или отдельных ее компонент) нужно ее предварительно выделить (мышкой). После этого в заголовке окна "Properties" появится название формы Form1 (либо соответствующей компоненты).



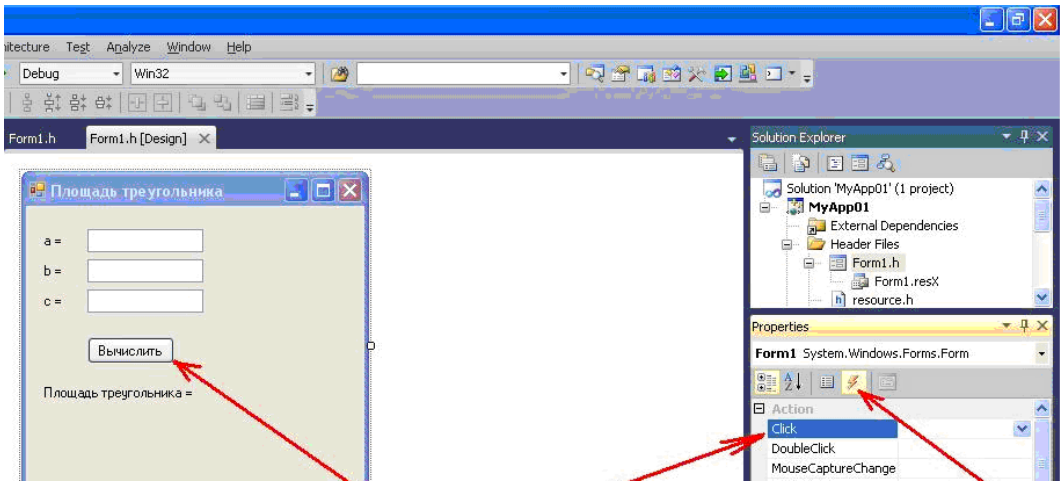
Затем в поле Text установить значение, например: "Площадь треугольника". Аналогично переписываются названия остальных компонентов формы:



Результатом работы программы может быть, например, вычисление площади треугольника, заданного сторонами a, b и c. Вычисление происходит в момент, когда пользователь делает клик на кнопке "Вычислить". В результате формируется событие Click. В программе этому событию отвечает фрагмент программного кода, который называется обработчиком события (event handler). Этот фрагмент формируется средствами Microsoft Visual Studio.

Для вызова обработчика события клика на кнопке button1 нужно выполнить следующие действия:

- выделить компонент button1;
- в окне Properties активировать вкладку «Events» (события);
- в списке событий, в поле ввода события Click сделать двойной клик мышкой.



В результате выполненных действий, система Microsoft Visual Studio сформирует обработчик события и переключит окно в режим ввода программного кода.

Обработчик события (метод) имеет название button1_Click. Формируется следующий программный код:

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
}
```

Между фигурными скобками записывается требуемый код программы (например, вычисления площади треугольника):

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
     float a, b, c, p, s, t;
     a = System::Double::Parse(this->textBox1->Text);
     b = System::Double::Parse(this->textBox2->Text);
     c = c.Parse(this->textBox3->Text);
     p = (a + b + c)/2;
     t = p * (p-a) * (p-b) * (p-c);
     if (t<0)
         label4->Text = "Ошибка ввода данных!";
     else
     {
         s = (float)Math::Sqrt(t);
         label4->Text = s.ToString();
     }
}
```

Здесь для перевода значения компонента textBox1->Text из строки в соответствующий вещественный тип float используется метод Parse() из класса System::Double.

Чтобы занести значение строки с textBox1->Text в переменную *a* типа float можно воспользоваться одним из трех способов:

- `a = System::Double::Parse(this->textBox1->Text);`
- `a = a.Parse(this->textBox1->Text);`
- `a = Convert::ToDouble(this->textBox1->Text).`

Указатель this есть указатель на данный класс (класс Form1).

Для вычисления квадратного корня используется функция Sqrt() из класса Math:

```
s = (float)Math::Sqrt(t);
```

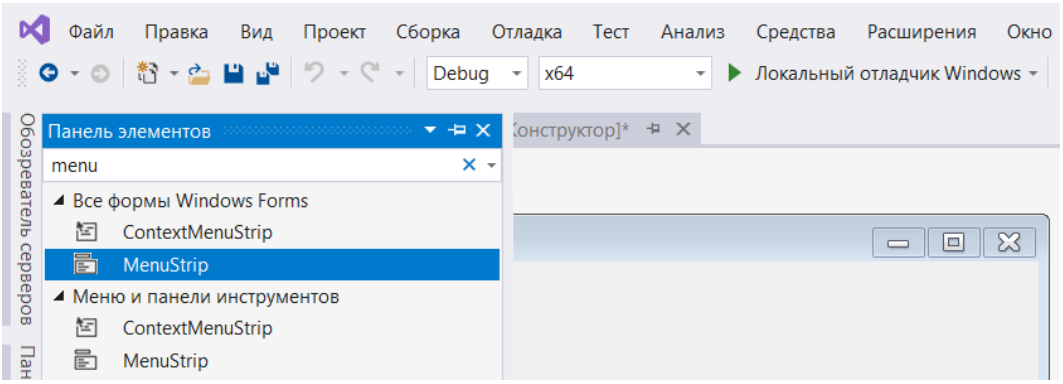
Для преобразования из типа float в строчный (string) используется метод ToString():

```
label4->Text = s.ToString();
```

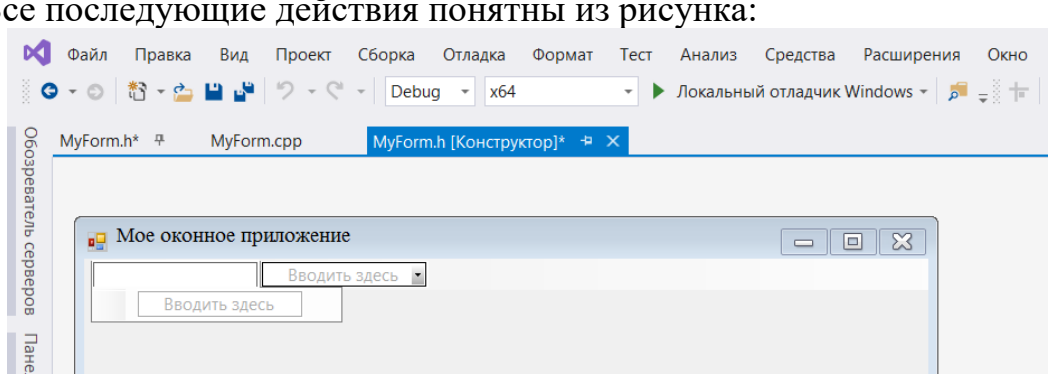
или

```
label4->Text = Convert::ToString(s);
```

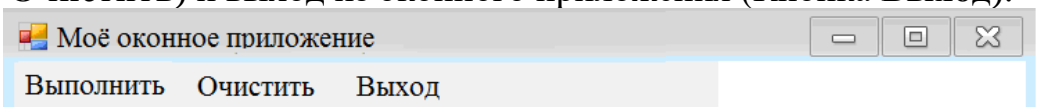
Для организации более удобной работы с оконным приложением может быть создано МЕНЮ окна. Для этого на Панели инструментов (Toolbox) выбирается Компонент MenuStrip (Строка Меню).



Все последующие действия понятны из рисунка:



В результате будет получен компонент (меню) предназначенный для легкого доступа к основным действиям программы, например: запуск вычислительного процесса и выведение в соответствующее поле окна его результатов (кнопка **Выполнить**), очистка полей для повторного выполнения программы (кнопка **Очистить**) и выход из оконного приложения (Кнопка **Выход**).



Для каждой кнопки меню пишется соответствующая функция, запускаемая при наступлении соответствующего события (нажатия на кнопку).
Например, Код компонента меню "Очистить":

```
private: System::Void ОчиститьToolStripMenuItem_Click(System::Object^  sender,
                                                       System::EventArgs^  e)
      {
            for (int i = 0; i < N; i++)
                  for (int j = N; j < 2 * N; j++)
                  {
                        dataGridView2->Rows[i]->Cells[j - N]->Value = " ";
                        dataGridView1->Rows[i]->Cells[j - N]->Value = " ";
                        label4->Text = " ";
                  }
      }
```

Код компонента меню "Выход":

```
private: System::Void ВыходToolStripMenuItem_Click(System::Object^  sender,
                                                   System::EventArgs^  e)
      {
            Application::Exit();
      }
```

Отдельно стоит упомянуть о событии Load. Оно выполняется при запуске формы, и с его помощью задаются начальные значения во все ячейки:

```
private: System::Void main_form_Load(System::Object^ sender, System::EventArgs^
e)
           {
              this->textBox1->Text = "4";
              this->textBox2->Text = "5";
              this->textBox3->Text = "6";
           }
```

Ниже в примерах приводятся полезные функции для создания некоторых приложений.

Пример 1. По кнопке button1_Click вызывается стандартный метод обработки события. Внутри метода создаются и инициализируются все необходимые элементы графического интерфейса (рабочего окна) и вызываются функции работы с элементами массива.
size — порядок массива;
param[] — одномерный массив;
dataGridView1 — таблица для вывода значений массива на экран;
dataGridView2 — таблица для ввода одномерного массива;
radioButton1 — параметр заполнения таблицы;

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^
e)          {
            // Получаем порядок полинома
            int size = Convert::ToInt32(numericUpDown1->Value);
            // Создаем массив
            int * param = new int[size + 1];//массив коэффициентов
            // Обнуляем элементы массива
            for (int i = 0; i < size + 1; i++)
            {
                  param[i] = 0;
            }
            // Создаем таблицу1
            dataGridView1->RowCount = size + 1;
            dataGridView1->ColumnCount = size / 2 + 2;
            dataGridView1->TopLeftHeaderCell->Value = "Массив";

            // Создаем таблицу2
            dataGridView2->RowCount = 1;
            dataGridView2->ColumnCount = size + 1;

            //Если выбран вариант Заполнить таблицу коэффициентами
            if (radioButton1->Checked) {
                  typing(size, param);
                  show1(size, param);
            }
            // Выравниваем ячейки таблицы
            dataGridView1->AutoResizeRowHeadersWidth
                    (DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);
            dataGridView1->AutoResizeColumns();
            dataGridView2->AutoResizeRowHeadersWidth
```

```
                (DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);
        dataGridView2->AutoResizeColumns();
        // Удаляем из памяти массив
        for (int j = 0; j < size + 1; j++)
        delete[] param;
}
```

В функции обработки события `button1_Click` использованы вспомогательные функции:

void clear() - функция очистки таблиц.

```
private: void clear() {
        for (int i = 0; i < dataGridView1->Rows->Count; i++)
        {
                for (int j = 0; j < dataGridView1->Columns->Count; j++)
                {
                        dataGridView1->Rows[i]->Cells[j]->Value = 0;
                }
        }
        for (int i = 0; i < dataGridView2->Columns->Count; i++)
        {
                dataGridView2->Rows[0]->Cells[i]->Value = 0;
        }
        textBox1->Text = "";
}
```

void typing(int size, int * param) —ввод элементов одномерного массива.
В цикле считываются значения элементов массива param из графического элемента dataGridView2

```
private: void typing(int size, int * param)
{
        int j = dataGridView2->Columns->Count;
        for (int i = 0; i < size + 1; i++)
        {
           dataGridView2->TopLeftHeaderCell->Value = "Степень";
           dataGridView2->Columns[i]->HeaderCell->Value = Convert::ToString(--j);
           param[i] = Convert::ToInt32(dataGridView2->Rows[0]->Cells[i]->Value);
        }
}
```

void show1(int size, int* param) — заполнение двух строк таблицы datagridview1 значениями элементов массива param.

```
private: void show1(int size,  int* param)
{
        int p = 0;
        for (int j = 0; j < 2; j++)
        {
                for (int k = 1; k < size / 2 + 2 && p <= size; k++)
                {
                        dataGridView1->Rows[j]->Cells[k]->Value = param[p];
                        p += 2;
                }
                p = 1;
        }
}
```

Пример 2. Создание таблицы в форме (окне)

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
      int n;
      n = System::Int16::Parse(textBoxsize->Text);
                                          //чтение размера матрицы из окна textbox
            matrix->ColumnCount = n;
            matrix->RowCount = n + 1;
}
```

Пример 3. Функция заполнения матрицы значениями из таблицы

```
void inputmatrix(float* A, DataGridView^ nametable,int n)
{
      int i, j, t;
      for (i = 0; i < n; i++)
      {
            for (j = 0; j < n; j++)
            {
            t = i * n + j;
            *(A + t) = System::Convert::ToSingle(nametable->Rows[i]->Cells[j]->Value);
            }
      }
}
```

Пример 4. Функция вывода матрицы в таблицу формы

```
void output(int n, DataGridView^ nametable,float *f)
{
      int i, j;
      for (i = 0; i < n; i++)
      {
            for (j = 0; j <=n + 1; j++)
            {
                  nametable->Rows[i]->Cells[j]->Value =(*(f + i * 10 + j)) ;
            }
      }
}
```

Пример 5. Изменение свойств компонентов окна (формы)

```
private: System::Void button4_Click(System::Object^  sender, System::EventArgs^  e)
{
      this->textBox1->ReadOnly = false;
      this->textBox2->ReadOnly = false;
      this->button5->Visible = true;
      this->button4->Visible = false;
}
private: System::Void button5_Click(System::Object^  sender, System::EventArgs^  e)
{
      this->textBox1->ReadOnly = true;
      this->textBox2->ReadOnly = true;
      this->button5->Visible = false;
      this->button4->Visible = true;
}
```

Пример 6. Открытие окна (формы) по нажатию кнопки в главном окне

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
      Incomeform ^income_form = gcnew Incomeform(tt, gg, ff, ii, ee);
      income_form->Show();
}
```

Пример 7. Заполнение таблицы в форме информацией из списка

```
// Заполнение заголовочной таблицы
int Readers::FillingOfTheTable(Readers *p_list, DataGridView^ MyTable)
{
      Readers *p;
      p = p_list;
      p->RenewalTable(MyTable);
      MyTable->ColumnCount = 8;
      MyTable->Columns[1]->HeaderCell->Value = "Фамилия";
      MyTable->Columns[2]->HeaderCell->Value = "Имя";
      .   .   .
      MyTable->Columns[7]->HeaderCell->Value = "№ чит. билета";
      if (p->GetInf().surname == "") return 1;
      // Разрешить добавление
      MyTable->AllowUserToAddRows = true;
      int str = 1;
      // Заполнение таблицы информацией из списка
      while (p != NULL)
      {
         MyTable->Rows->Add(1);
         MyTable->Rows[str - 1]->Cells[0]->Value = p->GetInf().ID;
         MyTable->Rows[str - 1]->Cells[1]->Value =
                                        gcnew String(YYY(p->GetInf().surname));
         MyTable->Rows[str - 1]->Cells[2]->Value =
                                          gcnew String(YYY(p->GetInf().name));
         .   .   .
         MyTable->Rows[str - 1]->Cells[7]->Value = p->GetInf().libCard;
         p = p->Next(p);
         str++;
      }
      MyTable->AllowUserToAddRows = false;
      MyTable->AutoResizeColumns();
      return 0;
}
```

Пример 8. Преобразование строки из ячейки таблицы в строку – массив символов **char**[].

Параметры функции: системная строка – входной, строка символов – выходной, максимальный размер выходной строки – входной.

```
#include < vcclr.h >
int MyStringToStr(String^ str, char *ch, int maxSise) // Конвертор String в char*
{                 // wch  - вспомогательная строка юникодовских (жирных)символов
      pin_ptr<const wchar_t> wch = PtrToStringChars(str);
      size_t convertedChars = 0;    //будущий реальный размер символьной строки
      size_t  sizeInBytes = ((str->Length + 1) * 2); // требуемый размер строки
      if (sizeInBytes>255) sizeInBytes=maxSise; //защита от переполнения строки
      errno_t err = 0;                              // код ошибки
      // преобразование “жирной” *wch строки в обычную *ch  (WideCharToMultiByte)
      err = wcstombs_s(&convertedChars,  ch, sizeInBytes,  wch,  sizeInBytes);
```

```
	return (int)err;                                    // возврат кода ошибки
}
```

Пример 9. Преобразование строки из ячейки таблицы в строку – массив символов **char**[] (вариант 2).

```
int MyStringToStr(String^ str, char *ch, int maxSise) // Конвертор String в char*
{
	pin_ptr<const wchar_t> wch = PtrToStringChars(str);
	size_t  sizeInBytes = ((str->Length + 1) * 2);
	if (sizeInBytes>255) sizeInBytes=maxSise;
	int err = 0;
	err= WideCharToMultiByte(CP_ACP,0,wch,-1,ch,sizeInBytes,NULL,NULL);
	return (int)!err;
}
```

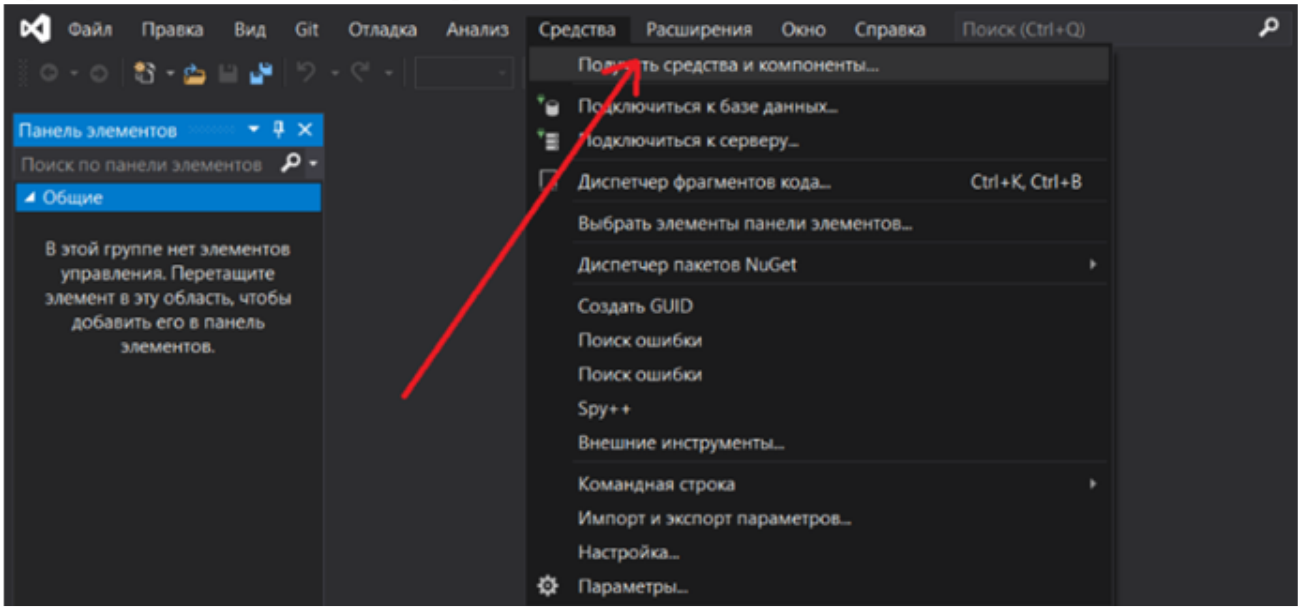
Функция WideCharToMultiByte возвращает не ошибку, а размер строки. Для совместимости вызова оставлена переменная err, которая инвертирована при возврате. Таким образом, возвращаемое значение функции также равно 0 при нормальном преобразовании.

В заключение, для каждой формы необходимо создать **заголовочный файл**, объявив пользовательские типы данных и прототипы всех используемых функций, и, по необходимости, отредактировать файл Form1.h (и других).

Запуск окна выполняется при нажатии кнопки F5.

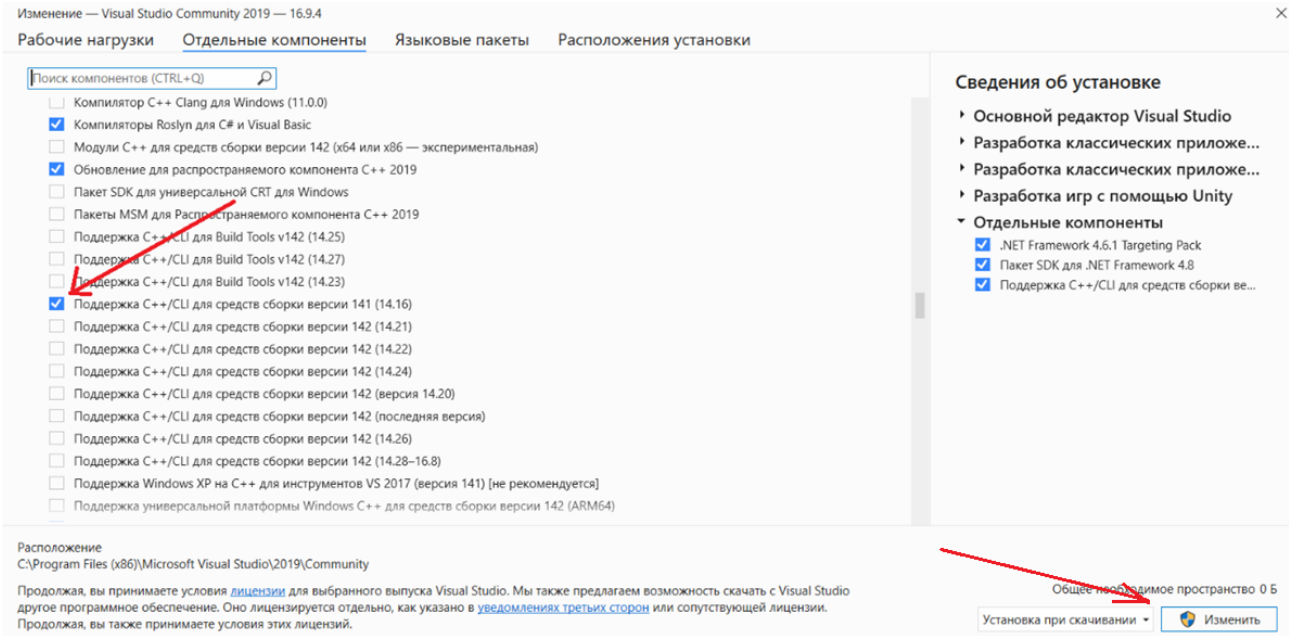
П 13. Особенности создания окон в MS Visual Studio последних версий

Перед созданием оконного приложения MS Visual Studio необходимо установить дополнительное расширение. Для этого переходим к закладке меню:
`Средства->Получить средства и компоненты`



В открывшемся окне ищем требуемую поддержку C++ для средств сборки:

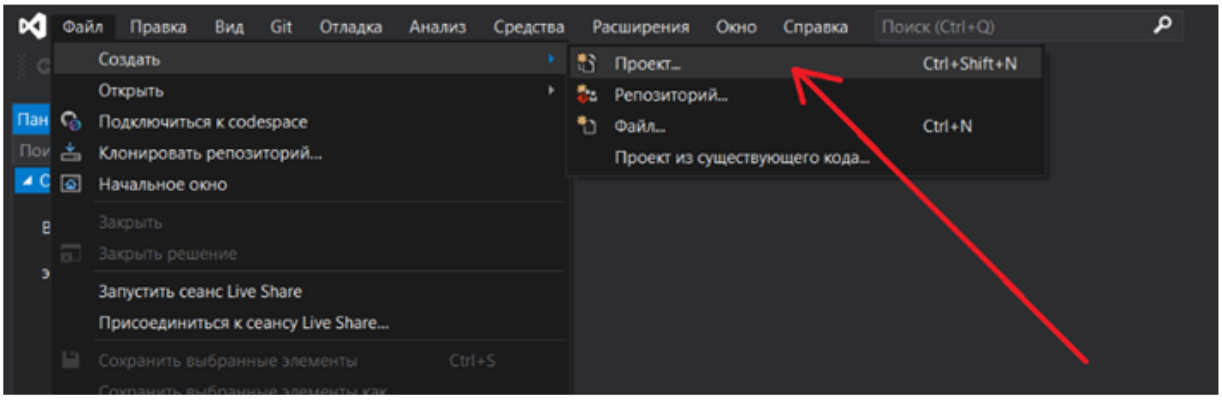
`Поддержка C++/CLI для средств сборки версии 141 (14.16)`



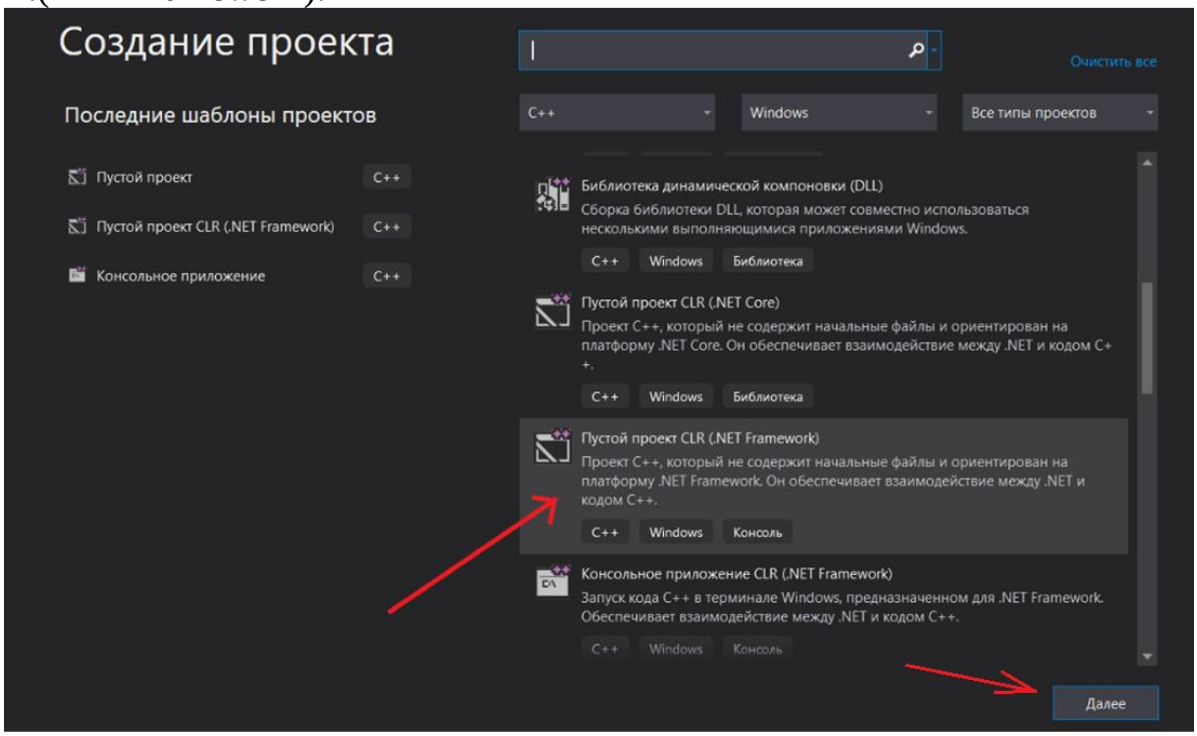
После установки "галочки" у соответствующей поддержки, нажимаем кнопку “Изменить” для установки данного расширения.

Для создания нового проекта на языке C++ последовательно выбирается команда меню:

`Файл->Создать->Проект`



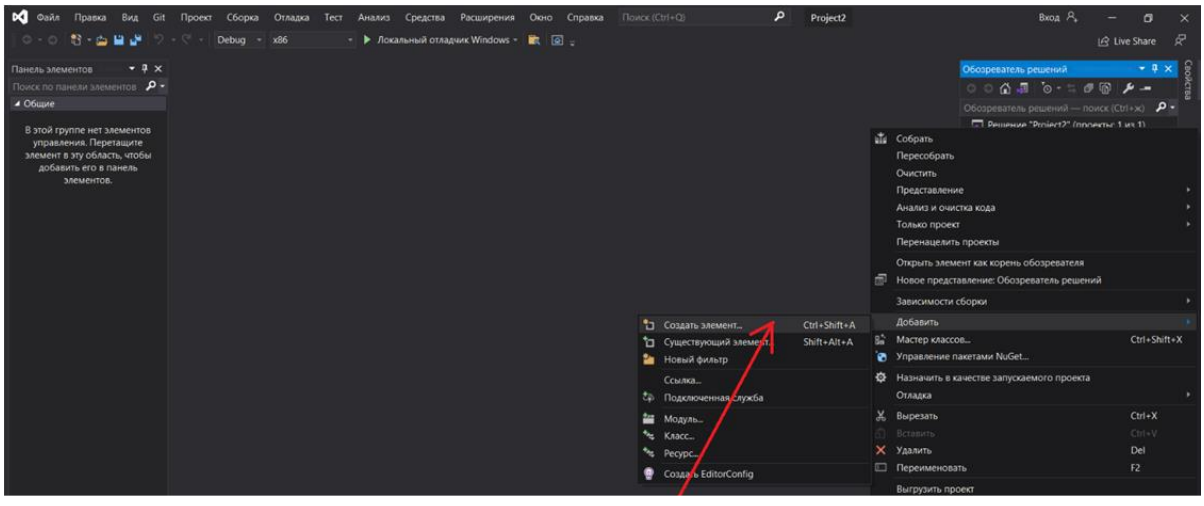
В открывшемся окне ищем необходимый шаблон – Пустой проект CLR.(NET Framework):



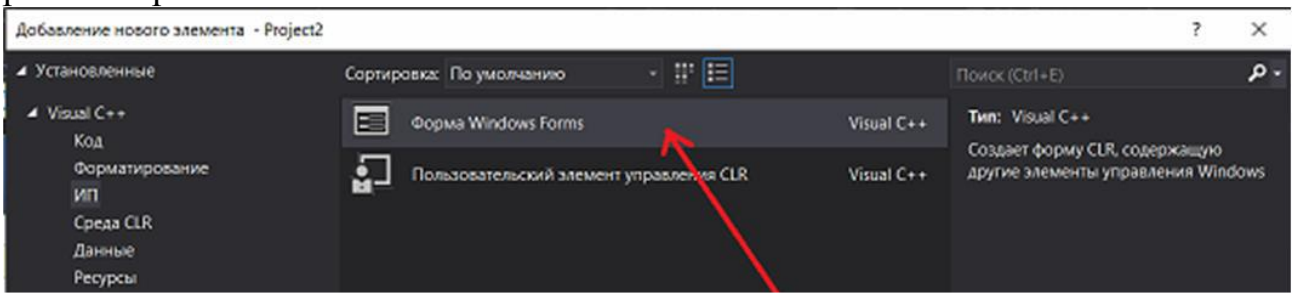
Нажимаем кнопку “Далее”.

На данном этапе имеем пустой проект “Project2”. Для добавления (создания) формы нажимаем правой кнопкой мыши по нашему проекту, активизируя контекстное меню (НЕ ПО РЕШЕНИЮ!!!), из которого выбираем команду:

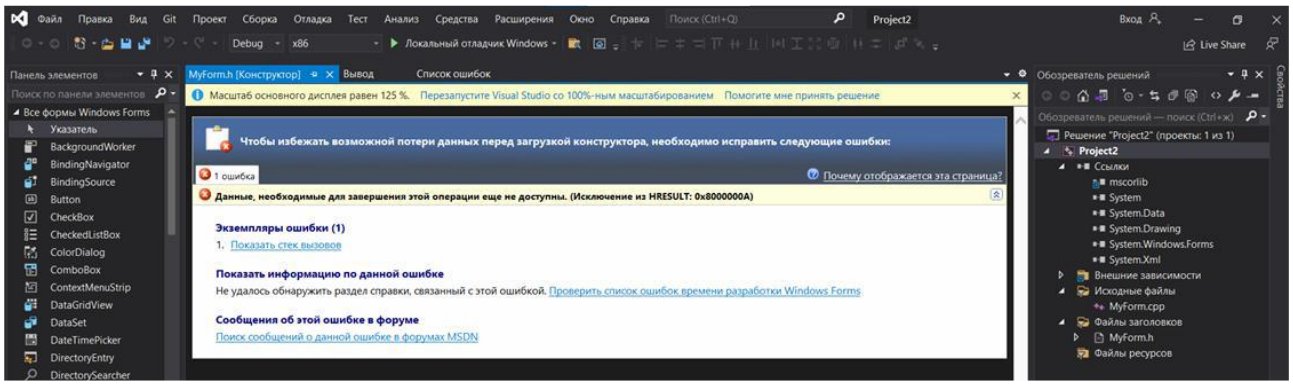
`Добавить->Создать элемент...`



В открывшемся окне в разделе “Установленные” заходим в “ИП” и выбираем “Форма Windows Forms”:

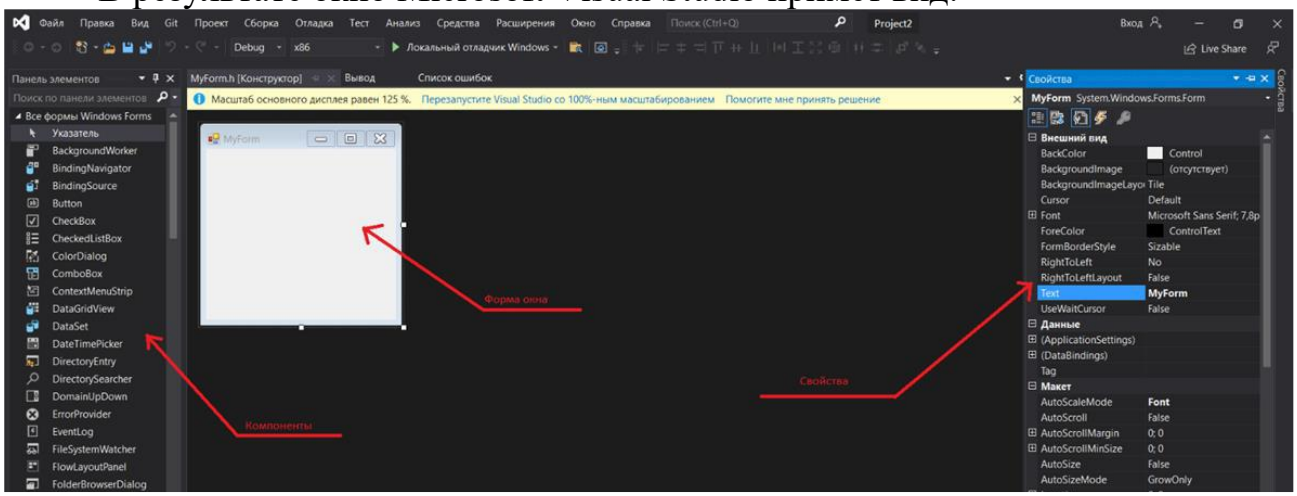


После того нажатия кнопки “ДОБАВИТЬ” обычно появляется сообщение об ошибке:



Для её устранения достаточно перезапустить MS Vusial Studio.

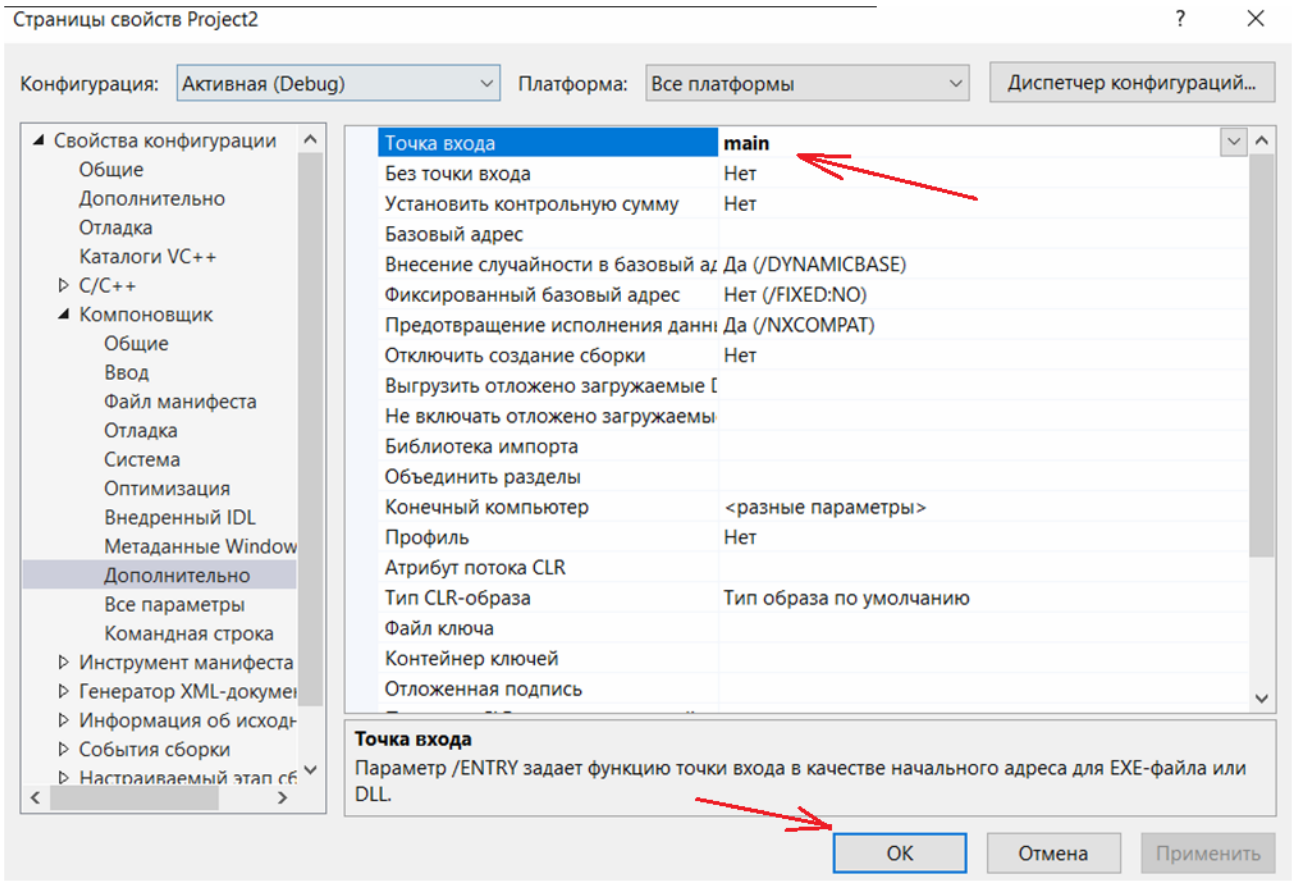
В результате окно Microsoft Visual Studio примет вид:



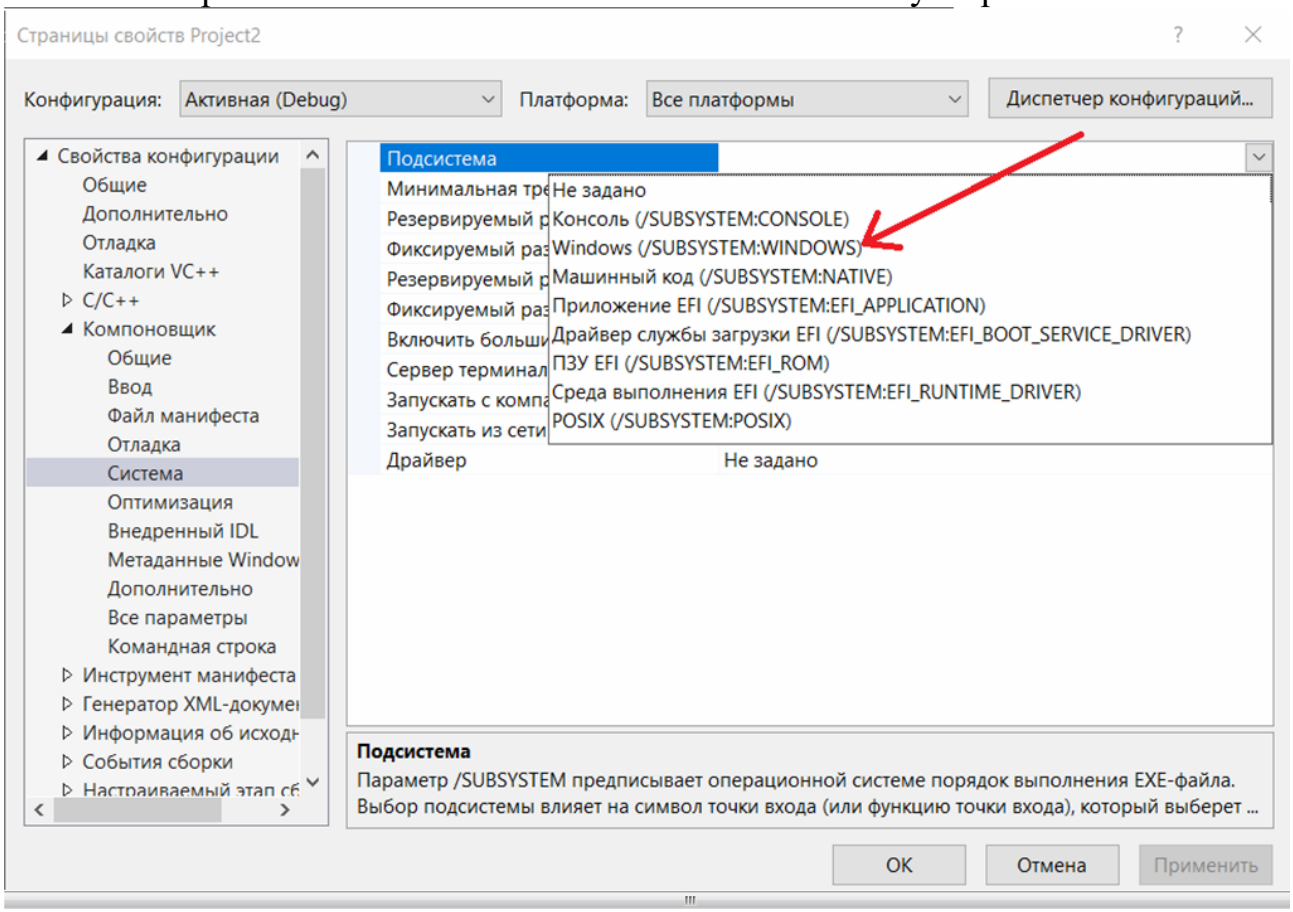
Перед началом работы необходимо проделать следующие действия: нажать правой кнопкой мыши на создаваемый проект (НЕ НА РЕШЕНИЕ!!!) и в появившемся контекстном меню выбрать команду `->Свойства.`

(данная команда является самой нижней командой этого меню)

В разделе “Компоновщик”, подразделе “Дополнительно”, в пункте “Точка входа” пишем название главной функции. В нашем случае это “main”:



Нажимаем кнопку “ОК”. Затем в подразделе “Система” в пункте “Подсистема” выбираем “Windows”. Не забываем нажать кнопку “Применить”.



Это делается для того чтобы система понимала, что работаем с оконным приложением и запуск консоли не требуется.

Однако, если выбрать “Консоль”, то вместе с нашим приложением будет запускаться консоль, где можно производить отладку в случае ошибок программы, что является полезным дополнением.

Для запуска главной формы создаётся C++ файл “MyForm.cpp” в котором прописывается открытие формы. Данное действие может быть реализовано с помощью кода:

```
#include "MyForm.h"
using namespace System;
using namespace System::Windows::Forms;

[STAThread] void main(array<String^>^ args)
    {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Project2::MyForm form;
    Application::Run(% form);
    }
```

Основная (главная) форма может быть создана и с помощью кода:

```
#include "MyForm.h"
#include "windows.h"

using namespace Project2;

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, in)
    {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Application::Run(gcnew MyForm);
    return 0;
    }
```

В этом случае в пункте “Точка входа” ничего указывать не нужно, а в пункте “Подсистема” выбирается только “Windows”, так как при выборе “Консоль” программа не скомпилируется.

Принцип дальнейших действий по созданию (разработке) оконного приложения соответствует описанному в Приложении П.12.

П 14. Задания на перегрузку

1. а) Написать программу, использующую перегрузку функций сложения целых и вещественных чисел, строк и целочисленных и вещественных массивов.

   б) Определить операторы сложения, вычитания и скалярного перемножения векторов.

2. а) Написать программу, использующую перегрузку функций перемножения целых и вещественных чисел, скалярного произведения целочисленных и вещественных векторов.

   б) Определить операторы поиска слова в тексте, подстроки в тексте.

3. а) Написать программу, использующую перегрузку функций вычисления нормы (абсолютной величины) целых и вещественных чисел, соответствующих векторов и матриц.

   б) Определить операторы вычисления количества слов в строке, символов в строке для класса строк.

4. а) Написать программу, использующую перегрузку функций нахождения метрики целых и вещественных чисел (абсолютная величина), вектора (длина), строки (количество символов).

   б) Определить операторы нахождения строки максимальной или минимальной длины.

5. а) Написать программу, использующую перегрузку функций вычисления расстояния между целыми, вещественными числами, символами, векторами (как норма разности).

   б) Определить операторы сложения, вычитания, умножения и деления комплексных чисел.

6. а) Написать программу, использующую перегрузку функций нахождения максимума в векторе, многомерном массиве, наибольшего символа в числе, строке.

   б) Определить операторы увеличения или уменьшения даты на число дней для класса дат.

7. а) Написать программу, использующую перегрузку функций перестановки значений целых и вещественных переменных, символов, строковых переменных.

   б) Определить операторы сложения и вычитания дат для класса дат.

8. а) Написать программу, использующую перегрузку функций перестановки элементов в целочисленных, вещественных и символьных векторе, матрице, строке.

   б) Определить операторы нахождения целых лет, целых месяцев, целых дней между двумя датами для класса дат.

9. а) Написать программу, использующую перегрузку функций упорядочивания по убыванию элементов векторов, строк матрицы (всех типов), символов в строке.

   б) Определить операторы сложения и деления векторов на скаляр.

10. а) Написать программу, использующую перегрузку функций упорядочивания по убыванию элементов векторов, строк матрицы (всех типов), символов в строке.

    б) Определить операторы нахождения строки максимальной или минимальной длины.

11. а) Написать программу, использующую перегрузку функций перестановки элементов в целочисленных, вещественных и символьных векторе, матрице, строке.

    б) Определить операторы сложения и вычитания матриц для класса матриц.

12. а) Написать программу, использующую перегрузку функций перестановки значений целых и вещественных переменных, символов, строковых переменных.

    б) Определить операторы умножения и деления матрицы на скаляр для класса матриц.

13. а) Написать программу, использующую перегрузку функций нахождения максимума в векторе, многомерном массиве, наибольшего символа в числе, строке.

    б) Определить операторы сложения, вычитания, умножения и деления комплексных чисел.

14. а) Написать программу, использующую перегрузку функций сложения целых и вещественных чисел, строк и целочисленных и вещественных массивов.

    б) Определить операторы сравнения, нахождения минимального и максимального по норме для класса векторов.

15. а) Написать программу, использующую перегрузку функций вычисления расстояния между целыми, вещественными числами, символами, векторами (как норма разности).

    б) Определить операторы нахождения минимального и максимального элемента вектора для класса вещественных векторов.

16. а) Написать программу, использующую перегрузку функций нахождения метрики целых и вещественных чисел (абсолютная величина), вектора (длина), строки (количество символов).

    б) Определить операторы объединения двух строк в одну и удаления из одной строки подстроки для класса строк.

17. а) Написать программу, использующую перегрузку функций вычисления нормы (абсолютной величины) целых и вещественных чисел, соответствующих векторов и матриц.

    б) Определить операторы сложения, вычитания и скалярного перемножения векторов.

18. а) Написать программу, использующую перегрузку функций перемножения целых и вещественных чисел, скалярного произведения целочисленных и вещественных векторов.

    б) Определить операторы сложения и вычитания дат для класса дат.

19. а) Написать программу, использующую перегрузку функций сравнения двух переменных целого, вещественного, символьного типа. Функция возвращает значения +1, -1, 0, в зависимости от того, какое значение больше (Либо равны друг другу).

    б) Определить операторы, изменяющие для класса дат день месяца (добавляя или вычитая целое число)

20. а) Написать программу, использующую перегрузку функций сравнения друг с другом строковых переменных либо структур с числовыми и символьными полями. Функция возвращает значения +1 или -1, если объекты различны, но их можно упорядочить по содержимому, либо 0, если содержимое параметров одинаково.

    а) Определить операторы, изменяющие для класса дат месяц (добавляя или вычитая целое число)

21. а) Написать программу, использующую перегрузку функций нахождения среднего арифметического двух целых, вещественных, либо переменных double типа.

б) Определить операторы, изменяющие все элементы массива на одно целое число (например, "+" либо "-").

22. а) Написать программу, использующую перегрузку функций нахождения среднего геометрического двух целых, вещественных, либо переменных double типа.

    б) Определить операторы сравнения двух символьных массивов (">", "<", "==")

23. а) Написать программу, использующую перегрузку функций нахождения среднего арифметического нескольких параметров вещественного типа.

    б) Определить операторы, выделяющие из даты день, месяц и год.

24. а) Написать программу, использующую перегрузку функций нахождения максимального значения среди нескольких параметров вещественного типа.

    б) Определить операторы сложения, вычитания, деления и перемножения обыкновенных дробей.

25. а) Написать программу, использующую перегрузку функций нахождения минимального значения среди нескольких параметров целого типа.

    б) Определить операторы умножения и деления обыкновенной дроби на целое число.

26. а) Написать программу, использующую перегрузку функций заполнения массивов каким-либо одним значением (например, обнулить). Массивы могут быть одномерные либо двумерные. Типы данных в массиве - символьные, целые или вещественные.

    б) Определить операторы, расставляющие элементы в одномерном массиве по убыванию или возрастанию.

П 15. Задания на создание базы данных

1. Записная книжка (фамилия, имя, отчество, муж/жен, дата рождения, место работы, должность, хобби, дети, телефон, намечаемые встречи, прошлые встречи,…). Предусмотреть перенос на новую дату не состоявшихся встреч.

2. Семейная бюджет-книга (члены семьи, статьи дохода и расхода, нормы расхода, дата операции, сумма, остаток в кассе, ...). Обязательно с формированием итогов за период и записью их с возможностью перерасчета.

3. Студенты института (фамилия, имя, отчество, группа, муж/жен, паспортные данные, адрес, №зачетной книжки, кафедры, предметы, преподаватели, расписание занятий...).

4. Сессия в институте (студенты, группы, преподаватели, расписание экзаменов и зачетов, оценки).

5. Сведения об автомобилях и их владельцах (марка, даты выпуска, покупки, регистрации, пробег, техосмотр, все о владельцах, информация о нарушениях).

6. Библиотека (книги, читатели, библиотекари - обо всех подробно, выдача книг, возврат, предельный срок пользования книгой, ...)

7. Отдел кадров предприятия (структура предприятия, штатное расписание каждого из отделов, работник, стаж, образование, начальники подразделений, телефоны, вакансии, взыскания, поощрения ...)

8. Торговая база (покупатели, поставщики, их реквизиты, товары, цены, количество, стоимость, дата получения и отгрузки, дата оплаты, задолженность...). Формировать задолженность по поставщикам и покупателям на конец отчетного периода. И на начало соответственно.

9. Пицца по интернету (ассортимент, покупатели, заказы, курьеры, доставка, оплата...).

10. Бухгалтерская база «Зарплата» (персонал, штатное расписание отделов, структура предприятия, льготы по подоходному налогу, зарплата по месяцам, начисления, выдача, удержания по исполнительному листу, оплата товаров в кредит, оплата коммунальных платежей, виды начислений, удержаний, ...)

11. Производство (номенклатура продукции, калькуляция каждого вида - стоимость изготовления и компоненты, в количественной и стоимостной

оценке, наличие на складе материалов (количество и стоимость) и готовой продукции...)

12. Пенсионный Фонд г.СПб - организация работы с работодателями и плательщиками страховых взносов (предприятия с их реквизитами, ежемесячные отчеты о доходах и страховых взносах, граждане с их данными, годовые сведения о доходах граждан, ...)

13. Налоговая инспекция какого-нибудь района г.СПб (предприятия-налогоплательщики, виды налогов, перечисления, отчеты, ...)

14. Налоговая инспекция района - работа с физическими лицами (предприятия, налогоплательщики с их реквизитами и ИНН, сведения о доходах граждан от организаций, сведения о расходах от нотариусов, декларации, ...)

15. Подготовительные курсы: список слушателей с адресами, телефонами, паспортными данными; перечень учебных групп; список преподавателей, ведущих занятия; расписание занятий; учет оплаты обучения; результаты олимпиад; результаты поступления; соответствующие отчеты...

16. База данных выпускников кафедры - ф.и.о., дата рождения, дата окончания кафедры, послужной список, семейное положение (с динамикой, детьми, супругами), адрес проживания, телефон, E-mail, связи с кафедрой (факты), интересы.

17. База данных кинолога (кличка, порода, хозяин, родословная, участие в выставках и занятые места).

18. Доставка товаров на дом (заказы, ассортимент товаров, дата, время, адрес, объема товара, оплата, курьер).

19. Справочник улиц и способы проезда к ним (улицы, адреса, виды транспорта, маршруты, остановки,...).

20. Справочник лекарств в аптеках (лекарства, аптеки, районы, наличие лекарств в аптеках, болезни, симптомы, показания,...).

21. Продажа билетов на поезда (расписание поездов, маршруты, места в поезде, пассажиры, бронирование мест, ...).

22. Отель (номера, цены, клиенты, бронирование мест, вселение,...).

23. Биржа труда (предприятия, должности, условия приема, вакансии, безработные, учет обращений по поиску работы,...).

24. Рекламная фирма (места рекламы, стоимость, рекламодатели, договора на рекламу,...).

25. Заказ билетов в театр (театры, места в театрах, репертуар, тип спектакля, стоимость билетов, покупатели,...).

26. Автомастерская (услуги, мастера, запчасти, калькуляция услуги, клиенты, заказы, ...).

27. Студенческий городок (общежития, комнаты, свободные комнаты, жильцы, оплата,...).