

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное образовательное учреждение высшего образования  
«Санкт-Петербургский государственный политехнический  
университет Петра Великого»  
Университетский политехнический колледж

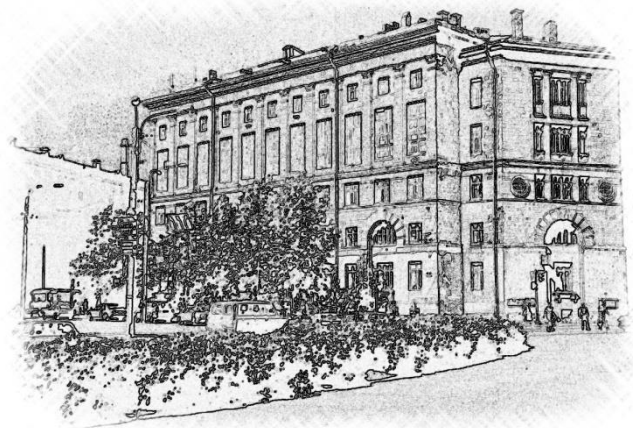
**Девятко Н.С.**

## **МЕТОДИЧЕСКИЕ УКАЗАНИЯ по учебной практике**

**УП.01.01 «Системное программирование»**

**ПМ.01**

**«Разработка программных модулей программного  
обеспечения для компьютерных систем»**



Санкт-Петербург  
2017 год

Автор:

*Девятко Наталья Сергеевна*  
преподаватель спецдисциплин,  
зам.председателя выпускающей ПЦК «Программное обеспечение»

Рецензенты:

*Ильин Юрий Петрович*  
доцент кафедры СПбГПУ, кандидат физико-математических наук

*Клименко Дмитрий Никитич*  
преподаватель СПбГПУ, кандидат наук

### **Девятко Н.С.**

Методические указания по УП.01.01 «Системное программирование» – СПб: ФГАОУ ВО СПбПУ Университетский Политехнический колледж, 2017, 57 с.

Методические указания по выполнению практических работ соответствуют Федеральному Государственному стандарту и рабочим программам профессионального модуля ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем», ориентировано на аудиторную и внеаудиторную работу студентов по формированию профессиональных компетенций в области системного программирования.

Методические указания содержат теоретические сведения, примеры решения задач для закрепления материала, и направлен на получение практических навыков по системному программированию: разработки алгоритмов и программ, созданию приложений в среде программирования Visual Studio на языке C/C++, отладке, тестированию, оптимизации программных модулей.

Методические указания предназначены для студентов по специальностям 09.02.03 «Программирование в компьютерных системах» (базовой подготовки) средних профессиональных учебных заведений.

## Содержание

Требования к оформлению отчета по практике УП.01.01 «Системное программирование»	4
Составление тестовых данных на основе потока данных .....	5
Генерация исключений .....	8
Арифметические операции языка С .....	9
Операции присваивания.....	9
Программы линейной структуры .....	10
Операции сравнения языка Си .....	12
Логические операции .....	12
Сложные условия.....	12
Условный оператор (полного ветвления).....	12
Условный оператор (сокращенного ветвления) .....	12
Вложение условий .....	13
Программирование циклических алгоритмов .....	18
Оператор цикла с известным числом повторений.....	18
Оператор цикла с неизвестным числом повторений.....	21
Обработка строк.....	22
Указатели .....	23
Обработка массивов .....	26
Функции программиста.....	32
Модуль программиста (библиотека).....	35
Обработка текстовых файлов .....	37
Обработка двоичных файлов .....	39
Динамические структуры данных.....	42
Однонаправленные (односвязные) списки.....	43
Создание однонаправленного списка .....	44
Вставка элемента в однонаправленный список.....	45
Удаление элемента из однонаправленного списка .....	46
Поиск элемента в однонаправленном списке .....	48
Стеки.....	48
Ключевые термины .....	51
Краткие итоги.....	52
Работа с файловой системой.....	55
Организация многопоточной обработки данных .....	57

## **Требования к оформлению отчета по практике УП.01.01«Системное программирование»**

3 курс  
Специальность 09.02.03

### ***Отчет должен содержать:***

1. Название темы
2. Цель работы
3. Математическую формулировку задачи (включая описание входных данных, функциональные характеристики программы, описание выходных данных)
4. Схемы основных функций, составленные в соответствии с ЕСПД
5. Текст программного модуля (модулей) с **комментариями** хода решения задачи
6. План тестирования (тестовый сценарий) на основе потока данных
7. Результаты испытаний (скриншоты)

*Все отчёты сохранять в один документ, с титульным листом, оглавлением, нумерацией страниц, колонтитулами.*

### ***Общие требования по оформлению:***

- Отчет должен содержать стандартный титульный лист, задание на практику, дневник прохождения практики и сам отчёт, сверху на отчёт кладем аттестационный лист (не прошивая). Все стандартные документы можно найти в папке «Документы для практики»;
- Все листы отчета, кроме титульного, должны быть пронумерованы;
- Все листы отчета, исключая титульный лист, задание на практику и дневник, должны иметь колонтитулы; колонтитул содержит ФИО студента и номер группы;
- Все заголовки одного уровня должны быть оформлены одним стилем (заголовок 1, заголовок 2 и т.д.), проверьте, чтобы не было «висячих» заголовков;
- Текст всех заданий должен быть оформлен одинаковым стилем, гарнитура Times New Roman, размер шрифта 12 пт., выравнивание основного текста по ширине; тексты программ имеют другую гарнитуру (установленную в Visual Studio);
- Сделать авто собираемое оглавление, включающее только темы (заголовки 1-го уровня) со ссылками на нужные страницы.

## Составление тестовых данных на основе потока данных

1. Для того чтобы составить тестовый набор на основе потока данных, необходимо разбить входной поток данных на классы эквивалентности. Каждому классу присваивается номер. Затем подготавливаются такие наборы входных данных, которые позволяют проверить выполнение программы по каждому классу эквивалентности хотя бы один раз.
2. Внимательно изучите условие задачи. Составьте структуру входных и выходных данных (введите обозначения, укажите типы данных). Подумайте, какие ограничения накладываются на входные данные.
3. Разбейте входной поток данных на классы эквивалентности. Присвойте каждому классу эквивалентности свой номер.
4. Запишите описание каждого класса в таблицу, в графу «Классы эквивалентности».
5. Запишите в таблицу наборы входных данных, соответствующих каждому классу эквивалентности, в графу «Тестовый набор».
6. В графу «Ожидаемые результаты» внесите результаты, рассчитанные вручную, для сравнения с теми результатами, которые будет выдавать программа.

*Условие ввода может задавать:*

- 1) Определенное значение
- 2) Диапазон значений
- 3) Множество конкретных величин
- 4) Булево условие

*Правило формирования основных классов эквивалентности:*

- 1) Если условие ввода задает диапазон значений  $[a; b]$ , то формируются один допустимый и два недопустимых класса эквивалентности (справа и слева от интервала)
- 2) Если условие ввода задает конкретное значение  $a$ , то определяется один допустимый (при  $x=a$ ) и два недопустимых класса эквивалентности (при  $x < a$  и при  $x > a$ )
- 3) Если условие ввода задает множество значений  $x \in \{a, b, c\}$ , то определяется один допустимый и один недопустимый класс эквивалентности (при  $x \neq a$  и  $x \neq b$  и  $x \neq c$ )
- 4) Если условие ввода задает булево значение, то определяется один допустимый  $\{true\}$  и один недопустимый  $\{false\}$  класс эквивалентности

*Пример:*

Вычислить значение функции при следующих условиях

$$F(x) = \begin{cases} x^2 - 1 & \text{если } 0 < x \leq 10 \\ 1 & \text{если } x = 0 \\ x & \text{если } -10 \leq x < 0 \\ \sqrt{x^2 + 1} & \end{cases}$$

Подготовим тестовый сценарий и тестовый набор на основе потока данных. Известно, что допустимые значения входных данных лежат в интервале от -10 до 10. Тогда недопустимые данные будут справа и слева от этого интервала. Имеем 2 класса недопустимых значений. Так как в допустимом диапазоне расчет производится по разным формулам, разобьем класс допустимых значений на 3 отдельных класса. Добавим ещё 2 дополнительных класса для проверки граничных условий (на границах допустимого интервала и вблизи границ).

Таблица 1

№	Классы эквивалентности	Тестовый набор	Ожидаемые результаты
1	Класс допустимых значений $x \in [-10;10], x > 0$	$X=5$	$F(x) = 24$ При положительных значениях $X$ в допустимом диапазоне должно вычисляться значение функции $F(x)$ по формуле $x^2-1$
2	Класс допустимых значений $x \in [-10;10], x < 0$	$X=-1,5$	$F(x) = -0,832$ При отрицательных значениях $X$ в допустимом диапазоне должно вычисляться значение функции $F(x)$ по формуле $\frac{x}{\sqrt{x^2+1}}$
3	Класс допустимых значений $x \in [-10;10], x = 0$	$X=0$	$F(x) = 1$ При $X=0$ по условию значение функции $F(x)=1$
4	Класс допустимых значений $x \in [-10;10]$ , граничные условия	$X = -10$	$F(x) = -0,995$ При $X=-10$ должно вычисляться значение функции $F(x)$ по формуле $\frac{x}{\sqrt{x^2+1}}$ $X=10$ $F(x) = 99$ При $X=10$ должно вычисляться значение функции $F(x)$ по формуле $x^2-1$
5	Класс недопустимых значений $x \in (-\infty;-10)$ , $x \in (10;+\infty)$ граничные условия	$X = -10.001$ $X=10.001$	<i>Ожидаемый ответ:</i> При недопустимых значениях $X$ должен быть выдан ответ «Функция не определена»
6	Класс недопустимых значений $x \in (-\infty;-10)$ ,	$X=-25$	<i>Ожидаемый ответ:</i> При недопустимых значениях $X$ должен быть выдан ответ «Функция не определена»
7	Класс недопустимых значений $x \in (10;+\infty)$	$X=124$	<i>Ожидаемый ответ:</i> При недопустимых значениях $X$ должен быть выдан ответ «Функция не определена»

## Как проверить данные, введенные пользователем

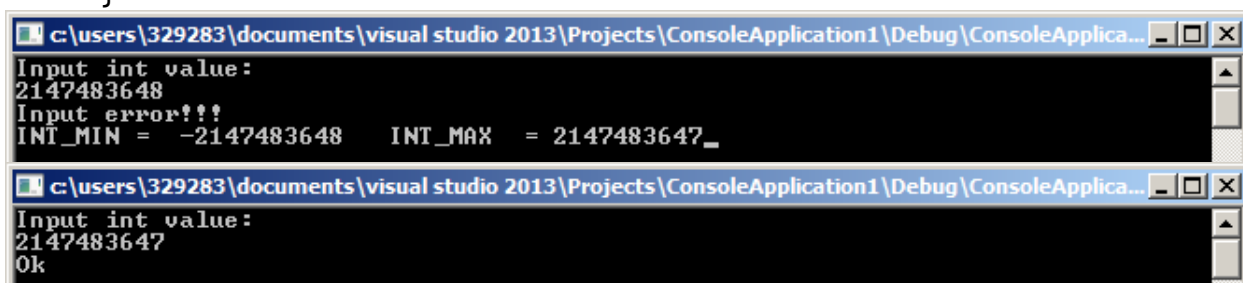
Например, допустимыми являются все значения типа `int`. Проверим, принадлежат ли введенные данные диапазону этого типа?

Воспользуемся стандартным классом `std` (C++), в котором определены все максимальные и минимальные значения для каждого типа.

```
#include <iostream>
#include <conio.h>
#include <limits.h>

using namespace std;

void main()
{
    int i; // проверит любой тип данных
    cout << "Input int value:" << endl;
    cin >> i;
    if (!cin)
    {
        cout << "Input error!!!"<<endl;
        cout <<"INT_MIN = " << INT_MIN <<"    INT_MAX = " <<
INT_MAX;
    }
    else
        cout << "Ok";
    _getch();
}
```



Тип	Размер в байтах	Диапазон значений
char	1	от -128 до 126
unsigned char	1	от 0 до 255
short	2	от -32 768 до 32 767
unsigned short	2	от 0 до 65536
enum	2	от -2 147 483 648 до 2 147 483 647
long	4	от -2 147 483 648 до 2 147 483 647
unsigned long	4	от 0 до 4 294 967 295
int	4	от -2 147 483 648 до 2 147 483 647
unsigned int	4	от 0 до 4 294 967 295
float	4	от $3,4 * 10^{-38}$ до $3,4 * 10^{38}$
double	8	от $1,7 * 10^{-308}$ до $1,7 * 10^{308}$
long double	10	от $3,4 * 10^{-4932}$ до $1,1 * 10^{4932}$
bool	1	true или false

## Генерация исключений

Что важно запомнить об исключениях:

- try-блок — так называемый блок повторных попыток. В нем надо располагать код, который может привести к ошибке и аварийному закрытию программы;
- throw генерирует исключение. То, что остановит работу try-блока и приведет к выполнению кода catch-блока. Тип исключения должен соответствовать типу принимаемого аргумента catch-блока;
- catch-блок — улавливающий блок, поймает то, что определил throw и выполнит свой код. Этот блок должен располагаться непосредственно под try-блоком.
- если в try-блоке исключение не генерировалось, catch-блок не сработает. Программа его обойдет.

```
int num1, num2;
cout << "Введите значение num1: ";
cin >> num1;
cout << "Введите значение num2: ";
cin >> num2;
cout << "num1 + num2 = " << num1 + num2 << endl;
cout << "num1 / num2 = ";

try //код, который может привести к ошибке, располагается тут
{
    if (num2 == 0)
    {
        throw "Ошибка - на 0 делить нельзя!!!!"; //генерировать символьную строку
    }
    cout << num1 / num2 << endl;
}
catch (char *str) //сюда передается строка
{
    cout << str << endl;
}
```

Если вы разрабатываете собственную функцию, надо исходить из того, что параметры, поданные ей на вход, могут быть ошибочными. Внутри тела функции необходимо проверить входные данные, и если корректный результат получить невозможно, надо сгенерировать исключение, которой может быть перехвачено в главной функции.

```
float division(int n1, int n2)
{
    if (n2 == 0)
    {
        throw 99; //генерировать ошибку
    }
    return (float) n1 / n2;
}

void main()
{
    try //вызов функции может привести к ошибке
    {
        cout << division(5, 0);
        cout << endl;
    }
    catch (int i) //сюда передается номер ошибки
    {
        cout << "Ошибка №" << i << " Деление на ноль!" << endl;
    }
}
```



## Арифметические операции языка C

Знак	Операция	Типы операндов	Тип результата
+	Сложение	Целые Хотя бы один вещественный	Целый Вещественный
—	Вычитание	Целые Хотя бы один вещественный	Целый Вещественный
*	Умножение	Целые Хотя бы один вещественный	Целый Вещественный
/	Деление	Только целые Целые и вещественные (хотя бы одно вещественное)	Целый Вещественный
%	Остаток от деления	Только целые	Целый

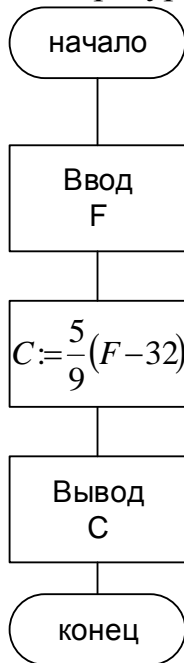
## Операции присваивания

Назначение	Инструкция	Соответствующая обычная инструкция
Увеличение на 1	$x++$	$x = x + 1$
Уменьшение на 1	$x--$	$x = x - 1$
Добавление $y$	$x += y$	$x = x + y$
Уменьшение на $y$	$x -= y$	$x = x - y$
Умножение на $y$	$x *= y$	$x = x * y$
Деление на $y$	$x /= y$	$x = x / y$
Остаток от деления $x$ на $y$	$x \% = y$	$x = x \% y$

## Программы линейной структуры

### Пример 1.

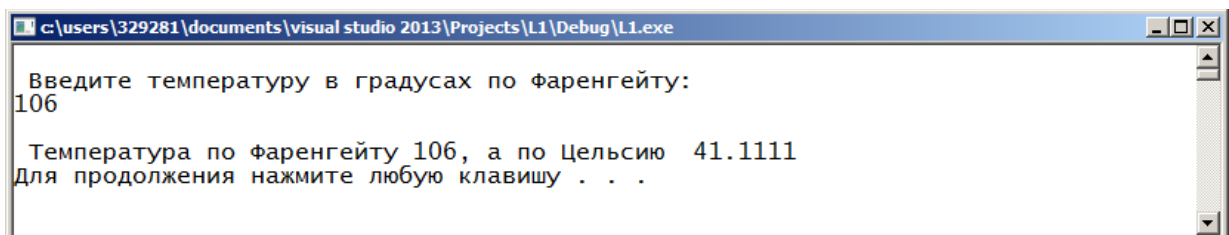
Задать с клавиатуры температуру в градусах по Фаренгейту. Перевести температуру в градусы по Цельсию.



```
#include "stdafx.h"
#include "iostream"
#include "clocale"
#include "windows.h"

using namespace std;

void main()
{
    float f, c;    //объявление переменных вещественного типа
    setlocale(LC_ALL, "RUS");
    cout<<"\n Введите температуру в градусах по Фаренгейту: \n";
    cin>>f;
    c = 5 * (f - 32) / 9;
    cout<<"\n Температура по Фаренгейту "<<f<<"", а по Цельсию "<<c<<endl;
    system("pause");
}
```



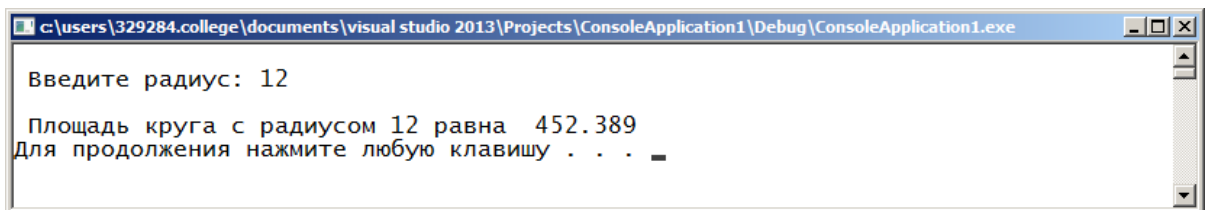
## Пример 2.

Вычислить площадь круга по заданному значению радиуса.

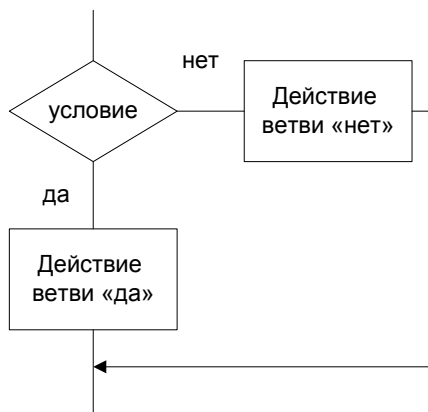
```
#include "stdafx.h"
#include "iostream"
#include "clocale"
#include "windows.h"
#define _USE_MATH_DEFINES // for C++
#include <math.h>

using namespace std;

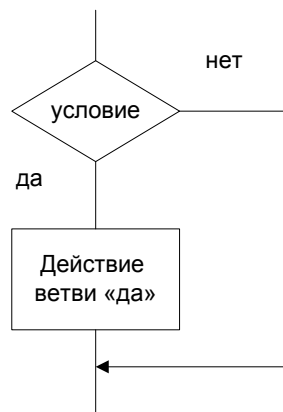
void main()
{
    float r, S; //объявление переменных вещественного типа
    setlocale(LC_ALL, "RUS");
    cout<<"\n Введите радиус: ";
    cin>>r;
    S = M_PI*r*r; //используем константу PI
    cout<<"\n Площадь круга с радиусом "<<r<<" равна "<<S<<endl;
    system("pause");
}
```



## Организация ветвления



Структура полного ветвления



Структура сокращенного ветвления

## Операции сравнения языка Си

>	больше
<	меньше
>=	больше либо равно
<=	меньше либо равно
==	проверка на равенство
!=	проверка на неравенство

## Логические операции

Обозначение операции	Название операции	Выполняемое действие
&&	И	Логическое умножение
	ИЛИ	Логическое сложение
!	НЕ	Логическое отрицание

## Сложные условия

(a>-5 && a<5) - и

(a<=-5 || a>=5) - или

## Условный оператор (полного ветвления)

```
if (условие)
{   // операторы ветви «да»   }
else
{   // операторы ветви «нет»  }
```

```
if (a>b)
{ a=a+1; }
else
{ b=b+1; }
```

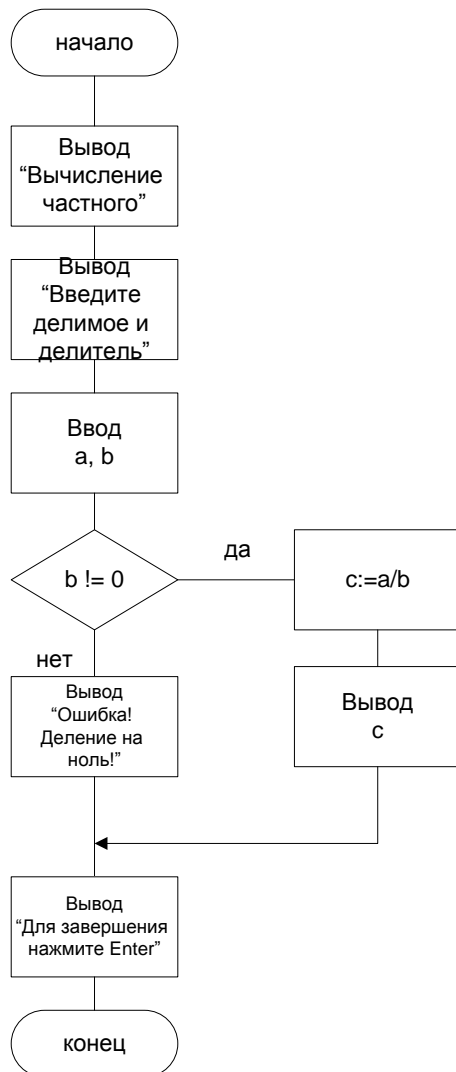
## Условный оператор (сокращенного ветвления)

```
if (условие)
```

```
{ // операторы ветви «да» }
```

### Пример 3.

Вычислить значение частного двух чисел.



## Вложение условий

### 1). if (условие 1)

```
{ операторы ветви «да1» }  
else  
{ if (условие 2)  
  { операторы ветви «да2» }  
  else  
  { операторы ветви «нет2» } }
```

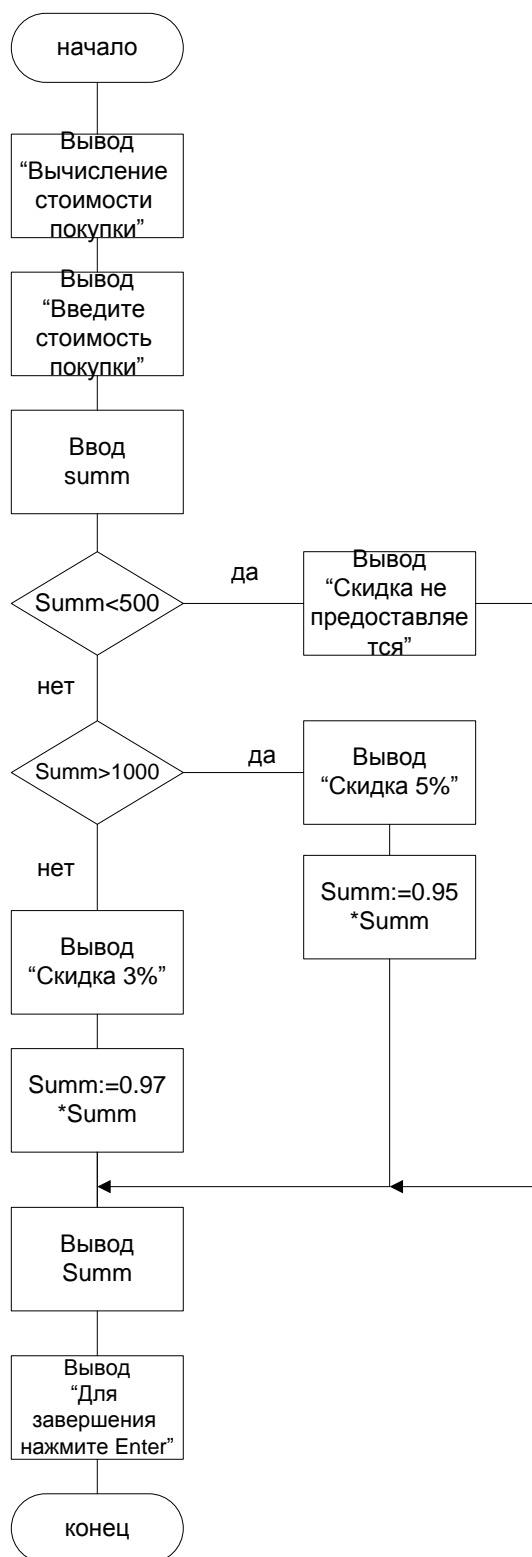
### 2). if (условие 1)

```
{ if (условие 2)  
  { операторы ветви «да2» }  
  else  
  { операторы ветви «нет2» }  
else  
{ операторы ветви «нет1» } }
```



#### Пример 4.

Вычислить стоимость покупки с учетом скидки. Скидка 3% предоставляется на сумму свыше 500 рублей, скидка 5% на сумму свыше 1000 рублей.



```

#include "stdafx.h"
#include "iostream"
#include "clocale"
#include "math.h"
#include "windows.h"

using namespace std;

void main()
{
    float summ;    // стоимость покупки
    setlocale(LC_ALL, "RUS");
    cout<<"\n * Вычисление стоимости покупки * \n";
    cout<<"\n Введите стоимость покупки: \n";
    cin>>summ;
    if (summ<500) cout<<"\n Скидка не предоставляется \n";
    else
        if (summ>1000)
        {
            cout<<"\n Скидка 5% \n";
            summ = summ*0.95;
        }
        else
        {
            cout<<"\n Скидка 3% \n";
            summ = summ*0.97;
        }
    cout<<"\n summ= "<<summ<<endl;
    system("pause");
}

```

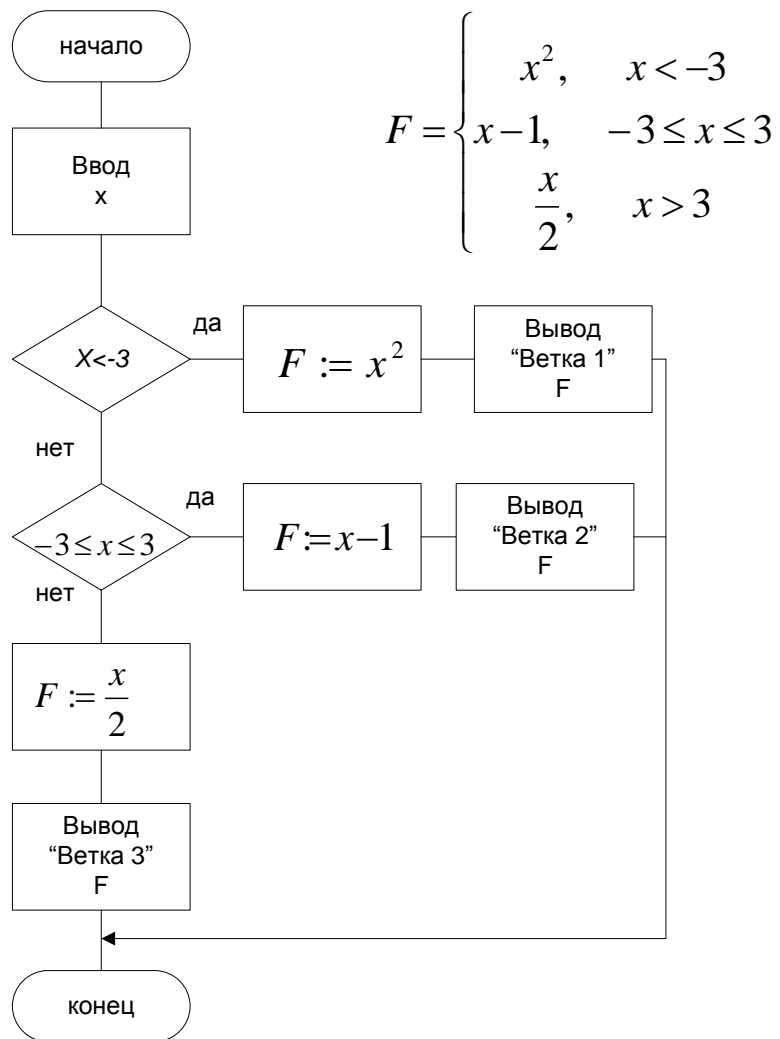
```

c:\users\329281\documents\visual studio 2013\Projects\L1\Debug\L1.exe
* Вычисление стоимости покупки *
Введите стоимость покупки:
2080
Скидка 5%
summ= 1976
Для продолжения нажмите любую клавишу . . .

```



### Пример 5.



## Программирование циклических алгоритмов

Циклы по своей структуре подразделяются на:

- Циклы с известным числом повторений (когда заранее известно, сколько раз будет повторяться цикл)
- Циклы с неизвестным числом повторений (в этом случае цикл завершается при достижении некоторых условий)

### Оператор цикла с известным числом повторений

for (инициализация; условие; изменение) оператор;

#### Например:

```
for (int i = 0; i < 10; i++)
```

```
for (int i = 9; i >= 0; i--)
```

```
unsigned char ch;
```

```
for (ch = 'a'; ch <= 'z'; ch++) cout<<ch<<"\n";
```

#### Пример 6.

Вывод на экран таблицы умножения на 7

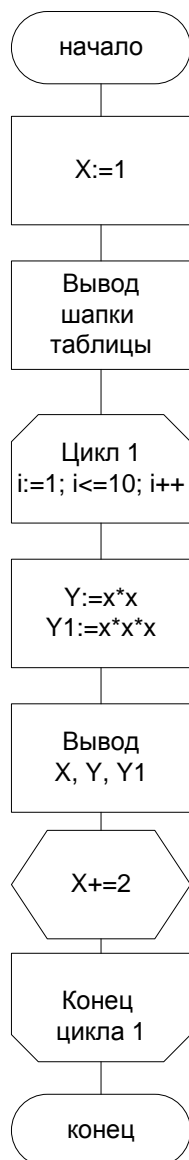
```
for (int x = 0; x <= 10; x++)          //цикл перебора аргумента
{
    y = x*7;        //расчет значения функций для каждого значения x
    cout<<x <<"\t"<< y<<"\n";        //вывод каждой строки таблицы
}
```

### Пример 7.

Вывести на экран таблицу значений  $y=x^2$  и  $y1=x^3$  только для нечетных значений  $x$  в интервале от 1 до 21.

//Фрагмент программы (цикл)

```
int y, y1, x = 1;
for (int i = 1; i <= 10; i++)      //цикл перебора номеров строк таблицы
{
    y = x*x;      //расчет значения функций для каждого значения x
    y1 = x*x*x;
    cout<<x <<"\t"<< y<<"\t" << y1<<"\n";    //вывод каждой строки таблицы
    x += 2;      //добавление шага
}
```



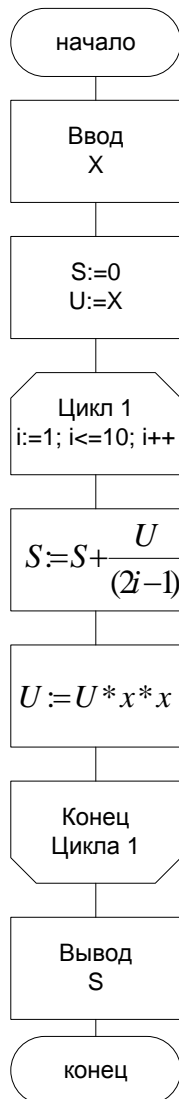
c:\users\329281\documents\visual studio 2013\Projects\L1\Debug\L1.exe

1	1	1
3	9	27
5	25	125
7	49	343
9	81	729
11	121	1331
13	169	2197
15	225	3375
17	289	4913
19	361	6859

Для продолжения нажмите любую клавишу . . .

### Пример 8.

Вычислить сумму ряда:  $S = x + \frac{x^3}{3} + \frac{x^5}{5} + \dots + \frac{x^{19}}{19}$



//Фрагмент программы (цикл)

```
for (int i = 1, S = 0, U = x; i <= 19; i += 2)
{
    S += U / i;
    U *= x*x;
}
```

### Пример 9.

Вычислить сумму 10 чисел, вводимых с клавиатуры.

```
int i, x, sum = 0; //сумма до пересчета равна нулю
for (i = 1; i <= 10; i++) //цикл повторяется 10 раз
{
    cin>>x; //ввод очередного значения x
    sum += x; //добавление x в общую сумму
}
cout<<"Сумма чисел "<< sum<<endl;
```

## Оператор цикла с неизвестным числом повторений

### 1). Цикл с предусловием

```
while (условие)
{
    операторы цикла
}
```

### 2). Цикл с постусловием

```
do {
    операторы цикла
} while (условие);
```

### Пример 10.

Сколько степеней двойки надо просуммировать, чтобы сумма превысила 10000?

```
int    sum = 0, U = 1, n = 0; //начальные значения переменных
while (sum <= 10000)          //цикл повторяется, пока сумма меньше 10000
{
    sum += U;                  //добавляем в сумму очередное слагаемое
    n++;                       //считаем количество слагаемых
    U *= 2;                    //увеличение степени двойки
}
cout << n << endl; //вывод ответа, сколько степеней двойки просуммировано
```

### Пример 11.

Вычислить сумму чисел, вводимых с клавиатуры, окончание работы – ввод числа ноль.

```
int    x, sum = 0, n = 0;
do
{
    cin>>x;                    //ввод очередного значения x
    sum += x;                  //добавление x в общую сумму
    n++;                       //считаем количество введенных чисел
} while (x != 0); //цикл с постусловием, продолжается пока x не равен нулю
cout<<"Было введено "<<n<<" чисел. Сумма чисел "<<sum<<endl;
```

## Обработка строк

Строка – это последовательность символов произвольной длины.

### Пример 12.

```
char S1[80], S2[80] = "\0";           //объявление строк
int i, k1 = 0, k2 = 0;
gets(S1);                             //ввод строки с клавиатуры
for (i = 0; i <= strlen(S1); i++)      //цикл перебирает номера символов строки
    if (S1[i] == 'a') k1++;           //подсчет количества букв «а» в строке

for (i = 0; i <= strlen(S1); i++)
    if (S1[i] >= '0' && S1[i] <= '9') k2++;
                                     //подсчет количества встречающихся цифр в строке

int n = 0;
for (i = 0; i <= strlen(S1); i++)
    if (S1[i] != ' ') { S2[n] = S1[i]; n++; }
    //перенесем во вторую строку все символы первой строки, кроме пробелов
```

## Указатели

Указатель – это переменная, которая содержит адрес некоторого объекта.

*Объявление указателя:*

**тип \*имя переменной**

При этом тип определяет тип объекта, на который указывает указатель (адрес которого содержит).

*Операции над указателями:*

&     взять адрес

\*     взять значение, расположенное по указанному адресу

```
char s[80];           //объявление строки
char S1[80], S2[80] = "\0"; //объявление строк
char *p1, *p2, *p3;   //объявление указателей
int k = 0;             //переменная для подсчета количества пробелов
gets(s);              //ввод строки
int L1 = strlen(s);    //определение длины строки
p1 = &s[0];            //ставим указатель p1 на начало строки
p2 = &s[L1];           //ставим указатель p2 на конец строки
for (char* p = p1; p <= p2; p++)
    if (*p == ' ') k++; //подсчет количества пробелов
p1 = &S1[0];           // устанавливаем указатели
p3 = &S2[0];
while (p1)             //цикл, пока не дошли до конца 1 строки
{
    if (*p1 < '0' || *p1 > '9')
    {
        *p3 = *p1; // присваиваем значениям 2 строки значения 1 строки
        p3++;      //добавляем адрес строки 2
    }
    p1++;           // добавляем адрес строки 1
}
*p3 = '\0';
```

### Пример 13.

Перенести во вторую строку символы исходной строки, стоящие между первым и последним пробелом.

```
#include "stdafx.h"
#include "iostream"
#include "clocale"
#include "windows.h"

using namespace std;

void main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    char s1[255], s2[255]; //объявление строк
```

```

char *p1, *p2, *p, *pp; //объявление указателей
cout << "введите строку"<<endl;
gets_s(s1);           //ввод исходной строки
p1 = &s1[0];
p = NULL;
while (*p1 != '\0')
{
    if (*p1 == ' ') //найдем позицию первого пробела
    {
        p = p1;
        break;
    }
    p1++;
}

p1 = &s1[0];
pp = NULL;
while (*p1 != '\0')
{
    if (*p1 == ' ') pp = p1; //найдем позицию последнего пробела
    p1++;
}
if (p == NULL || pp == NULL)
{
    cout << "Мало пробелов!"<<endl;
    system("pause");
    return;
}
p1 = s1;
p2 = s2;
while (p1 != '\0') //от позиции первого пробела до последнего
{
    if ((p1 >= p) && (p1 <= pp))
    {
        *p2 = *p1; //перенесём символы строки во вторую строку
        p2++;
    }
    p1++;
}
*p2 = '\0';
cout << s2 << endl;
system("pause");
}

```

```

c:\users\329284.college\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
введите строку
Привет, мир! Доброго дня!
мир! Доброго
Для продолжения нажмите любую клавишу . . .

```



#### Пример 14.

Присоединить вторую половину первой заданной строки ко второй строке.

```
#include "stdafx.h"
#include "iostream"
#include "clocale"
#include "windows.h"

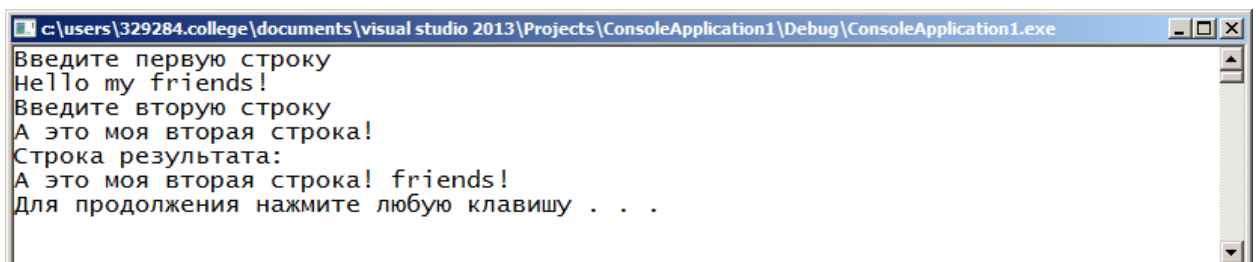
using namespace std;

void main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    int L1, L2, q;
    char str1[80], str2[80];           // объявление строк
    char *p1, *p2, *p3;               // объявление указателей

    cout<<"Введите первую строку"<<endl;
    gets_s(str1);
    cout << "Введите вторую строку"<<endl;
    gets_s(str2);

    L1 = strlen(str1);                //получение длин строк
    L2 = strlen(str2);
    q = L1 / 2;                       //определяем середину 1 строки
    p1 = &str1[q];                    //устанавливаем указатели
    p2 = &str2[L2];
    p3 = &str1[L1];

    while (p1<p3)                     //цикл, пока не дошли до конца 1 строки
    {
        *p2 = *p1; //присваиваем 2 строке значения 1 строки
        p1++;      //добавляем адрес 1 строки
        p2++;      //добавляем адрес 2 строки
    }
    *p2 = '\0';
    cout <<"Строка результата:"<<endl<<str2<< endl;
    system("pause");
}
```



```
c:\users\329284.college\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Введите первую строку
Hello my friends!
Введите вторую строку
А это моя вторая строка!
Строка результата:
А это моя вторая строка! friends!
Для продолжения нажмите любую клавишу . . .
```

## Обработка массивов

Массивом называется упорядоченная совокупность однотипных данных, имеющая общее имя.

*Объявление массива:*

```
int m[10];           //одномерный массив из 10 целых элементов
float a[4][5];       //двумерный массив из 4 строк и 5 столбцов (таблица)
```

Элементы массива пронумерованы по порядку и хранятся в соседних ячейках памяти. Поэтому к элементам массива можно обращаться по номеру: m[0] m[1] m[2] и т.д. или через указатели.

Любая обработка массива, в том числе ввод и вывод, происходит при помощи цикла. Двумерные массивы обрабатываются при помощи двойного вложенного цикла.

### Пример 15(а).

Вычислить сумму и количество элементов одномерного массива, значения которых кратны 3.

```
#include "stdafx.h"
#include "iostream"
#include "clocale"
#include "windows.h"
#include <time.h>

using namespace std;

void main()
{
    int m[10]; //объявление одномерного массива из 10 элементов
    int i, sum = 0, k = 0;
    srand(time(NULL)); //инициализация генератора случайных чисел

    for (i = 0; i<10; i++) //цикл для перебора номеров массива
    {
        m[i] = rand()%100 - 50; //элементам массива присвоим случайные
значения
        cout<< m[i]<<" "; //вывод элементов массива на экран
    }
    cout << endl; //перевод строки
    for (i = 0; i<10; i++) // цикл для перебора номеров массива
    {
        if (m[i] % 3 == 0)
        {
            k++; //вычисляем количество элементов кратных 3
            sum += m[i]; //вычисляем сумму элементов кратных 3
        }
    }
    cout<<"k = "<<k<<" sum = "<<sum<<endl; //вывод ответов
```

```

        delete[] m; // освобождение памяти;
        system("pause");
    }

```

```

c:\users\329284.college\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
33 -23 -24 -8 -6 29 26 49 -23 -27
k = 4 sum = -24
Для продолжения нажмите любую клавишу . . .

```

### Пример 15(б).

Приведем пример программы с тем же заданием, но массив будем обрабатывать с помощью указателей.

```

void main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    int *m; //объявление одномерного динамического массива
    int i, N, sum = 0, k = 0;
    srand(time(NULL)); //инициализация генератора случайных чисел
    cout << "Введите размер массива" << endl;
    cin >> N;
    m = new int[N];
    for (i = 0; i < 10; i++) //цикл для перебора номеров массива
    {
        *(m + i) = rand()%100 - 50; //элементам массива присвоим
        случайные значения
        cout << *(m + i) << " "; //вывод элементов массива на экран
    }
    cout << endl; //перевод строки
    for (i = 0; i < 10; i++) // цикл для перебора номеров массива
    {
        if (*(m + i) % 3 == 0)
        {
            k++; //вычисляем количество элементов кратных 3
            sum += *(m + i); //вычисляем сумму элементов кратных 3
        }
    }
    cout<<"k = "<<k<<" sum = "<<sum<<endl; //вывод ответов
    delete[] m; // освобождение памяти;
    system("pause");
}

```

```

c:\users\329284.college\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Введите размер массива
12
-17 -3 9 -34 -24 -19 -3 -44 -48 41
k = 5 sum = -69
Для продолжения нажмите любую клавишу . . .

```

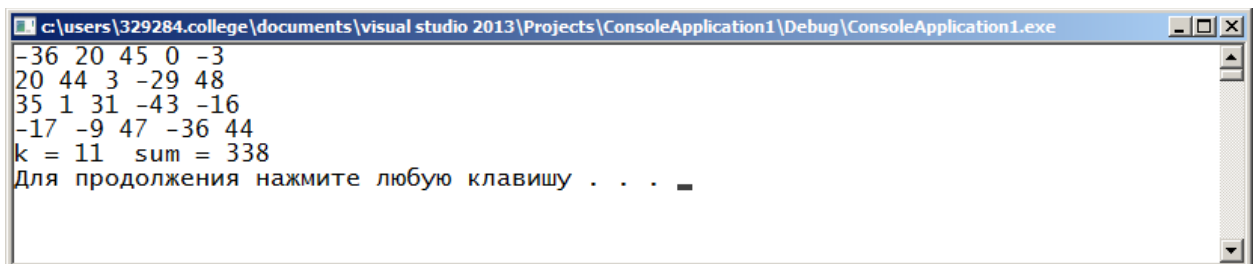
### Пример 16(а).

Вычислить сумму положительных элементов двумерного массива.

```
void main()
{
    int m[4][5]; //объявление двумерного массива из 4 строк и 5 столбцов
    int sum = 0, k = 0;
    srand(time(NULL)); //инициализация генератора случайных чисел

    for (int i = 0; i < 4; i++) //цикл для перебора номеров строк массива
    {
        for (int j = 0; j < 5; j++) //цикл для перебора номеров столбцов
        {
            m[i][j] = rand() % 100 - 50; //элементам массива присвоим
случайные значения
            cout<< m[i][j]<<" "; //вывод элементов массива на экран
        }
        cout << endl; //перевод строки, чтобы получилась таблица
    }

    for (int i = 0; i < 4; i++) //цикл для перебора номеров строк массива
    {
        for (int j = 0; j < 5; j++) //цикл для перебора номеров столбцов
        {
            if (m[i][j] > 0)
            {
                sum += m[i][j]; //вычисляем сумму положительных
                k++; //вычисляем количество положительных элементов
            }
        }
    }
    cout << "k = " << k << " sum = " << sum << endl; //вывод ответов
    system("pause");
}
```



```
c:\users\329284.college\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
-36 20 45 0 -3
20 44 3 -29 48
35 1 31 -43 -16
-17 -9 47 -36 44
k = 11 sum = 338
Для продолжения нажмите любую клавишу . . .
```

### Пример 16(б).

Вычислить сумму положительных элементов каждого столбца двумерного массива.

```
#include "stdafx.h"
#include "iostream"
#include "clocale"
#include "windows.h"
#include <time.h>
#include <iomanip>

using namespace std;
```

```

void main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    int m[4][5]; //объявление двумерного массива из 4 строк и 5 столбцов
    int sum = 0, k = 0;
    srand(time(NULL)); //инициализация генератора случайных чисел

    for (int i = 0; i < 4; i++) //цикл для перебора номеров строк массива
    {
        for (int j = 0; j < 5; j++) //цикл для перебора номеров столбцов
        {
            m[i][j] = rand() % 100 - 50; //элементам массива присвоим
случайные значения
            cout setw(4)<<<< m[i][j]; //вывод элементов массива
        }
        cout << endl; //перевод строки, чтобы получилась таблица
    }
    cout << endl;
    for (int j = 0; j < 5; j++) //цикл для перебора номеров столбцов
    {
        sum = 0;
        k = 0;
        for (int i = 0; i < 4; i++) //цикл для перебора номеров строк массива
        {
            if (m[i][j] > 0)
            {
                sum += m[i][j]; //вычисляем сумму положительных
                k++; //вычисляем количество положительных элементов
            }
        }
        cout << "Столбец: " << j << " k = " << k << " sum = " << sum << endl;
    }

    system("pause");
}

```

```

c:\users\329284.college\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
-34 13 -10 37 -25
-22 -6 34 -43 -33
-36 23 -10 39 34
-39 -21 -6 39 -43

Столбец: 0 k = 0 sum = 0
Столбец: 1 k = 2 sum = 36
Столбец: 2 k = 1 sum = 34
Столбец: 3 k = 3 sum = 115
Столбец: 4 k = 1 sum = 34
Для продолжения нажмите любую клавишу . . .

```

### Объявление динамического массива и работа с ним через указатели:

Выделение памяти для двумерного динамического массива можно выполнить следующим образом. Сначала память выделяется под массив указателей на каждую строку. Затем в цикле выделяется память под соответствующее количество элементов каждой строки.

#### Пример 16 (в).

Вычислить сумму положительных элементов каждого столбца двумерного массива. Обработка массива происходит с использованием указателей.

```
#include "stdafx.h"
#include "iostream"
#include "clocale"
#include "windows.h"
#include <time.h>
#include <iomanip>

using namespace std;

void main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    int N, M;
    cout << "Введите количество строк" << endl;
    cin >> N;
    cout << "Введите количество столбцов" << endl;
    cin >> M;
    int **m = new int*[N]; //выделение памяти под массив указателей
    int sum = 0, k = 0;
    srand(time(NULL)); //инициализация генератора случайных чисел

    for (int i = 0; i < N; i++) //цикл для перебора номеров строк массива
    {
        m[i] = new int[M]; //выделение памяти под каждую строку
        for (int j = 0; j < M; j++) //цикл для перебора номеров столбцов
        {
            (*(m + i) + j) = rand() % 100 - 50;
            cout << setw(4) << (*(m + i) + j) << " "; //вывод элементов массива
        }
        cout << endl; //перевод строки, чтобы получилась таблица
    }
    cout << endl;
    for (int j = 0; j < M; j++) //цикл для перебора номеров столбцов
    {
        sum = 0;
        k = 0;
        for (int i = 0; i < N; i++) //цикл для перебора номеров строк
        {
            if (*(m + i) + j > 0)
            {
                sum += (*(m + i) + j); //вычисляем сумму положительных
                k++; //вычисляем количество положительных
            }
        }
    }
}
```

```

    }
    cout << "Столбец: " << j << " k = " << k << " sum = " << sum << endl;
}
delete m; //освобождение памяти
system("pause");
}

```

```

c:\users\329284.college\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Введите количество строк
6
Введите количество столбцов
8
 25  -19  -22  -16   12   -3   -1   48
-44  -38  -19   -3  -39  -43  -24   18
 22   43   10  -20  -26   -8  -11   37
 -8   28   17   41  -31   3   -6   12
-28  -34   27   11   15   15   12  -21
 48  -23    5  -12   22   41   15   11

Столбец: 0 k = 3 sum = 95
Столбец: 1 k = 2 sum = 71
Столбец: 2 k = 4 sum = 59
Столбец: 3 k = 2 sum = 52
Столбец: 4 k = 3 sum = 49
Столбец: 5 k = 3 sum = 59
Столбец: 6 k = 2 sum = 27
Столбец: 7 k = 5 sum = 126
Для продолжения нажмите любую клавишу . . .

```

### Пример 17.

Приведем фрагмент программы, показывающий окрашивание элементов массива. Здесь будут окрашены все минимальные элементы массива.

```

HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
int k = 0;
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < M; j++)
    {
        if (*(m + i) + j) == min)
        {
            SetConsoleTextAttribute(console, ((WORD)(15) << 4) | (WORD)(1));
            k++;
        }
        else
        {
            SetConsoleTextAttribute(console, ((WORD)(15) << 4) | (WORD)(0));
        }
        cout << setw(5) << setprecision(2) << (*(m + i) + j);
    }
    cout << endl;
}

```

## Функции программиста

Функция – это поименованная группа инструкций, выполняющих определённую задачу. К функции можно обратиться по имени, передать ей значения и получить из неё результат. Функцию можно вызывать несколько раз, причём с разными фактическими параметрами. Функции нужны для упрощения структуры программы. Разбив задачу на подзадачи и оформив каждую из них в виде функции, мы можем значительно упростить её решение.

### Описание функции в общем виде

```
Тип Имя(тип1 параметр1, тип2 параметр2, ... типК параметрК )
{
    // здесь объявление локальных переменных функции
    // здесь инструкции функции
    return=Выражение; // здесь передача результата через возвращаемое значение
}
```

### Пример описания функции без результата:

```
// функция распечатывает строку из заданного количества символов
void printchar(char ch, int n)
{
    for (int i = 0; i < n; i++)    // тело функции
        cout << ch;
    cout << endl;
}
```

### Пример описания функции с результатом:

```
// функция находит максимальное из двух целых значений
int Max(int a, int b)
{
    if (a > b) return a;    // тело функции
    else return b;
}
```

### Вызов функций:

```
printchar('*', 80);
int m = Max(int a, int b);
```

Вся информация, которая передаётся в функцию и обратно, должна отражаться в её заголовке. Это требование не синтаксиса, а хорошего стиля программирования. В теле функции **крайне нежелательно** использовать глобальные переменные!



Входные данные передаются в функцию по значению, а результат работы функции – через возвращаемое значение. При необходимости передачи в качестве результатов более одной величины, результаты передаются через параметры по ссылке или через указатели.



### Пример передачи результатов по ссылке:

```
// функция вычисляет корни квадратного уравнения
// результат функции – количество корней
// вычисленные корни квадратного уравнения передаются по ссылке
int Kvadr1(float a, float b, float c, float &x1, float &x2)
{
    float D;    // D-локальная переменная
    D = b*b - 4 * a*c;
    if (D>0)
    {
        x1 = (-b + sqrt(D)) / (2 * a);
        x2 = (-b - sqrt(D)) / (2 * a);
        return 2;    // вычисляем два корня квадратного уравнения
    }
    else
        if (D == 0)
        {
            x1 = (-b) / (2 * a);
            x2 = x1;
            return 1;    // вычисляем один корень квадратного уравнения
        }
        else return 0;
}    // иначе корней нет
```

### Пример передачи результатов через указатели:

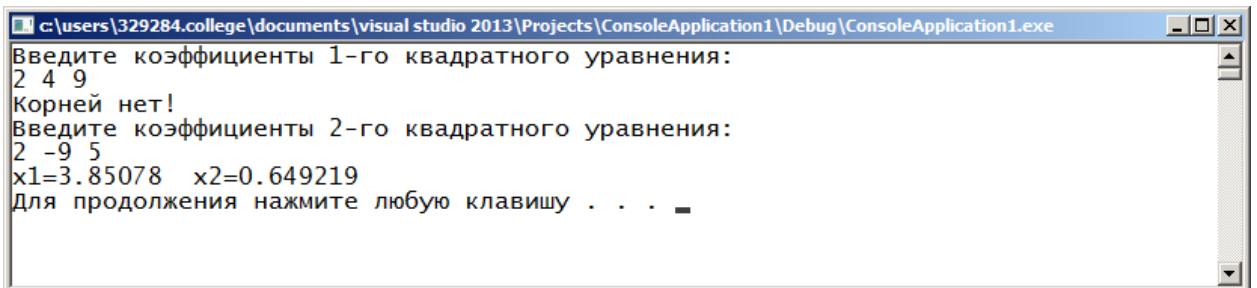
```
// функция вычисляет корни квадратного уравнения
// функция передаёт результат – количество корней
// вычисленные корни квадратного уравнения передаются через указатели
int Kvadr2(float a, float b, float c, float *x1, float *x2)
{
    float D;    //D-локальная переменная
    D = b*b - 4 * a*c;
    if (D>0)
    {
        *x1 = (-b + sqrt(D)) / (2 * a);
        *x2 = (-b - sqrt(D)) / (2 * a);
        return 2;    // вычисляем два корня квадратного уравнения
    }
    else
        if (D == 0)
        {
            *x1 = (-b) / (2 * a);
            *x2 = *x1;
            return 1;    // вычисляем один корень квадратного уравнения
        }
        else return 0; // иначе корней нет
}
```

### Вызов этих функций:

```
void main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    float a, b, c, x1, x2;
    cout << "Введите коэффициенты 1-го квадратного уравнения:" << endl;
    cin >> a >> b >> c;
    int r = Kvadr1(a, b, c, x1, x2);
    if (r == 2)
        cout << "x1=" << x1 << " x2=" << x2 << endl;
    else
        if (r == 1)
            cout << "x1=" << x1 << endl;
        else
            cout << "Корней нет!" << endl;

    cout << "Введите коэффициенты 2-го квадратного уравнения:" << endl;
    cin >> a >> b >> c;
    r = Kvadr2(a, b, c, &x1, &x2);
    if (r == 2)
        cout << "x1=" << x1 << " x2=" << x2 << endl;
    else
        if (r == 1)
            cout << "x1=" << x1 << endl;
        else
            cout << "Корней нет!" << endl;

    system("pause");
}
```



Можно также многократно вызывать одну и ту же функцию с различными значениями параметров.

## Модуль программиста (библиотека)

Наличие модулей в языке C/C++ позволяет программировать и отлаживать программу по частям, создавать *библиотеки подпрограмм и данных*.

При создании многофайлового проекта задача разбивается на несколько подзадач с распределением подзадач по разным модулям (файлам).

Заголовочный файл с расширением `.h` или `.hpp` содержит интерфейс для некоторого набора функций, а исходный файл с расширением `.c` или `.cpp` – реализацию этих функций.

**В заголовочном файле следует размещать:**

- ✓ Определение типов, задаваемых программистом, констант, шаблонов;
- ✓ Объявления (прототипы) функций;
- ✓ Определение внешних (глобальных) переменных с модификатором `extern`;
- ✓ Пространства имён.

**Пример описания заголовочных файлов и модулей программиста:**

```
// файл Point.h //
#ifndef POINT_H
#define POINT_H
// объявление типов
struct Point
{
    int x;
    int y;
};

// прототипы функций
void SetXY(Point &point, int x, int y);
int GetX(Point &point);
int GetY(Point &point);
#endif /*POINT_H*/

// файл Rect.h //
#ifndef RECT_H
#include <Point.h>
#define RECT_H

// объявление типов
struct Rect
{
    Point LeftPoint;
    Point RightPoint;
};

// прототипы функций
void SetLR(Rect &rect, Point lp, Point rp);
int GetL(Rect &rect, Point &lp);
int GetR(Rect &rect, Point &rp);
#endif /*RECT_H*/
```

```

// файл Point.cpp //
#include <Point.h>
void SetXY(Point &point, int x, int y)
{
    point.x = x;
    point.y = y;
}

int GetX(Point &point)
{
    return point.x;
}

int GetY(Point &point)
{
    return point.y;
}

// файл Rect.cpp //
#include <Point.h>
void SetLR(Rect &rect, Point lp, Point rp)
{
    rect.LeftPoint = lp;
    rect.RightPoint = rp;
}

int GetL(Rect &rect, Point &lp)
{
    lp = rect.LeftPoint
}

int GetR(Rect &rect, Point &rp)
{
    rp = rect.RightPoint
}

// файл Main.cpp //
#include <stdio.h>
#include <Point.h>
#include <Rect.h>
void main()
{
    Point pt1, pt2, lt, rt;
    Rect rect1;
    SetXY(pt1, 2, 5);
    SetXY(pt2, 10, 14);
    SetLR(rect1, pt1, pt2);
    GetL(rect1, lt);
    GetR(rect1, rt);
    cout<<lt.x<<" "<<lt.y;
    cout << rt.x<<" "<<rt.y;
}

```

## Обработка текстовых файлов

**Файл** (от англ. file-досье, документ) - это произвольная последовательность данных некоторой длины, имеющая имя. Другими словами, файл - это поименованное место на внешнем носителе.

**Текстовые файлы** являются файлами *последовательного* доступа. Их можно считывать только последовательно, от первой строки до последней.  
Запись информации в текстовый файл происходит либо от начала файла (при этом вся информация, находившаяся там ранее, стирается), либо в конец файла (добавление информации в файл).

### Назначение команд работы с файлами:

<pre>ifstream f1; ofstream f2;</pre>	Объявление потоков (файловых переменных)
<pre>f2.open("file1.txt", ios::out);</pre>	Открытие файла в режиме записи
<pre>f1.open("file1.txt", ios::in);</pre>	Открытие файла в режиме чтения
<pre>f2.open("file1.txt", ios::app);</pre>	Открытие файла в режиме добавления информации в конец файла
<pre>int x; f1.getline(x);</pre>	Чтение из файла целого значения
<pre>char str[120]; f1.getline(str, 120);</pre>	Чтение из файла строки
<pre>f2 &lt;&lt; str &lt;&lt; endl;</pre>	Запись в файл строки
<pre>if (!f1.is_open())     cout &lt;&lt; "Такого файла нет!" &lt;&lt; endl;</pre>	Проверка открытия файла
<pre>char str[120]; while (!f1.eof()) {     f1.getline(str, 120); }</pre>	Цикл для чтения из текстового файла всех строк
<pre>char ch; cin &gt;&gt; ch; while (!f1.eof()) {     f1.getline(str, 120);     if (str[0]==ch)         f2 &lt;&lt; str &lt;&lt; endl; }</pre>	Чтение из одного файла всех строк и запись в другой файл строк, начинающихся на заданную букву
<pre>f1.close();</pre>	Закрытие файла

## Пример 1

Создать в редакторе текстовый файл, внести в него 10 строк произвольного текста. Разработать программу, которая определяет номер строки в файле, в которой содержится больше всего пробелов.

Код программного модуля:

```
#include "stdafx.h"           //Объявление библиотек
#include "iostream"
#include "fstream"
#include "clocale"
#include "windows.h"

using namespace std;

int Kolichество(const char *c); //Функция подсчета количества пробелов в строке
void Vivod(string n);          //Функция вывода содержимого файлов на экран

void main()
{
    int kol = 0, max = 0, numstr, m = 0, kolstr, k = 1;
    setlocale(LC_ALL, "RUS");
    char str[120]; //Строка, в которую будет считываться содержимое файлов
    ifstream f1;   //Открытие потоков для записи информации
    f1.open("file1.txt", ios::in); //Привязка переменной к файлу
    while (!f1.eof()) //Пока не конец первого или второго файла
    {
        m++;
        f1.getline(str, 120); //Считывание строки в переменную str
        kol = Kolichество(str); //Вызов функции подсчета количества
        пробелов в строке
        if (max < kol)
        {
            max = kol;
            numstr = m;
        }
    }
    f1.close(); //Закрытие файлов

    Vivod("file1.txt"); //Функция вывода содержимого файла на экран
    cout << endl << endl;
    cout << "Максимальное кол - во пробелов в " << numstr << " строке и их
кол-во = " << max << endl;
    system("pause");
}
```

## Обработка двоичных файлов

Бинарные (двоичные) файлы – это файлы, в которых информация хранится в двоичном виде, то есть во внутренней форме представления. Двоичные файлы используются для их последующей обработки программными средствами.

Двоичные файлы являются файлами *прямого* доступа, то есть мы можем обратиться к любой компоненте файла с номером N (указатель текущей позиции файла настраивается на компоненту с заданным номером).

### Режимы открытия файлов при работе с потоками

В таблице 1 перечислены флаги управления режимами открытия файлов, определенные в классе **ios\_base**. Флаги относятся к типу **openmode** и группируются в битовые маски по аналогии с флагами **fmtflags**.

Таблица 1. Флаги открытия файлов	
Флаг	Описание
<b>in</b>	Открытие файла для чтения (используется по умолчанию для <b>ifstream</b> )
<b>out</b>	Открытие файла для записи (используется по умолчанию для <b>ofstream</b> )
<b>app</b>	Запись данных производится только в конец файла
<b>ate</b>	Позиционирование в конец файла после открытия (" <b>at end</b> ")
<b>trunc</b>	Удаление старого содержимого файла
<b>binary</b>	Специальные символы не заменяются

Флаг **binary** запрещает преобразование специальных символов или символьных последовательностей (например, конца строки или конца файла). В операционных системах типа **MS-DOS** или **OS/2** конец логической строки в тексте обозначается двумя символами (**CR** и **LF**). При открытии файла в обычном текстовом режиме (сброшенный флаг **binary**) символы новой строки заменяются последовательностью из двух символов, и наоборот. При открытии файла в двоичном режиме (с установленным флагом **binary**) эти преобразования не выполняются.

Флаг **binary** должен использоваться всегда, когда файл не содержит чисто текстовой информации и обрабатывается как двоичные данные. Пример - копирование файла с последовательным чтением символов и их записью без модификации. Если файл обрабатывается в текстовом виде, флаг **binary** не устанавливается, потому что в этом случае символы новой строки нуждаются в специальной обработке.

В некоторых реализациях имеются дополнительные флаги типа **nocreate** (файл должен существовать при открытии) и **noreplace** (файл не должен существовать). Однако эти флаги отсутствуют в стандарте, поэтому их использование влияет на переносимость программы.

Флаги объединяются оператором `|`. Полученный результат типа **openmode** может передаваться конструктору во втором аргументе. Например, следующая команда открывает файл для присоединения текста в конце:

```
std::ofstream file("xyz.out", std::ios::out | std::ios::app);
```

В таблице 2 представлены различные комбинации флагов и их аналоги - строковые обозначения режимов, используемые функцией открытия файлов **fopen()** в интерфейсе языка C. Комбинации с флагами **binary** и **ate** не приводятся. Установленный флаг **binary** соответствует строке с присоединенным символом **b**, а установленный флаг **ate** соответствует позиционированию в конец файла немедленно после открытия. Другие комбинации, отсутствующие в таблице (например, **trunc|app**), недопустимы.

Таблица 2. Описание режимов открытия файлов в C++		
Флаги ios_base	Описание	Обозначения режимов в C
<b>in</b>	Чтение (файл должен существовать)	"r"
<b>out</b>	Стирание и запись (файл создается при необходимости)	"w"
<b>out trunc</b>	Стирание и запись (файл создается при необходимости)	"w"
<b>out app</b>	Присоединение (файл создается при необходимости)	"a"
<b>in out</b>	Чтение и запись с исходным позиционированием в начало файла	"r+"
<b>in out trunc</b>	Стирание, чтение и запись (файл создается при необходимости)	"w+"

Открытие файла для чтения и/или записи не зависит от класса соответствующего объекта потока данных. Класс лишь определяет режим открытия по умолчанию при отсутствии второго аргумента. Это означает, что файлы, используемые только классом **ifstream** или **ofstream**, могут открываться для чтения и записи. Режим открытия передается соответствующему классу потокового буфера, который открывает файл. Тем не менее операции, разрешенные для данного объекта, определяются классом потока данных.

Также существуют три функции для открытия и закрытия файлов, принадлежащих файловым потокам данных (таблица 3).

Таблица 3. Функции открытия и закрытия файлов	
Функция	Описание
<b>open(имя)</b>	Открытие файла для потока в режиме по умолчанию
<b>open(имя, флаги)</b>	Открытие файла для потока в режиме, определяемом переданными флагами
<b>close()</b>	Закрытие файлового потока
<b>is_open()</b>	Проверка открытия файла



### Пример работы с потоками

Заполнить бинарный файл целыми числами с клавиатуры. Найти сумму чётных по значениям компонент файла и записать её в начало файла, сдвинув все компоненты вперёд. Массив не использовать.

```
#include "stdafx.h"
#include "conio.h"
#include "iostream"
#include "fstream"
#include "clocale"
#include "windows.h"
using namespace std;

int main()
{
    setlocale(LC_ALL, "Rus");
    int n = 0; //n - количество чисел
    int x;
    //открытие бинарного файла на запись
    ofstream F("first.bin", ios::binary);
    cout << "Введите n: ";
    cin >> n;
    cout << endl << "Целые числа:" << endl;
    //ввод целых чисел и запись их в файл
    for (int i = 0; i < n; i++)
    {
        cin >> x;
        F.write((char*)&x, sizeof(int)); //запись целых чисел в файл
    }
    F.close(); //закрытие потоков

    fstream S("first.bin", ios::binary | ios::in | ios::out);
    int pos = 0;
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        S.read((char*)&x, sizeof(int));
        if (x % 2 == 0) //если число чётное
        {
            sum += x;
        }
    }
    cout << endl;

    for (int i = n-1; i >= 0; i--) // цикл для сдвига элементов вперёд
    {
        S.seekp(i*sizeof(int), S.beg); //устанавливаем указатель файла на
        позицию i
        S.read((char*)&x, sizeof(int));
        S.seekp((i+1)*sizeof(int), S.beg); //устанавливаем указатель
        файла на позицию i+1
        S.write((char*)&x, sizeof(int)); //запись целых чисел в файл
    }
    S.seekp(0, S.beg);
```

```
S.write((char*)&sum, sizeof(int)); // запись суммы вместо первой  
компоненты
```

```
cout << endl;  
  
S.seekp(0, S.beg); // читаем файл с начала  
for (int i = 0; i < n+1; i++)  
{  
    S.read((char*)&x, sizeof(int));  
    cout << x << " ";  
}  
cout << endl;  
S.close(); //закрытие потоков  
_getch();  
return 0;  
}
```



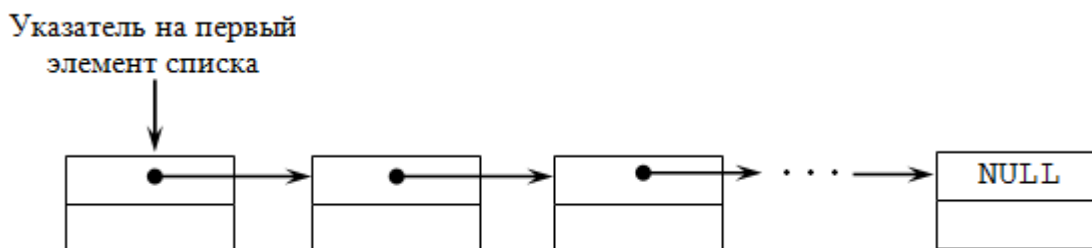
```
c:\users\329284\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Введите n: 5
Целые числа:
1
2
3
4
5
6 1 2 3 4 5
```

## Динамические структуры данных

### Однонаправленные (односвязные) списки

Наиболее простой динамической структурой является однонаправленный список, элементами которого служат объекты структурного типа.

**Однонаправленный (односвязный) список** – это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка. В последнем элементе указатель на следующий элемент равен NULL.



**Рисунок 1.** Линейный однонаправленный список

Описание простейшего элемента такого списка выглядит следующим образом:

```
struct имя_типа { информационное поле; адресное поле; };
```

где информационное поле – это поле любого, ранее объявленного или стандартного типа; адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка.

```
struct Node {  
    int key;//информационное поле  
    Node*next;//адресное поле  
};
```

Информационных полей может быть несколько.

```
struct point {  
    char*name;//информационное поле  
    int age;//информационное поле  
    point*next;//адресное поле  
};
```

Каждый элемент списка содержит ключ, который идентифицирует этот элемент. Ключ обычно бывает либо целым числом, либо строкой.

Основными операциями, осуществляемыми с однонаправленными списками, являются:

- создание списка;
- печать (просмотр) списка;
- вставка элемента в список;
- удаление элемента из списка;
- поиск элемента в списке
- проверка пустоты списка;
- удаление списка.

Особое внимание следует обратить на то, что при выполнении любых операций с линейным однонаправленным списком необходимо обеспечивать позиционирование какого-либо указателя на первый элемент. В противном случае часть или весь список будет недоступен.

Рассмотрим подробнее каждую из приведенных операций.

Для описания алгоритмов этих основных операций используется следующее объявление:

```
struct Single_List { //структура данных
    int Data; //информационное поле
    Single_List *Next; //адресное поле
};

. . . . .

Single_List *Head; //указатель на первый элемент списка

. . . . .

Single_List *Current;

//указатель на текущий элемент списка (при необходимости)
```

## Создание однонаправленного списка

Для того, чтобы создать список, нужно создать сначала первый элемент списка, а затем при помощи функции добавить к нему остальные элементы. При относительно небольших размерах списка наиболее изящно и красиво использование рекурсивной функции. Добавление может выполняться как в начало, так и в конец списка.

```
//создание однонаправленного списка (добавления в конец)
void Make_Single_List(int n,Single_List** Head){
```

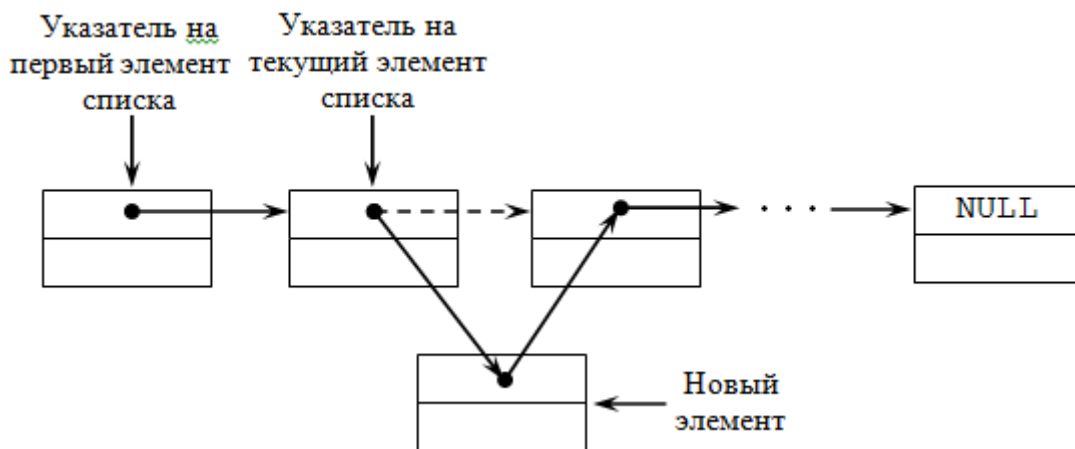
```

if (n > 0) {
    (*Head) = new Single_List();
    //выделяем память под новый элемент
    cout << "Введите значение ";
    cin >> (*Head)->Data;
    //вводим значение информационного поля
    (*Head)->Next=NULL; //обнуление адресного поля
    Make_Single_List(n-1, &((*Head)->Next));
}
}

```

## Вставка элемента в однонаправленный список

В динамические структуры легко добавлять элементы, так как для этого достаточно изменить значения адресных полей. Вставка первого и последующих элементов списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, сначала осуществляется проверка, на какое место вставляется элемент. Далее реализуется соответствующий алгоритм добавления.



**Рисунок 2.** Вставка элемента в однонаправленный список

```

/*вставка элемента с заданным номером в однонаправленный список*/
Single_List* Insert_Item_Single_List(Single_List* Head,
    int Number, int DataItem){
    Number--;
    Single_List *NewItem=new(Single_List);
    NewItem->Data=DataItem;
    NewItem->Next = NULL;

```

```

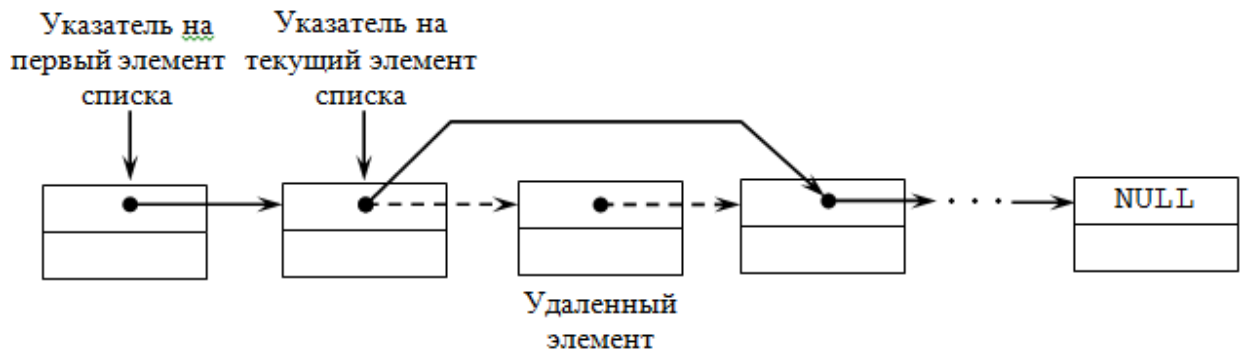
if (Head == NULL) { //список пуст
    Head = NewItem; //создаем первый элемент списка
}
else { //список не пуст
    Single_List *Current=Head;
    for(int i=1; i < Number && Current->Next!=NULL; i++)
        Current=Current->Next;
    if (Number == 0){
        //вставляем новый элемент на первое место
        NewItem->Next = Head;
        Head = NewItem;
    }
    else { //вставляем новый элемент на непервое место
        if (Current->Next != NULL)
            NewItem->Next = Current->Next;
        Current->Next = NewItem;
    }
}
return Head;
}

```

## Удаление элемента из однонаправленного списка

Из динамических структур можно удалять элементы, так как для этого достаточно изменить значения адресных полей. Операция удаления элемента однонаправленного списка осуществляет удаление элемента, на который установлен указатель текущего элемента. После удаления указатель текущего элемента устанавливается на предшествующий элемент списка или на новое начало списка, если удаляется первый.

Алгоритмы удаления первого и последующих элементов списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, осуществляется проверка, какой элемент удаляется. Далее реализуется соответствующий алгоритм удаления.



**Рисунок 3.** Удаление элемента из однонаправленного списка

```

/*удаление элемента с заданным номером из однонаправленного списка*/
Single_List* Delete_Item_Single_List(Single_List* Head,
    int Number){
    Single_List *ptr;//вспомогательный указатель
    Single_List *Current = Head;
    for (int i = 1; i < Number && Current != NULL; i++)
        Current = Current->Next;
    if (Current != NULL){//проверка на корректность
        if (Current == Head){//удаляем первый элемент
            Head = Head->Next;
            delete(Current);
            Current = Head;
        }
        else{//удаляем непервый элемент
            ptr = Head;
            while (ptr->Next != Current)
                ptr = ptr->Next;
            ptr->Next = Current->Next;
            delete(Current);
            Current=ptr;
        }
    }
    return Head;
}

```

## Поиск элемента в однонаправленном списке

Операция поиска элемента в списке заключается в последовательном просмотре всех элементов списка до тех пор, пока текущий элемент не будет содержать заданное значение или пока не будет достигнут конец списка. В последнем случае фиксируется отсутствие искомого элемента в списке (функция принимает значение false ).

```
//поиск элемента в однонаправленном списке
bool Find_Item_Single_List(Single_List* Head, int DataItem){
    Single_List *ptr; //вспомогательным указатель
    ptr = Head;
    while (ptr != NULL){ //пока не конец списка
        if (DataItem == ptr->Data) return true;
        else ptr = ptr->Next;
    }
    return false;
}
```

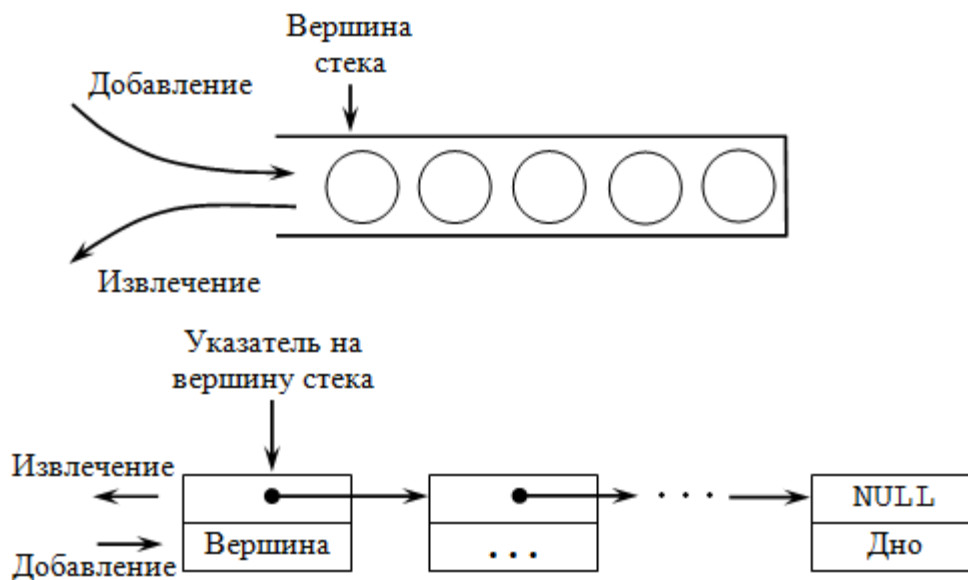
## Стеки

В списках доступ к элементам происходит посредством адресации, при этом доступ к отдельным элементам не ограничен. Но существуют также и такие списковые структуры данных, в которых имеются ограничения доступа к элементам. Одним из представителей таких списковых структур является стековый список или просто стек.

**Стек** (англ. *stack* – стопка) – это структура данных, в которой новый элемент всегда записывается в ее начало (вершину) и очередной читаемый элемент также всегда выбирается из ее начала. В стеках используется метод доступа к элементам *LIFO* ( *Last Input – First Output*, "последним пришел – первым вышел"). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно сначала взять верхнюю.

Стек – это список, у которого доступен один элемент (одна позиция). Этот элемент называется *вершиной стека*. Взять элемент можно только из вершины стека, добавить элемент можно только в вершину стека. Например, если записаны в стек числа 1, 2, 3, то при последующем извлечении получим 3,2,1.





**Рисунок 4.** Стек и его организация

Описание стека выглядит следующим образом:

```
struct имя_типа {
    информационное поле;
    адресное поле;
};
```

где информационное поле – это поле любого ранее объявленного или стандартного типа; адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента стека.

```
struct list {
    type pole1;
    list *pole2;
} stack;
```

Стек как динамическую структуру данных легко организовать на основе линейного списка. Поскольку работа всегда идет с заголовком стека, то есть не требуется осуществлять просмотр элементов, удаление и вставку элементов в середину или конец списка, то достаточно использовать экономичный по памяти линейный однонаправленный список. Для такого списка достаточно хранить указатель вершины стека, который указывает на первый элемент списка. Если стек пуст, то списка не существует, и указатель принимает значение NULL.

Описание элементов стека аналогично описанию элементов линейного однонаправленного списка. Поэтому объявим стек через объявление линейного однонаправленного списка:

```

struct Stack {
    Single_List *Top;//вершина стека
};

. . . . .

Stack *Top_Stack;//указатель на вершину стека

```

### **Основные операции, производимые со стеком:**

- создание стека;
- печать (просмотр) стека;
- добавление элемента в вершину стека;
- извлечение элемента из вершины стека;
- проверка пустоты стека;
- очистка стека.

Реализацию этих операций рассмотрим в виде соответствующих функций, которые, в свою очередь, используют функции операций с линейным однонаправленным списком. Обратим внимание, что в функции создания стека используется функция добавления элемента в вершину стека.

//создание стека

```

void Make_Stack(int n, Stack* Top_Stack){
    if (n > 0) {
        int tmp;//вспомогательная переменная
        cout << "Введите значение ";
        cin >> tmp; //вводим значение информационного поля
        Push_Stack(tmp, Top_Stack);
        Make_Stack(n-1,Top_Stack);
    }
}

```

//добавление элемента в вершину стека

```

void Push_Stack(int NewElem, Stack* Top_Stack){
    Top_Stack->Top =Insert_Item_Single_List(Top_Stack->Top,1,NewElem);
}

```

//извлечение элемента из вершины стека

```

int Pop_Stack(Stack* Top_Stack){
    int NewElem = NULL;

```

```

    if (Top_Stack->Top != NULL) {
        NewElem = Top_Stack->Top->Data;
        Top_Stack->Top = Delete_Item_Single_List(Top_Stack->Top, 0);
        //удаляем вершину
    }
    return NewElem;
}

//проверка пустоты стека
bool Empty_Stack(Stack* Top_Stack) {
    return Empty_Single_List(Top_Stack->Top);
}

//очистка стека
void Clear_Stack(Stack* Top_Stack) {
    Delete_Single_List(Top_Stack->Top);
}

```

## Ключевые термины

**FIFO (First Input – First Output)** – это метод доступа к элементам очереди по принципу "первый пришёл – первый вышел".

**LIFO (Last Input – First Output)** – это метод доступа к элементам стека по принципу "последним пришел – первым вышел"

**Вершина стека** – это доступный элемент стека.

**Конец очереди** – это позиция доступного для вставки в очередь элемента.

**Начало очереди** – это позиция доступного для извлечения из очереди элемента.

**Очередь** – это структура данных, представляющая собой последовательность элементов, образованная в порядке их поступления.

**Стек** – это структура данных, в которой новый элемент всегда записывается в ее начало (вершину) и очередной читаемый элемент также всегда выбирается из ее начала.

## Краткие итоги

1. Стек и очередь – это частные случаи линейного списка.
2. Стек является списком, у которого доступен один элемент, называемый вершиной стека. Поместить или извлечь элемент можно только из вершины списка.
3. Стек и очередь как динамические структуры данных можно организовать на основе линейного списка.
4. Основными операциями со стеком являются: создание стека; печать (просмотр) стека; добавление элемента в вершину стека; извлечение элемента из вершины стека; проверка пустоты стека; удаление стека.
5. Очередь является списком, у которого доступны два элемента: начало и конец очереди. Поместить элемент можно только в конец очереди, а взять элемент только из ее начала.
6. Основными операциями с очередью являются: создание очереди; печать (просмотр) очереди; добавление элемента в конец очереди; извлечение элемента из начала очереди; проверка пустоты очереди; удаление очереди.
7. Стек и очередь более экономно расходуют адресное пространство по сравнению с однонаправленными и двунаправленными списками.

## Пример

Составить программу обработки динамической структуры данных: в стек, элементы которого упорядочены по возрастанию, вставить новый элемент так, чтобы сохранить упорядоченность.

### Математическая модель:

На вход поступает несколько чисел типа Int, формирующих стек;

На выходе получаем отсортированный стек с добавленным пользователем элементом.

### Код программного модуля:

```
#include "stdafx.h"
#include "iostream"
#include <time.h>

using namespace std;

struct stek
{
    int d;
    struct stek *next; // указатель на следующий элемент стека
};

void push(stek* &next, int d); // функция будет помещать элемент в стек
// next – указатель на вершину стека
int pop(stek* &next); // функция будет извлекать элемент из стека
```

```
// вершина которого - next
```

```
void push(stek* &next, int d)
{
    stek *pv = new stek;           // объявляем новую динамическую переменную типа stek
    pv->d = d;                      // записываем значение, которое помещается в стек
    pv->next = next;                // связываем новый элемент стека с предыдущим
    next = pv;                     // новый элемент стека становится его вершиной
}
```

```
int pop(stek* &next)
{
    int temp = next->d;             // извлекаем в переменную temp значение в вершине стека
    stek *pv = next;               // запоминаем указатель на вершину стека, чтобы затем
    // освободить выделенную под него память
    next = next->next;              // вершиной становится предшествующий топ элемент
    delete pv;                     // освобождаем память, тем самым удалили вершину
    return temp;                   // возвращаем значение, которое было в вершине
}
```

```
int output(stek* &next)
{
    int out = next->d;
    next = next->next;
    return out;
}
```

```
void main()
{
    srand(time(NULL));
    int size, elem, k = 0, k2 = 0, k3 = 0;
    stek *s1 = new stek;
    stek *s2 = new stek;
    stek *s3 = new stek;
    cout << "Stack: ";
    for (int i = 0; i < 10; i++)
    {
        elem = rand() % 50;
        push(s1, elem);
        cout << elem << " ";
        k++;
    }
    cout << endl;
    int max, tek;
    while (k != 0)
    {
        max = pop(s1);
        k--;
        push(s2, max);
        k2++;
        while (k != 0)
        {
            tek = pop(s1);
            k--;
        }
    }
}
```

```

        if (tek > max) { max = tek; }
        push(s2, tek);
        k2++;
    }
    while (k2 != 0)
    {
        tek = pop(s2);
        k2--;
        if (tek == max){ push(s3, tek); k3++; }
        else{ push(s1, tek); k++; }
    }
}
cout << "Stack sort: ";
while (k3 != 0){
    int o = output(s3);
    cout << o << " ";
    push(s1, o);
    k++;
    k3--;
}
k++;
cout << endl;
int el = 0;
cout << "Enter element: ";
cin >> el;
push(s1, el);
k2 = 0; k3 = 0;
while (k != 0)
{
    max = pop(s1);
    k--;
    push(s2, max);
    k2++;
    while (k != 0)
    {
        tek = pop(s1);
        k--;
        if (tek > max) { max = tek; }
        push(s2, tek);
        k2++;
    }
    while (k2 != 0)
    {
        tek = pop(s2);
        k2--;
        if (tek == max){ push(s3, tek); k3++; }
        else{ push(s1, tek); k++; }
    }
}
while (k3 != 0){
    cout << output(s3) << " ";
    k3--;
}
cout << endl;
system("pause");
}

```

## Работа с файловой системой

Составить программу по работе с файловой системой, которая выводит в центре экрана системную дату и время, а также обеспечивает ввод с клавиатуры названия каталога; и если он существует, выводит список всех файлов из указанного каталога, созданных сегодня.

### Код программного модуля:

```
#include "stdafx.h"
#include <time.h>
#include <stdio.h>
#include <iostream>
#include <Windows.h>
#include "tchar.h"
#include <string>

using namespace std;

int main() {
    HANDLE hCon;
    COORD cPos;

    hCon = GetStdHandle(STD_OUTPUT_HANDLE);
    cPos.X = 20;
    cPos.Y = 10;
    SetConsoleCursorPosition(hCon, cPos);
    char buffer[80];
    time_t seconds = time(NULL);
    tm timeinfo;
    localtime_s(&timeinfo, &seconds);
    char* format = "%A, %B %d, %Y %H:%M:%S";
    strftime(buffer, 80, format, &timeinfo);
    cout << buffer << endl;

    setlocale(LC_ALL, "Russian");

    WIN32_FIND_DATAW findData;
    HANDLE hf;
    SYSTEMTIME timeF, timeS;
    cPos.X = 20;
    cPos.Y = 11;
    SetConsoleCursorPosition(hCon, cPos);
    cout << "Vvedite put': ";
    wstring path;
    wcin >> path;
    hf = FindFirstFileW(path.c_str(), &findData);
    cout << endl;
    if (hf == INVALID_HANDLE_VALUE)
    {
        cout << "Cannot find file" << endl;
        return -1;
    }
    cPos.X = 20;
    cPos.Y = 12;
```

```

int k = 0;
if (INVALID_HANDLE_VALUE != hf)
{
    do
    {
        if (!wcsncmp(findData.cFileName, L".", 1) ||
!wcsncmp(findData.cFileName, L"..", 2))
            continue;
        FileTimeToSystemTime(&findData.ftCreationTime, &timeF);
        SetConsoleCursorPosition(hCon, cPos);
        GetLocalTime(&timeS);
        if (timeF.wDay == timeS.wDay){
            std::wcout << findData.cFileName << ' ' <<
timeF.wDay << '-' << timeF.wMonth << '-' << timeF.wYear << std::endl;
            cPos.Y += 1;
        }
    } while (NULL != FindNextFileW(hf, &findData));

    FindClose(hf);
}

FindClose(hf);
system("pause");
}

```



## Организация многопоточной обработки данных

### Пример синхронизации потоков с использованием критических секций

Напишите программу, которая создает поток. Используйте атрибуты по умолчанию. Родительский и вновь созданный поток должны распечатать десять строк текста так, чтобы вывод родительского и дочернего потока был синхронизован: сначала родительский поток выводил бы 3 строки, затем дочерний одну, затем родительский 3 строки и т.д. Используйте критические секции.

#### Математическая модель:

На вход поступает файл в формате .txt; thread, hThread – потоки, связанные с этим файлом.

На выходе имеем несколько строк в формате char[80], поочередно выводимых из этого файла через буферную переменную buff.

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>
#include "fstream"

using namespace std;
HANDLE hThread;
DWORD IDThread;
CRITICAL_SECTION CS; //доступ к общему ресурсу
bool flag;

DWORD WINAPI thread(LPVOID) //дочерний поток
{
    setlocale(LC_ALL, "Rus");
    ifstream file2;
    char buff[50];
    file2.open("C:\\\\file.txt");
    while (true) //дочерний поток выводит по одной строке
    {
        if (flag) //при выходе из цикла в главном потоке он не заходит
        {
            EnterCriticalSection(&CS);
            file2.getline(buff, sizeof(buff));
            cout << "22 " << buff << endl;
            flag = false; //если вывод произведен, то дается
разрешение на вывод из главного потока
            LeaveCriticalSection(&CS);
            Sleep(500);
        }
    }
    file2.close();
    return 0;
}
```

```

int main()
{
    setlocale(LC_ALL, "Rus");
    int i = 0;
    char buff[50];    //буфер для вывода из файла
    flag = false;
    InitializeCriticalSection(&CS);
    hThread = CreateThread(NULL, 0, thread, NULL, 0, &IDThread);
    if (hThread == NULL)    //вывод номера ошибки при несоздании потока
        GetLastError();
    ifstream file1;
    file1.open("C:\\\\file.txt");    //открытие файла
    if (file1)
    {
        while (!file1.eof())    //главный поток выводит по 3 строки
        {
            if (!flag)
            {
                EnterCriticalSection(&CS);
                file1.getline(buff, sizeof(buff));
                cout << "11 " << buff << endl;
                file1.getline(buff, sizeof(buff));
                cout << "11 " << buff << endl;
                file1.getline(buff, sizeof(buff));
                cout << "11 " << buff << endl;
                flag = true;    //если вывод произведен, то дается
разрешение на вывод из дочернего потока
                LeaveCriticalSection(&CS);
                Sleep(500);
            }
        }
    }
    else
        cout << "Файл не найден!" << endl;
    file1.close();
    DeleteCriticalSection(&CS);    //разрушение критической секции
    CloseHandle(hThread);    //закрытие потока
    system("pause");
    return 0;
}

```

```

C:\Учеба\Практика №3\Программы\Задача 8\Debug\Задача 8
11 1 - строка
11 2 - строка
11 3 - строка
22 1 - строка
11 4 - строка
11 5 - строка
11 6 - строка
22 2 - строка
11 7 - строка
11 8 - строка
11 9 - строка
22 3 - строка
Для продолжения нажмите любую клавишу . . .

```

```

C:\Учеба\Практика №3\Программы\Задача 8\Debug\Задача 8
Файл не найден?
Для продолжения нажмите любую клавишу . . .

```

## Пример синхронизации потоков с использованием мьютексов

Напишите программу, которая создает поток. Используйте атрибуты по умолчанию. Родительский и вновь созданный поток должны распечатать строки текста так, чтобы вывод родительского и дочернего потока был синхронизован: сначала родительский поток выводил бы последнюю строку, затем дочерний первую, затем родительский предпоследнюю строку, а дочерний вторую и т.д. Используйте мьютексы.

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>
#include "fstream"
#include <thread>

using namespace std;
HANDLE Mutex; //мьютекс
int cycle, amount; //кол.проходов дочернего потока
bool tr;

void dstream() //дочерний поток
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    ifstream F1;
    char anotherfile1[80]; //строка 1
    F1.open("\\1.txt"); //открытие файла
    int number = 0; //порядк.номер строки
    Mutex = OpenMutex(SYNCHRONIZE, FALSE, (LPCWSTR)"Mutex"); //открытие
    мьютекса
    while (tr) //родительский поток выводит по 2 строки с начала
    {
        F1.getline(anotherfile1, sizeof(anotherfile1));
        if (number == amount)
        {
            Sleep(1500); //задержка экрана
            SetConsoleTextAttribute(console, ((WORD)(15) << 4) | (WORD)(4));
            cout << "Child -> " << anotherfile1 << endl;
            tr = false;
            cycle ++; //+ строка
        }
        number++;
    }
    F1.close(); //закрытие файла
}

void main()
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    char firstfile[80]; //буфер для вывода из файла
    ifstream F1;
    tr = false;
    cycle = 0, amount = 0; //номера строк
    F1.open("\\1.txt"); //открытие файла
```

```

int number = 0, k = 0; //кол.строк в файле 2 (обе перем.)
if (F1)
{
    while (!F1.eof())
    {
        F1.getline(firstfile, sizeof(firstfile));
        number++; //подсчёт кол.строк
    }
    Mutex = CreateMutex(NULL, FALSE, (LPCWSTR)"Mutex"); //создание мьютекса
    F1.clear(); //возврат в начало файла
    F1.seekg(0);
    while (!F1.eof()) //до конца файла
    {
        if (!tr)
        {
            WaitForSingleObject(Mutex, INFINITE);
            F1.getline(firstfile, sizeof(firstfile));
            k++; //подсчёт кол.строк
            if (k == (number - cycle)) //если это - искомая строка
            {
                Sleep(500); //задержка
                SetConsoleTextAttribute(console, ((WORD)(15) << 4) | (WORD)(1));
                cout << "Parent -> " << firstfile << endl;
                tr = true;
                thread t(dstream);
                t.join(); //вызов дочернего потока
                F1.clear(); //возврат в начало файла
                F1.seekg(0);
                k = 0; //кол.строк обнул.
                amount++;
            }
        }
    }
    ReleaseMutex(Mutex); //освобожд.мьютекса
}
else
    cout << "The file wasn't found!" << endl;
F1.close(); //заккрытие файла F1
CloseHandle(Mutex); //заккрытие потока (дескриптора объекта)
_gettch();
}

```

```

c:\users\329284.college\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Parent -> 9 the witching hour
Child -> 1 Noweverypieceisinplace
Parent -> 8 We're fast approaching
Child -> 2 And all that's left to erase before I
Parent -> 7 and she canrun but can'thide.
Child -> 3 take over all the power
Parent -> 6 She's been a thorn in my side
Child -> 4 Is every trace of dear Alice
Parent -> 5 then the queen
Child -> 5 then the queen
Parent -> 4 Is every trace of dear Alice
Child -> 6 She's been a thorn in my side
Parent -> 3 take over all the power
Child -> 7 and she canrun but can'thide.
Parent -> 2 And all that's left to erase before I
Child -> 8 We're fast approaching
Parent -> 1 Noweverypieceisinplace
Child -> 9 the witching hour

```