

C++ THEORY

Оглавление

№1 escape - последовательности	5
№ 2 типы данных	5
№3 инкремент и декремент	5
№4 if else	6
№5 switch	6
№6 while.....	6
№7 do while.....	6
№8 for.....	7
№9 break	7
№10 continue	7
№11 goto (лучше не использовать)	7
№12 Массивы	9
№13 Матрицы.....	9
№14 Передача массива в функцию	9
№15 Видимость переменных	10
№16 Параметры функций по умолчанию.....	12
№17 Перегрузка функций.....	12
№18 Фича компилятора – красивая строка	12
№19 Шаблонные функции	13
№20 Рекурсия.....	13
№21 Указатели	13
№22 Ссылки	14
№23 Прямой запрос памяти (new/delete)	15
№24 Массив указателей (создание и чистка).....	16
№25 Матрица указателей (создание и чистка + транспонирование)	16
№26 Конкатенация (объединение) строк	17
№27 Указатель на функцию	18
№28 Директива препроцессора, макросы (#define).....	19
№29 Тернарный оператор (аналог if else)	19
№30 Параметры argc и argv[]	20
№31 Начало ООП	20
№32 Классы (подобие БД).....	20
№33 Методы (функции) классов	21

№34 Модификаторы доступа классов	22
№35 Геттеры и сеттеры (инкапсуляция)	22
№36 Кофемашина (инкапсуляция)	24
№37 Конструктор класса	24
№38 Перегрузка конструктора класса.....	25
№39 Деструктор класса	25
№40 Ключевое слово «this»	27
№41 Конструктор копирования	27
№42 Перегрузка оператора присваивания.....	28
№43 Перегрузка оператора равенства и неравенства для класса Point	30
№44 Перегрузка оператора сложения для класса Point	30
№45 Перегрузка инкремента и декремента	30
№46 Перегрузка оператора индексирования	31
№47 Дружественные функции.....	31
№48 Определение методов вне класса.....	32
№49 Метод класса, дружественный другому	33
№50 Дружественные классы	34
№51 Static переменная и метод класса	34
№52 Вложенные классы.....	36
№53 Массив объектов класса	37
№54 Агрегация и композиция	39
№55 Наследование классов в ООП	40
№56 Модификаторы доступа при наследовании	41
№57 Порядок вызова конструкторов (и деструкторов) при наследовании	44
№58 Конструктор базового класса из дочернего.....	44
№59 Виртуальные методы класса (virtual и override), полиморфизм.....	45
№60 Абстрактный класс	47
№61 Виртуальный деструктор класса (работа с дин. памятью в наследниках).....	49
№62 Полностью виртуальный деструктор класса.....	51
№63 Делегирующий конструктор класса.....	51
№64 Обращение к виртуальной функции базового класса	53
№65 Множественное наследование.....	54
№66 Интерфейс к классам	56
№67 Ромбовидное наследование.....	57
№67 Работа с файлами.....	59
№68 Запись объекта класса в файл	61

№69 Чтение и запись fstream.....	65
№70 Перегрузка операторов ввода и вывода >> и <<.....	66
№71 Обработка исключений try catch	67
№72 Генерация исключений.....	67
№73 Свой exception	69
№74 Enum	70
№75 Пространство имён	71
№76 Шаблонные классы и их специализации	71
№77 Структуры vs классы	72
№78 Умные указатели	72
№79 Умные указатели и динамические массивы.....	76
№80 Вектор.....	76
№81 Итераторы	78
№82 Ключевое слово auto.....	81
СОРТИРОВКИ.....	82
1. Пузырёк.....	82
2. Вставками.....	82
3. Выбором.....	83
ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ	85
0. Два вида односвязных списков:.....	85
1. Односвязный список	86
Реализация односвязного списка	86
2. Двусвязный список.....	89
3. Бинарное дерево.....	91
4. Стек	92
5. Очередь	95
№82 Многофайловый проект	95
№83 Цикл for each.....	96
№84 Порядок удаления объектов (static, глобальных и простых)	97
№85 Консоль (формат)	98
БИБЛИОТЕКА STL	98
1. array STL.....	98
2. Итераторы STL.....	101
3. deque STL.....	103
4. list STL	103
5. set STL	104

6. multiset STL.....	107
7. map STL.....	108
8. multimap STL.....	110

№1 escape - последовательности

\b	Удаление последнего выведенного символа
\n	Перейти на начало новой строки
\t	Перейти к следующей позиции табуляции

\\	Вывести обратную черту \
\"	Вывести двойную кавычку "
'	Вывести одинарную кавычку '

№ 2 типы данных

Числа с плавающей точкой			
Тип	Размер в байтах	Пояснение	
float	4	описывает вещественные числа одинарной точности	
double	8	описывает вещественные числа двойной точности	

Целые числа			
Тип	Размер в байтах	Пояснение	Диапазон значений
int	4	описывает целые числа	от -2147483648 до 2147483647
short	2	описывает короткие целые числа	от -32768 до 32767
long	4	описывает длинные целые числа	от -2147483648 до 2147483647

Символьный тип			
Тип	Размер в байтах	Пояснение	
char	1	описывает символы	

Логический тип			
Тип	Размер в байтах	Пояснение	Значения
bool	1	описывает логические значения	true false

№3 инкремент и декремент

```
#include <iostream>
using namespace std;

void main()
{
    int a = 1,b;
    b = ++a*a++;
    cout << b << endl;
    cout << a << endl;
}
```

4
3
Для

№4 if else

```
if (a % 2 == 0)
    cout << "Число "<<a<<" чётное!\n";
else
    cout << "Число "<<a<<" нечётное!\n";
```

№5 switch

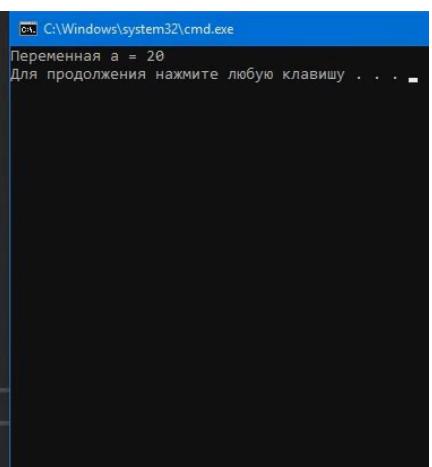
```
11     switch (выражение) {
12         case значение1:
13             действие1;
14             break;
15         case значение2:
16             действие2;
17             break;
18         case значение3:
19             действие3;
20             break;
21         .....
22     default:
23         действие_по_умолчанию;
24         break;
25     }
```

№6 while

```
11     while (утверждение)
12     {
13         действие для повторения;
14     }
```

№7 do while

```
7     void main()
8     {
9         setlocale(LC_ALL, "ru");
10
11     int a=20;
12
13     do
14     {
15         cout << "Переменная a = " << a << endl;
16         a++;
17
18     } while (a<20);
```



№8 for

```
7     for (int i = 0; i < 5; i++)
8     {
9         cout << "Текст\n";
10    }
```

№9 break

```
void main()
{
    setlocale(LC_ALL, "ru");
    int a=0;
    while (true) {
        cout << a<<endl;
        a++;
        if (a == 5)
        {
            cout << a << endl;
            cout << "Stop!\n";
            break;
        }
    }
    cout << "\nEnd of program.\n";
}
```

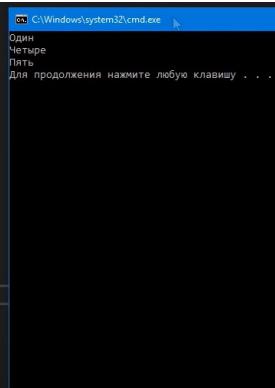
№10 continue

```
//печатаем только нечётные
#include <iostream>
using namespace std;
void main()
{
    setlocale(LC_ALL, "ru");
    int a, b;
    cout << "Enter the left border of the range: ";
    cin >> a;
    cout << "Enter the right border of the range: ";
    cin >> b;
    for (a; a <= b; a++)
    {
        if (a % 2 == 0)
            continue;
        cout << "a=" << a << endl;
    }

    cout << "\nEnd of program.\n";
}
```

№11 goto (лучше не использовать)

```
6 void main()
7 {
8     setlocale(LC_ALL, "ru");
9
10    cout << "Один" << endl;
11
12    goto link;
13
14    cout << "Два" << endl;
15
16    cout << "Три" << endl;
17
18
19    link:
20
21    cout << "Четыре" << endl;
22
23    cout << "Пять" << endl;
24
```



№12 Массивы

```
//инициализация (нулями)
int arr[]{};  
  
//вывод
const int SIZE=5;
int arr[SIZE]{ 1,2,3,4,5 };  
  
for (int i=0; i<SIZE; i++)
{
    cout << arr[i] << " ";  

}  
  
//считаем кол-во эл-тов
int arr[]{ 1,2,3,4,5 };
int n = sizeof(arr) / sizeof(arr[0]);
```

№13 Матрицы

```
#include <iostream>
#include <ctime>
using namespace std;  
  
int main()
{
    setlocale(LC_ALL, "ru");
    srand(time(NULL));
    const int n = 3, m = 4;
    int i, j, arr[n][m]{};//иниц. нулями  
  
    for (i = 0; i < n; i++) //заполнение
    {
        for (j = 0; j < m; j++)
        {
            arr[i][j] = rand() % 10; //рандомим
        }
    }
    cout << "Your matrix:\n\n";
    for (i = 0; i < n; i++) //вывод
    {
        for (j = 0; j < m; j++)
        {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
    cout << "\nEnd of program.\n";
    return 1;
}
```

№14 Передача массива в функцию

Ничем не отличается от простого Си, в функцию автоматически предаётся указатель на массив

```
#include <iostream>
```

```

#include <ctime>
using namespace std;
void fillArray(int arr[], const int SIZE);
void printArray(int arr[], const int SIZE);

int main()
{
    setlocale(LC_ALL, "ru");
    srand(time(NULL));
    const int SIZE = 6;
    int i, arr[SIZE];
    fillArray(arr, SIZE);
    printArray(arr, SIZE);
    cout << "\n\nEnd of program.\n";
    return 1;
}

void fillArray(int arr[], const int SIZE)
{
    for (int i = 0; i < SIZE; i++) //заполнение
    {
        arr[i] = rand() % 10; //рандомим
    }
}

void printArray(int arr[], const int SIZE)
{
    cout << "Your array:\n\n";

    for (int i = 0; i < SIZE; i++) //выводим
    {
        cout << arr[i] << " ";
    }
}

```

№15 Видимость переменных

(тут а – глобальная переменная, рез-т = 2)

```

6 int a;
7
8
9 void foo()
10 {
11     a++; // Local 'a' declaration
12 }
13
14 void main()
15 {
16     foo(); // Calls the function
17     a++; // Global 'a' is incremented again
18
19     cout << a << endl;
20 }

```

(а тут foo работает с переданной переменной, рез-т = 1)

```
6  int a;
7
8
9  void foo(int a)
10 {
11     a++; // ≤ 1 мс прошло
12 }
13
14 void main()
15 {
16     foo(2);
17     a++;
18
19     cout << a << endl;
20 }
```

№16 Параметры функций по умолчанию

(Можно изначально указывать значение переменной [по умолчанию], но, тем не менее, оно изменяется, можно просто на это место передать своё значение. Если в объявлении функции есть переменная без значения по умолчанию, то она должна стоять первой [что логично]).

```
4  /*Аргументы (параметры) по умолчанию*/
5
6  void foo(int q, int a = 5, double b = 0.5)
7  {
8      for (int i = 0; i < a; i++)
9      {
10          cout << "#" << endl;
11      }
12  }
13
14  void main()
15  {
16      foo(4);
17 }
```

№17 Перегрузка функций

(Суть в том, что несколько функций можно назвать одинаково, но давать им разные входные значения. По передаваемым параметрам компилятор сам возьмёт ту функцию, которая подойдёт).

```
7  int Sum(int a, int b, int c)
8  {
9      a ++ ;
10     return a + b + c;
11 }
12
13 int Sum(int a, int b)
14 {
15     return a + b;
16 }
17
18
19 double Sum(double a, double b)
20 {
21     return a + b;
22 }
```

№18 Фича компилятора – красивая строка

Фишка



Ggaming Год назад

А какой комбинацией клавиш красиво ide оформляет строку?



ОТВЕТИТЬ



Mukha Vertoleт Год назад

@Ggaming ctrl + k + d



ОТВЕТИТЬ

№19 Шаблонные функции

(Позволяют не писать кучу перегрузок для функций, а делать их универсальными для разных типов)

```
//ШАБЛОННАЯ ФУНКЦИЯ
#include <iostream>
using namespace std;

template <class T1, class T2, class T3>
void print(T1 a, T2 b, T3 c)
{
    cout << a << endl
        << b << endl
        << c << endl;
}

int main()
{
    setlocale(LC_ALL, "ru");
    print(2, 3.4, "слово");
    print('q', 3, 5.66);
}
```

№20 Рекурсия

```
//Факториал рекурсией
#include <iostream>
using namespace std;

int fact(int a);

int main()
{

    int a;
    cout << "Enter the number for factorial: ";
    cin >> a;
    setlocale(LC_ALL, "ru");
    cout << fact(a) << endl;
    return 0;
}

int fact(int a)
{
    if (a == 0)
        return 1;
    return a * fact(a - 1);
}
```

№21 Указатели

(не отличаются от си)

```

7  /*
8
9
10 void Foo(int *pa)
11 {
12     (*pa) = 555;
13 }
14
15 void main()
16 {
17     int a = 0;
18     cout << a << endl;
19
20     Foo(&a);
21
22     cout << a << endl;
23 }
```

0
555
Для про

№22 Ссылки

//РАЗЛИЧИЯ ССЫЛОК, УКАЗАТЕЛЕЙ И ПЕРЕМЕННЫХ

```

#include <iostream>
using namespace std;

void func(int& a)
{
    a = 10;
    return;
}

int main()
{
    setlocale(LC_ALL, "ru");
    int a = 5;
    int& aRef = a;
    int* pa = &aRef;

    cout << "Значение:\tАдрес:\n";
    cout << "a=" << a << "\t\t" << &a << endl;
    cout << "aRef=" << aRef << "\t\t" << &aRef << endl;
    cout << "pa=" << pa << "\t\t" << &pa << endl;

    func(aRef);

    cout << "\nFunc was\n\n";

    cout << "a=" << a << "\t\t" << &a << endl;
    cout << "aRef=" << aRef << "\t\t" << &aRef << endl;
    cout << "pa=" << pa << "\t\t" << &pa << endl;

    return 0;
}
```

//ПЕРЕДАЧА ССЫЛОК, УКАЗАТЕЛЕЙ И ПЕРЕМЕННЫХ В ФУНКЦИИ

```

#include <iostream>
using namespace std;

void func1(int a)
{
    a = 1;
}
void func2(int& a)
{
    a = 2;
}
void func3(int* a)
```

```

{
    *a = 3;
}

int main()
{
    setlocale(LC_ALL, "ru");
    int value = 5;
    cout << "value=" << value << endl << endl;

    cout << "func1" << endl;
    func1(value);
    cout << "value=" << value << endl << endl;

    cout << "func2" << endl;
    func2(value);
    cout << "value=" << value << endl << endl;

    cout << "func3" << endl;
    func3(&value);
    cout << "value=" << value << endl << endl;

    return 0;
}

```

№23 Прямой запрос памяти (new/delete)

```

#include <iostream>
using namespace std;

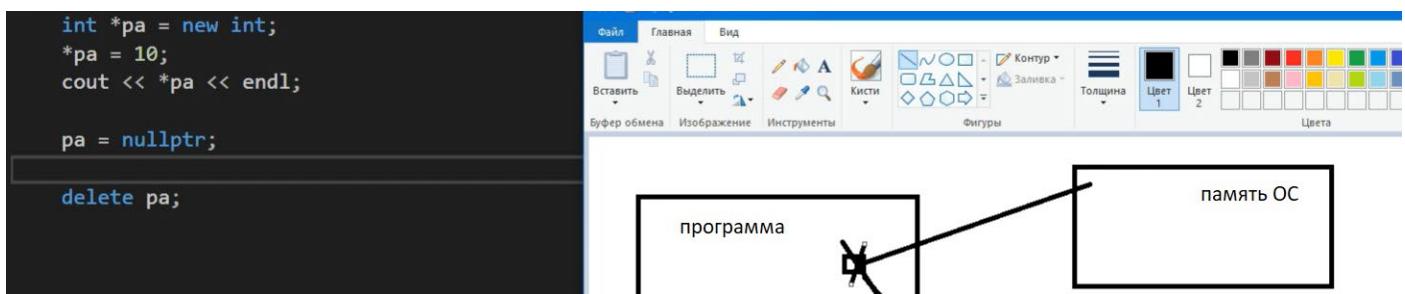
int main()
{
    int* pa = new int; //напрямую попросили у ОС память
    *pa = 10;

    //альтернативное присвоение
    //int* pa = new int(10);

    cout << *pa << endl;
    delete pa; //очистили за собой память (аналог free)
    return 0; //delete удаляет значение, содержащееся в pa
}

```

ВАЖНОЕ ПРО ОЧИСТКУ ПАМЯТИ!



ТАК НЕЛЬЗЯ ↑ - указатель уйдёт, а данные в ОС останутся до окончания программы!

- `delete` затирает данные в ОС
- `nullptr` (как и `NULL`, только не в форме `int`) затирает сам указатель (в программе)
- **стек** – статическая память (размер известен до компиляции)
- **куча** – динамическая память (может меняться в процессе выполнения программы)

№24 Массив указателей (создание и чистка)

```
#include <iostream>
using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");
    int size;
    cout << "Введите кол-во элементов массива: ";
    cin >> size;
    int* arr = new int[size];
    cout << "Массив готов!\n\n";

    for ( int i = 0; i < size; i++) //заполняем массив, выводим его значения и адреса
    {
        arr[i] = i;
        cout << arr[i] << "\t" << arr + i << endl;
    }

    delete[] arr;
    arr = nullptr;
    cout << "\nИ уже почищен...\n";
    return 0;
}
```

№25 Матрица указателей (создание и чистка + транспонирование)

```
#include <iostream>
#include <ctime>
using namespace std;

int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "ru");
    int rows, cols, i, j;
    cout << "Введите кол-во строк матрицы: ";
    cin >> rows;
    cout << "Введите кол-во столбцов матрицы: ";
    cin >> cols;

    int** arr = new int* [rows]; //создаём матрицу
    for (i = 0; i < rows; i++)
    {
        arr[i] = new int[cols];
    }
    cout << "\nМатрица готова!\n\n";

    for (i = 0; i < rows; i++) //заполняем и сразу выводим
    {
        for (j = 0; j < cols; j++)
        {
            arr[i][j] = rand() % 10;
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    cout << endl;
    int temp, temp1;
    if (rows != cols) cout << "Матрица не квадратная, транспонировать нельзя!" << endl;
```

```

else {
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            if (i < j)
            {
                temp = arr[i][j];
                arr[i][j] = arr[j][i];
                arr[j][i] = temp;
            }
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
}

/*
//отдельно печать матрицы
for ( i = 0; i < rows; i++)
{
    for ( j = 0; j < rows; j++)
    {
        cout << arr[i][j] << " ";
    }
    cout << endl;
}
*/



for (i = 0; i < rows) //чистим память, занятую матрицей
{
    delete[] arr[i];
    arr[i] = nullptr;
}
delete[] arr;

arr = nullptr;
cout << "\nМатрица почищена!\n";
return 0;
}

```

№26 Конкатенация (объединение) строк

```

//Работа со строками
#include <iostream>
#include <string>

using namespace std;
void PrintString(const string a, const string b, const string c)
{
    string Fname = "Фамилия: " + a + "\nИмя: " + b + "\nОтчество: " + c;
    cout << Fname;
}

void main()
{
    setlocale(LC_ALL, "ru");
    string str1 = "Дугин";
    string str2 = "Мартин";
    string str3 = "Игоревич";
    PrintString(str1, str2, str3);
}

```

```
    cout << endl;
}
```

№27 Указатель на функцию

Смысл в том, что с помощью указателя на функцию мы можем вызывать какую-либо функцию, не привязываясь к её имени. Указатель должен иметь тот же тип, что и его функция, и принимать те же значения

По сути мы заново пишем прототип функции, но вместо имени нашей функции пишем новое со *

Имя со * (*func, например) – это просто универсальное имя

```
5  /*  
6  * Указатели на функции  
7  */  
8  
9  
10 //тип функции(*имя_указателя)(спецификация_параметров);  
11  
12
```

//ПРИМЕР ПРИМЕНЕНИЯ УКАЗАТЕЛЕЙ НА ФУНКЦИИ

```
#include <iostream>  
#include <string>  
  
using namespace std;  
string GetDataFromBD()  
{  
    return "DataFromBD";  
}  
string GetDataFromWebServer()  
{  
    return "DataFromWebServer";  
}  
string GetDataFromAstral()  
{  
    return "DataFromAstral";  
}  
int MultTo2(int a)  
{  
    return a *= 2;  
}  
void ShowInfo(string(*func)())  
{  
    cout << func() << endl;  
}  
  
void main()  
{  
    string(*fooPtr)(); //создали указатель на функцию  
    fooPtr = GetDataFromBD; //присвоили  
    cout << fooPtr() << endl; //вывели указатель  
  
    int(*funcPtr)(int a); //аналогично с другой  
    funcPtr = MultTo2;  
    cout << funcPtr(2) << endl;  
  
    ShowInfo(GetDataFromWebServer); //вызвали функцию, принимающую другую функцию  
}
```

№28 Директива препроцессора, макросы (#define)

```
//МАКРОС
#include <iostream>
#define sum(x,y) (x+y)
using namespace std;

void main()
{
    cout << sum(5.6, 6) << endl;
}
```

№29 Тернарный опреатор (аналог if else)

```
12
13     int a;
14
15     cout << " введите значение переменной а " << endl;
16     cin >> a;
17
18
19     if (a < 10)
20     {
21         cout << "а меньше 10" << endl;
22     }
23     else
24     {
25         cout << "а больше 10" << endl;
26     }
27
28
29     (a < 10) ? (cout << "а меньше 10" << endl) : (cout << "а больше 10" << endl);
```

else if через оператор

```
19     if (a < 10)
20     {
21         cout << "а меньше 10" << endl;
22     }
23     else if (a > 10)
24     {
25         cout << "а больше 10" << endl;
26     }
27     else
28     {
29         cout << "а равна 10" << endl;
30     }
```

```
19     if (a < 10)
20     {
21         cout << "а меньше 10" << endl;
22     }
23     else
24     {
25         if (a > 10)
26         {
27             cout << "а больше 10" << endl;
28         }
29         else
30         {
31             cout << "а равна 10" << endl;
32         }
33     }
34
35
36     (a < 10) ? (cout << "а меньше 10" << endl) : (a > 10) ? (cout << "а больше 10" << endl) : (cout << "а равна 10" << endl);
```

№30 Параметры argc и argv[]

1 – кол-во параметров, переданных в функцию

2 какие именно параметры (param1, например)

The screenshot shows a code editor with a C++ file containing the following code:

```
4  /*
5   *      int argc, char* argv[]
6   */
7
8
9 void main(int argc, char* argv[])
10 {
11     for (int i = 0; i < argc; i++)
12     {
13         cout << argv[i] << endl;
14     }
15
16     system("pause");
17 }
```

Two red arrows point to the variables `argc` and `argv[]` in the `main` function signature. To the right of the code, a terminal window shows the execution of the program and its output. The first run shows the output of `www.google.com`. The second run shows the output of `param1 param2`. The third run shows the output of `dsds dsdsdsad dsadadasd dodoqnb01 da08dha0sd`.

№31 Начало ООП

Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Инкапсуляция неразрывно связана с понятием интерфейса класса. По сути, всё то, что не входит в интерфейс, инкапсулируется в классе.

Наследование – (заимствование функций другого класса) это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом.

Полиморфизм – (что-то вроде перегрузки функции, способность класса работать в разных ситуациях) это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

№32 Классы (подобие БД)

```
//Классы человек
#include <iostream>
#include <string>
using namespace std;

class Human //класс
{
public: //тело класса
    int age;
    int weight;
```

```

        string name;
};

void main()
{
    Human first; //объект класса
    first.age = 30;
    first.weight = 70;
    first.name = "Rudnitskiy Oleg Anatolievich";

    Human second; //объект класса
    second.age = 18;
    second.weight = 65;
    second.name = "Rudnitskiy Artem Olegovich";

    cout << first.age << endl << first.weight << endl << first.name << endl
        << "-----\n"
        << second.age << endl << second.weight << endl << second.name << endl;
}

```

№33 Методы (функции) классов

Простая функция, которая пишется в теле класса и может работать с его переменными.

```

//Функции класса
#include <iostream>
#include <string>
using namespace std;

class Human //класс
{
public: //тело класса
    int age;
    int weight;
    string name;

    void print() //функция класса
    {
        cout << "Возраст: " << age << "\nВес: " << weight << "\nИмя: " << name << endl <<
    endl;
    }

    void conc()
    {
        cout << name + " Clown ahaha\n";
    }
};

void main()
{
    setlocale(LC_ALL, "ru");
    Human first; //объект класса
    first.age = 30;
    first.weight = 70;
    first.name = "Rudnitskiy Oleg Anatolievich";

    Human second;
    second.age = 18;
    second.weight = 65;
    second.name = "Rudnitskiy Artem Olegovich";

    first.print();
    second.print();
    second.conc();
}

```

№34 Модификаторы доступа классов

public: – то, что можем видеть и вызывать из main

private: – то, что видим в классе, можем там использовать, но не видим в main

protected: – как private, только может стать public при наследовании (???)

Видимость модификатора – до встречи другого.

```
28  {
29      public:
30          int x;
31
32      void Print()
33      {
34          cout << "y = " << y << "\nX = " << x << "\n" << "z = " << z << endl;
35          PrinY();
36      }
37
38  private:
39      int y;           | Сокрытие части функционала - пример декапсуляции
40      int z;
41
42      void PrinY()
43      {
44          cout << y << endl;
45          cout << "PrinY()" << endl;
46      }
```

№35 Геттеры и сеттеры (инкапсуляция)

```
//Геттеры и сеттеры
#include <iostream>
using namespace std;

class Point
{
private:
    int x;
    int y;
public:
    int GetX() //геттер (для x) - функция получения чего-либо
    {
        return x;
    }

    void SetX(int valueX) // сеттер (для x) - функция присвоения значения
    {
        x = valueX;
    }

    void Print() //т.о. в public создали интерфейс класса
    {
        cout << "X= " << x << "\tY= " << y << endl << endl;
    }
};

void main()
```

```
{  
    setlocale(LC_ALL, "ru");  
    Point a;  
    a.SetX(5);  
    a.Print();  
}
```

№36 Кофемашина (инкапсуляция)

```
//Кофемашина
#include <iostream>
#include <string>
using namespace std;

class CoffeGrinder
{
private:
    bool CheckVoltage() //скрытая (инкапсулированная) функция кофемашины, которую мы будем
    //вызывать при запуске
    {
        srand(time(NULL));
        int check = rand() % 2;
        if (check == 1)
            return true;
        else
            return false;
    }

public:
    void Start()
    {
        if (CheckVoltage()) //если напряжение в порядке
            cout << "Вжуух!\n"; //завестись
        else
            cout << "Бип-Бип!\n"; //иначе сигнал
    }
};

void main()
{
    setlocale(LC_ALL, "ru");
    CoffeGrinder kitchen; //объект класса кофемашины
    kitchen.Start(); //используем функцию кофемашины "старт"
}
```

№37 Конструктор класса

Используется для того, чтобы описать способ инициализации объекта класса

```
52     class Point
53     {
54
55     private:
56         int x;
57         int y;
58
59     public:
60
61     Point(int valueX, int valueY)
62     {
63         x = valueX; ≤ 1мс прошло
64         y = valueY;
65     }
66     void SetY(int valueY){ ... }
67
68     int GetY(){ ... }
69
70     int GetX(){ ... }
71
72     void SetX(int valueX){ ... }
73
74     void Print()
```

конструктор

```
98     int main()
99     {
100        Point a(5,11);[ ]
101        a.Print();
102
103        return 0;
```

№38 Перегрузка конструктора класса

Как и перегрузка функции, часть полиморфизма

```
117  int main()
118  {
119      Point a;
120      a.Print();
121
122      Point b(4, 14);
123      b.Print();
124
125      Point c(22, false);
126      c.Print();
```



```
57  public:
58          описанный конструктор по умолчанию
59  Point()
60  {
61      x = 0;
62      y = 0;
63  }
64
65  Point(int valueX, int valueY)
66  {
67      x = valueX;
68      y = valueY;
69  }
```



```
71  Point(int valueX, bool boolean)
72  {
73      x = valueX;
74
75      if (boolean)
76      {
77          y = 1;
78      }
79      else
80      {
81          y = -1;
82      }
83
84 }
```

№39 Деструктор класса

То же, что и конструктор, но вызывается при окончании функции, в которой создавался класс

Деструкция происходит как в стеке – в порядке, обратном порядку конструкции

```
//Деструктор в действии!
```

```

#include <iostream>
using namespace std;

class Tost
{
    int data;

public:

    Tost(int value)
    {
        data = value;
        cout << "Объект " << data << " Вызывался конструктор" << endl;
    }

    ~Tost()
    {
        cout << "Объект " << data << " Вызывался деструктор " << endl;
    }
};

void Making()
{
    cout << "Начало функции\n";
    Tost three(3);
    cout << "Конец функции\n";
}

void main()
{
    setlocale(LC_ALL, "ru");
    Tost one(1);
    Making();
    Tost two(2);
}

```

//Использование деструктора на практике (удаление массива, создаваемого в конструкторе)

```

#include <iostream>
using namespace std;

class Tost
{
    int* arr;
    int number;

public:

    Tost(int size)
    {
        number = size;
        cout << "Объект " << number << " Вызывался конструктор" << endl;
        arr = new int[size];
        for (int i = 0; i < size; i++)
        {
            arr[i] = i+1;
            cout << arr[i] << " ";
        }
        cout << endl;
    }

    ~Tost()
    {
        delete[] arr;
    }
}

```

```

        cout << "Объект " << number << " Вызвался деструктор " << endl;
    }
};

void Making()
{
    cout << "Начало функции\n";
    Tost three(3);
    cout << "Конец функции\n";
}

void main()
{
    setlocale(LC_ALL, "ru");
    Tost one(20);
}

```

№40 Ключевое слово «this»

`this->` используется в теле класса, содержит в себе адрес объекта. С помощь. конструкции `->` можно указать именно ту переменную, что принадлежит классу (будь то `public` или `private`)

```

75     void SetY(int y)
76     {
77         this->y = y;
78     }

```

№41 Конструктор копирования

Примеры, когда срабатывает конструктор копирования:

- при передаче объекта в функцию по значению
- при возврате (через `return`) объекта из функции – в `main` создаётся пустой объект (именно с помощью конструктора копирования), в который копируется возвращенное значение
- когда при создании объекта мы передаём в конструктор как аргумент какой-то объект `a(b)`

Смысл в том, что конструктор копирования, как и другие, пишется компилятором по умолчанию, но не всегда может подходить. Например, если объект содержит какой-нибудь массив, то при копировании этого объекта указатель на массив будет просто скопирован → передаст адрес массива. Это значит, что в деструкторе, в котором мы обязаны почистить за собой память, будет дважды чиститься один и тот же участок памяти, что недопустимо. Чтобы понять, о чём здесь говорится, достаточно просто удалить прописанный конструктор копирования ☺

```

//Значимость конструктора копирования (тут прописанный)
#include <iostream>
using namespace std;

class Tost
{
private:
    int* arr;

```

```

int size;

public:
    Tost(int value)
    {
        cout << "Объект " << this << " Вызвался конструктор" << endl;
        this->size = value;
        this->arr = new int[value];
        for (int i = 0; i < value; i++)
        {
            arr[i] = i+1;
            cout << arr[i] << " ";
        }
        cout << endl;
    }

    ~Tost()
    {
        cout << "Объект " << this << " Вызвался деструктор " << endl;
        delete[] arr;
    }

    Tost(const Tost& other) //прописываем конструктор копирования
    {
        cout << "Объект " << this << " Вызвался конструктор копирования" << endl;
        this->size = other.size; //для полного подобия необходимо присвоить размер
        this->arr = new int[other.size]; //создаём массив такого же размера, что и у
        копируемого объекта

        for (int i = 0; i < other.size; i++) //поячеично копируем
        {
            this->arr[i] = other.arr[i];
        }
    };

void main()
{
    setlocale(LC_ALL, "ru");
    Tost one(3); //создаём объект
    Tost two(one); //копируем первый объект во второй
}

```

№42 Перегрузка оператора присваивания

<https://itnan.ru/post.php?c=1&p=308890> – перегрузка всех операторов

Опять же возможно возникновение проблемы: если мы просто присваиваем один объект другому, то их динамическая память объединяется, что приводит к ошибке во время delete. Необходимо отдельно это прописать.

```

Tost& operator = (const Tost& other)
{
    this->size = other.size; // меняем размер массива

```

```
if (this->arr != nullptr) //если указатель не пустой, то чистим
{
    delete[] arr;
}

this->arr = new int[this->size]; // заново выделяем память

for (int i = 0; i < other.size; i++) // поочеично копируем
{
    this->arr[i] = other.arr[i];
}

return *this;
}
```

```
134     MyClass & operator = (const MyClass &other)
135     {
136
137         cout << "Вызвался оператор = " << this << endl;
138
139         this->Size = other.Size;
140
141         if (this->data!=nullptr)
142         {
143             delete[] this->data;
144         }
145
146         this->data = new int[other.Size];
147
148         for (int i = 0; i < other.Size; i++)
149         {
150             this->data[i] = other.data[i];
151         }
152
153         return *this; [ ]
154     }
```

№43 Перегрузка оператора равенства и неравенства для класса Point

```
71
72     bool operator ==(const Point & other)
73     {
74         return this->x == other.x && this->y == other.y;
75     }
76
77     bool operator !=(const Point & other)
78     {
79         return !(this->x == other.x && this->y == other.y);
80     }
81 }
```

```
bool operator ==(const Point& other) //перегрузка оператора равенства
{
    return this->x == other.x && this->y == other.y;
}

bool operator !=(const Point& other) //перегрузка оператора неравенства
{
    return !(this->x == other.x && this->y == other.y);
}
```

№44 Перегрузка оператора сложения для класса Point

```
82     Point operator +(const Point & other)
83     {
84         Point temp;
85         temp.x = this->x + other.x;
86         temp.y = this->y + other.y;
87         return temp;
88     }
89 }
```

```
Point operator +(const Point& other) //перегрузка оператора сложения
{
    Point temp;
    temp.x = this->x + other.x;
    temp.y = this->y + other.y;
    return temp;
}

Point operator -(const Point& other) //перегрузка оператора вычитания
{
    Point temp;
    temp.x = this->x - other.x;
    temp.y = this->y - other.y;
    return temp;
}
```

№45 Перегрузка инкремента и декремента

```
135     Point & operator --()
136     {
137         this->x--;
138         this->y -= 1;
139     }
140     return *this;
141 }
142
143     Point & operator --(int value)
144     {
145         Point temp(*this);
146
147         this->x--;
148         this->y--;
149
150         return temp;
151     }
```

```
//предфиксный инкремент
Point & operator ++()
{
    this->x++;
    this->y += 1;
    return *this;
}

//постфиксный инкремент
Point & operator ++(int value)
{
    Point temp(*this);
    this->x++;
    this->y++;
    return temp;
}
```

№46 Перегрузка оператора индексирования

```
227 class TestClass
228 {
229 public:
230
231     int & operator[](int index)
232     {
233         return arr[index];
234     }
235
236 private:
237     int arr[5]{5,44,4,987,69};
238 };
```

№47 Дружественные функции

```
//Дружественные функции
#include <iostream>
using namespace std;

class Point;

class Test
{
    int data;

    friend void ChangeValues(Point& value, Test& test); //дружим функции и классы
```

```

public:
    Test()
    {
        data = 33;
    }

    void Print()
    {
        cout << "data = " << this->data << endl;
    }

};

class Point
{
    int x;
    int y;

public:

    friend void ChangeValues(Point& value, Test& test); //дружим функции и классы

    Point()
    {
        x = 0;
        y = 0;
    }

    void Print()
    {
        cout << "X = " << this->x << "\tY = " << this->y << endl;
    }
};

void ChangeValues(Point& value, Test & test) //дружественная функция по отношению к Point и Test
{
    value.x += 1;
    test.data += 1;
}

void main()
{
    setlocale(LC_ALL, "ru");
    Point a;
    a.Print();
    Test b;
    b.Print();
    ChangeValues(a, b);
    a.Print();
    b.Print();
}

```

№48 Определение методов вне класса

```

294     class MyClass
295     {
296         public:
297             void PrintMessage();
298
299     };
300
301     I
302
303
304
305     void MyClass::PrintMessage()
306     {
307         cout << "Hello!" << endl;
308

```

№49 Метод класса, дружественный другому

```

//Дружественный метод класса
#include <iostream>
#include <string>
using namespace std;

class Apple;

class Human //важно, чтобы Apple описывалось после Human (????)
{
public:
    void TakeApple(Apple& apple);
};

void Human::TakeApple(Apple& apple) //чтобы метод сделать дружественным, надо вынести его из класса
и
{
    cout << "Take apple\n" << "weight is " << apple.weight << "\ncolor is " << apple.color <<
endl;
}

class Apple
{
public:

    Apple(int weight, string color)
    {
        this->weight = weight;
        this->color = color;
    }

    friend void Human::TakeApple(Apple& apple); // и указать то, что эта вынесенная функция friend

private:

    int weight;
    string color;
};

void main()
{
    setlocale(LC_ALL, "ru");
    Apple first(150, "Green");
    Human one;
    one.TakeApple(first);
}

```

№50 Дружественные классы

Открывает дружественному классу доступ к private полям.

The screenshot shows a code editor with the following C++ code:

```
272 class Apple {
273     friend Human; ← объявление дружественности
274
275 public:
276     Apple(int weight, string color)
277     {
278         this->weight = weight;
279         this->color = color;
280     }
281
282 private:
283     int weight;
284     string color;
285
286 };
287
288 void Human::TakeApple(Apple & apple)
289 {
290     cout << "TakeApple " << "weight = " << apple.weight << " color = " << apple.color << endl;
291     apple.|
292 }
```

A red arrow points from the word " friend Human;" to the text "объявление дружественности". A tooltip appears over the cursor at line 291, showing the declaration "private : std::string Apple::color" and the file "Файл: Source.cpp".

№51 Static переменная и метод класса

Суть в том, что это общая переменная на все объекты класса, т.е. при изменении её в каком-то объекте она меняется во всех объектах тоже. Особенность static в том, что к ним можно обращаться не от объекта, а хоть от класса (`cout << Apple::GetCount() << endl;` - пример)

Static методы используются так же, как и переменные. Они общие для всех объектов, могут вызываться не от объекта.

```

Apple(int weight, string color)
{
    this->weight = weight;
    this->color = color;
    Count++;
    id = Count;
}

int GetId()
{
    return id;
}

static int GetCount()
{
    return Count;
}

```

private:

```

    static int Count;
    int weight;
    string color;
    int id;

```

```

277 class Apple {
278
279     public:
280
281         static int Count;           статическая переменная
282
283
284
285     Apple(int weight, string color)
286     {
287         this->weight = weight;
288         this->color = color;
289     }
290
291     private:
292         int weight;
293         string color;
294     };
295
296
297     int Apple::Count = 0;        инициализация этой
                                переменной

```

```
cout << Apple::GetCount() << endl;
```

Использование (пример):

```

//Простейший id объектов
#include <iostream>
#include <string>
using namespace std;

```

```

class Apple
{
public:

    Apple(int weight, string color)
    {
        this->weight = weight;
        this->color = color;
        count++;
        id = count;
    }

    static int GetCount() //static метод, общий для всех объектов класса, поэтому не может
    использовать, например, this (но исп. static переменные)
    {
        return count;
    }

    int GetId() //простой метод, может использовать this (и static переменные)
    {
        return id;
    }

private:

    int id;
    int weight;
    string color;
    static int count;

};

int Apple::count = 0;

void main()
{
    setlocale(LC_ALL, "ru");
    Apple apple1(150, "Green");
    Apple apple2(100, "Red");
    Apple apple3(150, "Green");
    cout << apple1.GetId() << endl;
    cout << Apple::GetCount() << endl;
}

```

№52 Вложенные классы

```

//ВЛОЖЕННЫЕ КЛАССЫ
#include <iostream>
#include <string>
using namespace std;

class Image
{
public:

    void GetImageInfo()
    {
        for (int i = 0; i < LENGTH; i++)
        {
            cout << "#" << i << " " << pixels[i].GetInfo() << endl;
        }
    }

private:

```

```

//вложенный в Image класс Pixel
//вложенные классы ведут себя так же, как и простые, только вложенный можно, например,
спрятать в другом
class Pixel
{
public:
    Pixel(int r, int g, int b)
    {
        this->r = r;
        this->g = g;
        this->b = b;
    }

    string GetInfo()
    {
        return "Pixel: r = " + to_string(r) + " g = " + to_string(g) + " b = " +
to_string(b);
    }

private:
    int r;
    int g;
    int b;
};

static const int LENGTHT = 5;

Pixel pixels[LENGTHT]
{
    Pixel(1,2,3),
    Pixel(4,5,6),
    Pixel(7,8,9),
    Pixel(10,11,12),
    Pixel(13,14,15)
};

void main()
{
    setlocale(LC_ALL, "ru");
    Image Repin;
    Repin.GetImageInfo();
}

```

№53 Массив объектов класса

Статический:

```

391     const int LENGTH = 5;
392
393     //Pixel p(11, 44, 112);
394
395     Pixel arr[LENGTH]
396     {
397         Pixel(11,12,123),
398         Pixel(123,411,144),
399         Pixel(114,141,114),
400         Pixel(7575,12,623),
401         Pixel(135,411,134),
402     };

```

Динамический:

```

384     [];};
385
386
387     int main()
388     {
389         setlocale(LC_ALL, "ru");
390
391         int LENGTH = 5;
392
393         //Pixel p(11, 44, 112);
394
395         Pixel *arr = new Pixel[LENGTH];
396
397         arr[0] = Pixel(11, 151, 44);
398
399         cout << arr[0].GetInfo() << endl;
400
401         delete []arr;

```

Pixel: r = 11 g = 151 b = 44
Для продолжения нажмите любую клавишу . . .

По сути ничем не отличается от объявления и использования простых массивов. (для объявления пустого необходимо прописанный конструктор)

```

//СТАТИЧЕСКИЙ И ДИНАМИЧЕСКИЙ МАССИВЫ
#include <iostream>
#include <string>
using namespace std;

class Pixel
{
public:

    Pixel()
    {
        r = g = b = 0;
    }

    Pixel(int r, int g, int b)
    {
        this->r = r;
        this->g = g;
        this->b = b;
    }

    string GetInfo()
    {
        return "Pixel: r = " + to_string(r) + " g = " + to_string(g) + " b = " + to_string(b);
    }

private:
    int r;
    int g;
}

```

```

        int b;
};

void main()
{
    setlocale(LC_ALL, "ru");
    const int SIZE = 3;
    Pixel arr_s[SIZE]{ Pixel(1,2,3), Pixel(4,5,6), Pixel(7,8,9) };
    cout << arr_s[1].GetInfo() << endl;

    int size = 2;
    Pixel* arr_d = new Pixel[size];
    cout << arr_d[1].GetInfo() << endl;
}

```

№54 Агрегация и композиция

Композиция – (пример, когда один класс вложен в другой) прямая зависимость классов друг от друга. Один класс не может существовать без второго (т.е. если мы удалим Human, то Brain уничтожится)

Агрегация – один класс использует другой, но используемый класс не зависит от использующего, может существовать и без него (т.е. если мы удалим Human, то Cap останется таким, какой есть).

Метод класса вызывает метод другого класса – это делегирование.

```

//КОМПОЗИЦИЯ И АГРЕГАЦИЯ
#include <iostream>
#include <string>
using namespace std;

class Cap
{
public:
    string GetColor()
    {
        return color;
    }

private:
    string color = "red";
};

//композиция с классом Brain и агрегация с классом Cap
class Human
{
public:

    void Think()
    {
        brain.Think();
    }

    void LookAtTheCap()
    {
        cout << "Human's cap have a " << cap.GetColor() << " color!\n";
    }

private:

```

```

class Brain
{
public:
    void Think()
    {
        cout << "Я думаю!\n";
    }
};

Brain brain;
Cap cap;

};

//пример агрегации с классом Cap
class Doll
{
public:
    void LookAtTheCap()
    {
        cout << "Doll's cap have a " << cap.GetColor() << " color!\n";
    }
private:
    Cap cap;
};

void main()
{
    setlocale(LC_ALL, "ru");
    Human human;
    human.Think();
    human.LookAtTheCap();
    Doll dolly;
    dolly.LookAtTheCap();
}

```

№55 Наследование классов в ООП

Базовый (родительский) класс – тот, от которого происходило наследование

Производный (дочерний) класс – тот, который наследовал от базового

Производный класс наследует всё то, что есть в родительском. Грубо говоря, кусочек с кодом родительского класса вставляется в код дочернего.

```

//НАСЛЕДОВАНИЕ КЛАССОВ
#include <iostream>
#include <string>
using namespace std;

class Human
{
private:
    string name="No Name";

public:
    string GetName()
    {
        return name;
    }

    void SetName(string name)
    {
        this->name = name;
    }
}

```

```

};

//дочерний класс от Human
class Student : public Human
{
public:
    string group;

    void learn()
    {
        cout << "I'm studying!\n";
    }
};

//дочерний класс от Student (и от Human соответственно тоже)
class ExtramuralStudent : public Student
{
public:
    void learn()
    {
        cout << "I study less often than a normal student..\n";
    }
};

//дочерний класс от Human
class Professor : public Human
{
public:
    string subject;
};

void main()
{
    setlocale(LC_ALL, "ru");
    Student st; //создаём экземпляры разных классов
    Professor pr;
    ExtramuralStudent ex;
    ex.learn(); //используем функции, прописанные не в самих классах, а в их родительских
    cout << ex.GetName() << endl;
    pr.SetName("Bob Calls");
    cout << pr.GetName() << endl;
}

```

№56 Модификаторы доступа при наследовании

для полей и методов в классах

public: – даёт доступ в классе, наследниках и любых объектах.
private: – даёт доступ в классе, но не даёт в наследниках и любых объектах.
protected: – даёт доступ в классе и в наследниках, но не даёт доступа в любых объектах.

```

//НАСЛЕДОВАНИЕ КЛАССОВ
#include <iostream>
#include <string>
using namespace std;

class A
{

    //можно использовать везде, даже в объектах

```

```

public:
    string msgOne = "Сообщение один";
    //можно использовать только в классе A
private:
    string msgTwo = "Сообщение два";
    //можно использовать в наследованных классах
protected:
    string msgThree = "Сообщение три";
};

class B : protected A
{
public:
    void PrintMsg()
    {
        cout << msgOne << endl;
    }
};

class C : public B
{
public:
    void PrintMsg()
    {
        cout << msgThree << endl;
    }
};

void main()
{
    setlocale(LC_ALL, "ru");
    B b;
    b.msgOne;
    b.PrintMsg();
    C c;
    c.msgThree;
}

```

для классов

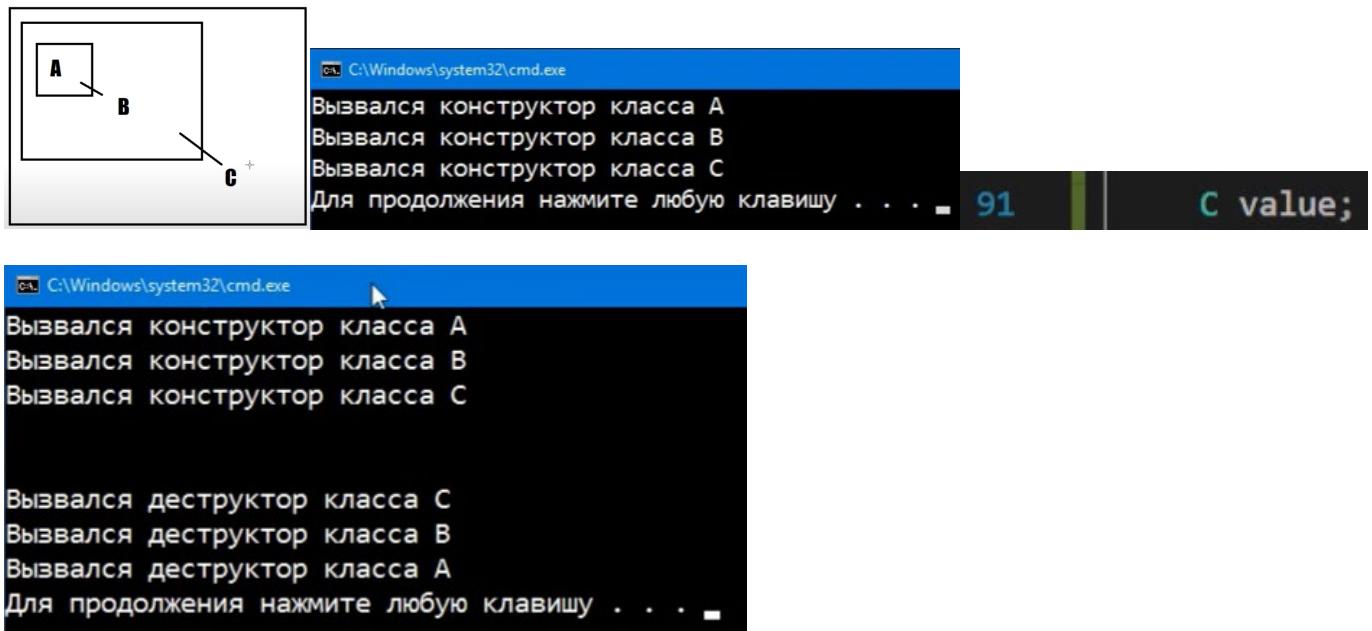
		Исходный модификатор доступа		
		public	private	protected
public-наследование	public	private	protected	
private-наследование	private	private	private	
protected-наследование	protected	private	protected	

```
23  class B : public A
24  {
25      public:
26      void PrintMsg()    может быть
27      {                  private
28          cout << msgThree << endl;
29      }
30  };
31
```



может быть
private
protected

№57 Порядок вызова конструкторов (и деструкторов) при наследовании



№58 Конструктор базового класса из дочернего

//ВЫЗОВ КОНСТРУКТОРА БАЗОВОГО КЛАССА ИЗ КОНСТРУКТОРА КЛАССА-НАСЛЕДНИКА

```
#include <iostream>
#include <string>
using namespace std;

class A
{
public:
    A()
    {
        msg = "Пусто";
    }
    A(string msg)
    {
        this->msg = msg;
    }

    void PrintMsg()
    {
        cout << msg << endl;
    }

private:
    string msg;
};

class B : public A
{
public:
    B()
    {
    }

    /*B() :A("Это В")
```

```
36  B() : A("Это В")
37  {
38
39 }
```

```

{
}*/
```

B(string msg):A(msg)

```

{
```

```

}
```

```

};
```

void main()

```

{
    setlocale(LC_ALL, "ru");
    B b("Hello!");
    b.PrintMsg();
}
```

№59 Виртуальные методы класса (virtual и override), полиморфизм

virtual позволяет переопределять метод базового класса в производных (если его убрать, то с базукой метод shoot игрока будет выводить «BANG»)

override даёт указание компилятору проверять, соблюдены ли все условия переопределения (одинаковые параметры: имя, входные и выходные данные). Не обязательен, но удобен.

```
#include <iostream>

using namespace std;

class Gun
{
public:
    virtual void Shoot() //будет полиморфным методом
    {
        cout << "BANG!";
    }
};

class SubmachineGun : public Gun
{
public:
    void Shoot() override //"полиморфизируется"
    {
        cout << "BANG! BANG! BANG!" << endl;
    }
};

class Bazooka : public Gun
{
public:
    void Shoot() override //"полиморфизируется"
    {
        cout << "BOOOOM!" << endl;
    }
};

class Player
{
public:
```

```
void Shoot(Gun *gun)
{
    gun->Shoot();
}

int main()
{
    setlocale(LC_ALL, "ru");
    Gun gun;
    SubmachineGun machinegun;
    Bazooka bazooka;
    Player player;
    player.Shoot(&bazooka);
    /*Gun* weapon = &machinegun;
    weapon->Shoot();*/
}

return 0;
}
```

№60 Абстрактный класс

Абстрактным (может использоваться для объединения, обобщения предметов – классов) класс становится тогда, когда в нём есть хотя бы один чисто виртуальный метод .

!Нельзя создать объект виртуального класса.

*Абстрактный класс также может содержать и не полностью виртуальные методы, они тоже будут унаследованы. (пример в конце: knife.func();)

*Используется для того, чтобы через указатель на этот класс использовать его наследников (Player shoot, например)

```
#include <iostream>
using namespace std;

class Weapon //тот самый абстрактный класс
{
public:
    virtual void Shoot() = 0; //та самая полностью виртуальная функция
    void func()
    {
        cout << "func is here!\n";
    }
};

class Gun : public Weapon
{
public:
    void Shoot() override //будет полиморфным методом
    {
        cout << "BANG!";
    }
};

class SubmachineGun : public Gun
{
public:
    void Shoot() override //"полиморфизируется"
    {
        cout << "BANG! BANG! BANG!" << endl;
    }
};

class Bazooka : public Weapon
{
public:
    void Shoot() override //"полиморфизируется"
    {
        cout << "BOOOOM!" << endl;
    }
};

class Knife : public Weapon
{
public:
    void Shoot() override //"полиморфизируется"
    {
        cout << "VJUH!" << endl;
    }
};
```

```

class Player
{
public:
    void Shoot(Weapon *gun)
    {
        gun->Shoot();
    }
};

int main()
{
    setlocale(LC_ALL, "ru");
    Knife knife;
    Player player;
    player.Shoot(&knife);
    knife.func();
    return 0;
}

```

Ещё пример наследования от абстрактного класса (цепочка)

Важно! Если виртуальный метод абстрактного класса не перегружен в потомке, то потомок тоже становится абстрактным

```

#include <iostream>
using namespace std;

class A
{
public:
    virtual void printMsg(string msg) = 0;
    virtual void SetRandom() = 0;

protected:
    float x;
};

class B : public A
{
public:
    void printMsg(string msg) override
    {
        cout << msg << endl;
    }

    void SetRandom() override
    {
        this->y = 1;
    }
};

protected:
    float y;
};

class C : public B
{
public:
    void printMsg(string msg) override
    {
        cout << '{' + msg + '}' << endl;
    }
};

```

```

    }

    void SetRandom() override
    {
        this->x = 1;
    }

protected:

    float z;
};

int main()
{
    setlocale(LC_ALL, "ru");

    A* a;

    B b;

    C c;

    a = &c;

    a->SetRandom();

    a->printMsg("Первая строка");

    b.printMsg("Строка");

    return 0;
}

```

№61 Виртуальный деструктор класса (работа с дин. памятью в наследниках)

Если как-то используется динамическая память, то необходимо создавать виртуальный деструктор, чтобы корректно чистить память. (при комбинировании классов могут возникать утечки памяти)

```

int main()
{
    setlocale(LC_ALL, "ru");
    A *bptr=new B;
    delete bptr;
    return 0;
}

```

```

68     class A
69     {
70     public:
71     A()
72     {
73         cout << "выделена динамическая память, объект класса A" << endl;
74     }
75     virtual ~A() ←
76     {
77         cout << "овобождена динамическая память, объект класса A" << endl;
78     }
79 };
80
81 class B: public A
82 {
83 public:
84 B()
85 {
86     cout << "выделена динамическая память, объект класса B" << endl;
87 }
88 ~B() override ←
89 {
90     cout << "овобождена динамическая память, объект класса B" << endl;
91 }
92 };
93

```

```
#include <iostream>

using namespace std;

class A
{
public:
    A();
    virtual ~A();
    void printMsg(string msg);
};

class B : public A
{
public:
    B();
    ~B() override;
};
```

```

int main()
{
    setlocale(LC_ALL, "ru");

    A* ptr = new B;
    ptr->printMsg("Bababa");
    delete ptr;

    return 0;
}

void A::printMsg(string msg)
{
    cout << msg << endl;
}

A::A()
{
    cout << "Выделение динамической памяти А" << endl;
}

A::~A()
{
    cout << "Освобождение динамической памяти А" << endl;
}

B::B()
{
    cout << "Выделение динамической памяти В" << endl;
}

B::~B()
{
    cout << "Освобождение динамической памяти В" << endl;
}

```

№62 Полностью виртуальный деструктор класса

Делает класс обстректным (со всеми плюшками). Очень даже аналогичен полностью виртуальной функции, использоваться может ~так же.

```

68     class A
69     {
70     public:
71     A()
72     {
73     }
74
75     virtual ~A() = 0; ←
76 };
77
78 A::~A() {}; ←
79
80

```

№63 Делегирующий конструктор класса

Очень похоже на наследование классов. т.е. мы можем вставить кусок кода одного конструктора в другой, не дублируя код.

```
69 class Human
70 {
71     public:
72
73     Human(string Name)
74     {
75         this->Name = Name;
76         this->Age = 0;
77         this->Weight = 0;
78     }
79
80     Human(string Name, int Age) :Human(Name)
81     {
82         this->Age = Age;
83     }
84
85     Human(string Name, int Age, int Weight) :Human(Name, Age)
86     {
87         this->Weight = Weight;
88     }
89
90     string Name;
91     int Age;
92     int Weight;
93 };
94
95 
```

```
#include <iostream>
using namespace std;

class Human
{
public:
    Human(string Name, int Age, int Weight)
    {
        this->Name = Name;
        this->Age = Age;
        this->Weight = Weight;
    }

    Human(string Name, int Age) :Human(Name, Age, 0) {}

    Human(string Name) :Human(Name, 0, 0) {}

    void ShowInfo()
    {
        cout << "Имя: " << Name << "\nВозраст: " << Age << "\nВес: " << Weight << endl;
    }

private:
    string Name;
    int Age;
    int Weight;
};

int main()
{
    setlocale(LC_ALL, "ru");

    Human human("Андрей");
    human.ShowInfo();

    return 0;
}
```

№64 Обращение к виртуальной функции базового класса

Проблема в том, что в дочернем классе может понадобиться вызов виртуальной функции (которая переопределена **override**) базового класса. Для того, чтобы указать из какого именно класса функция, нужно использовать синтаксис: **ИмяКласса::ИмяФункции**

```
4  class Msg
5  {
6  public:
7      Msg(string msg)
8      {
9          this->msg = msg;
10     }
11
12     virtual string GetMsg() ← ВОТ ОНА
13     {
14         return msg;
15     }
16
17 private:
18     string msg;
19 };
20
21 class BracketsMsg : public Msg
22 {
23 public:
24     BracketsMsg(string msg) :Msg(msg) {}
25
26     string GetMsg() override ← обращение именно к вирт.
27     {
28         return "[" + Msg::GetMsg() + "]";   функции базового класса
29     }
30 };
```

Пример:

```
#include <iostream>
using namespace std;

class Msg
{
public:
    Msg(string msg)
    {
        this->msg = msg;
    }

    virtual string GetMsg()
    {
        return msg;
    }

private:
    string msg;
};

class BracketsMsg : public Msg
{
public:
    BracketsMsg(string msg) :Msg(msg) {}

    string GetMsg() override
    {
        return "[" + Msg::GetMsg() + "]";
    }
};

class Printer
{
public:
    void Print(Msg* msg)
    {
        cout << msg->GetMsg() << endl;
    }
};
```

```

};

int main()
{
    setlocale(LC_ALL, "ru");
    BracketsMsg msg("Hello, world!");
    Printer printer;
    printer.Print(&msg);

    return 0;
}

```

№65 Множественное наследование

Характеризуется так же, как и простое наследование, но просто класс-наследник может наследовать все поля и методы не одного класса, а сразу нескольких.

```

//МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
#include <iostream>
using namespace std;

class Car
{
public:
    void Drive()
    {
        cout << "Я еду!" << endl;
    }
    string strC = "Поле класса Car";
};

class AirPlain
{
public:
    void Fly()
    {
        cout << "Я лечу!" << endl;
    }
    string strA = "Поле класса AirPlain";
};

class FlyingCar : public Car, public AirPlain {} //класс-наследник классов Car и AirPlain одновременно

int main()
{
    setlocale(LC_ALL, "ru");
    FlyingCar fc;
    fc.Fly(); //у такого наследника есть все методы
    cout << fc.strA << endl; //и все поля родителей
    return 0;
}

```

Порядок вызова конструкторов при множественном наследовании:

Порядок вызова конструкторов зависит от порядка наследований:

```

96 class FlyingCar : public Airplain, public Car
97 {
98     public:
99         FlyingCar()
100     {
101         cout << "Вызван Конструктор FlyingCar" << endl;
102     }
103 };

```

C:\Windows\system32\cmd.exe
Вызван Конструктор Airplain
Вызван Конструктор Car
Вызван Конструктор FlyingCar
Для продолжения нажмите любую клавишу . . .

№66 Интерфейс к классам

Мы можем написать обстрактный класс (который будет интерфейсом) и наследников к нему.

```
//ПИШЕМ ИНТЕРФЕЙС К КЛАССАМ
#include <iostream>
using namespace std;

class iBicycle //интерфейс ко всем велосипедам
{
public:
    void virtual TwistWheels() = 0;
    void virtual Ride() = 0;
};

class SimpleBicycle : public iBicycle //простой велосипед
{
public:
    void TwistWheels() override
    {
        cout << "*Спокойно крутим педальки of SimpleBicycle*\n";
    }

    void Ride() override
    {
        cout << "*Спокойно райдим на SimpleBicycle*\n";
    }
};

class SportBicycle : public iBicycle //простой велосипед
{
public:
    void TwistWheels() override
    {
        cout << "*Быстро крутим педальки of SportBicycle*\n"
            << "Переключаем скорости\n";
    }

    void Ride() override
    {
        cout << "*Быстро райдим на SportBicycle*\n";
    }
};

class Human
{
public:
    void RideOn(iBicycle& bicycle)
    {
        cout << "Крутим педали!\n";
        bicycle.TwistWheels();
        cout << "Поехали!\n";
        bicycle.Ride();
    }
};

void main()
{
    setlocale(LC_ALL, "ru");

    Human human;
    SimpleBicycle sib;
    human.RideOn(sib);
    cout << "\n\nA потоооом...\n\n";
}
```

```

    SportBicycle spb;
    human.RideOn(spb);
}

№67 Ромбовидное наследование
//ПРОСТОЕ МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
#include <iostream>
using namespace std;

class Component
{
public:
    Component(string companyName)
    {
        cout << "Конструктор Component\n";
        this->companyName = companyName;
    }
    string companyName;
};

class GPU : public Component
{
public:
    GPU(string companyName):Component(companyName)
    {
        cout << "Конструктор GPU\n";
    }
};

class Memory : public Component
{
public:
    Memory(string companyName) :Component(companyName)
    {
        cout << "Конструктор Memory\n";
    }
};

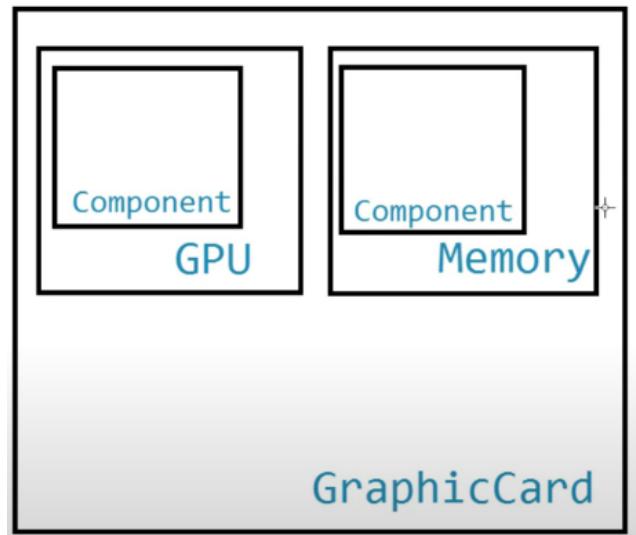
class GraphicCard : public GPU, public Memory
{
public:
    GraphicCard(string GPU companyName, string Memory companyName):GPU(GPU companyName),
Memory(Memory companyName)
    {
        cout << "Конструктор GraphicCard\n";
    }
};

int main()
{
    setlocale(LC_ALL, "ru");

    GraphicCard gc("AMD", "Samsung"); //экземпляр gc имеет в себе два класса component
    cout<<((Memory)gc).companyName<<endl;

    return 0;
}

```



Суть же ромбовидного множественного наследования, что их общий предок создаётся один раз и все поля, общего предка этих virtual наследованных становятся единственными и общими (пример cout << boss.HP << endl – если убрать virtual, то boss.HP будет неоднозначным)


```

//РОМБОВИДНОЕ МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
#include <iostream>
using namespace std;

class Character
{
public:
    Character()
    {
        cout << "Конструктор Character\n";
    }
    int HP=100;
};

class Orc :public virtual Character
{
public:
    Orc()
    {
        cout << "Конструктор Orc\n";
    }
};

class Warrior :public virtual Character
{
public:
    Warrior()
    {
        cout << "Конструктор Warrior\n";
    }
};

class OrcWarrior :public Orc, public Warrior
{
public:
    OrcWarrior()
    {
        cout << "Конструктор OrcWarrior\n";
    }
};

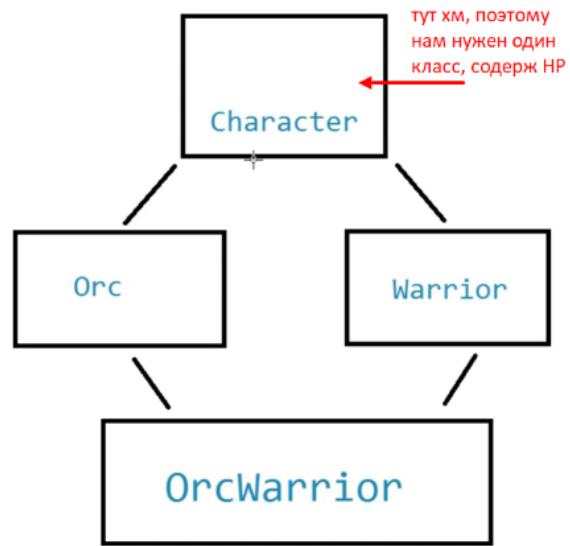
int main()
{
    setlocale(LC_ALL, "ru");

    OrcWarrior boss;

    cout << boss.HP << endl;

    return 0;
}

```



№67 Работа с файлами

КЛАСС для записи файлов – **ofstream** (output file)

```

//ЗАПИСЬ В ФАЙЛ
#include <iostream>
#include <string>
#include <fstream> //ОБЯЗАТЕЛЬНО включить при работе с файлами
using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");

```

```

/*string str;
cout << "Ввод: ";
getline(cin, str);
cout << str<<endl;*/

char choice; //для выбора, как записывать данные в файл

string path = "myFile.txt"; //имя файла можно записывать в переменную

ofstream fout; //создали класса ofstream, отвечающий за вывод в файл

cout << "Очистить старые данные из файла?(y/n)\n"
    << "Введите: ";
cin >> choice;

if (choice == 'y' || choice == 'Y')
    fout.open(path); //открываем наш файл с перезаписью
else if (choice == 'n' || choice == 'N')
    fout.open(path, ofstream::app); //открываем наш файл с записью поверх
else
{
    cout << "Начните думать и пользоваться клавиатурой, пожалуйста...\n";
    return -1;
}

if (!fout.is_open())
{
    cout << "Ошибка открытия файла!" << endl;
    return -1;
}
else
{
    cout << "Введите новые данные:\n ";
    getline(cin, path);
    getline(cin, path);
    fout << path;
    fout << endl;
}

fout.close();
cout << "Готово!"
    << "Завершение работы.\n";

return 0;
}

```

```

//ЧТЕНИЕ ИЗ ФАЙЛА
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");

    string path = "myFile.txt";
    ifstream fin;
    fin.open(path);

    if (!fin.is_open())
    {

```

```

        cout << "Ошибка при раскрытии файла!\n";
        return -1;
    }

    char ch;
    while (fin.get(ch))
    {
        cout << ch;
    }

    fin.close();

    return 0;
}

```

```

23     char ch;
24     while (fin.get(ch)) ПОСИМВОЛЬНЫЙ ВЫВОД
25     {
26         cout << ch;
27     }

```

```

23     string str;
24     while (!fin.eof()) построчный вывод
25     {
26         str = "";
27         getline(fin, str);
28         cout << str << endl;
29     }
30

```

№68 Запись объекта класса в файл

- `fout.write((char*)&point, sizeof(Point))` – записывает по одному объекту
- `fin.read((char*)&point, sizeof(Point))` – считывает по одному объекту и засовывает его в данный

```

//СОХРАНЕНИЕ ОБЪЕКТА В ФАЙЛ
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

class Point
{
public:
    Point()
    {
        x = y = z = 0;
    }
    Point(float x, float y, float z)
    {
        this->x = x;
        this->y = y;
        this->z = z;
    }
    void Print()
    {
        printf("X: %.1f\tY: %.1f\tZ: %.1f\n", x, y, z);
    }
    void SetValues()
    {
        cout << "Введите X: ";
        cin >> x;
        cout << "Введите Y: ";
        cin >> y;
    }
}

```

```

        cout << "Введите Z: ";
        cin >> z;
    }
private:
    float x, y, z;
};

int main()
{
    setlocale(LC_ALL, "ru");
    int n;

    cout << "Сколько точек хотите записать в файл?\n";
    do {
        cout << "Введите: ";
        cin >> n;
        if (n < 1)
            cout << "Ошибка ввода. Кол-во должно быть больше 0\n";
    } while (n < 1);

    ofstream fout;
    string path = "myFile.txt";
    Point point;
    char choice;

    cout << "Очистить старые данные из файла?(y/n)\n"
        << "Введите: ";
    cin >> choice;
    cout << endl;

    if (choice == 'y' || choice == 'Y')
        fout.open(path); //открываем наш файл с перезаписью
    else if (choice == 'n' || choice == 'N')
        fout.open(path, ofstream::app); //открываем наш файл с записью поверх
    else
    {
        cout << "Научитесь думать и пользоваться клавиатурой, пожалуйста...\n";
        return -1;
    }

    if (!fout.is_open())
    {
        cout << "Ошибка открытия файла!\n";
        return -1;
    }

    //ТАК МЫ ЗАПИСЫВАЕМ ОБЪЕКТЫ
    for (int i = 0; i < n; i++)
    {
        printf("Точка #%d\n", i + 1);
        point.SetValues();
        fout.write((char*)&point, sizeof(Point));
    }
    fout.close();

    cout << "\nВот, что вы записали:\n\n";

    ifstream fin;
    fin.open(path);

    if (!fin.is_open())
    {
        cout << "Ошибка открытия файла!\n";
        return -1;
    }
}

```

```

//ТАК ОБЪЕКТЫ ВЫВОДИМ
int i = 0;
while (fin.read((char*)&point, sizeof(Point)))
{
    printf("Точка #%d\n", i + 1);
    i++;
    point.Print();
    cout << endl;
}
fin.close();

cout << "\nЗавершение работы..\n";

return 0;
}

```

Ещё вариант записи:

```

#include <iostream>
#include <fstream>
#include <cstdlib>

#include <map>
#include <sstream>
#include "string"

using namespace std;

int main() {

    ofstream outClientFile("clients.dat", ios::out);

    if (!outClientFile) {
        cerr << "File could not be opened" << endl;
        exit(1);
    }

    cout << "Enter the account, name and balance." << endl
        << "Enter end-of-file to end input.\n? ";

    int account;
    char name[30];
    double balance;

    //прочитать из cin счёт, имя и баланс, а затем записать в файл
    while (cin >> account >> name >> balance) {
        outClientFile << account << ' ' << name << ' ' << balance << endl;
        cout << "? ";
    }

    return 0;
}

```

Режим	Описание
ios::app	Дописать весь вывод в конец файла. (дописываем)
ios::ate	Открыть файл для вывода и переместиться в конец файла (обычно применяется для дописывания данных в конец файла). Данные могут быть записаны в любое место файла.
ios::in	Открыть файл для ввода.
ios::out	Открыть файл для вывода. (чистим перед записью)
ios::trunc	Отбросить содержимое файла, если он существует (это также по умолчанию делается для ios::out).
ios::binary	Открыть файл для двоичного (то есть нетекстового) ввода или вывода.

.seekg (*число*)	переводит на указанный байт ifstream
.seekp(*число *)	переводит на указанный байт ofstream

Как **ifstream**, так и **ofstream** предусматривают элемент-функции для переустановки *указателя позиции файла* (номера байта в файле, который будет читаться или записываться следующим). Этими функциями являются **seekg** («seek get») для **ifstream** и **seekp** («seek put») для **ofstream**. Каждый объект **ifstream** имеет «get-указатель», указывающий номер байта в файле, с которого будет происходить следующий ввод, а каждый объект **ofstream** имеет «put-указатель», указывающий номер байта в файле, начиная с которого будет помещаться следующий вывод. Оператор

```
inClientFile.seekg( 0 );
```

переустанавливает указатель позиции на начало (позицию 0) файла, прикрепленного к **inClientFile**. Аргументом **seekg** обычно является длинное целое. Может указываться второй аргумент, задающий *направление поиска*. Направление поиска может иметь значения **ios::begin** (по умолчанию) для позиционирования относительно начала потока, **ios::cur** для позиционирования относительно текущей позиции или **ios::end** для позиционирования относительно конца потока. Указатель позиции файла является целым значением, задающим положение в файле как число байтов от начала файла (его называют также *смещением* относительно начала файла). Вот несколько примеров позиционирования «get-указателя» файла:

```
// позиционирует на n-й байт fileObject (подразумевается ios::begin)
fileObject.seekg( n );

// позиционирует fileObject на n байтов вперед
fileObject.seekg( n, ios::cur );

// позиционирует на n байтов назад от конца fileObject
fileObject.seekg( n, ios::end );

// позиционирует на конец fileObject
fileObject.seekg( 0, ios::end );
```

Те же самые операции можно производить с помощью функции **seekp** класса **ofstream**. Для определения текущего положения «get»- и «put»-указателей предусмотрены соответственно элемент-функции **tellg** и **tellp**. Следующий оператор присваивает значение «get»-указателя позиции файла переменной **location** типа **long**:

```
location = fileObject.tellg();
```

№69 Чтение и запись fstream

```
//ЗАПИСЬ И ЧТЕНИЕ ОДНИМ КЛАССОМ
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");

    int input;
    cout << "Что вы хотите сделать?\n"
        << "1. Записать новые данные в файл\n"
        << "2. Просмотреть имеющиеся\n"
        << "Ведите: ";
    cin >> input;
    fstream fs;
    string path = "newFile.txt";
    char choice;
    cout << "Очистить старые данные из файла?(y/n)\n"
        << "Ведите: ";
    cin >> choice;
    cout << endl;

    if (choice == 'y' || choice == 'Y')
        fs.open(path, fstream::in | fstream::out | fstream::trunc);
    else if (choice == 'n' || choice == 'N')
        fs.open(path, fstream::in | fstream::out | fstream::app);
    else
    {
        cout << "Научитесь думать и пользоваться клавиатурой, пожалуйста...\n";
        return -1;
    }

    if (!fs.is_open())
    {
        cout << "Ошибка открытия файла!\n";
        return -1;
    }

    switch (input)
    {
    case 1:
        cout << "Теперь введите данные: ";
        getline(cin, path);
        getline(cin, path);
        fs << path << endl;
        break;
    case 2:
        char ch;
        cout << "\nВаши данные:\n\n";
        while (fs.get(ch))
        {
            cout << ch;
        }
        break;
    default:
        cout << "\nНекорректный ввод.\n"
            << "Завершение работы..";
        return -1;
    }
    fs.close();
    cout << "\nГотово!\n"
        << "Завершение работы..\n";
}
```

Константа	Описание
ios_base::in	открыть файл для чтения
ios_base::out	открыть файл для записи
ios_base::ate	при открытии переместить указатель в конец файла
ios_base::app	открыть файл для записи в конец файла
ios_base::trunc	удалить содержимое файла, если он существует
ios_base::binary	открытие файла в двоичном режиме

```

        return 0;
}

№70 Перегрузка операторов ввода и вывода >> и <<

#include <iostream>
#include <fstream>
using namespace std;

class Point
{
public:
    Point()
    {
        x = y = z = 0;
    }
    Point(float x, float y, float z)
    {
        this->x = x;
        this->y = y;
        this->z = z;
    }

private:
    float x, y, z;
    friend ostream& operator << (ostream& os, const Point& point);
    friend istream& operator >>(istream& is, Point& point);
};

ostream& operator << (ostream& os, const Point& point)
{
    os << point.x << " " << point.y << " " << point.z;
    return os;
}

istream& operator >>(istream& is, Point& point)
{
    is >> point.x >> point.y >> point.z;
    return is;
}

int main()
{
    setlocale(LC_ALL, "ru");
    Point p(77, 9, 32);
    Point pnt;
    fstream fs;
    string path = "new(2)File.txt";
    fs.open(path, fstream::in | fstream::out | fstream::app);

    if (!fs.is_open())
    {
        cout << "Ошибка открытия файла! \n";
        return -1;
    }

    //fs << p << "\n"; //ЦЕ ЗАПИСЬ В ФАЙЛ

    while (true)//ЦЕ СЧИТЫВАЕМ
    {
        fs >> pnt;
        if (fs.eof())
            break;
        cout << pnt << endl;
    }
}

```

```

    }

    fs.close();

    cout << "\nЗавершение программы..\n";
    return 0;
}

```

№71 Обработка исключений try catch

```

218     ifstream fin;
219     fin.exceptions(ifstream::badbit | ifstream::failbit);
220
221     try
222     {
223         cout << "Попытка открыть файл!" << endl;
224
225         fin.open(path);
226
227         cout << "Файл успешно открыт!" << endl;
228     }
229     catch (const std::exception & ex) стандартно для всех
230     {
231         cout << ex.what() << endl;
232         cout << "Ошибка открытия файла!" << endl;
233     }

```

Для работы с файлами

```

218     ifstream fin;
219     fin.exceptions(ifstream::badbit | ifstream::failbit); чтобы выводил инфу об ошибке
220
221     try ← обязательно к выполнению
222     {
223         cout << "Попытка открыть файл!" << endl;
224
225         fin.open(path);
226         ←
227         cout << "Файл успешно открыт!" << endl;
228     }
229     catch (const ifstream::failure & ex) ← если что-то случится
230     {
231         cout << ex.what() << endl;
232         cout << ex.code() << endl;
233         cout << "Ошибка открытия файла!" << endl;
234     }

```

№72 Генерация исключений

```

//ГЕНЕРАЦИЯ ИСКЛЮЧЕНИЙ
#include <iostream>
#include <ctime>
using namespace std;

void Func(int value)
{
    if (value < 0)
    {
        throw exception("Число меньше 0 !!!");
    }
    cout << "Переменная = " << value << endl;
}

int main()
{
    setlocale(LC_ALL, "ru");

    try
    {
        Func(-20);
    }
    catch (const exception &ex)

```

```
{  
    cout << ex.what() << endl;  
}  
return 0;  
}
```

№73 Свой exception

```
12 class MyException :public exception
13 {
14     public:
15     MyException(char *msg,int dataState):exception(msg)
16     {
17         this->dataState = dataState;
18     }
19     int GetDataState()
20     {
21         return dataState;
22     }
23     private:
24     int dataState;
25 };
```

```
28 void Foo(int value)
29 {
30
31     if (value < 0)
32     {
33         throw exception("Число меньше 0 !!!");
34     }
35
36     if (value == 1)
37     {
38         throw MyException("Число = 1 !!!",value);
39     }
40
41     cout << "Переменная = " << value << endl;
42 }
```

```
51     try
52     {
53         Foo(1);
54     }
55     catch (MyException &ex) ←
56     {
57         cout << "Блок 1 Мы поймали " << ex.what() << endl;
58         cout << "Состояние данных " << ex.GetDataState() << endl;
59     }
60     catch (exception &ex) ← равнозначные, т.к. MyExs. наследник exs.
61     {
62         cout << "Блок 1 Мы поймали " << ex.what() << endl;
63     }
```

```
#include <iostream>
using namespace std;

class MyException :public exception
{
public:
    MyException(const char* msg, int dataState) :exception(msg)
    {
        this->dataState = dataState;
    }

    int GetData()
    {
        return dataState;
    }

private:
    int dataState;
};

void Func(int value)
{
    if (value < 0)
```

```

    {
        throw exception("Число меньше 0 !!!");
    }

    if (value == 1)
    {
        throw MyException("Число = 1 !!!", value);
    }

    cout << "Переменная = " << value << endl;
}

int main()
{
    setlocale(LC_ALL, "ru");

    try
    {
        Func(1);
    }
    catch (MyException& ex)
    {
        cout << "Блок 1 (MyExc.) " << ex.what() << endl;
        cout << "Состояние данных: " << ex.GetData() << endl;
    }
    catch (exception& ex)
    {
        cout << "Блок 1 (exc.) " << ex.what() << endl;
    }

    return 0;
}

```

№74 Enum

```

#include <iostream>
using namespace std;

class PC
{
public:
    enum PCState
    {
        OFF,
        ON,
        SLEEP
    };

    PCState GetState()
    {
        return State;
    }

    void SetState(PCState State)
    {
        this->State = State;
    }

private:
    PCState State;
};

```

```

int main()
{
    setlocale(LC_ALL, "ru");

    PC pc;

    pc.SetState(PC::PCState::OFF);

    switch (pc.GetState())
    {
    case PC::PCState::OFF:
        cout << "Выключен!\n";
        break;
    case PC::PCState::ON:
        cout << "Включен!\n";
        break;
    case PC::PCState::SLEEP:
        cout << "Спит!\n";
        break;
    }

    return 0;
}

```

№75 Пространство имён

```

12  namespace firsNS
13  {
14      void Foo()
15      {
16          cout << "Foo firstNS" << endl;
17      }
18  }
19
20  namespace secondNS
21  {
22      void Foo()
23      {
24          cout << "Foo secondNS" << endl;
25      }
26  }

30  int main()
31  {
32      setlocale(LC_ALL, "ru");
33
34      firsNS::Foo();
35
36      return 0;
37  }

```

№76 Шаблонные классы и их специализации

Шаблонные классы работают с любыми типами данных (как и функции)

Специализации нужны, чтобы отдельный тип данных как-то по-особенному расписать

```

//ШАБЛОНЫЕ КЛАССЫ, СПЕЦИАЛИЗАЦИЯ КЛАССА
#include <iostream>

using namespace std;

```

```

template<typename T> //шаблонный класс
class Printer
{
public:

    void Print(T value)
    {
        cout << value << endl;
    }
};

template<> //специализированный класс шаблона
class Printer<string>
{
public:
    void Print(string value)
    {
        cout << "___" << value << "___" << endl;
    }
};

int main()
{
    setlocale(LC_ALL, "ru");

    Printer<int> p;
    p.Print(5);

    Printer<string> pr;
    pr.Print("Hello, world!");

    return 0;
}

```

№77 Структуры vs классы

	class	struct
Поля по умолчанию	private	public
При наследовании	Private	public

Структуры были ещё в языке С...но не было модификаторов доступа(те по сути инкапсуляции), не было наследования и полиморфизма, а также не было функций (методов)...потом появился C++, ООП и новый тип данных класс, в котором появилось ООП, инкапсуляция, полиморфизм, наследование и методы. Но поскольку язык C++ включает в себя язык С в нем оставили структуры, наделив их свойствами классов, с той лишь разницей, что в структуре все поля по умолчанию public

№78 Умные указатели

Умные указатели нужны для автоматизации процесса чистки памяти – прописав в деструкторе `delete`, никогда не забудешь это сделать

```

//простейший SmartPointer
#include <iostream>
using namespace std;

template <class T>
class SmartPointer
{
public:

    SmartPointer(T* ptr) //присваивает нашему указателю указатель из new int
    {
        this->ptr = ptr;
    }
}
```

```

        cout << "Constructor" << endl;
    }

~SmartPointer() //чишит память при удалении
{
    delete ptr;
    cout << "Destructor" << endl;
}

T& operator * () //перегрузка операции разыменования
{
    return *ptr;
}

private:
    T* ptr; //это наш указатель
};

int main()
{
    setlocale(LC_ALL, "ru");

    SmartPointer<int> pointer = new int(5); //new int возвращает указатель, так что мы
присваиваем его

    //альтернативное присвоение
    //SmartPointer<int> pointer(new int(5));

    cout << *pointer << endl;

    return 0;
}

```

3 вида умных указателей:

- auto_ptr

при присваивании в другой указатель, затирается

```

103     int main()
104     {
105
106         setlocale(LC_ALL, "ru");
107
108         auto_ptr<int> ap1(new int(5));
109
110         auto_ptr<int> ap2(ap1);

```

The screenshot shows a debugger interface with two memory locations. The first location, labeled 'ap1|empty', contains a small icon of a computer monitor with a red 'X' over it, indicating it's empty. The second location, labeled 'ap2|auto_ptr 5', contains a similar icon followed by the value '5', indicating it's pointing to the memory address of an integer variable with value 5.

- unique_ptr

не позволяет просто так присваивать другой указатель

```

108     unique_ptr<int> p1(new int(5));
109
110     unique_ptr<int> p2(p1);

```

но с помощью move это всё-таки можно сделать

```

112     p2 = move(p1);

```

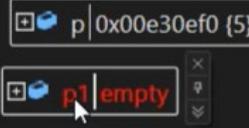
или с помощью swap

```

112     p2.swap(p1);

```

метод `reset` чистит полностью умный указатель и все данные в нём а `release` только сам умный указатель



```
103     int main()
104     {
105
106         setlocale(LC_ALL, "ru");
107
108         int *p = new int(5);
109
110         unique_ptr<int> p1(p);
111         p1.release();
```

- `share_ptr`

решена проблема чистки памяти

```
//share вручную
#include <iostream>
using namespace std;

template <class T>
class SmartPointer
{
public:

    SmartPointer(T* ptr) //присваивает нашему указателю указатель из new int
    {
        count++;
        this->ptr = ptr;

        cout << "Constructor" << endl;
    }

    ~SmartPointer() //чистит память при удалении
    {
        count--;
        if (count == 0)
            delete ptr;

        cout << "Destructor" << endl;
    }

    SmartPointer(const SmartPointer& other) //прописываем конструктор копирования
    {
        count++;
        this->ptr = other.ptr;

        cout << "Copying" << endl;
    }

    T& operator * () //перегрузка операции разыменования
    {
        return *ptr;
    }

private:
    T* ptr; //это наш указатель
    static int count;
```

```
};
```

```
template <class T>
int SmartPointer<T>::count = 0;
```

```
int main()
{
    setlocale(LC_ALL, "ru");

    SmartPointer<int> p1 = new int(5); //new int возвращает указатель, так что мы присваиваем его
    SmartPointer<int> p2 = p1;

    //альтернативное присвоение
    //SmartPointer<int> pointer(new int(5));

    cout << *p2 << endl;

    printf("Объект 1: %p\n", p1);
    printf("Объект 2: %p\n", p2);

    return 0;
}
```

№79 Умные указатели и динамические массивы

Умные указатели можно использовать как указатели на динамические массивы – только в <> нужно обязательно указать []

ЭКВИВАЛЕНТНЫЕ ЗАПИСИ:

```
101  int main()
102  {
103
104      setlocale(LC_ALL, "ru");
105
106      int SIZE = 5;
107      int *arr = new int[SIZE] {1, 6, 44, 9, 8};
108
109      shared_ptr<int[]> ptr(arr);
```

(для понимания)

```
101  int main()
102  {
103
104      setlocale(LC_ALL, "ru");
105
106      int SIZE = 5;
107
108      shared_ptr<int[]> ptr(new int[SIZE] {1, 6, 44, 9, 8});
```

(так надо)

Code:

```
//умный указатель + массив
#include <iostream>
using namespace std;

int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "ru");

    int size;
    cout << "Введите размер массива: ";
    cin >> size;

    shared_ptr<int[]> ptr(new int[size]);

    for (int i = 0; i < size; i++)
    {
        ptr[i] = rand() % 10;
        cout << ptr[i] << " ";
    }

    return 0;
}
```

№80 Вектор

Общее описание – динамический массив «на стероидах»: его не надо чистить и у него куча функций

```
14 int main()
15 {
16     setlocale(LC_ALL, "ru");
17     vector<int> myVector = {0,484,484,995};
18     myVector.reserve(10000);
19     myVector.push_back(2);
20     myVector.push_back(44);
21     myVector.push_back(77);
22     myVector.push_back(9);
23
24     cout << "количество элементов в векторе: " << myVector.size();
25     cout << "capacity вектора: " << myVector.capacity();
26
27     myVector.shrink_to_fit();
28     cout << "shrink_to_fit() " << endl;
29     cout << "количество элементов в векторе: " << myVector.size();
30     cout << "capacity вектора: " << myVector.capacity();
31 }
```

Сколько должна быть capacity

показать резерв

уменьшить capacity до size

команды как в листе, вернуть кол-во эл-том вектора

Способ инициализации:

The screenshot shows two code snippets in a C++ IDE and their corresponding outputs in the terminal window.

Top Snippet:

```
12 int main()
13 {
14     setlocale(LC_ALL, "ru");
15
16     vector<int> myVector(20,55)
17
18     cout << "количество элементов в векторе 20"
19     cout << "capacity вектора 20"
20
21     for (int i = 0; i < myVector.size(); i++)
22     {
23         cout << myVector[i] << endl;
24     }
25
26     cout << "количество элементов в векторе 20"
27     cout << "capacity вектора 20"
28
29     return 0;
30 }
```

Output (Top Terminal):

```
количество элементов в векторе 20
capacity вектора 20
55
55
55
55
55
55
55
55
55
55
55
55
55
55
55
55
55
55
55
55
55
Для продолжения нажмите любую клавишу . . .
```

Bottom Snippet:

```
14 int main()
15 {
16     setlocale(LC_ALL, "ru");
17
18     vector<int> myVector;
19
20     cout << "количество элементов в векторе 0"
21     cout << "capacity вектора 0"
22
23     myVector.resize(20,448); ← как заново создать
24
25     cout << "resize количество элементов в векторе 20"
26     cout << "capacity вектора " << myVector.capacity();
27
28     for (int i = 0; i < myVector.size(); i++)
29     {
30         cout << myVector[i] << endl;
31     }
32 }
```

Output (Bottom Terminal):

```
количество элементов в векторе 0
capacity вектора 0
resize количество элементов в векторе 20
capacity вектора 20
448
448
448
448
448
448
448
448
448
448
448
448
448
448
448
448
448
448
448
448
448
ещё есть myVector.empty() - возвращает true или false
```

A red arrow points from the line `myVector.resize(20,448);` to the explanatory text "как заново создать". Another red arrow points from the word "ещё" in the output to the explanatory text "ещё есть myVector.empty() - возвращает true или false".

Code:

```
//вектор
#include <vector>
#include <iostream>
using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");

    vector<int> MyVector(20, 5);

    for (int i = 0; i < MyVector.size(); i++)
    {
        cout << MyVector[i] << " ";
    }
    cout << "\n\nCapacity before: " << MyVector.capacity()<<endl;
```

```

MyVector.reserve(100);

cout << "\nCapacity after: " << MyVector.capacity() << endl;

MyVector.resize(50, 3);

cout << "\nCapacity after (2): " << MyVector.capacity() << endl;
cout << "\n Size after(2): " << MyVector.size() << endl;
cout << "\n Is this empty? - " << MyVector.empty()<<endl;
cout << "\nShrink yo fit...\n";
MyVector.shrink_to_fit();
cout << "\nCapacity after (3): " << MyVector.capacity() << endl;
cout << "\n Size after(3): " << MyVector.size() << endl<<endl;

for (int i = 0; i < MyVector.size(); i++)
{
    cout << MyVector[i] << " ";
}

return 0;
}

```

№81 Итераторы

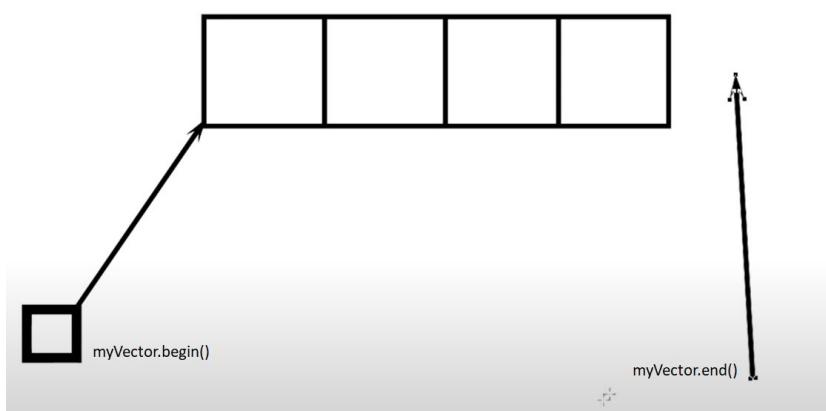
Итератор – штукга, имеющее устройство обычного указателя, только лишь с тем смыслом, что она автоматически подстраивается под шаблон STL, позволяя по нему итерироваться, вне зависимости от того, как устроена связь между элементами в самом шаблоне.

Позволяет пользоваться арифметикой указателей в шаблонах.

```

14 int main()
15 {
16     setlocale(LC_ALL, "ru");
17
18     vector<int> myVector = { 1,9,44,422,676,78 };
19
20     vector<int>::iterator it;
21
22     it = myVector.begin();
23
24     *it = 1000; // поменяется
25
26     cout << *it << endl;
27
28     return 0;
29 }
30

```



```
19     vector<int> myVector = { 1,9,44,422,676,78 };
20
21     vector<int>::iterator it;
22
23     it = myVector.begin();
24
25     it+=2;
26
27     it-=2;      I
28
29     cout << *it << endl;
```

```
19     vector<int> myVector = { 1,9,44,422,676,78 };
20
21
22
23
24
25     for (vector<int>::iterator i = myVector.begin(); i != myVector.end(); i++)
26     {
27         cout << *i << endl;
28     }          I
```

Константный итератор нужен для того, чтобы только итерироваться, но не редактировать по адресу

```
25     for (vector<int>::const_iterator i = myVector.cbegin(); i != myVector.cend(); i++)
26     {
27         cout << *i << endl;
28     }          I
```

iterator наследуется от **const_iterator**, поэтому **const** в простой положить нельзя

const_iterator

iterator

Обратный ход (как вариант). Начинает с конца, при **reverse_iterator**++ смещается к началу

```
23     for (vector<int>::reverse_iterator i = myVector.rbegin(); i != myVector.rend(); i++)
24     {
25         cout << *i << endl;
26     }          I
```

Итерирование с помощью **advance**

14 int main()
15 {
16 setlocale(LC_ALL, "ru");
17
18 vector<int> myVector = { 1,9,44,422,676,78 };
19
20 vector<int>::iterator it = myVector.begin();
21
22 //cout << *(it + 3) << endl;
23
24 advance(it, 3); ← ЭКВИВАЛЕНТО
25 cout << *it << endl;
26
27 }

The screenshot shows a code editor with a dark theme. A red arrow points from the text 'ЭКВИВАЛЕНТО' to the line 'advance(it, 3);'. In the top right corner of the IDE window, there is a status bar with the text '422 для'.

insert и erase

```
29     vector<int>::iterator it = myVector.begin();  
30  
31     advance(it, 5);  
32  
33     myVector.insert(it, 999); ← ВСТАВИТЬ ЭЛ-Т  
34  
35     for (vector<int>::iterator i = myVector.begin(); i != myVector.end(); i++)  
36     {  
37         cout << *i << endl;  
38     }  
39  
40     cout << endl << "erase " << endl << endl;  
41  
42     vector<int>::iterator itErase = myVector.begin();  
43  
44  
45  
46     myVector.erase(itErase, itErase+3); ← ФУНКЦИЯ УДАЛЕНИЯ (МОЖНО ДИПАЗОН)  
47  
48     for (vector<int>::iterator i = myVector.begin(); i != myVector.end(); i++)  
49     {  
50         cout << *i << endl;  
51     }
```

The screenshot shows a code editor with a dark theme. A red arrow points from the text 'ВСТАВИТЬ ЭЛ-Т' to the line 'myVector.insert(it, 999);'. Another red arrow points from the text 'ФУНКЦИЯ УДАЛЕНИЯ (МОЖНО ДИПАЗОН)' to the line 'myVector.erase(itErase, itErase+3);'.

code

```
#include <iostream>  
#include <vector>  
using namespace std;  
  
void printintVector(vector<int>& myVector);  
  
int main()  
{  
    setlocale(LC_ALL, "ru");  
  
    srand(time(NULL));  
  
    cout << "Hello!";  
  
    vector<int> myVector;  
  
    vector<int>::iterator it = myVector.begin();  
  
    for (int i=0; i < 10; i++)  
    {  
        myVector.insert(it, rand()%10);  
    }
```

```

        it = myVector.begin();
    }

printintVector(myVector);

int j = 1;
for (; it != myVector.end(); it++)
{
    *it = j;
    j++;
}

printintVector(myVector);

myVector.erase(it = myVector.begin() + (rand() % (myVector.size() - 1)));

printintVector(myVector);

cout << endl;

return 0;
}

void printintVector(vector<int>& myVector)
{
    cout << endl << endl;
    for (vector<int>::iterator it = myVector.begin(); it != myVector.end(); it++)
    {
        cout << *it << " ";
    }
    cout << "\nРазмер вектора: " << myVector.size();
}

```

№82 Ключевое слово auto

Автоматическое определение типа данных. Не использовать для примитивных типов данных! Можно использовать для длинных типов типа итераторов.

```

std::vector<int> myVector = { 11,46,9 };

std::vector<int>::iterator it = myVector.begin();
||

auto it2 = myVector.begin();

for (auto it = myVector.begin(); it != myVector.end(); it++)
{
    std::cout << *it << std::endl;
}

```

СОРТИРОВКИ

<https://academy.yandex.ru/posts/osnovnye-vidy-sortirovok-i-primerы-ikh-realizatsii>

0	1	2	3	4
3	2	4	1	5



1. Пузырёк

- i нужен для того, чтобы отрезать конец массива – после первого прохода цикла по j самый большой элемент окажется в конце массива (его уже проверять не надо). С помощью цикла j мы сравниваем с первым элементом соседние поочерёдно.

- Code

```
void bubbleSort(int* arr, int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - 1 - i; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int t = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = t;
            }
        }
    }
}
```

2. Вставками

Эффективна для массивов до 10 элементов

- Аналогия с картами – берём новую карту и ищем ей место. Конкретно: с помощью **for** мы выбираем «карту» (по очереди каждый элемент массива), а с помощью **while** мы ищем ей место (перезаписываем в правый элемент левый, а когда условие, что левый элемент меньше, чем наша новая «карта», записываем в этот левый элемент нашу новую «карту» - левый элемент уже был продублирован в правый в прошлой итерации, так что мы ничего не потеряли)

- Code:

```
void insertionSort(int* arr, int size)
{
    int x;
    for (int i = 1; i < size; i++)
    {
        x = arr[i];
        int j = i - 1;
        while (x < arr[j])
```



```

{
    arr[j + 1] = arr[j];
    j--;
}
arr[j + 1] = x;
}

```

- Пример, как происходит сортировка массива 95361:

» i=1 \Rightarrow j=0

x=5

5<9 \Rightarrow заходим в while, j=0: 99361

j=-1: сравнивать не с чем – 59361

» i=2 \Rightarrow j=1

x=3

3<9 \Rightarrow заходим в while, j=1: 59961, j-- \Rightarrow j=0

3<4 \Rightarrow заходим в while, j=0 55961, j-- \Rightarrow j=-1 сравнивать не с чем – 35961

аналогично далее:

» i=3 \Rightarrow j=2

x=6

35991

35691 (тут всё закончилось на j=1 \Rightarrow x присвоили в j+1 т.е. в arr[2])

» i=4 \Rightarrow j=3

x=1

35699

35669

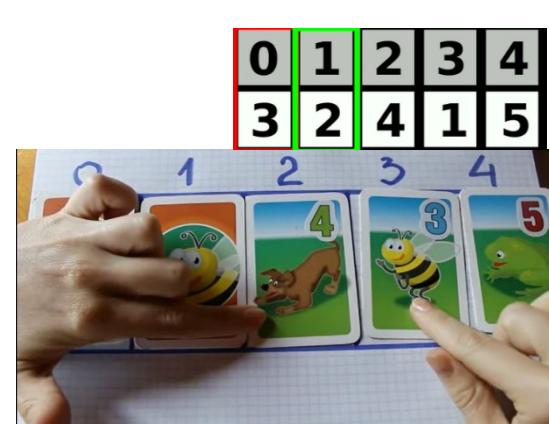
35569

33569

13569 (тут всё закончилось на j=-1 \Rightarrow x присвоили в j+1 т.е. в arr[0])

3. Выбором

- Суть в том, что мы ставим минимальный элемент из всего (оставшегося) массива в начало (оно смещается по ходу счётчика i). Первым делом в k мы храним номер минимального элемента (изначально полагая его первым [зависит от i] т.е. если мы минимального в if не найдём, то менять ничего и не придётся, всё будет готово), а в x мы храним значение минимального элемента. с помощью цикла for по j и if мы находим из всего массива минимальный элемент и после цикла начальный элемент сохраняем в ячейку минимального, а значение минимального сохраняем в начальный (просто меняем взаимно их значения)



- Code

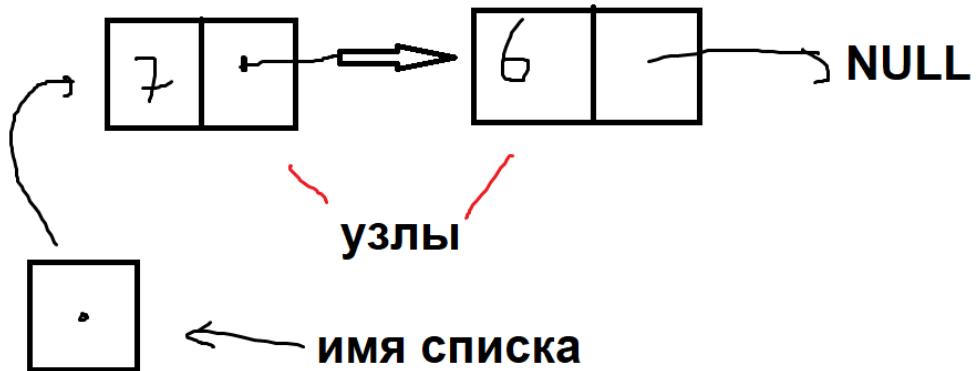
```
void selectionSort(int* arr, int size)//сортировка выбором
{
    int k, x;
    for (int i = 0; i < size; i++)
    {
        k = i; //предполагаем первый эл-т минимальным
        x = arr[i];
        for (int j = i + 1; j < size; j++)
        {
            if (arr[j] < x)
            {
                k = j;
                x = arr[j];
            }
        }
        arr[k] = arr[i]; //в тот, который min присваиваем начальный
        arr[i] = x; //в начальный присваиваем значение минимального
    }
}
```

ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

0. Два вида односвязных списков:

[**РАЗЛИЧИЕ** В МЕТОДАХ ДОБАВЛЕНИЯ/УДАЛЕНИЯ ЭЛ-ТА]

1. СТЕК (LIFO – last input first output)



2. ОЧЕРЕДЬ (голова|хвост)

Голова – касса (адрес первого элемента)

Хвост – очередь

Удаление с головы (для головы создаём temp=head, чтобы потом почистить старую голову)

Добавление в конец tail



Деревья:

Корень - первое число в списке **Дерево поиска**

Левый потомок меньше

Правый потомок больше

Если то же (повтор) то слева

(обходы)

Печать: левый потомок, корень, правый потомок (**порядковый**)

Печать: корень, левый, правый (**прямой**) [правый печатается тогда, когда слева ничего не осталось]

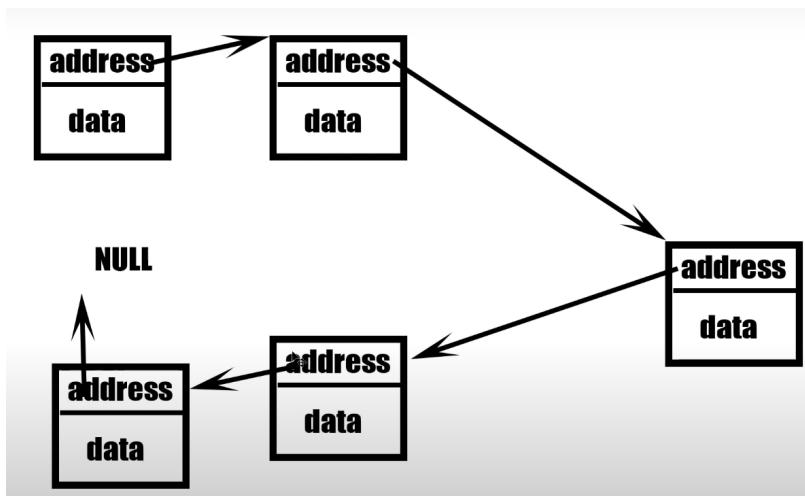
Печать: левый, правый, корень (**обратный**)

корень не печатается, пока есть листья!

1. Односвязный список

Характеристика:

- Каждый элемент списка содержит 2 поля:
 1. Своё значение
 2. Адрес следующего элемента
- Позволяет быстро добавлять/удалять элементы (куда угодно, т.к. список – не сплошной кусок памяти)
- Медленно достаёт данные из элементов, потому что ему приходится проходить по каждому адресу отдельно



Реализация односвязного списка

```
//односвязный список

#include <iostream>
using namespace std;

template<typename T> //шаблонный
class List //односвязный список

{
public:
    List();
    ~List();
    void push_back(T data);
    void push_front(T data);
    void insert(T data, int index);
    void pop_back();
    void pop_front();
    void removeAt(int index);

    void clear();
    int GetSize() { return Size; }
    T& operator [](const int index);
}
```

private:

```
template<typename T>
class Node
{
```

```

public:
    Node* pNext;
    T data;

    Node(T data = T(), Node* pNext = nullptr)
    {
        this->data = data;
        this->pNext = pNext;
    }
};

Node<T>* head;
int Size;
};

template<typename T>
List<T>::List()
{
    Size = 0;
    head = nullptr;
}

template<typename T>
List<T>::~List()
{
    clear();
}

template<typename T>
void List<T>::pop_back()
{
    removeAt(Size - 1);
}

template<typename T>
void List<T>::pop_front()
{
    Node < T >* temp= head;
    head = head->pNext;
    delete temp;
    Size--;
}

template<typename T>
void List<T>::removeAt(int index)
{
    if (index == 0)
    {
        pop_front();
    }
    else
    {
        Node<T>* previous = head;
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext; //нашли тот, который перед индексированным
        }

        Node<T>* toDelete = previous->pNext; //сохранили элемент для удаления
        previous->pNext = toDelete->pNext; //в предыдущий сохранили адрес следующего после
удаляемого
        delete toDelete; //удалили
        Size--;
    }
}

```

```

template<typename T>
void List<T>::push_back(T data) //элемент в конец списка
{
    if (head == nullptr)
    {
        head = new Node<T>(data);
    }
    else
    {
        Node<T>* current = this->head;
        while (current->pNext != nullptr)
        {
            current = current->pNext; //наш эл-т берётся как след.
        }
        current->pNext = new Node<T>(data); //когда нашли объект с адресом nullptr, присвоили туда data значение ( и вызвался конструктор Node, который создаёт следующую ячейку nullptr
    }
    Size++;
}

template<typename T>
void List<T>::push_front(T data)
{
    head = new Node<T>(data, head);
    Size++;
}

template<typename T>
void List<T>::insert(T data, int index)
{
    if (index == 0)
    {
        push_front(data);
    }
    else
    {
        Node<T>* previous = head;
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext; //нашли тот, который перед индексированным
        }

        Node<T>* newNode = new Node<T>(data, previous->pNext); //создаём новый элемент и чтобы прицепить его к следующему исп. pNext для конструктора

        previous->pNext = newNode; //ранее pNext указывал индексированный эл-т, теперь будет указывать новый
        //previous->pNext = new Node<T>(data, previous->pNext);

        Size++;
    }
}

template<typename T>
void List<T>::clear()
{
    while (Size)
    {
        pop_front();
    }
}

```

```

template<typename T>
T& List<T>::operator[](const int index)
{
    int counter = 0;
    Node<T>* current = this->head;
    while (current != nullptr)
    {
        if (counter == index)
        {
            return current->data;
        }
        current = current->pNext;
        counter++;
    }
}

int main()
{
    setlocale(LC_ALL, "ru");

    List<int> lst;

    cout << "Введите кол-во ячеек: ";
    int numbersCount;
    cin >> numbersCount;
    for (int i = 0; i < numbersCount; i++)
    {
        lst.push_back(rand() % 10);
    }
    cout << endl;

    for (int i = 0; i < lst.GetSize(); i++)
    {
        cout << lst[i] << endl;
    }

    cout << "До: " << lst.GetSize() << endl;

    lst.removeAt(2);
    lst.insert(99, 2);
    lst.pop_back();

    cout << "После: " << lst.GetSize() << endl;

    for (int i = 0; i < lst.GetSize(); i++)
    {
        cout << lst[i] << endl;
    }

/*lst.push_back(1);
lst.push_back(2);
lst.push_back(3);
cout << lst.GetSize() << endl;
cout << lst[1] << endl;*/

    return 0;
}

```

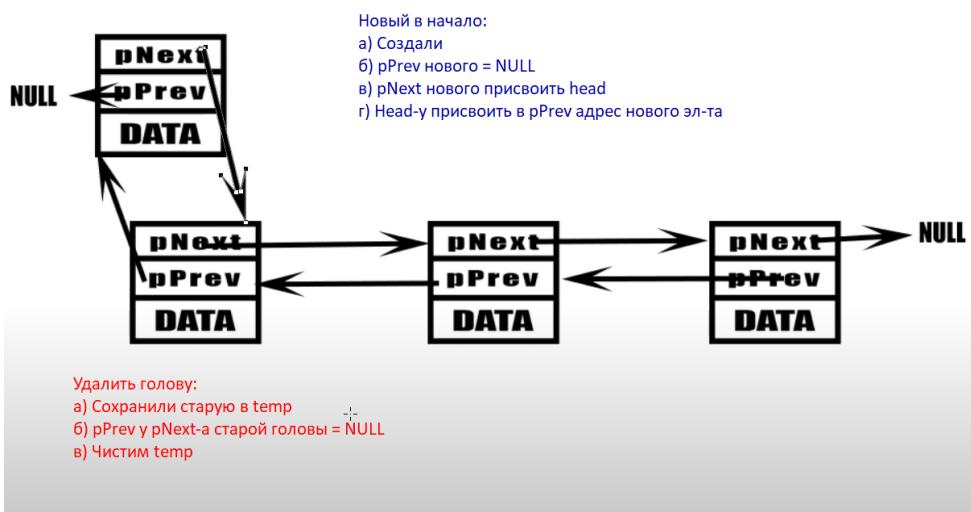
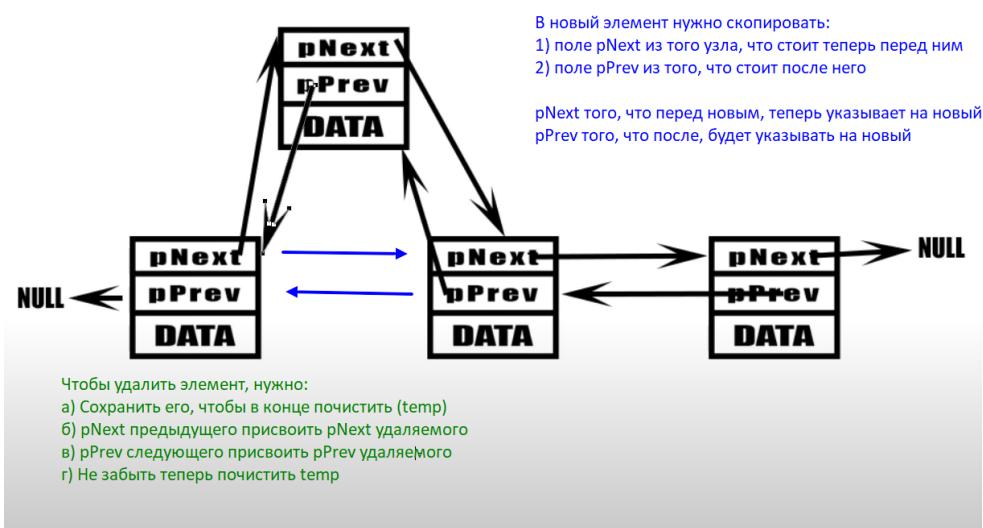
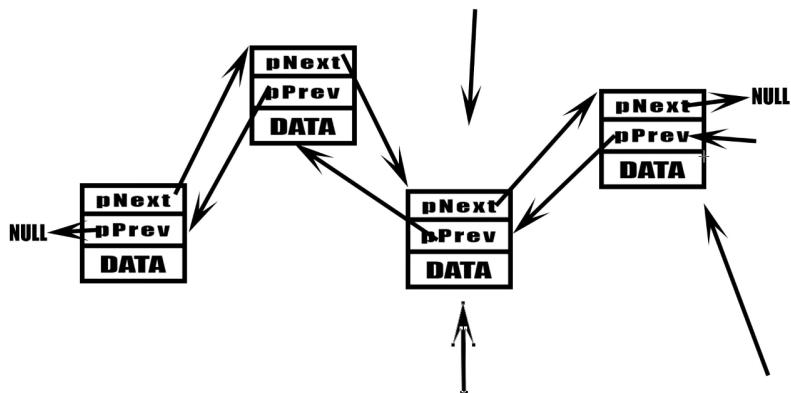
2. Двусвязный список

Отличие от односвязного заключается в том, что вместо двух полей (в односвязном списке) узел двусвязного содержит 3 поля:

- Адрес следующего узла

- Адрес предыдущего узла
- Данные

Имеет Head|Tail т.е. возможность пробегаться по узлам как с начала списка, так и с конца – это ускоряет передвижение, ведь можно проверять (вводимый) индекс с помощью Size (кол-ва эл-тов в списке) на то, ближе узел к концу или началу списка. В общем, чем ближе к центру, тем дольше нам идти.



Аналогично с хвостом:

- а) Создаём новый
- б) pNext нового = NULL
- в) pNext старой головы = адрес нового
- г) pPrev нового = старая голова

Если не нужно работать со всем списком (стек, пример), то односвязный список будет лучше

Значимые отличия односвязного и двусвязного списков в том, что процедура изменения в двусвязном посложнее (больше полей нужно менять), но скорость доступа намного выше.

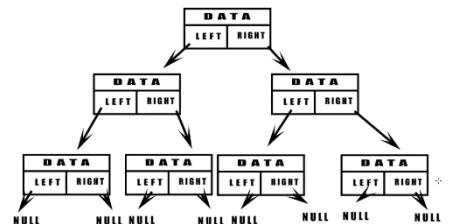
3. Бинарное дерево

РЕКУРСИВНАЯ динамическая структура данных – дерево состоит из деревьев, корень один, но потомки одного корня так же могут быть корнями

Корень – узел, у которого есть потомки

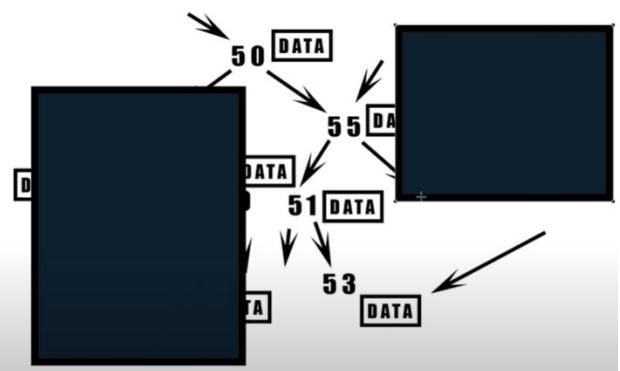
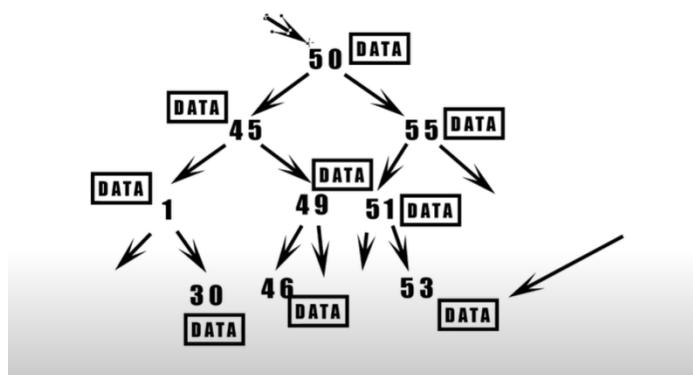
Листья – узлы без потомков

Слева меньшие потомки, справа – большие



Особенность бинарного дерева – очень быстрый доступ к узлам за счёт отсевания целых ветвей проверкой (если этот корень больше, чем искомый элемент, то налево, else (т.е. если корень меньше, то направо)

Ищем 53



(<https://www.youtube.com/watch?v=HBMIhZAOh0I>)

Методы дерева (дерево поиска)

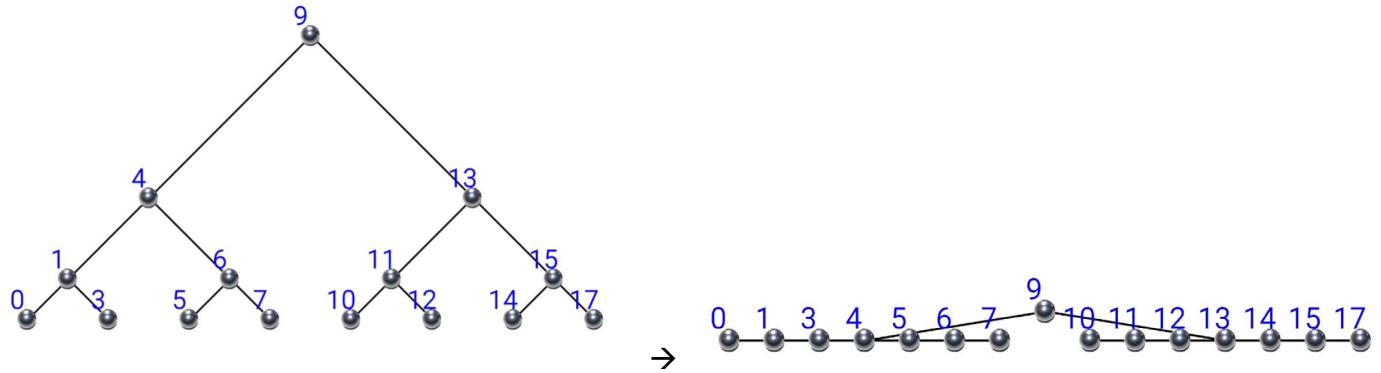
TREE-MINIMUM(x)

```
1 while  $x.left \neq NIL$  // ходим пока следующ. магазин не тупик
2      $x = x.left$       // идем наЛЕВО;
3 return  $x$            // ура! нашли, где дешево!
```

TREE-MAXIMUM(x)

```
1 while  $x.right \neq NIL$  // ходим пока следующ. магазин не тупик
2      $x = x.right$      // идем наПРАВО;
3 return  $x$            // ура! нашли антиквариат
```

ITERATIVE-TREE-SEARCH(x, k) //ходим пока (не тупик и
1 while $x \neq NIL$ и $k \neq x.key$ // не нашли нужную цену)
2 if $k < x.key$ // если нужно дешевле
3 $x = x.left$ // идем наЛЕВО
4 else $x = x.right$ // иначе
5 return x // идем наПРАВО
//пришли (либо нет такого магазина)

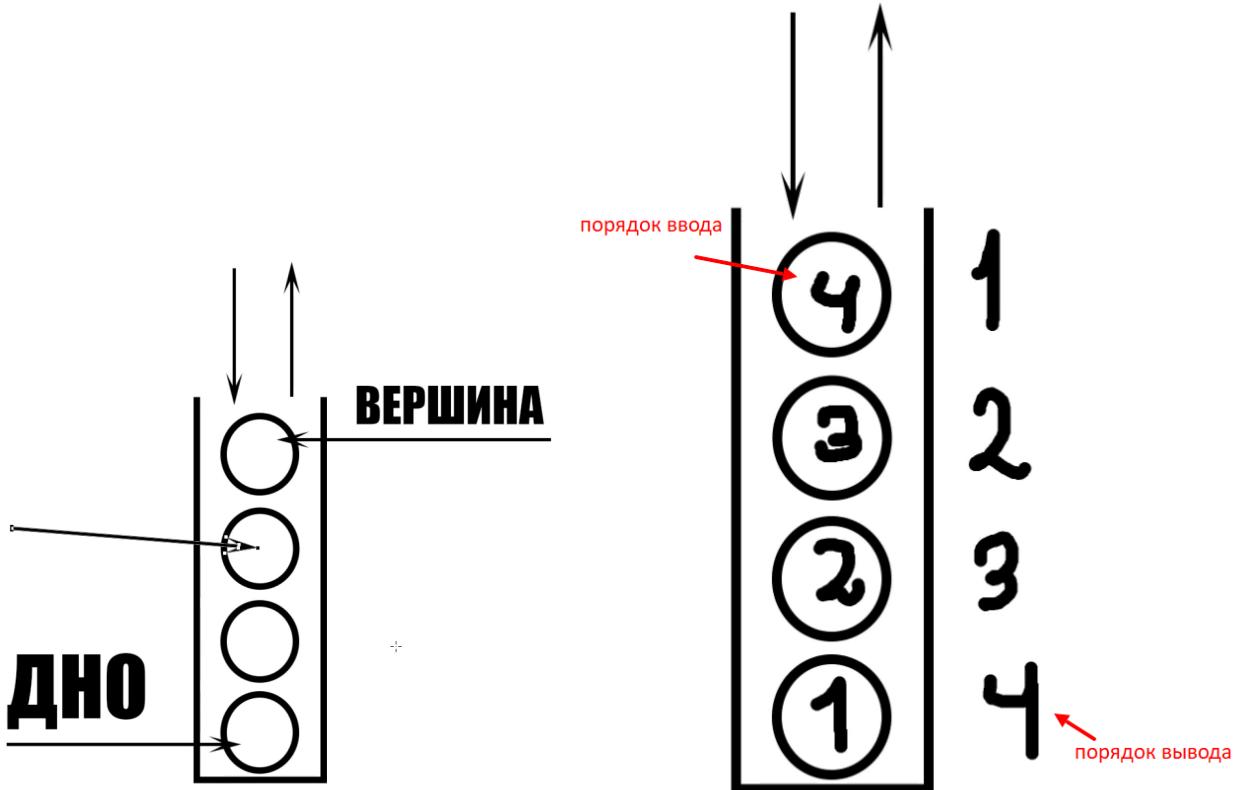


4. Стек

Суть стека в том, что это просто односвязный список с первым элементом `head`, у которого есть методы

- 1) `void push_front(T data);`
- 2) `T pop_front();`

По сути это урезанный односвязный список



```
//стек

#include <iostream>
using namespace std;

template<typename T> //шаблонный
class List //односвязный список

{
public:
    List();
    ~List();
    void push_front(T data);
    T pop_front();
    void clear();
    int GetSize() { return Size; }

private:
    //структура узла
    template<typename T>
    class Node
    {
public:
    Node* pNext;
    T data;

    Node(T data = T(), Node* pNext = nullptr)
    {
        this->data = data;
        this->pNext = pNext;
    }
};

Node<T>* head; //адрес первого узла
int Size; //размер стека
};

//конструктор
```

```

template<typename T>
List<T>::List()
{
    Size = 0;
    head = nullptr;
}

//деструктор
template<typename T>
List<T>::~List()
{
    clear();
}

//очистка стека
template<typename T>
void List<T>::clear()
{
    while (Size)
    {
        pop_front();
    }
}

//удаление вершины (и вытащить data)
template<typename T>
T List<T>::pop_front()
{
    T time = head->data;
    Node < T >* temp = head;
    head = head->pNext;
    delete temp;
    Size--;
    return time;
}

//новая вершина
template<typename T>
void List<T>::push_front(T data)
{
    head = new Node<T>(data, head);
    Size++;
}

int main()
{
    setlocale(LC_ALL, "ru");
    srand(time(NULL));

    List<int> lst;

    cout << "Введите кол-во ячеек: ";
    int numbersCount, random;
    cin >> numbersCount;
    cout << "\nПорядок рандома:\n";
    for (int i = 0; i < numbersCount; i++)
    {
        random = rand() % 10;
        lst.push_front(random);
        cout << random << " ";
    }
    cout << "\nПорядок извлечения из стека:\n";

    numbersCount = lst.GetSize();
}

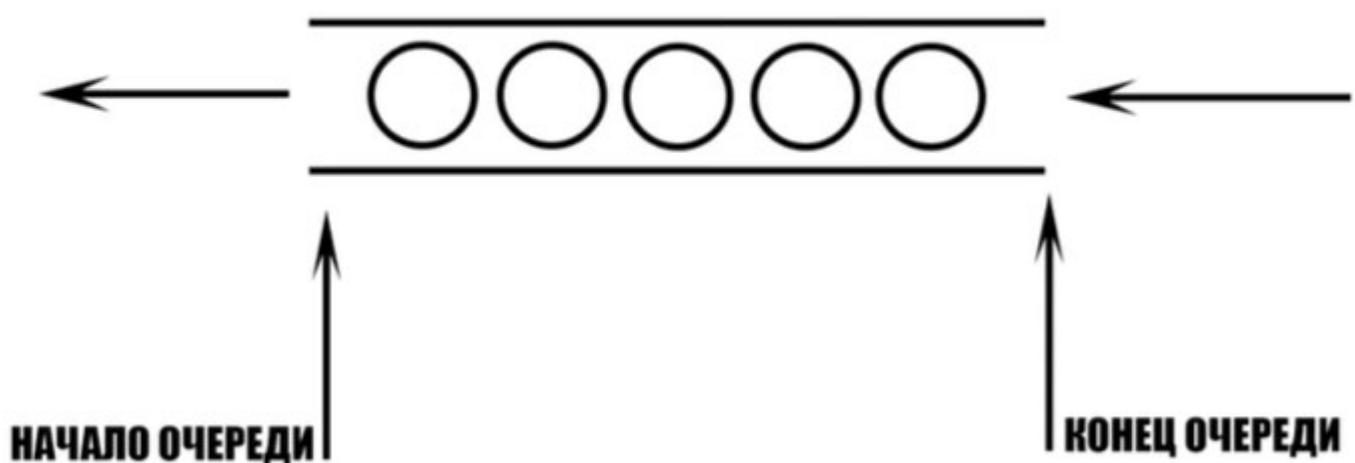
```

```

for (int i = 0; i < numbersCount; i++)
{
    cout << lst.pop_front() << " ";
}
cout << endl;
}

```

5. Очередь



№82 Многофайловый проект

Перенос функции

В хедере (file.h) лежит прототип – в новом new.cpp лежит реализация.

В source.cpp с main() пишем строку #include "file.h"

```

Sum.cpp      Sum.h  ✘  Source.cpp*
❶ #pragma once
❷ int Sum(int a, int b);

```

Перенос класса

Аналогично переносу функции:

В хедере (class.h) лежит класс с прототипами

В class.cpp лежат реализации и пишем строку #include "class.h"

ВАЖНО:

1. Всякие `#include <iostream>` и `using namespace std` должны лежать в хедере, чтобы их можно было использовать при описании класса в `class.cpp`
2. Из хедера можно обратиться к методу: просто тыкнуть курсор и нажать f12
3. Если классы (аналогично функции) называются одинаково, то их можно ограничить пространствами имён (`namespace`). Для этого и хедер, и `cpp` (реализацию) нужно поместить в пространство. Это есть на скринах, без пространств всё так же будет работать.

The screenshot shows two code editor windows side-by-side. The left window contains `MyClass.h` with the following code:1 #pragma once
2 #include<iostream>
3
4 namespace myNamespace {
5
6 class MyClass
7 {
8 public:
9 void PrintMessage(char str[]);
10 };
11}The right window contains `MyClass.cpp` with the following code:1 #include "MyClass.h"
2
3 namespace myNamespace {
4
5 void MyClass::PrintMessage(char str[])
6 {
7 std::cout << str << std::endl;
8 }
9}

№83 Цикл for each

Используется для итерации по контейнеру (перебор всей коллекции).

```
33     int arr[] = { 5,11,94,99,44 };  
34  
35     for each (auto var in arr)  
36     {  
37         cout << var << endl;  
38     }
```

На консоль будет выведен весь массив

То был синтаксический сахар, вот стандарт:

```
40     for (auto element : arr)  
41     {  
42         cout << element << endl;  
43     }
```

откуда берём эл-ты
куда временно их пишем

Для изменения эл-тов

```
40     for (const auto &element : arr)
41     {
42         //element = -1;
43         cout << element << endl;
44     }
```

№84 Порядок удаления объектов (static, глобальных и простых)

Создаются все по порядку вызова

Удаляются вот так:

1. В ФУНКЦИЯХ

- Простые переменные удаляются по выходу из функции
- Static переменные в функциях удаляются после удаления всех простых в main в порядке, обратном созданию

2. В MAIN

- Сначала удаляются простые переменные в обратном порядке
- (Потом все static из функций)
- Потом все static из main

ПОСЛЕ ВСЕГО ЭТОГО УДАЛЯЮТСЯ ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ (нет разницы, static они или нет)

Код, на котором можно всё понять:

```
#include <iostream>
using namespace std;

class Test
{
public:
    Test(int number);
    ~Test();

private:
    int number;
};

Test::Test(int number)
{
    this->number = number;
    cout << "Created object " << number << endl;
}

Test::~Test()
{
    cout << "Deleted object " << number << endl;
}

void foo1()
{
```

```

    Test d(4);
    static Test e(5);
}

void foo2()
{
    Test f(6);
    static Test g(7);
}

Test a(1);
static Test o(0);

void main()
{
    Test b(2);
    static Test c(3);
    foo1();
    foo2();
}

```

№85 Консоль (формат)

<http://cppstudio.com/post/319/>

БИБЛИОТЕКА STL

Общие методы:

Метод	Описание
.push_back(*элемент*)	добавить элемент в конец
.pop_back()	удалить последний элемент
.size()	кол-во элементов
.empty()	предикат пустоты контейнера

1. array STL

Динамический массив на максимальках

Инициализация:

```
array<int, 4> arr = { 0,1,2,3};
```

или

```
array<int> muVector(20) //20 нулей
```

или

```
array<int> muVector(20,55) //20 чисел 55
```

Методы:

Метод	Описание
<code>.at(*элемент*)</code>	аналогичен [], только бросает исключение (try catch проверка) <u>код</u> : cout<< arr.at(2) << endl; → 2
<code>.clear()</code>	[вместимость], чистит вектор
<code>.capacity()</code>	кол-во подготовленных ячеек памяти
<code>.reserve(*число*)</code>	значение capacity вручную, выделение памяти вектора заранее
<code>.shrink_to_fit()</code>	capacity = size
<code>.resize(*число*)</code>	аналогично первоначальной инициализации <u>код</u> : myVector.resize(20,55)
<code>.begin()</code>	возвращает адрес первого эл-та
<code>.end()</code>	возвращает адрес ПОСЛЕ последнего эл-та
<code>.insert(*итератор с указателем, куда поместить*, *элемент для размещения*)</code>	вставляет элемент по адресу (ничего не удаляет)
<code>.erase(*итератор с указателем, какой удалить*)</code>	удаляет элемент по адресу
<code>.front</code>	возвращает <u>значение первого</u> эл-та контейнера
<code>.back</code>	возвращает <u>значение последнего</u> эл-та контейнера

код с исключением:

```
try{
    cout << arr.at(10) << endl;
}

catch (const std::exception& ex){
    cout << ex.what() << endl;
}
```

- Метод `size` показывает кол-во эл-тов
- Метод `fill` заполняет весь массив каким-либо значением
- Метод `front` – первый эл-т массива

- Метод `back` – последний эл-т массива

Общее правило: контейнеры можно сравнивать

```
array<int, 4> arr1 = { 4,1,2,3 };
array<int, 4> arr2 = { 3,2,1,0 };           -> 1 (т.к. < и > сравнивают по очереди)
cout << (arr1 > arr2) << endl;
```

2. Итераторы STL

Грубо говоря, универсальный для всех контейнеров STL указатель

```
vector<int> myVector = { 1,9,44,422,676,78 };      ← инициализация вектора
vector<int>::iterator it;   ← инициализация итератора
it = myVector.begin();    ← ставим итератор на начало вектора
*it = 1000;
cout << *it << endl;
```

- Итераторы поддерживают арифметику указателей:
`it += 2;`
- С помощью итератора можно пройти по контейнеру:
`for (vector<int>::iterator it = myVector.begin(); it != myVector.end(); it++){
 cout << "Итератор: " << *it << endl;
}`
- Итератор можно сделать константным. Он не допускает изменения по своему адресу:
`vector<int>::const_iterator it = myVector.begin()`
- Обычный итератор – наследник константного
т.е. в константный итератор можно поместить обычный и тот станет константным.
Пример: `vector<int>::const_iterator it = myVector.begin()`
Метод `.begin()` возвращает обычный итератор, а всё равно `it` константный.



- `reverse_iterator` - итератор наоборот

```
vector<int> myVector = { 1,9,44,422,676,78 };

for (vector<int>::reverse_iterator i = myVector.rbegin(); i != для
{
    cout << *i << endl;
}
```

78
676
422
44
9
1

- Функция advance – арифметика указателей.
Принимает: advance(*итератор*, *+ сколько*)

```
vector<int> myVector = { 1,9,44,422,676,78 };

vector<int>::iterator it = myVector.begin();

//cout << *(it + 3) << endl;
advance(it, 3);

cout << *it << endl;
```

422
Для

- Применяем

Код:

```
#include <iostream>
#include <locale>

#include <vector>

using namespace std;

int main() {

    setlocale(LC_ALL, "Russian");

    vector<int> myVector = { 1,2,3,4,5,6,7 };

    cout << "Начальный вектор: " << endl;

    for (vector<int>::iterator i = myVector.begin(); i != myVector.end(); i++) {
        cout << *i << endl;
    }

    vector<int>::iterator it = myVector.begin();

    advance(it, 7);

    cout << endl << "Insert" << endl << endl;

    myVector.insert(it, 999);

    for (vector<int>::iterator i = myVector.begin(); i != myVector.end(); i++) {
        cout << *i << endl;
    }

    cout << endl << "Erase" << endl << endl;
```

```

vector<int>::iterator itErase = myVector.begin();

itErase++;

myVector.erase(itErase);

for (vector<int>::iterator i = myVector.begin(); i != myVector.end(); i++) {
    cout << *i << endl;
}

cout << endl << "Range Erase" << endl << endl;

itErase = myVector.begin(); //снова в начало

myVector.erase(itErase,itErase+2);

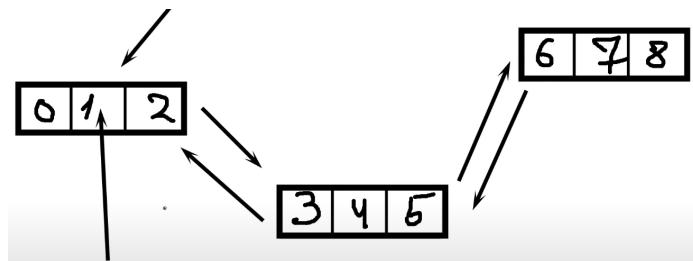
for (vector<int>::iterator i = myVector.begin(); i != myVector.end(); i++) {
    cout << *i << endl;
}

return 0;
}

```

3. deque STL

Двусвязный список векторов



Быстрее, чем лист, но медленнее вектора

4. list STL

<code>.splice(*итератор листа,куда прицепить*,*имя другого листа*)</code>	перенести второй лист к первому
<code>.push_front(*элемент листа*)</code>	добавить эл-т в начало листа
<code>.pop_front()</code>	удалить эл-т из начала
<code>.remove(*элемент листа*)</code>	удаляет все переданные эл-ты их листа

<code>.remove_if()</code>	
<code>.unique()</code>	удаляет дубликаты в листе (лист нужен упорядоченный)
<code>.merge(*имя другого листа*)</code>	упорядоченно перецепляет другой лист
<code>.reverse()</code>	перевернуть лист
<code>.sort()</code>	отсортировать лист
<code>.assign(*итератор начала*, *итератор конца*)</code>	заменить эл-ты листа 1 на эл-ы листа 2
<code>.swap(*имя другого листа*)</code>	поменять имена листов

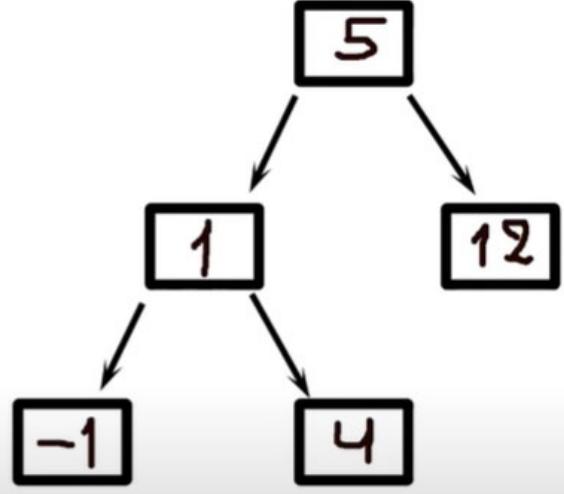
5. set STL

- Бинарное дерево с методами.
- Хранит только уникальные элементы.
- Вывод элементов – в порядке возрастания
- Нельзя изменить какой-то элемент – нарушится структура. Вместо этого – заменить его другим

```
int main()
{
    setlocale(LC_ALL, "ru");

    set<int> mySet;

    mySet.insert(5);
    mySet.insert(1);
    mySet.insert(12);
    mySet.insert(4);
    mySet.insert(-1);
```



```

int main() {
    setlocale(LC_ALL, "ru");

    set<int> mySet;

    mySet.insert(5);
    mySet.insert(2);
    mySet.insert(7);

    //for each (auto var in mySet)
    //{
    //    cout << var;
    //}

    for (auto &var : mySet)
    {
        cout << var << " ";
    }

    return 0;
}

```

Консоль отладки Microsoft Visual Studio
2 5 7
C:\Users\artem\Desktop\For_Experiments2\For_Experiments.exe (процесс 8468) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно.

.find(*элемент сета*)	найти элемент, возвращает ИТЕРАТОР (если не нашёл эл-т, то .end)
.erase (*элемент сета*)	удалить элемент, возвращает bool
.insert (*элемент сета*)	вставить эл-т, возвращает pair(*элемент*, bool)

Код №1

```

#include <iostream>
using namespace std;

#include <set>
#include <algorithm>
#include <iterator>

void showSet(set<int>& mySet);

int main() {
    setlocale(LC_ALL, "ru");

    set<int> mySet;

    char choice;
    int value;

    do {
        cout << "Добавим элемент в сет? (y/n)" << endl << "Введите: ";
        cin >> choice;

        if (choice == 'y' || choice == 'Y')
        {
            cout << "Какой элемент хотите добавить?" << endl << "Введите: ";
            cin >> value;
            auto result = mySet.insert(value);
            if (result.second == 0) cout << "Элемент не добавлен - такой уже есть!" <<
endl;
            showSet(mySet);
            continue;
        }
        if (choice == 'n' || choice == 'N') break;
    }
}

```

```

        cout << "Неверный ввод, повторите!" << endl;
        choice = 'y';

    } while (choice == 'y' || choice == 'Y');

    cout << "Попробуем отыскать какое-нибудь число в сете?" << endl << "Введите: ";
    cin >> value;

    if (mySet.find(value) != mySet.end()) cout << "Число " << value << " найдено!" << endl;
    else cout << "Число " << value << " не найдено!" << endl;

    showSet(mySet);

    cout << "Завершение работы..." << endl;

    return 0;
}

void showSet(set<int>& mySet)
{
    cout << "Содержание сета: ";
    if (mySet.empty()) cout << "сет пустой" << endl;
    else {
        for (auto& var : mySet)
        {
            cout << var << " ";
        }
        cout << endl;
    }
}

```

Код №2

```

#include <iostream>
using namespace std;

#include <set>
#include <algorithm>
#include <iterator>

int main() {
    setlocale(LC_ALL, "ru");

    set<int> mySet;

    int choice, value;

    do {
        cout << "Что хотите сделать?" << endl
            << "1.Распечатать сет" << endl
            << "2.Добавить элемент" << endl
            << "3.Удалить элемент" << endl
            << "4.Найти элемент" << endl
            << "5.Завершить работу" << endl
            << "Введите: ";
        cin >> choice;
        cout << endl;

        if (choice == 1)
        {
            cout << "Содержание сета: ";
            if (mySet.empty()) cout << "сет пустой" << endl << endl;
            else {
                for (auto& var : mySet)
                {

```

```

        cout << var << " ";
    }
    cout << endl << endl;
}
continue;
}

if (choice == 2)
{
    cout << "Какой элемент хотите добавить?" << endl << "Введите: ";
    cin >> value;
    auto result = mySet.insert(value);
    if (result.second == 0) cout << "Элемент не добавлен - такой уже есть!" << endl
<< endl;
    else cout << "Элемент " << value << " добавлен!" << endl << endl;
    continue;
}

if (choice == 3)
{
    cout << "Какой элемент хотите Удалить?" << endl << "Введите: ";
    cin >> value;
    auto result = mySet.erase(value);
    if (result == 0) cout << "Элемент не удалён - такого нет в сете!" << endl <<
endl;
    else cout << "Элемент " << value << " удалён!" << endl << endl;
    continue;
}

if (choice == 4)
{
    cout << "Какое элемент искать в сете?" << endl << "Введите: ";
    cin >> value;

    if (mySet.find(value) != mySet.end()) cout << "Элемент " << value << " найден!"
<< endl << endl;
    else cout << "Элемент " << value << " не найден!" << endl << endl;
    continue;
}

if (choice == 5) break;

cout << "Неверный ввод, повторите!" << endl << endl;
} while (choice != 5);

cout << "Завершение работы..." << endl;

return 0;
}

```

6. multiset STL

- То же, что и set, но хранит и одинаковые эл-ты тоже

.lower_bound(*элемент мультисета*)	найти первый элемент, возвращает ИТЕРАТОР (если не нашёл эл-т, то .end)
.upper_bound(*элемент мультисета*)	найти последний элемент, возвращает ИТЕРАТОР ПОСЛЕ НЕГО
.equal_range(*элемент мультисета*)	lower_bound и upper_bound в одном флаконе. Возвращает pair(*итератор

Код

```
#include <iostream>
using namespace std;

#include <set>
#include <algorithm>
#include <iterator>

int main() {
    setlocale(LC_ALL, "ru");

    multiset<int> mySet = { 5,4,2,5,5,6 };

    for (auto var : mySet)
    {
        cout << var << " ";
    }
    cout << endl;

    auto a = mySet.equal_range(6);

    cout << *a.second << endl;

    return 0;
}
```

7. map STL

- Бинарное дерево (по ключу)
- Хранит пару – ключ и значение
- Есть перегрузка [*ключ*] (возвращает значение)
- По ключу через [] можно присвоить значение и даже создать его (скрин ниже)

Пара – структура данных (шаблонная)

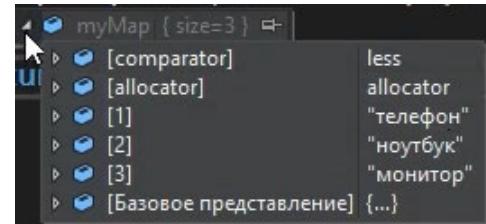
```
pair<int, string> p(0, "телефон");
cout << p.first << endl;
cout << p.second << endl;
```

```
map<int, string> myMap;

myMap.insert(make_pair(1,"телефон"));

myMap.insert(pair<int, string>(2, "ноутбук"));

myMap.emplace(3, "монитор");
```



.emplace (*ключ*, *элемент мэпа*)

аналогичен insert, возвращает пару и bool
ещё аналоги:
.insert(make_pair(*ключ*, *эл-т*))

	.insert(pair<*1тип*, *2тип*>(*ключ*, *эл-т*))
.at(*ключ*)	аналогично [], не создаст пару, если ключа нет. Выбрасывает исключение out_of_range
.erase(*ключ*)	удаляет пару по ключу

```

int main()
{
    setlocale(LC_ALL, "ru");

    map<string, int> myMap;

    myMap.emplace("Петя", 1313);
    myMap.emplace("Маша", 222);
    myMap.emplace("Миша", 4441);

    myMap["Вася"] = 9797;
}

```

```

int main()
{
    setlocale(LC_ALL, "ru");

    map<string, int> myMap;

    myMap.emplace("Петя", 1313);
    myMap.emplace("Маша", 222);
    myMap.emplace("Миша", 4441);

    myMap.at("Вася") = 3; ✖

    return 0;
}

```

Исключение не обработано
Возникло необработанное исключение по адресу 0x75DFAA12 в Lessons.exe: исключение Microsoft C++: std::out_of_range по адресу памяти 0x00EFF474.

Копировать подробности
Параметры исключений

Код

```

#include <iostream>
using namespace std;

#include <map>

int main() {
    setlocale(LC_ALL, "ru");

    map<string, int> myMap;

    myMap.emplace("Вася", 500);
    myMap["Артем"] = 300;

    try
    {
        myMap.at("Егор") = 1000;
    }
    catch (const std::exception& ex)
    {
        cout << ex.what() << endl << endl;
    }

    myMap.erase("Вася");
    if (myMap.erase("Вася") == false) cout << "Вася уже удалён" << endl;

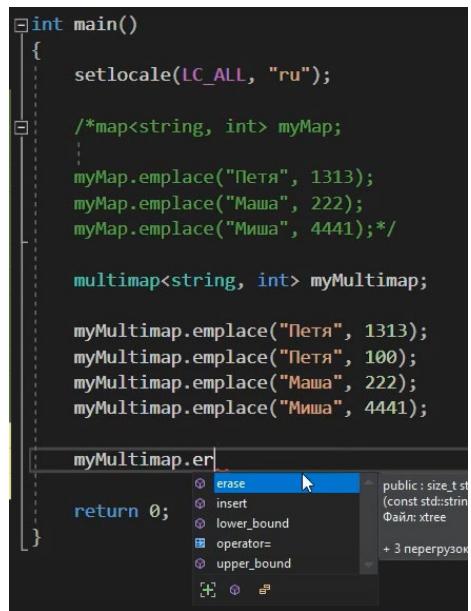
    cout << "Содержание myMap:" << endl << endl;
    for (auto var : myMap)
    {
        cout << var.first << "\t" << var.second << endl;
    }
}

```

```
    return 0;
}
```

8. multimap STL

- Тот же map, но ключи могут дублироваться
- Не перегружен оператор [] и нет метода .at()



```
int main()
{
    setlocale(LC_ALL, "ru");

    /*map<string, int> myMap;
    myMap.emplace("Петя", 1313);
    myMap.emplace("Маша", 222);
    myMap.emplace("Миша", 4441);*/

    multimap<string, int> myMultimap;

    myMultimap.emplace("Петя", 1313);
    myMultimap.emplace("Петя", 100);
    myMultimap.emplace("Маша", 222);
    myMultimap.emplace("Миша", 4441);

    myMultimap.er[erase]
    return 0;
}
```

The screenshot shows a code editor with C++ code. A tooltip is displayed over the `myMultimap.er` part of the code, listing several member functions of `multimap`: `erase`, `insert`, `lower_bound`, `operator=`, and `upper_bound`. The `erase` function is highlighted with a cursor.