

# Основы программирования на C++



## *Указатели, динамические массивы*

В C++ существуют динамические массивы – массивы переменной длины, они определяются с помощью указателей.

*Указатель* – переменная, значением которой является адрес памяти, по которому хранится объект определенного типа. При объявлении указателей всегда указывается тип объекта, который будет храниться по данному адресу.

**type \* name;**

Здесь name – переменная, объявляемая, как указатель. По этому адресу (указателю) храниться значение типа type.

Например:

**int \*i;**

Объявляем указатель (адрес) i. По этому адресу будет храниться переменная типа int. Переменная i указывает на тип данных int.

**float \*x,\*z;**

Объявляем указатели с именами x и z, которые указывают на переменные типа float.



## *Операции \* и & при работе с указателями*

Операция & возвращает адрес своего операнда.

Например, если объявлена переменная *a* следующим образом:

```
float a;
```

то оператор

```
adr_a=&a;
```

записывает в переменную *adr\_a* адрес переменной *a*,

переменная *adr\_a* должна быть указателем на тип *float*. Ее следует описать следующим образом:

```
float *adr_a;
```

Операция \* выполняет действие, обратное операции &. Она возвращает значение переменной, хранящееся по заданному адресу.

Например, оператор **a=\*adr\_a;**

записывает в переменную *a* вещественное значение, хранящееся по адресу **adr\_a**.

## *Операция присваивания указателей*

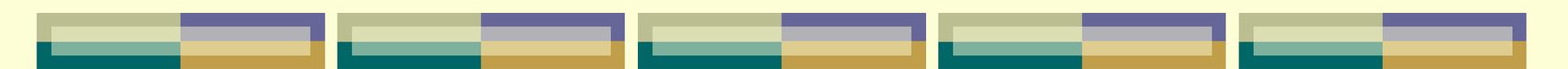
```
int main()
{ float PI=3.14159,*p1,*p2;
  p1=p2=&PI;
  printf("По адресу p1=%p хранится *p1=%g\n",p1,*p1);
  printf("По адресу p2=%p хранится *p2=%g\n",p2,*p2); }
```

В этой программе определены: вещественная переменная  $PI=3.14159$  и два указателя на тип `float` `p1` и `p2`. Затем в указатели `p1` и `p2` записывается адрес переменной `PI`. Операторы `printf` выводят на экран адреса `p1` и `p2` и значения, хранящиеся по этим адресам. Для вывода адреса используется спецификатор типа `%p`. В результате работы этой программы в переменных `p1` и `p2` будет храниться значение одного и того же адреса, по которому хранится вещественная переменная  $PI=3.14159$ .

## *Операция присваивания указателей*

Если указатели ссылаются на различные типы, то при присваивании значения одного указателя другому, необходимо использовать преобразование типов. Без преобразования можно присваивать любому указателю указатель `void *`.

```
int main()  
{ float PI=3.14159,*p1;  
double *p2; //В переменную p1 записываем адрес PI  
p1=&PI; //указателю на double присваиваем  
//значение, которое ссылается на тип float  
p2=(double *) p1;  
printf("По адресу p1=%p хранится *p1=%g\n",p1,*p1);  
printf("По адресу p2=%p хранится *p2=%e\n",p2,*p2);  
}
```

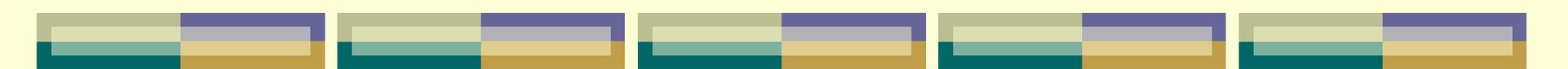


## *Операция присваивания указателей*

**По адресу  $p1=0012FF7C$  хранится  $*p1=3.14159$**

**По адресу  $p2=0012FF7C$  хранится  $*p2=2.642140e-308$**

В указателях  $p1$  и  $p2$  хранится один и тот же адрес, но значения, на которые они ссылаются, оказываются разными. Это связано с тем, указатель типа `*float` адресует 4 байта, а указатель `*double` – 8 байт. После присваивания  $p2=(double *)p1$ ; при обращении к  $*p2$  происходит следующее: к переменной, хранящейся по адресу  $p1$ , дописывается еще 4 байта из памяти. В результате значение  $*p2$  не совпадает со значением  $*p1$ .



Рассмотрим, что произойдет в результате следующей программы?

```
int main()
{ double PI=3.14159,*p1;
float *p2; p1=&PI; p2=(float *)p1;
printf("По адресу p1=%p хранится *p1=%g\n",p1,*p1);
printf("По адресу p2=%p хранится *p2=%e\n",p2,*p2); }
```

После присваивания `p2=(double *)p1`; при обращении к `*p2` происходит следующее: из переменной, хранящейся по адресу `p1`, выделяется только 4 байта. В результате и в этом случае значение `*p2` не совпадает со значением `*p1`.

**ВЫВОД.** При преобразовании указателей разного типа приведение типов разрешает только синтаксическую проблему присваивания. Следует помнить, что операция `*` над указателями различного типа, ссылающимися на один и тот же адрес, возвращает различные значения.



## *Арифметические операции над адресами*

Над адресами определены следующие операции:

- ❖ суммирование, можно добавлять к указателю целое значение;
- ❖ вычитание, можно вычитать указатели или вычитать из указателя целое число.

Некоторые особенности при выполнении арифметических операций:

**double \*p1;**

**float \*p2;**

**int \*i;**

**p1++**

**p2++;**

**i++;**

## Арифметические операции над адресами



Операция  $p1++$  увеличивает значение адреса на 8, операция  $p2++$  увеличивает значение адреса на 4, а операция  $i++$  на 2.

Операции адресной арифметики выполняются следующим образом:

- ❖ операция увеличения приводит к тому, что указатель будет сслаться на следующий объект базового типа (для  $p1$  – это `double`, для  $p2$  – `float`, для  $i$  – `int`);
- ❖ операция уменьшения приводит к тому, что указатель, ссылается на предыдущий объект базового типа.
- ❖ после операции  $p1=p1+n$ , указатель будет передвинут на  $n$  объектов базового типа;  $p1+n$  как бы адресует  $n$ -й элемент массива, если  $p1$  – адрес начала

## *Рекомендации по использованию указателей и динамического распределения памяти*

- ❖ Используйте указатели и динамическое распределение памяти только там, где это действительно необходимо. Проверьте, можно ли выделить память статически или использовать автоматическую переменную.
- ❖ Старайтесь локализовать распределение памяти. Если какой-либо метод выделяет память (в особенности под временные данные), он же и должен ее освободить.
- ❖ Там, где это возможно, вместо указателей используйте ссылки.
- ❖ Проверяйте программы с помощью специальных средств контроля памяти (Purify компании Rational, Bounce Checker компании Nu-Mega и т.д.)



## Ссылки

**Ссылка – это еще одно имя переменной.** Если имеется какая-либо переменная, например Complex  $x$ ; то можно определить ссылку на переменную  $x$  как `Complex& y = x`; и тогда  $x$  и  $y$  обозначают одну и ту же величину. Если выполнены операторы `x.real = 1`; `x.im = 2`; то `y.real` равно 1 и `y.im` равно 2.

**Ссылка – это адрес переменной** (поэтому при определении ссылки используется символ `&` - знак операции взятия адреса ), и в этом смысле она сходна с указателем, однако у ссылок есть свои особенности.

*Во-первых*, определяя переменную типа ссылки, ее необходимо инициализировать, указав, на какую переменную она ссылается.

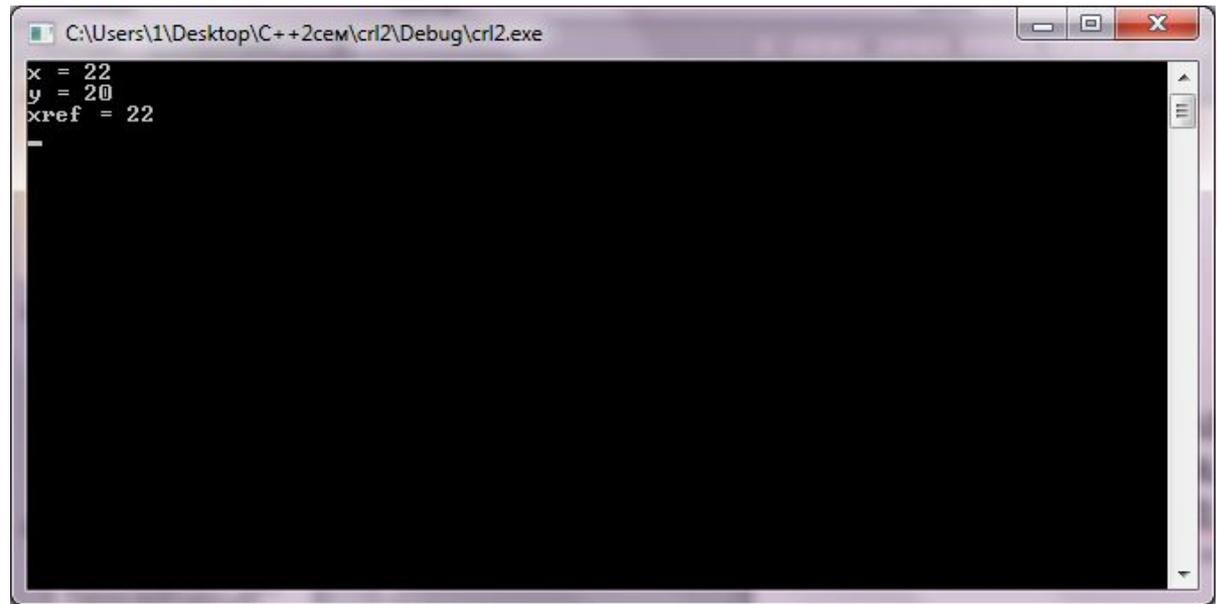
Нельзя определить ссылку `int& xref`; можно только

**`int& xref = x`;**

*Во-вторых*, нельзя переопределить ссылку, т.е. изменить на какой объект она ссылается. Если после определения ссылки `xref` выполним присваивание `xref = y`; то выполнится присваивание значения переменной  $y$  той переменной, на которую ссылается `xref`. Ссылка `xref` по-прежнему будет ссылаться на  $x$ .



# Ссылки. Пример



```
C:\Users\1\Desktop\C++2сем\сr12\Debug\сr12.exe
x = 22
y = 20
xref = 22
```

```
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ int x = 10; int y = 20; int& xref = x ;xref = y; x += 2;
  cout << "x = " << x << endl; cout << "y = " << y << endl;
  cout << "xref = " << xref << endl;   _getch(); }
```

*В-третьих*, синтаксически обращение к ссылке аналогично обращению к переменной. Если для обращения к атрибуту объекта, на который ссылается указатель, применяется операция `->`, то для подобной же операции со ссылкой применяется точка `"."`.

```
Complex a; Complex* aptr = &a;
```

```
Complex& aref = a;
```

```
aptr->real = 1; aref.im = 2;
```

Как и указатель, ссылка сама по себе не имеет значения. Ссылка должна на что-то ссылаться, тогда как указатель должен на что-то указывать.

## Динамические массивы



- Описать указатель (например, переменную  $p$ ) определенного типа.
- Начиная с адреса, определенного указателем, с помощью операции **new** выделить участок памяти определенного размера. После этого  $p$  будет адресом первого элемента выделенного участка оперативной памяти (0-й элемент массива),  $p+1$  будет адресовать – следующий элемент в выделенном участке памяти (1-й элемент динамического массива, ...,  $p+i$  является адресом  $i$ -го элемента). Необходимо только следить, чтобы не выйти за границы выделенного участка памяти. К  $i$ -му элементу динамического массива  $p$  можно обратиться одним из двух способов  **$*(p+i)$  или  $p[i]$**
- Когда участок памяти будет не нужен, его можно освободить с помощью операции **delete**.

## Найти сумму элементов массива



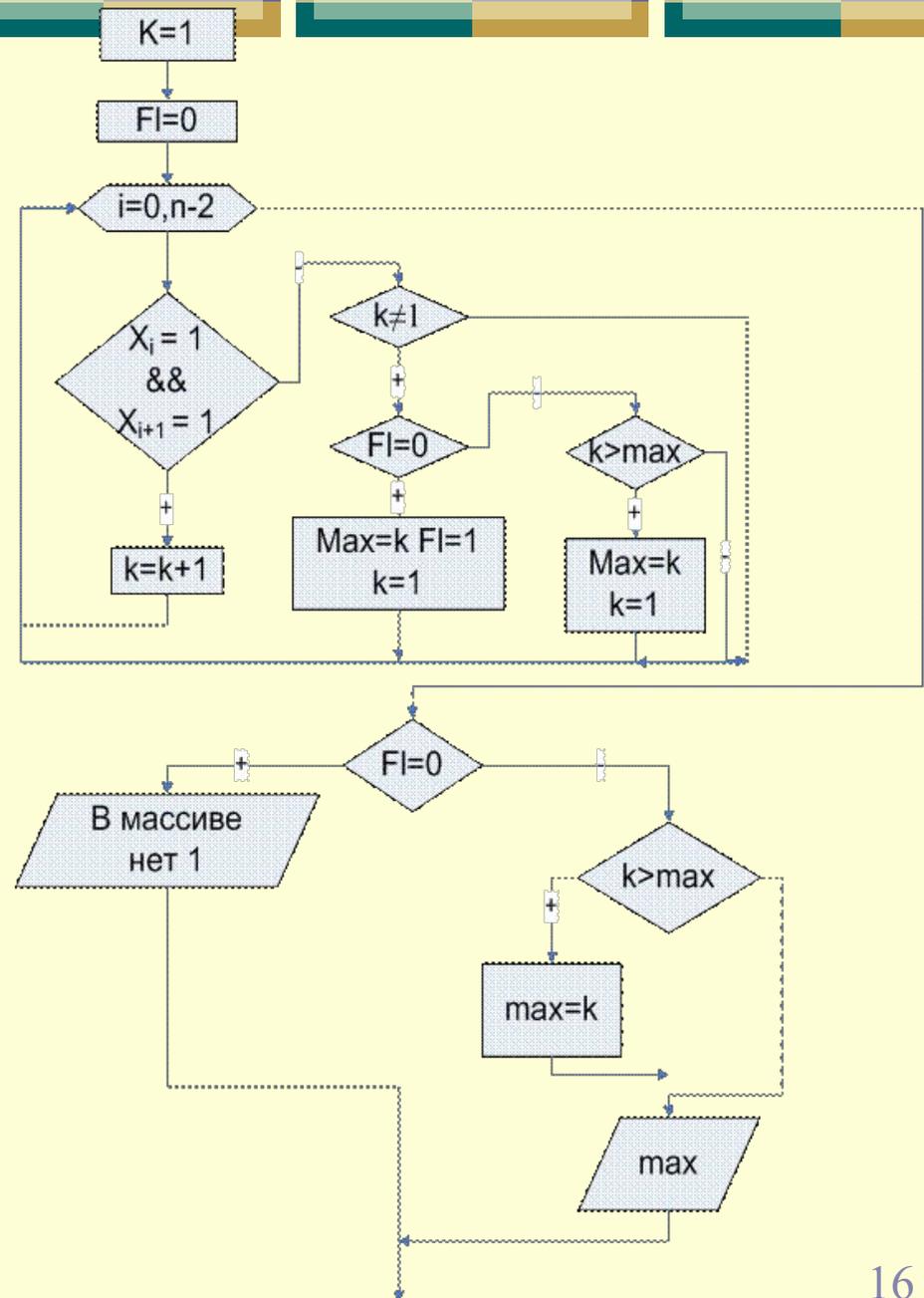
```
#include "....."  
#include <iostream>  
#include <cmath>  
using namespace System;  
int main()  
{  
int i, n; float *a, s; std::cout << "n=";  
std::cin >> n; a = new float[n];  
std::cout << "Vvedite massiv A";  
for (i = 0; i < n; i++)  
std::cin >> *(a + i);  
for (s = 0, i = 0; i < n; i++)  
s += *(a + i);  
std::cout << "S=" << s;  
delete[] a; return 0;  
}
```

A screenshot of a Windows command prompt window. The title bar shows the path "C:\Users\Master\Desktop\Лекции...". The window content shows the following text:

```
n=5  
Vvedite massiv A 1.3 1.1 1.36 1.95 -0.92  
S=4.79_
```

В заданном массиве  
найти длину самой  
длинной серии  
элементов,  
состоящей из  
единиц.

$F1 = 1$ , если были, и  $F1 = 0$ , если нет,  $k$  –  
длина текущей  
серии,  $max$  – самая  
длинная серия.



```

include "....."
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   int *x, max, i, k, fl, n, b;
    printf(" n="); scanf("%d", &n);
    x = new int[n];   printf("\n");
    for (i = 0; i<n; i++)
    {   printf("x(%d)=", i);
        scanf("%d", &b); *(x + i) = b; }
    for (k = 1, fl = 0, i = 0; i<n - 1; i++)
    {   if ((*x + i) == 1) && (*(x + i + 1) == 1))
        k++; else if (k != 1)
        { if (!fl) { max = k; fl = 1; k = 1; }
          else if (k>max) max = k; k = 1;
        } }
    if (fl == 0)   printf("V massive net seriyy iz 1"); else {
        if (k>max) max = k;   printf("max1= %d", max);
    } delete[] x;   return 0;}

```

```

n=7
x(0)=1
x(1)=0
x(2)=1
x(3)=1
x(4)=1
x(5)=1
x(6)=0
max1= 4

```

## *Пример*

*Написать программу для умножения матриц. Даны квадратные матрицы  $a$  и  $b$ , содержащие строк и  $n$  столбцов. Найти матрицу  $c$ , являющуюся результатом умножения матрицы  $a$  на матрицу  $b$ . Вводим только  $n$ , а матрицы заполняем случайными числами.*

```
#include...  
#include <iostream>  
#include <random>  
#include <time.h>  
using namespace std;  
// Функция вывода матрицы  
void output(int **a, int size, int size1)  
{ for (int i(0); i < size; i++) {  
    for (int j(0); j < size1; j++) {  
        cout << a[i][j] << ' '; } cout << endl; }  
}
```

```
int main()
{
    setlocale(LC_ALL, "rus");
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(-100, 100);
    int n;
    // Описание указателей на матрицы с инициализацией
    int **c = NULL; int **a = NULL; int **b = NULL;
    cout << "Введите число строк и столбцов квадратной матрицы: ";
    cin >> n;
    // Размещение матриц в динамической памяти
    a = new int *[n]; b = new int *[n]; c = new int *[n];
    for (int i(0); i < n; i++) { a[i] = new int[n]; }
    for (int i(0); i < n; i++) { b[i] = new int[n]; }
    for (int i(0); i < n; i++) { c[i] = new int[n]; }
}
```

```

// Заполнение матриц случайными числами
for (int i(0); i<n; i++) {
    for (int j = 0; j < n; j++) { a[i][j] = dist(gen); } }
for (int i(0); i < n; i++) {
    for (int j(0); j < n; j++) { b[i][j] = dist(gen); } }
cout << "1 матрица: " << endl; output(a, n, n); cout << endl;
cout << "2 матрица: " << endl; output(b, n, n);
int s; for (int i(0); i < n; i++) {
    for (int j(0); j < n; j++) {
s = 0; for (int k(0); k < n; k++) {
s += a[i][k] * b[k][j]; } c[i][j] = s; } }
cout << "Результат:" << endl;
for (int i(0); i < n; i++) {
    for (int j(0); j < n; j++) { cout << c[i][j] << "\t"; } cout << endl; }
//Освобождение памяти, выделенной под матрицы
    for (int i(0); i < n; i++) { delete a[i]; } delete[] a;
    for (int i(0); i < n; i++) { delete b[i]; } delete[] b;
    for (int i(0); i < n; i++) { delete c[i]; } delete[] c;
system("Pause"); return 0; }

```

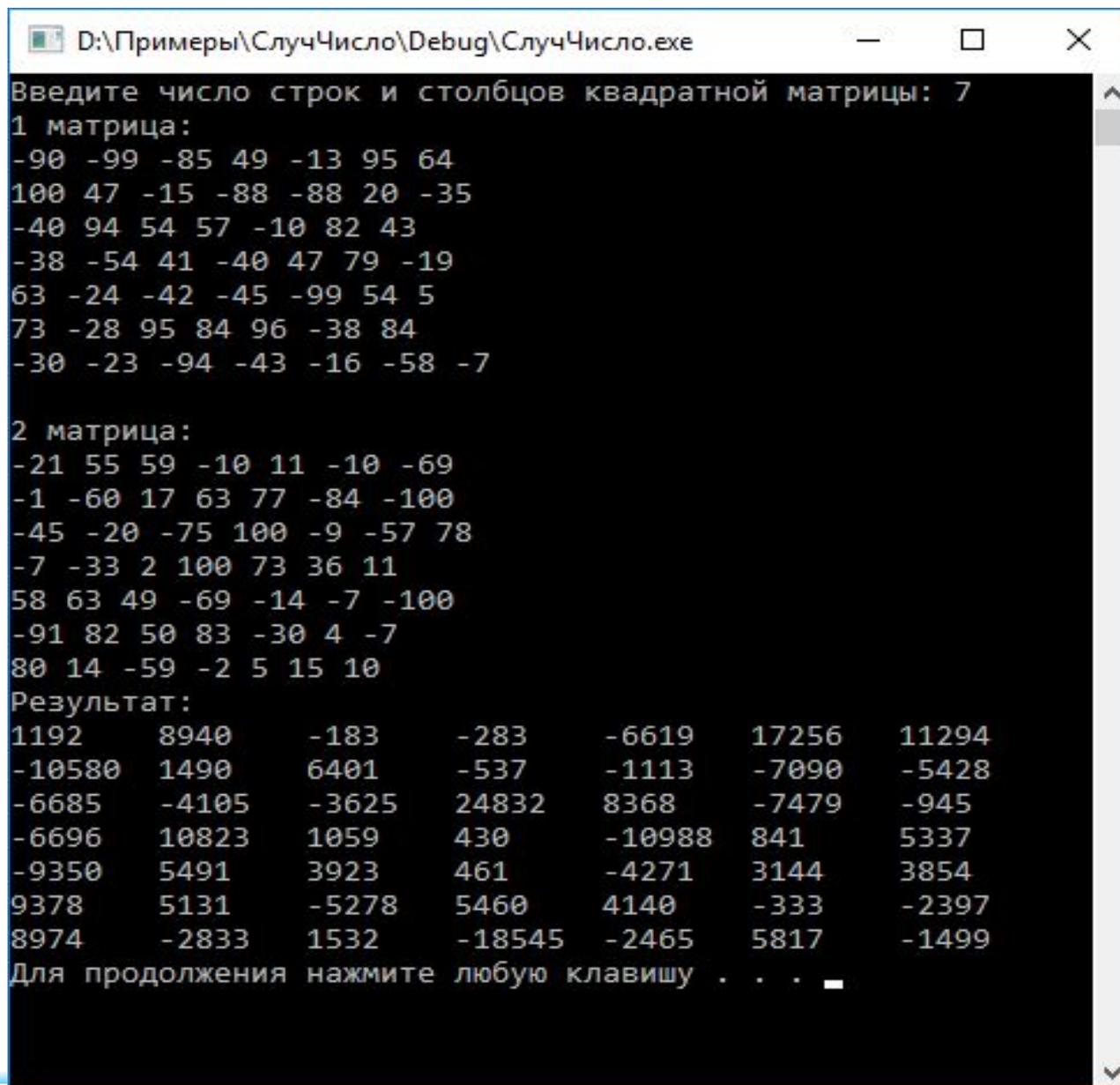
## Результаты первого запуска программы:

```
D:\Примеры\СлучЧисло\Debug\СлучЧисло.exe
Введите число строк и столбцов квадратной матрицы: 7
1 матрица:
-28 85 75 88 22 -86 -69
99 -34 69 -33 52 -61 -38
3 57 29 50 -3 77 -85
-81 37 -77 -91 -40 60 43
-53 -49 -13 -91 93 0 -58
75 4 3 61 75 21 88
54 -21 -1 -96 3 78 -70

2 матрица:
-82 54 -77 11 13 70 25
1 100 17 25 25 39 0
10 -39 39 -92 -61 45 -46
-80 -30 43 -24 31 -44 11
-78 91 84 -23 92 -54 -9
82 4 19 -3 17 89 1
-65 87 61 -68 -75 57 -60

Результат:
-8192   -2922   6315    -2751    5651    -11917   674
-11410  1427    -6038    -3746    1802    -242    689
8174   -4129    45       3208    8653    3708    4495
18434  5400    353      7118    -4137   6263   -1644
7963   -1108    3102    3377    8964   -10530  915
-20844 17068   9100    -8587   3440    5692   -3526
13933  -1770   -11218  6922    4114    9930   4591
Для продолжения нажмите любую клавишу . . .
```

## Результаты второго запуска программы:

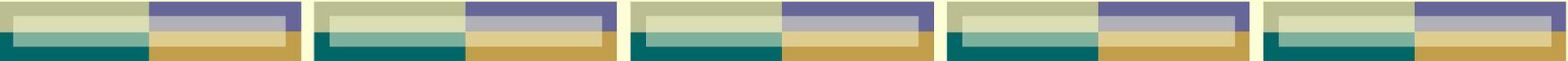


```
D:\Примеры\СлучЧисло\Debug\СлучЧисло.exe
Введите число строк и столбцов квадратной матрицы: 7
1 матрица:
-90 -99 -85 49 -13 95 64
100 47 -15 -88 -88 20 -35
-40 94 54 57 -10 82 43
-38 -54 41 -40 47 79 -19
63 -24 -42 -45 -99 54 5
73 -28 95 84 96 -38 84
-30 -23 -94 -43 -16 -58 -7

2 матрица:
-21 55 59 -10 11 -10 -69
-1 -60 17 63 77 -84 -100
-45 -20 -75 100 -9 -57 78
-7 -33 2 100 73 36 11
58 63 49 -69 -14 -7 -100
-91 82 50 83 -30 4 -7
80 14 -59 -2 5 15 10

Результат:
1192      8940      -183      -283      -6619      17256      11294
-10580     1490       6401     -537     -1113     -7090     -5428
-6685     -4105     -3625     24832     8368     -7479     -945
-6696     10823     1059      430     -10988     841       5337
-9350     5491      3923      461     -4271     3144     3854
9378      5131     -5278     5460     4140     -333     -2397
8974      -2833     1532     -18545   -2465     5817     -1499

Для продолжения нажмите любую клавишу . . .
```

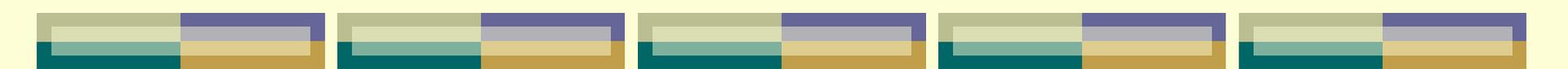


## *Обработка исключительных ситуаций*

*Исключение*—событие, возникающее во время выполнения программы, которое не позволяет корректное продолжение работы этой программы.

Различают два вида исключений:

- ❖ Аппаратные (структурные, *SE-Structured Exception*), которые генерируются процессором. В частности, к ним относятся:
    - деление на 0;
    - выход за границы массива;
    - обращение к невыделенной памяти;
    - переполнение разрядной сетки.
  - ❖ Программные (генерируемые операционной системой и прикладными программами).
- 



Для реализации обработки исключений в C++ применяют выражения *try*, *throw* и *catch*.

*try* {...} образует блок, содержащий операторы, при выполнении которых возможно возникновение исключительной ситуации.

Выражение *throw* используется только в программных исключениях и указывает на исключительную ситуацию в блоке *try*. В качестве операнда выражения *throw* можно использовать объект любого типа. Обычно этот объект используется для передачи информации об ошибке.

Обработка исключений осуществляется в специально создаваемых блоках *catch*, находящихся сразу после блока *try*. Каждый блок *catch* указывает тип исключения, которое он может обрабатывать.



*try {...}catch (тип\_исключительной\_операции){...}*

Когда внутри блока *try* возникает исключительная ситуация, то управление сразу же передается в соответствующий оператор *catch*.

*Первый метод идентификации исключительных ситуаций (пространство имен std).*

Идентификатор исключительной ситуации задается именем параметра аргумента *throw*. Если имя исключительной ситуации совпадает с именем аргумента *catch*, выполняется блок *catch*. Если совпадения не найдено, то происходит откат вызовов, до тех пор, пока либо не завершится программа, либо не встретится блок *catch* с подходящим типом аргумента. В блоке *catch* происходит обработка исключительной ситуации.



## Пример

Разработать *надежный* простейший калькулятор. Отличие от калькулятора, разработанного ранее, состоит в том, что при неправильном вводе данных или неверно выбранной операции, работа программы не будет завершаться аварийно.



Наиболее вероятные для рассматриваемой задачи исключительные ситуации:

- неверно выбрана операция - *throw (0)*,
- деление на 0 - *throw (1)*);
- неверно введено число - *throw (2)*.

```
#include ...
#include <iostream>
using namespace std;
int main()
{  setlocale (LC_ALL, "Russian");
   double a, b, res = 0;
   char op;
try { cout<< "  a,b= "; cin >> a >> b;
```

***if (!cin.good()) throw (2);***// проверка правильности ввода числа

```
cout << "  op= "; cin >> op;
switch (op) {
  case '+': {  res = a + b; break; }
  case '-': {  res = a - b; break; }
  case '*': {  res = a*b; break; }
```



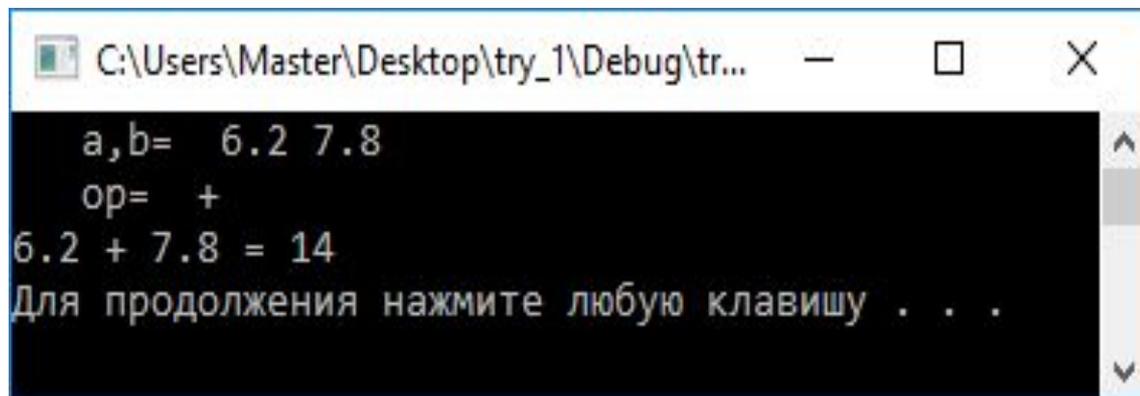
```

// Проверка деления на 0
case ':': { if (b == 0) throw (1); res = a / b; break; }
case '/': { if (b == 0) throw (1); res = a / b; break; }
default: {throw (0); break; } }
cout << a << " " << op << " " << b << " = " << res << endl;
system("Pause"); }
catch (int t)
{ switch (t) {
case 0: { cout<< "Неверно выбрана операция" <<endl;
break; }
case 1: {cout<< "Деление на 0" <<endl; break; }
case 2: {cout<< "Неверно введено число" <<endl; break;
}}
system("Pause"); return 0;
} }

```



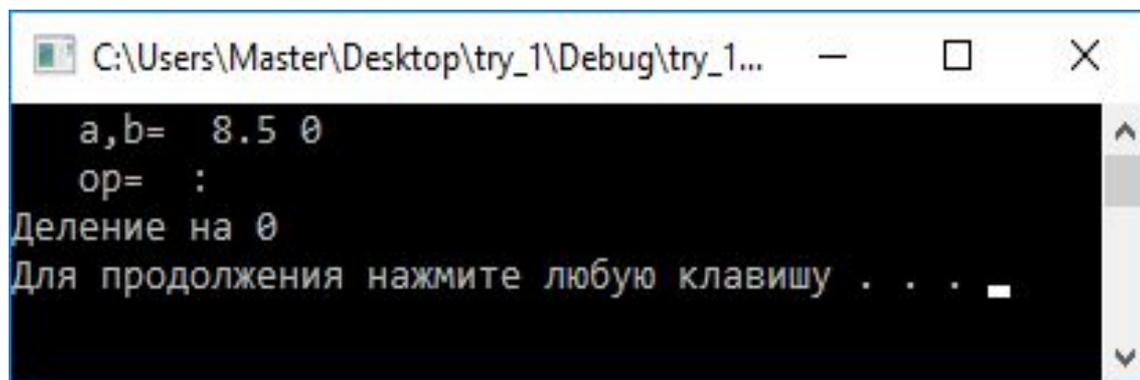
Результаты нормальной работы программы:



```
C:\Users\Master\Desktop\try_1\Debug\tr...  
a,b= 6.2 7.8  
op= +  
6.2 + 7.8 = 14  
Для продолжения нажмите любую клавишу . . .
```



Результаты запуска программы при возникновении ситуации деление на 0:



```
C:\Users\Master\Desktop\try_1\Debug\try_1...  
a,b= 8.5 0  
op= :  
Деление на 0  
Для продолжения нажмите любую клавишу . . .
```

Результаты запуска программы при возникновении ситуации неверно выбрана операция:



```
C:\Users\Master\Desktop\try_1\Debu...  
a,b= 7.8 6.7  
op= ^  
Неверно выбрана операция  
Для продолжения нажмите любую клавишу . . .  
_
```

Результаты запуска программы при возникновении ситуации неверно введено число:

```
C:\Users\Master\Desktop\try_1\Debug...  
a,b= 5,6 -7.3  
Неверно введено число  
Для продолжения нажмите любую клавишу . . .  
_
```

## *Класс Exception*

Другим методом идентификации исключительных ситуаций (класс *Exception*) является создание иерархии классов – по классу на каждый вид исключительной ситуации (пространство имен *System*).

В приведенной иерархии исключительные ситуации делятся на ситуации, связанные с работой базы данных (класс *DatabaseException*) и внутренние исключительные ситуации программы (класс *InternalException*). Ошибки базы данных, в свою очередь, делятся: на ошибки соединения (*ConnectDbException*) и ошибки чтения (*ReadDbException*). Среди внутренних исключительных ситуаций выделяют ситуации, связанные с нехваткой памяти (*NoMemoryException*), ситуации, когда при работе программы возникают недопустимые значения (*IllegalValException*).

# *Класс Exception*

