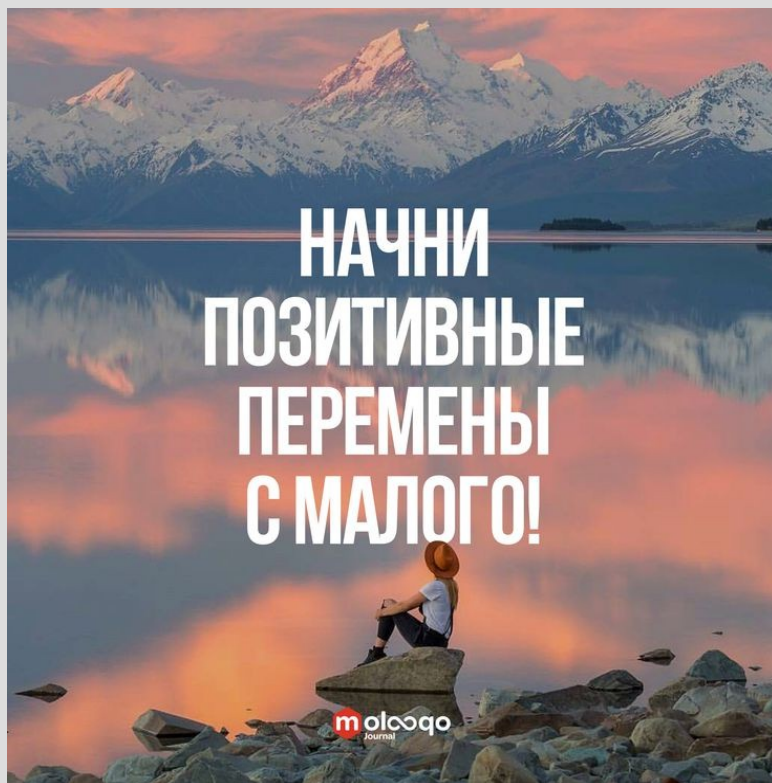


Основы C++. Вебинар №6.

Длительность: 1.5 - 2 ч.



GeekBrains

Что будет на уроке?

- Узнаете об Agile, SCRUM, «Водопаде», Jira и Trello.
- Узнаем, что представляет из себя процесс управления памятью, как выделять и освобождать динамическую память.
- Познакомимся с операторами выделения памяти в языке C.
- Научимся не допускать утечек памяти (memory leak) и изучим операторы new и delete.
- Рассмотрим работу с файловой системой и потоками ввода-вывода.

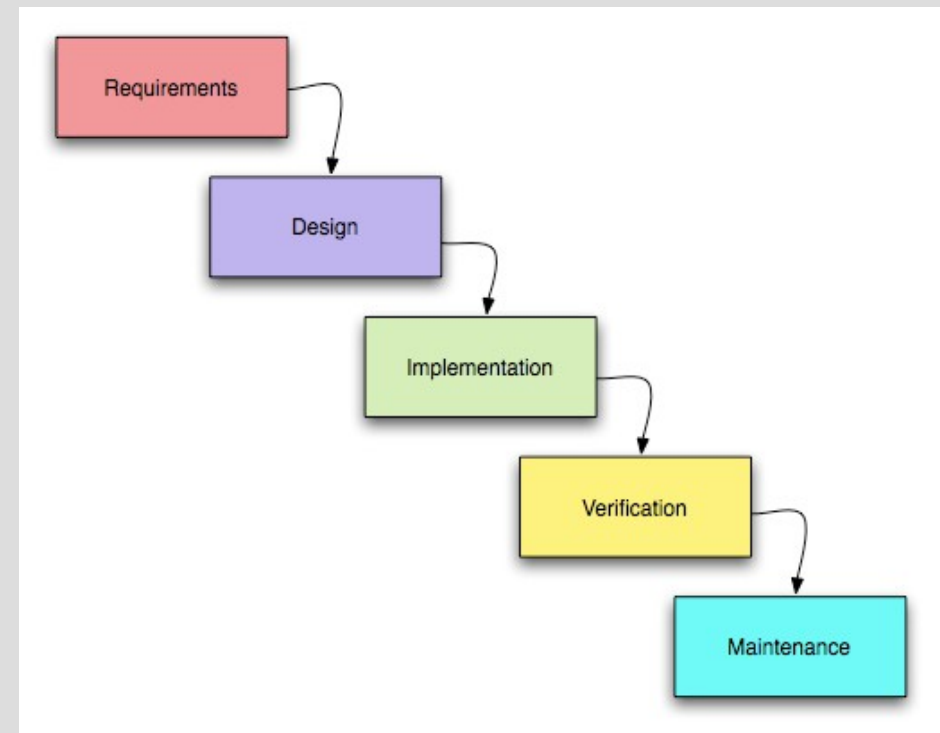


Каскадная модель разработки ПО (Водопад)

Каскадная модель (англ. waterfall model, иногда переводят как модель «**Водопад**») — модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.

В исходной каскадной модели **следующие фазы** шли в таком порядке:

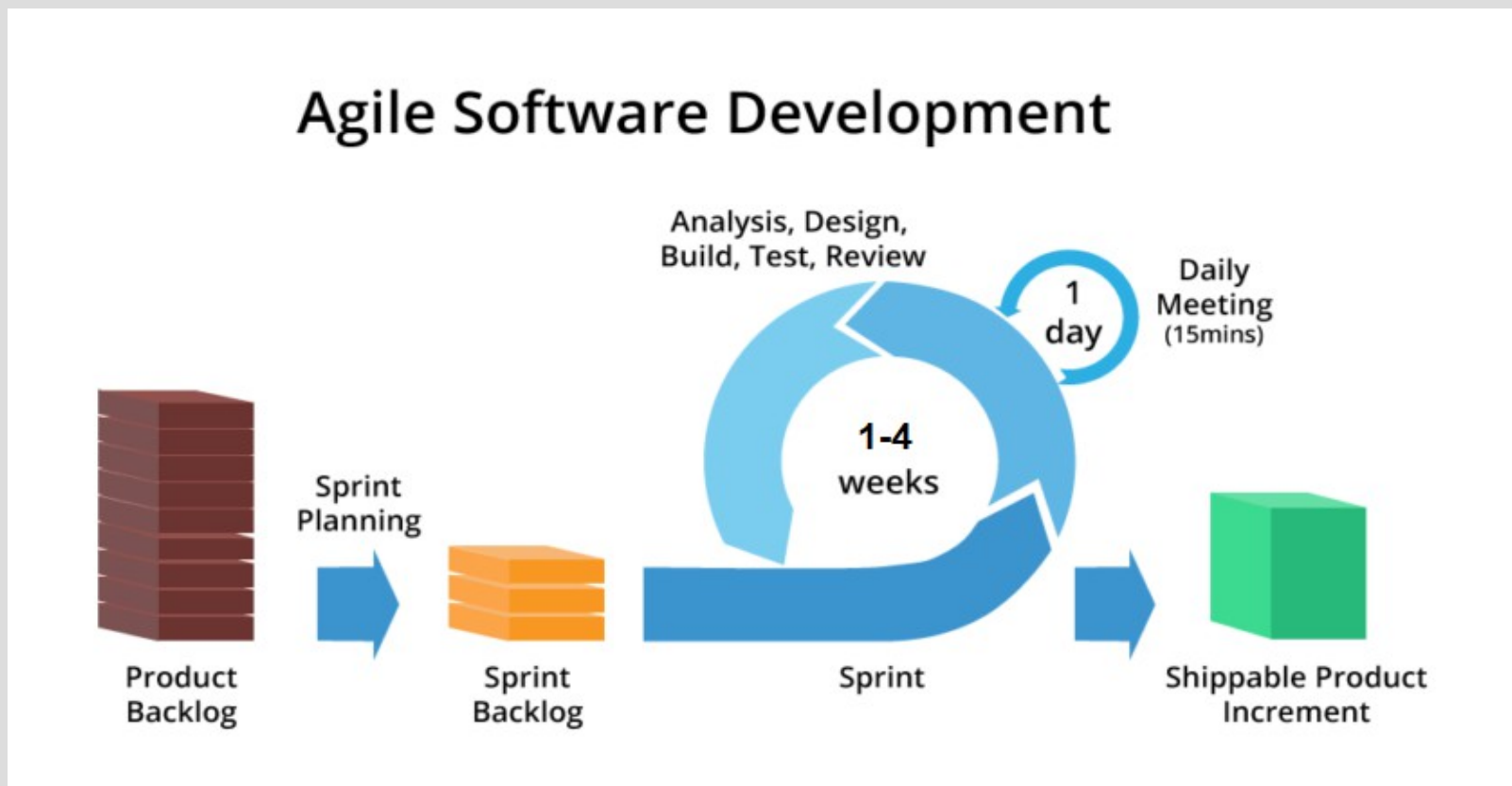
- Определение требований
- Проектирование
- Конструирование («реализация», «кодирование»)
- Воплощение
- Тестирование и отладка («верификация»)
- Инсталляция
- Поддержка



Гибкая методология разработки ПО - Agile

Гибкая методология разработки (англ. agile software development) — обобщающий термин для целого ряда подходов и практик, основанных на ценностях Манифеста гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе.

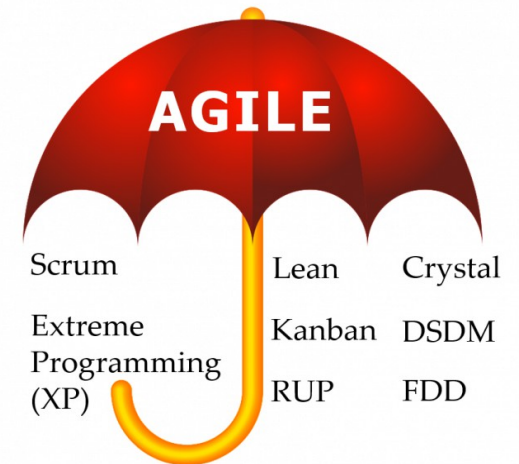
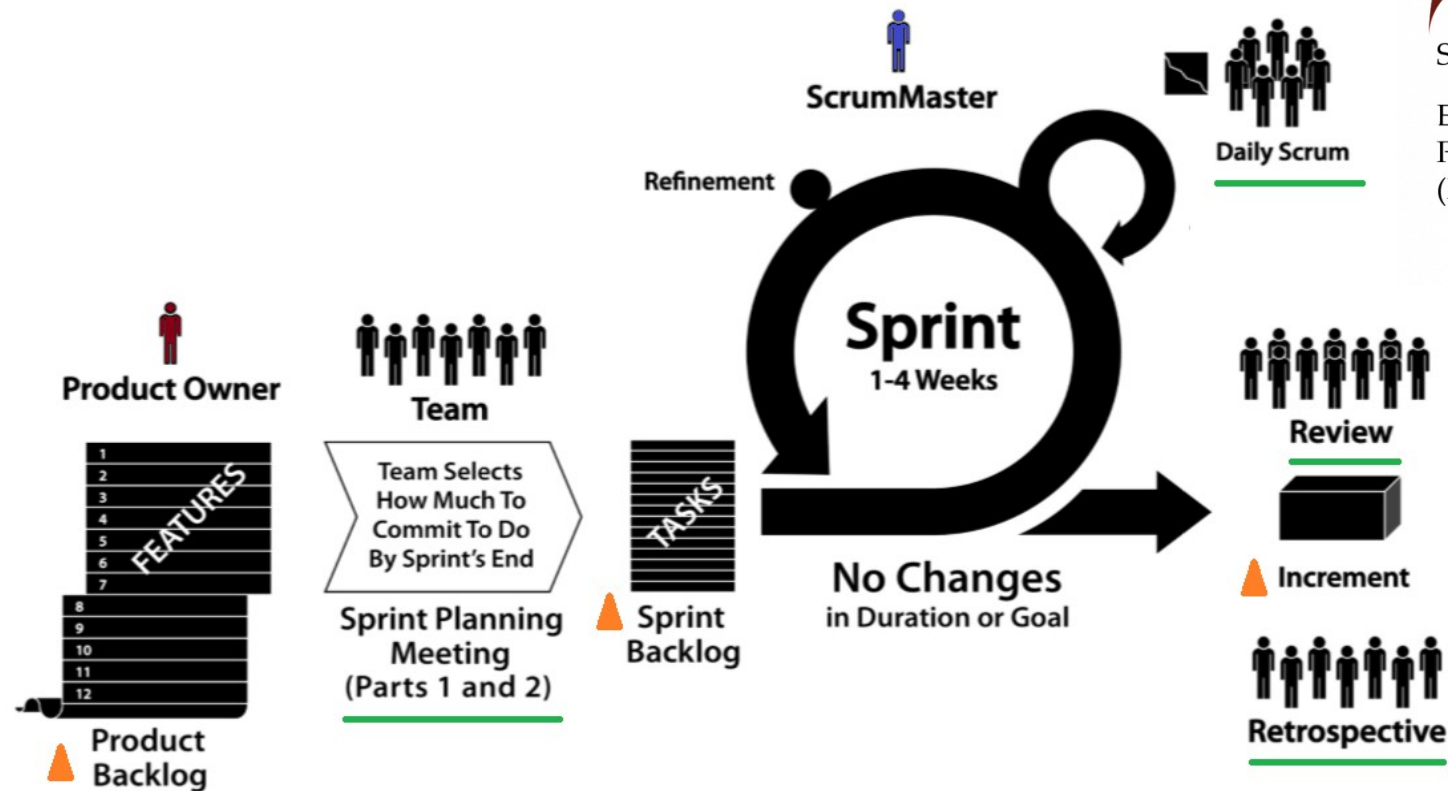
Подробнее: <https://weeek.net/ru/blog/chto-takoe-agile>



SCRUM — одно из воплощений Agile

Подробнее: <https://ru.wikipedia.org/wiki/SCRUM>

SCRUM



ПО управления проектами

Atlassian Jira — коммерческая система отслеживания ошибок (bug-tracker), предназначена для организации взаимодействия с пользователями, хотя в некоторых случаях используется и для управления проектами. Разработана компанией Atlassian, является одним из двух её основных продуктов (наряду с вики-системой Confluence). Имеет веб-интерфейс.





Первый выпуск — в 2002 году. Изначально применялась в процессах разработки программного обеспечения, впоследствии нашла применение в качестве инструмента управления проблемами, задачами, проектами в различных отраслях.

<https://ru.wikipedia.org/wiki/Jira>

Бесплатный аналог:

Trello — облачная программа для управления проектами небольших групп, разработанная Fog Creek Software.

Trello использует парадигму для управления проектами, известную как канбан, метод, который первоначально был популяризирован Toyota в 1980-х. Trello использует freemium-бизнес-модель, платные услуги были запущены в 2013 году. В 2017 году куплен Atlassian за 425 млн \$.

Jira	
 Jira Software	
Тип	система отслеживания ошибок, система управления проектами, проприетарное программное обеспечение и система отслеживания заявок ^[d]
Разработчик	Atlassian
Написана на	Java
Операционная система	UNIX-подобные операционные системы и Microsoft Windows
Первый выпуск	2002
Аппаратная платформа	Java Virtual Machine
Последняя версия	8.14.0 (23 ноября 2020) ^[1]
Лицензия	проприетарное программное обеспечение
Сайт	atlassian.com/sof...  (англ.) ru.atlassian.com/...  (рус.) fr.atlassian.com/...  (фр.)

Kanban доска в Jira

Teams in Space
Scrum: Teams in Space

Backlog

Agile board

Releases

Reports

All issues

Components

Add-ons

PROJECT SHORTCUTS

Mars Team HipChat Room

Space Station Dev Roadmap

Teams in Space Org Chart

Orbital Spotify Playlist

Hyperspeed Bitbucket Repo

+ Add shortcut

TIS-70 Scrum Board

QUICK FILTERS: [Critical partners](#) [Only my partners](#) [Recently updated](#)

12 To do

2 In progress

3 Done

▼ TIS Developer Love 3 issues

TIS-37

↑ Service should return prior trip details and info

SeeSpaceEZ plus

2

TIS-10

↑ Bad JSON data coming back from hotel API

SeeSpaceEZ plus

TIS-8

↑ Requesting flights is now taking > 5 seconds

SeeSpaceEZ plus

▼ Everything Else 21 issues

TIS-68

↑ Homepage footer uses an inline style-should use class

Large Team Support

TIS-20

↑ Engage Saturn Shuttle lines for group tours

Space Travel Partners

3

TIS-12

⊘ Create 90 day plans for all departments in Mars office

TIS-17

↑ Engage Saturn's Rings Resort as preferred

Space Travel Partners

TIS-56

↑ Add pointer to main css file to create child themes

Large Team Support

TIS-45

↑ Email non registered users to sign up with TIS

SeeSpaceEZ plus

7

Понятие кучи (heap)

Программа (исполнимый файл) состоит из:

- Сегменты кода (команды для CPU).
- Сегменты данных (глобальные переменные, статические переменные).
- Сегмент стека (локальные переменные, аргументы функций и пр.).

Все текущие вычисления CPU производит на регистрах. Регистры самая близкая к CPU память и самая быстрая это как его личные локальные переменные для расчетов.

Также программа может получать (выделять) для своей работы память из **свободной RAM** — эта область называется **кучей (heap)**. Не путать с кучей — структурой данных (дерево).

Для получения и освобождения памяти из кучи **в языке C** есть следующие функции:

`malloc`, `calloc`, `realloc` — выделение.

`free` — освобождение.

В языке C++:

`new` — выделение.

`delete` — освобождение.

Регистры процессора x86

Регистры данных

AH	AL	AX Аккумулятор
BH	BL	BX Базовый регистр
CH	CL	CX Счетчик
DH	DL	DX Регистр данных

Сегментные регистры

CS	Регистр сегмента команд
DS	Регистр сегмента данных
ES	Регистр дополнительного сегмента данных
SS	Регистр сегмента стека

Регистры-указатели

SI	Индекс источника
DI	Индекс приемника
BP	Указатель базы
SP	Указатель стека

Прочие регистры

IP	Указатель команд
FLAGS	Регистр флагов

Работа с кучей - C-style

Прототипы функций:

```
void * malloc( size_t sizeMem );           // Размер массива в байтах
void * calloc( size_t number, size_t size ); // Число элементов, размер элемента в байтах
void * realloc( void * ptrMem, size_t size ); // Выделяет новый блок памяти, копирует, освобождает старый указатель.

void free( void * ptrMem );
```

Пример программы:

```
#include <iostream>
#include <malloc.h>

using namespace std;

int main()
{
    int* a; // Указатель на динамически выделенный массив
    int n;
    cout << "Enter size of your array: ";
    cin >> n;

    a = (int*) malloc(n * sizeof(int)); // Выделение памяти

    a[0] = 100; // Работа с массивом

    free(a); // Освобождаем память
    a = NULL;

    return 0;
}
```

Операторы new и delete в C++

Безопасная работа с new (используем std::nothrow):

```
#include <iostream>

int main()
{
    int* ptrArr; // Указатель на массив
    int n;       // Количество элементов
    std::cout << "Enter size of your array: ";
    std::cin >> n;

    if (n > 0)
    {
        ptrArr = new (std::nothrow) int[n]; // Выделение памяти под одномерный массив

        if (ptrArr != nullptr) // Если память не выделена то указатель будет нулевой
        {
            // Работа с массивом
            ptrArr[0] = 100;
            std::cout << "ptrArr[0] = " << ptrArr[0];

            // освобождение памяти
            delete[] ptrArr;
            ptrArr = nullptr;
        }
        else
        {
            std::cout << "Error! Can not allocate memory!";
        }
    }

    return 0;
}
```

Операторы new и delete в C++

Можно проще, но программа упадет если память не может быть выделена:

```
#include <iostream>

int main()
{
    int* ptrArr; // Указатель на массив
    int n;        // Количество элементов
    std::cout << "Enter size of your array: ";
    std::cin >> n;

    if (n > 0)
    {
        ptrArr = new int[n]; // Выделение памяти под одномерный массив

        // Работа с массивом
        ptrArr[0] = 100;
        std::cout << "ptrArr[0] = " << ptrArr[0];

        // освобождение памяти
        delete[] ptrArr;
        ptrArr = nullptr;
    }

    return 0;
}
```

Выделение динамической памяти для двухмерного массива

```
int main()
{
    int** ptrArr; // Указатель на одномерный массив указателей на int
    const size_t m = 5; // Количество элементов m x n
    const size_t n = 5;

    // 1. Выделение памяти
    ptrArr = new int * [m]; // Выделение памяти под одномерный массив указателей
    for (size_t i = 0; i < m; i++) {
        ptrArr[i] = new int[n]; // Выделяем массив под одномерный int массив
    }

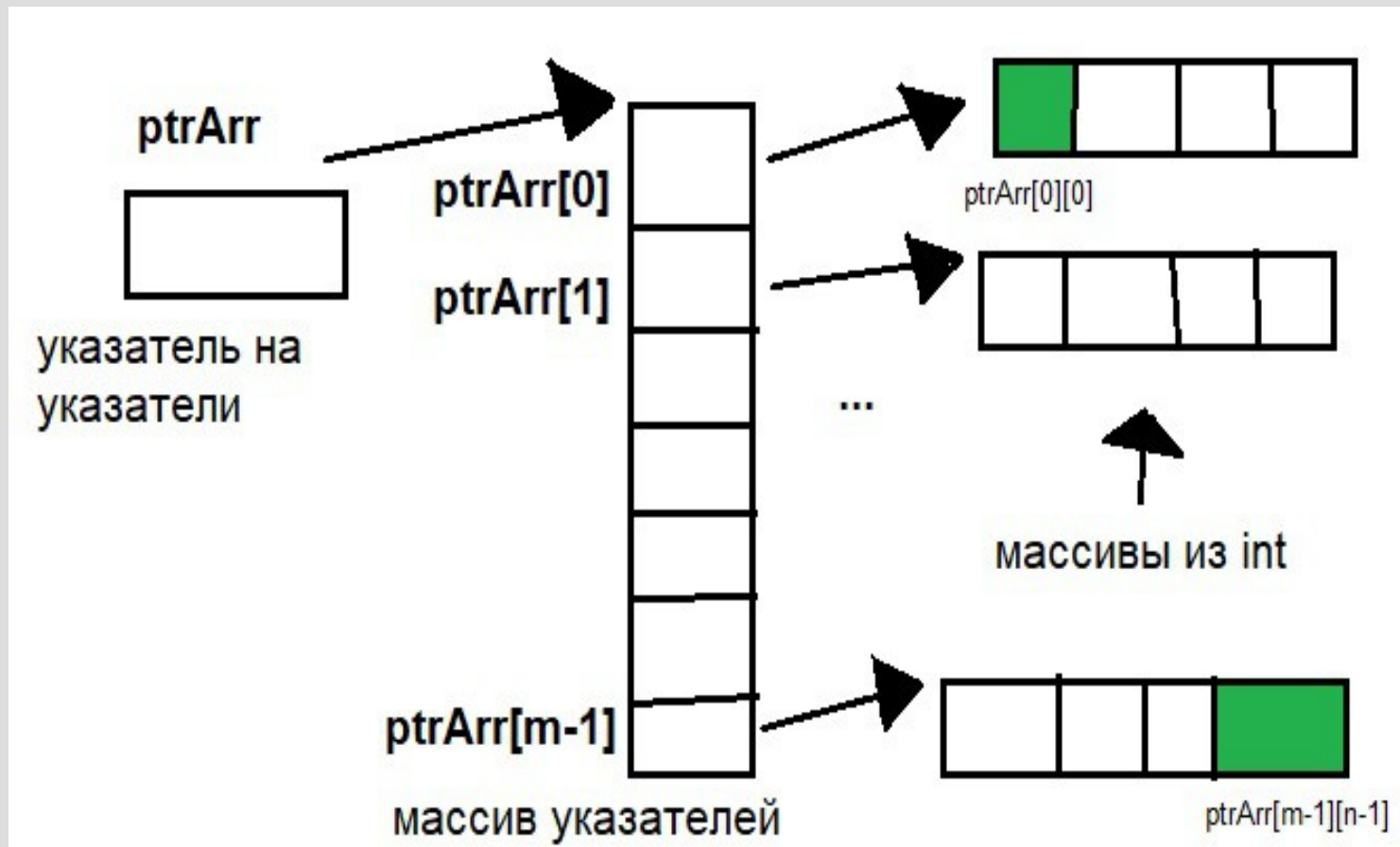
    // 2. Работа с массивом
    ptrArr[0][0] = 100; // Запись в первый элемента матрицы
    ptrArr[m-1][n-1] = 200; // Запись в последний элемент матрицы

    std::cout << "ptrArr[0][0]=" << ptrArr[0][0] << " ptrArr[m-1][n-1]=" << ptrArr[m - 1][n - 1];

    // 3. Освобождение памяти
    for (size_t i = 0; i < m; i++) {
        delete [] ptrArr[i]; // Освобождаем i-тый одномерный int массив
    }
    delete[] ptrArr; // Удаляем массив указателей
    ptrArr = nullptr;

    return 0;
}
```

Выделение динамической памяти для двухмерного массива



Создаем динамически структуру

```
struct Employee { // Новый тип данных Сотрудник
    long id; // ID сотрудника
    unsigned short age; // его возраст
    double salary; // его зарплата
};

enum CompanySize { CS_SMALL, CS_MIDDLE, CS_BIG }; // перечисление – размер компании

struct Company { // Новый тип данных Компания
    Employee people[30]; // Ее сотрудники (30 максимум)
    Employee director; // Директор
    CompanySize size; // Размер компании
    unsigned int PeopleNumber; // количество сотрудников
};

int main() {

    // 1. Выделение памяти для структуры
    Company * pCompany = new Company;

    // 2. Работаем со структурой
    pCompany->director = { 125093, 45, 350'000.0 };
    pCompany->size = CS_MIDDLE;
    pCompany->PeopleNumber = 215;
    pCompany->people[0] = pCompany->director;
    pCompany->people[1] = { 134578, 34, 60'000.0 };

    // 3. Освобождаем память
    delete pCompany;
    return 0;
}
```


Создаем динамически массив структур

```
#include <iostream>

struct Employee { // Новый тип данных Сотрудник
    long id;        // ID сотрудника
    unsigned short age; // его возраст
    double salary;   // его зарплата
};

int main() {

    // 1. Выделение памяти для массива структур
    const size_t size = 3;
    Employee* pArray = new (std::nothrow) Employee[size];

    if (pArray != nullptr)
    {
        // 2. Работаем с массивом структур
        pArray[0].age = 30;
        pArray[0].id = 55645435467;
        pArray[0].salary = 35'000.0;

        pArray[1] = { 78687678, 35, 40'000.0 };
        pArray[2] = { 97655667, 50, 80'000.0 };

        // 3. Освобождаем память
        delete[] pArray;
        pArray = nullptr;
    }

    return 0;
}
```

Вопросы на собеседованиях по C++ (информация для продвинутых)

1. Можно ли перегрузить операции new и delete?
2. Что такое new placement?



Борьба с memory leaks

(информация для продвинутых)

Умный указатель (англ. smart pointer) — идиома косвенного обращения к памяти, которая широко используется при программировании на языках высокого уровня: C++, Rust и так далее. Как правило, реализуется в виде специализированного класса (обычно — параметризованного), имитирующего интерфейс обычного указателя и добавляющего необходимую новую функциональность (например — проверку границ при доступе или очистку памяти).

Как правило, основной целью задеирования умных указателей является инкапсуляция работы с динамической памятью таким образом, чтобы свойства и поведение умных указателей имитировали свойства и поведение обычных указателей. При этом на них возлагается обязанность своевременного и аккуратного высвобождения выделенных ресурсов, что упрощает разработку кода и процесс отладки, исключая утечки памяти и возникновение висячих ссылок.

В C++11 появилось 3-и вида умных указателей:

- `std::unique_ptr`
- `std::shared_ptr`
- `std::weak_ptr`

Парадигма программирования RAII: Одна из лучших особенностей классов — это деструкторы, которые автоматически выполняются при выходе объекта класса из области видимости. При выделении памяти в конструкторе класса, вы можете быть уверены, что эта память будет освобождена в деструкторе при уничтожении объекта класса (независимо от того, выйдет ли он из области видимости, будет ли явно удален и т.д.). Это лежит в основе парадигмы программирования RAII (Resource Acquisition Is Initialization).

Запись в файл - ofstream

```
#include <iostream>
#include <fstream> // Для использования ofstream

using namespace std;

int main()
{
    const size_t size = 6;
    int array[size] = { 10, 20, 70, -90, 50, 80 };

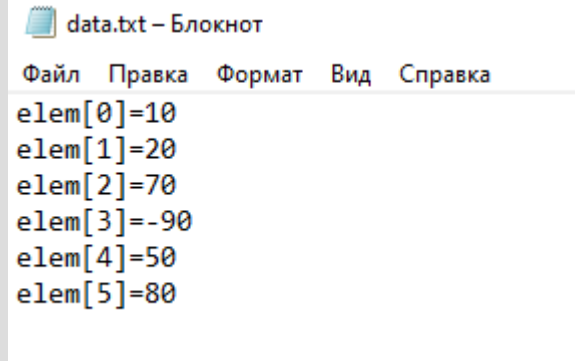
    ofstream fout("data.txt"); // 1. Создаем файл для записи данных

    // 2. Сохраняем в файл элементы массива
    int i = 0;
    for (int element : array) // Цикл по массиву array
    {
        fout << "elem[" << i++ << "]=" << element << endl;
    }

    fout.close(); // Закрываем файл
    return 0;
}
```

О ofstream подробнее:

https://en.cppreference.com/w/cpp/io/basic_ofstream



```
data.txt - Блокнот
Файл  Правка  Формат  Вид  Справка
elem[0]=10
elem[1]=20
elem[2]=70
elem[3]=-90
elem[4]=50
elem[5]=80
```

Чтение из файла - ifstream

```
#include <iostream>
#include <fstream>

using namespace std;

// Файл mydata.txt должен иметь следующий вид:
// 10 -10 30 50 80 -90

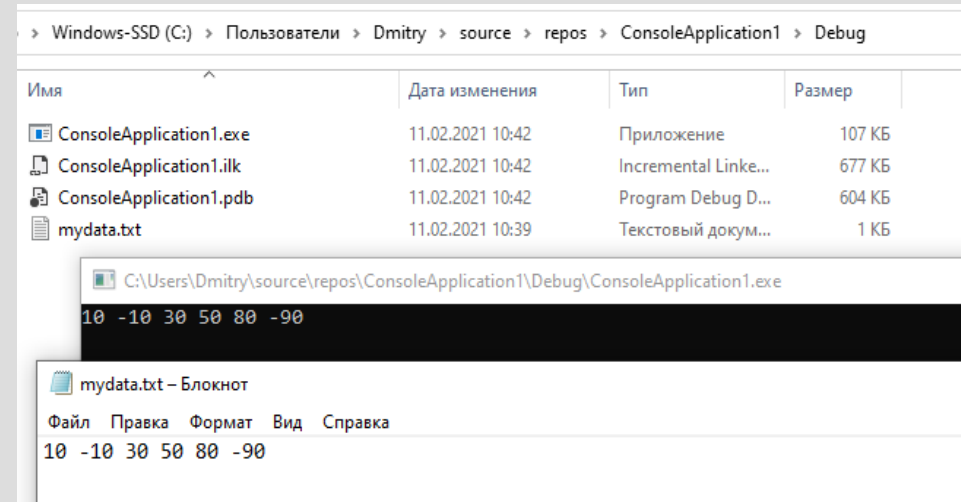
int main()
{
    const size_t size = 6;
    int array[size] = { 0 };

    ifstream fin("mydata.txt"); // 1. Открываем файл для чтения данных

    if (fin.is_open()) // Если файл был успешно открыт
    {
        // 2. Читаем числа из файла в массив
        for (int i = 0; i < size; i++)
        {
            fin >> array[i];          // Читаем из файла элемент массива
            cout << array[i] << " "; // Выводим на экран
        }

        fin.close(); // Закрываем файл
    }
    else
    {
        cout << "Error. Can not open file."; // Сообщение об ошибке
    }

    cin.get(); // Ждем нажатия клавиши для задержки закрытия программы
    return 0;
}
```



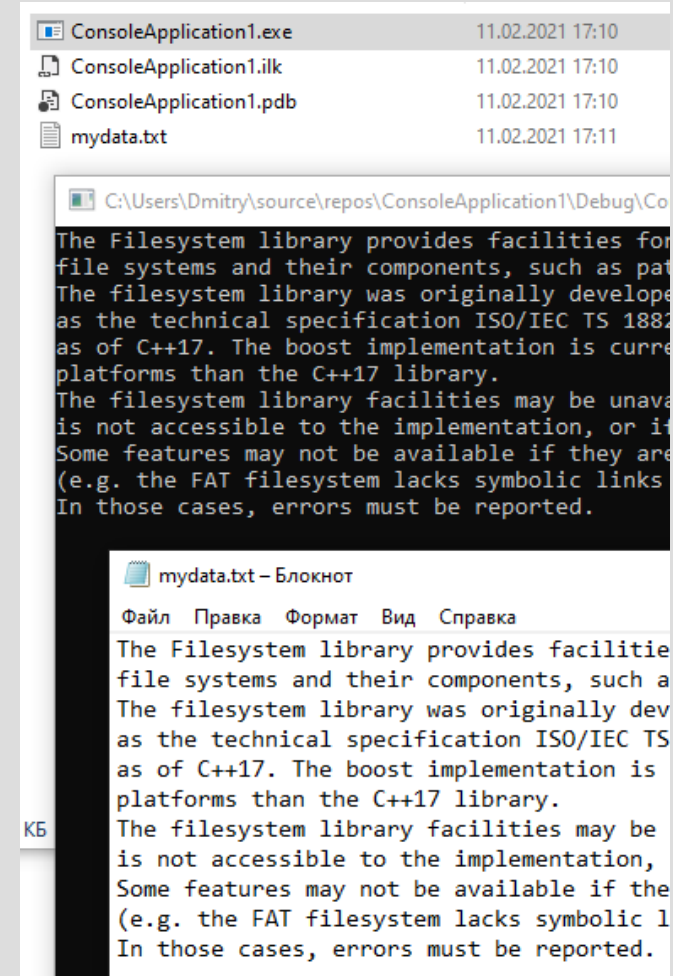
Читаем строку из ifstream, функция getline

```
int main()
{
    ifstream fin("mydata.txt"); // 1. Открываем файл для чтения данных

    if (fin.is_open()) // Если файл был успешно открыт
    {
        // 2. Выведем содержимое файла на экран
        const size_t size = 255;
        char buf[size];
        while (!fin.eof()) // Пока не конец файла
        {
            fin.getline(buf, size); // Читаем из файла строку
            cout << buf << endl;    // Выводим на экран
        }

        fin.close(); // Закрываем файл
    }
    else
    {
        cout << "Error. Can not open file."; // Сообщение об ошибке
    }

    cin.get(); // Ждем нажатия клавиши для задержки закрытия программы
    return 0;
}
```



Запись и чтение в бинарные файлы функции write и read

```
struct TEmployee { // Структура сотрудник
    int id;
    double salary;
    unsigned short age;
    char name[32];
};

int main()
{
    TEmployee e1 = { 12354321, 120'000.0, 32, "David Saimon"};
    const string FileName = "1.bin";

    // Запись структуры в бинарный файл с помощью функции write
    ofstream fout(FileName, ios_base::binary);
    if (fout.is_open())
    {
        fout.write((char*)&e1, sizeof(e1));
        fout.close();
    }

    TEmployee e2 = { 0 }; // Структура для считывания из файла

    // Чтение структуры из бинарного файла с помощью функции read
    ifstream fin(FileName, ios_base::binary);
    if (fin.is_open())
    {
        fin.read((char*)&e2, sizeof(e2));
        fin.close();
    }

    cout << e2.id << " " << e2.salary << " " << e2.age << " " << e2.name << endl;
    return 0;
}
```

Считываем безопасно число из `std::cin`, функции `fail`, `clear` и `ignore`

```
#include <iostream>

using namespace std;

int getIntNumber()
{
    while (true) // цикл продолжается до тех пор, пока пользователь не введёт корректное значение
    {
        cout << "Enter an integer value: ";
        int num;
        cin >> num;

        if (cin.fail()) // если предыдущее извлечение оказалось неудачным,
        {
            cout << "Error. Please enter an integer number!" << endl;
            cin.clear(); // то возвращаем cin в 'обычный' режим работы
            cin.ignore(32767, '\n'); // и удаляем значения предыдущего ввода из входного буфера
        }
        else // если всё хорошо, то возвращаем num
        {
            return num;
        }
    }
}

int main()
{
    cout << "Your int number: " << getIntNumber() << endl;
    return 0;
}
```

C++17 — библиотека filesystem

В ней много полезных функций для работы с файловой системой, директориями и файлами. Можно например получить список все файлов в любой папке.

Подробнее состав библиотеки: <https://en.cppreference.com/w/cpp/filesystem>

```
#include <iostream>
#include <filesystem>

namespace fs = std::filesystem; // Упрощаем namespace

using namespace std;

int main()
{
    const char filename [] = "mydata.txt";

    if (fs::exists(filename)) // Если файл существует
    {
        fs::remove(filename); // Удаляем его

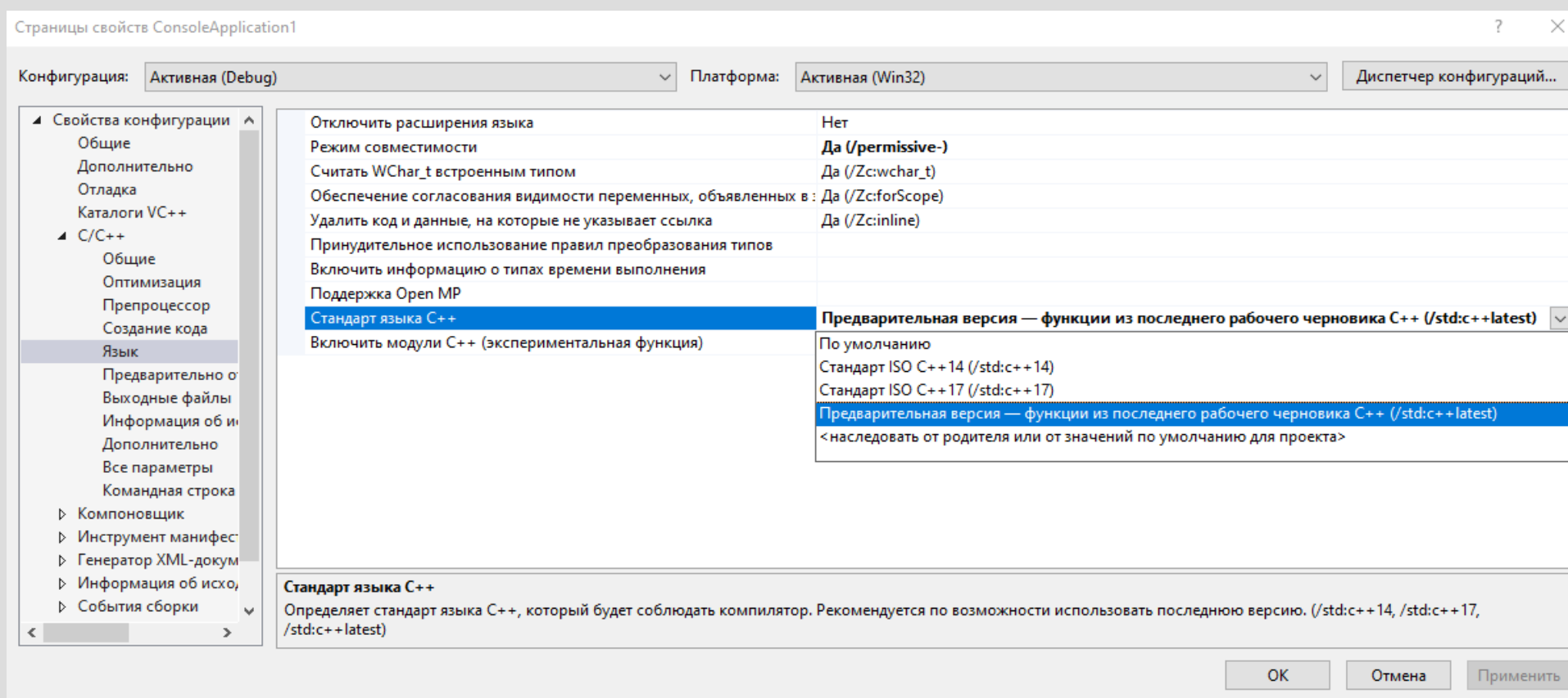
        cout << filename << " - was removed!" << endl;
    }

    cin.get(); // Ждем нажатия клавиши для задержки закрытия программы
    return 0;
}
```

Чтобы использовать эту библиотеку необходимо в свойствах проекта, в опциях компилятора в VS включить C++17 или выше.

C++17 — библиотека filesystem

Чтобы использовать эту библиотеку необходимо в свойствах проекта, в опциях компилятора в VS включить C++17 или выше.



Домашнее задание

1. Выделить в памяти динамический одномерный массив типа `int`. Размер массива запросить у пользователя. Инициализировать его числами степенями двойки: 1, 2, 4, 8, 16, 32, 64, 128 ... Вывести массив на экран. Не забыть освободить память. =) Разбить программу на функции.
2. Динамически выделить матрицу 4x4 типа `int`. Инициализировать ее псевдослучайными числами через функцию `rand`. Вывести на экран. Разбейте вашу программу на функции которые вызываются из `main`.
3. Написать программу с 2-я функциями, которая создаст два текстовых файла (*.txt), примерно по 50-100 символов в каждом (особого значения не имеет с каким именно содержимым). Имена файлов запросить у пользователя.
4. * Написать функцию, «склеивающую» эти файлы в третий текстовый файл (имя файлов спросить у пользователя).
5. * Написать программу, которая проверяет присутствует ли указанное пользователем при запуске программы слово в указанном пользователем файле (для простоты работаем только с латиницей).



Основы C++. Вебинар №6.

Успеха с домашним заданием!



GeekBrains