Санкт-Петербургский государственный университет

Вопросы

1. Что такое механизм обработки ошибок в C++?
2. Можно ли выбрасывать exception из конструктора? Приведите пример кода.
3. Какой будет результат выполнения кода? Объясните.
4. Можно ли выбрасывать exception из деструктора?
5. Что такое exception (std::exception)? Реализуйте свой класс исключение.
6. Что такое runtime_error (std::runtime_error)? Приведите иерархию основных исключений.
7. Как перехватывать исключение деления на 0?
8. Что делает ключевое слово noexcept?
9. Что такое паттерны программирования? Приведите примеры.
10. Для чего нужен Singleton? Преимущества и недостатки.

Санкт-Петербургский государственный университет

Вопросы

11. Что такое перегрузка функций? Какие у неё есть проблемы.
12. Что такое шаблоны в C++?
13. Что такое инстанциация шаблона?
14. Какие параметры бывают у шаблона?
15. Что такое специализация шаблона? Приведите пример использования.
16. Расскажите об имплементации шаблонных классов в cpp-файле.
17. Что такое структура данных? Какие основные СД можно выделить?
18. Что такое STL и из чего состоит?
19. Статический / динамический массив. Одномерный / двумерный.
20. Что такое std::size_t? Привести пример бесконечного цикла.

Санкт-Петербургский государственный университет

Вопросы

21. Какой контейнер является статическим массивом в STL?
22. Какая иерархия у памяти? Охарактеризуйте. Напишите примерные объёмы и время доступа.
23. Что такое виртуальная память? Для решения каких задач она была создана?
24. Как работают inline-функции? Может ли такая функция быть рекурсивной?
25. Что такое идиома remove-erase? Написать результат работы remove для {...}
26. Что такое явное и неявное приведение типов в C++?
27. Расскажите о функциях явного приведения типов в C++.
28. Как выделять и освобождать динамическую память в C? В чём разница с new и delete из C++?
29. Какая разница между calloc и malloc? Что делает realloc?
30. Что такое висячий указатель / висячая ссылка? Приведите примеры.

Санкт-Петербургский государственный университет

Вопросы

31. Что такое умный указатель? Какие умные указатели есть в стандартной библиотеке?
32. Как работает std::unique_ptr?
33. Что произойдёт? Приведите пример с сырыми указателями и объясните.
34. Что произойдёт в коде?
35. Как работает std::shared_ptr?
36. Что такое std::weak_ptr и концепция слабых ссылок?
37. Реализовать класс-шаблон, необходимый для выполнения кода.
38. Напишите класс дроби, выбрасывающий исключение при делении на 0. Исправьте main.
39. Напишите правильный обработчик исключений функции run_game().
40. Преобразуйте перегруженные функции в шаблоны.

Санкт-Петербургский государственный университет

Вопросы

41 Объясните зачем нужен const_cast<T> и какой и почему будет результат?

Санкт-Петербургский государственный университет

Практика. Вопросы

1. Напишите Singleton.
2. Напишите контейнер статический массив с КсПа(*начало, *конец).
3. Напишите алгоритм "remove_if" для массива из идиомы remove-erase.
4. Напишие 2 варианта умножение матриц: обычный и дружелюбный к кэшу. Объясните.
5. Напишите класс, у которого будет утечка памяти из-за циклического std::shared_ptr<T>.
6. Напишите удаление элемента из двухсвязного списка.

Санкт-Петербургский государственный университет

Ответы

1 Что такое механизм обработки ошибок в C++?

Механизм исключений создан, чтобы отделить описание нормального хода программы от описания реакции на сбои.

В C++ практически любое состояние, достигнутое в процессе выполнения программы, можно определить как особую ситуацию (исключение) и предусмотреть действия, которые нужно выполнить при ее возникновении. Для реализации механизма обработки исключений введены три ключевых слова: try (контролировать), catch (ловить), throw (выбрасывать).

try позволяет выделить в любом месте кода контролируемый блок.

throw формирует исключение. Исключение - это статический объект. Оператор throw передает управление за пределы контролируемого блока try.

В этом месте находятся один или несколько обработчиков исключений **catch**.

Исключения стоит использовать:
1. Заранее известно, что в данном месте кода и в месте вызова кода (функции) невозможно корректно обработать возникшую ситуацию. Тогда остается послать исключение "куда-повыше", в надежде, что "там" знают, что делать.
2. Если нужно передать информацию о проблемах при каскадном вызове функций с нижнего уровня на верхний.
3. При необходимости уведомить клиента о невозможности корректно создать объект - путем генерировании исключения в конструкторе.

Санкт-Петербургский государственный университет

Ответы

2 Можно ли выбрасывать exception из конструктора? Приведите пример кода.

Да. Более того: механизм исключений одной из своих решаемых задач и являет оповещение о том, что внутри конструктора возникла ситуация, в результате которой объект не может быть создан. Т.к. конструкторы не имеют возвращаемого значения, то механизм исключений остаётся единственно возможным для обработки таких ситуаций.

Если в конструкторе класса кидается исключение то объект не считается созданным и его деструктор не вызывается. Но при раскрутке стека будут вызваны деструкторы полей класса. Но надо помнить про выделение динамической памяти для полей: если их вручную перед выбросом исключения не очистить, то будет утечка памяти.

```
1  class A {
2      private:
3          int* arr;
4      public:
5          A() {
6              arr = new int[10];
7              throw 1; // утечка памяти *arr
8          }
9          ~A() {
10             delete [] arr;
11         }
12 }
```

Санкт-Петербургский государственный университет

Ответы

3 Какой будет результат выполнения кода? Объясните.

```
1 class A {
2     private:
3         int* arr;
4     public:
5         A() {
6             arr = new int[10];
7         }
8         ~A() {
9             delete [] arr;
10        }
11 }
12 class B: A {
13     public:
14         B() {
15             throw 1;
16         }
17 }
```

Утечки памяти нет.

Т.к. конструктор A отработал полностью до throw, то при раскрутке стека его деструктор будет вызван и тоже отработает.

Санкт-Петербургский государственный университет

Ответы

4 Можно ли выбрасывать exception из деструктора?

Нельзя.

Но если очень сильно хочется, то можно.

Бросать исключения из деструктора нельзя потому, что вызов деструктора мог произойти от раскрутки стека выброса исключения в другом месте. И получится ситуация "исключение на исключение". Это приведёт к аварийному завершению программы функцией terminate()

Если есть подозрение, что при выполнении деструктора может произойти исключение, то код деструктора надо обернуть в try + catch и там же в деструкторе его обработать, не выбрасывая наружу.

```
1 ~A() {
2     try {
3         // код, в котором может произойти исключение типа ЧЗИ
4     } catch (const ЧЗИ& e) {
5         // Обработка без аварийного завершения программы.
6         // Программы должна продолжить свой ход, который имела перед
7         // попаданием в этот деструктор.
8     }
9 }
```

Санкт-Петербургский государственный университет

Ответы

5 Что такое exception (std::exception)? Реализуйте свой класс исключение.

Все исключения в языке C++ описываются типом exception, который определен в заголовочном файле <exception>. И при обработке исключений мы также можем использовать данный класс, интерфейс которого выглядит следующим образом:

```cpp
namespace IBusko {
    class LongNumberException: public std::exception {
        private:
            std::string message;
        public:
            LongNumberException(const std::string& message) : message(message) {}
            const char* what() const noexcept override { return message.c_str(); }
    };
}
```

Описывать класс exception не надо.

```cpp
namespace std
{
    class exception
    {
    public:
        exception() noexcept;
        exception(const exception&) noexcept;
        exception& operator = (const exception&) noexcept;
        virtual ~exception(); // Destructor

        // возвращает сообщение об исключении
        virtual const char* what() const noexcept;
    };
}
```

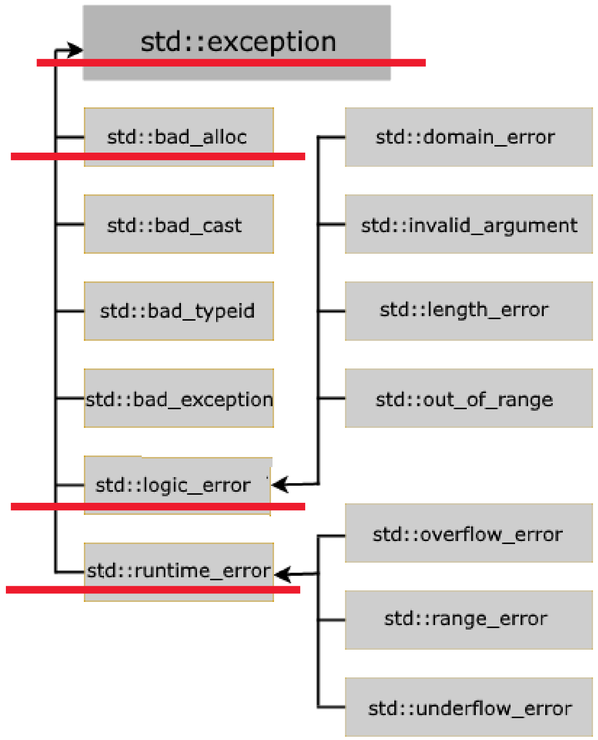
Санкт-Петербургский государственный университет

Ответы

6 Что такое runtime_error (std::runtime_error)? Приведите иерархию основных исключений.

Класс runtime_error определяет тип объекта, который бросается как исключение ошибок времени выполнения программы. Это относится к событиям, выходящих за рамки программы и, которые не могут быть легко предсказаны. Например, деление на 0. Эти ошибки оборачиваются в данный класс.

В этом вопросе надо вспомнить из иерархии отмеченные исключения и написать, что они значат!



runtime_error: общий тип исключений, которые возникают во время выполнения

range_error: исключение, которое возникает, когда полученный результат превосходит допустимый диапазон

overflow_error: исключение, которое возникает, если полученный результат превышает допустимый диапазон

underflow_error: исключение, которое возникает, если полученный в вычислениях результат имеет недопустимое отрицательное значение (выход за нижнюю допустимую границу значений)

logic_error: исключение, которое возникает при наличии логических ошибок к коде программы

domain_error: исключение, которое возникает, если для некоторого значения, передаваемого в функцию, не определен результат

invalid_argument: исключение, которое возникает при передаче в функцию некорректного аргумента

length_error: исключение, которое возникает при попытке создать объект большего размера, чем допустим для данного типа

out_of_range: исключение, которое возникает при попытке доступа к элементам вне допустимого диапазона

Санкт-Петербургский государственный университет

Ответы

7 Как перехватывать исключение деления на 0?

```
1 if (a == 0) {
2     throw MyZeroDivisionException(...);
3 } else {
4     return b / a;
5 }
```

Деление на 0 — это аппаратное прерывание, которое стандартный механизм не перехватывает. В таких ситуациях надо самому генерировать своё исключение через проверку делителя.

Санкт-Петербургский государственный университет

Ответы

8 Что делает ключевое слово noexcept?

В C++11 появилось много нововведений и одно из них — это ключевое слово noexcept. Оно указывает компилятору, что функция не будет выбрасывать исключений (оператором throw). Что приведёт к уменьшению размера итогового файла. Это связано с использованием по умолчанию потока для ошибок std::cerr компилятором для всех функций. До C++11 ряд компаний (не будет тыкать пальцем в Google) из-за этого даже запрещали использовать механизм обработки ошибок.

Но надо помнить, что если функция помеченная noexcept выпустит исключение наружу, то программа вызовет std::terminate() и завершится, не вызвав деструкторы для уже созданных переменных.

9 Что такое паттерны программирования? Приведите примеры.

Паттерны (шаблоны) проектирования — это типичные способы решения часто встречающихся задач при проектировании программ (при конструировании структур классов / отношений между классами).

Singleton.
— Используется для создания логгера, где только один объект должен управлять файлом, в который пишутся сообщения о работе приложения из всех её частей.
— Используется для объекта, который читает файл конфигурации приложения. И, т.о., тоже должен быть единственным, кто имеет доступ к файлу конфигурации, и должен быть доступным в любой части приложения.

Санкт-Петербургский государственный университет

Ответы

10 Для чего нужен Singleton? Преимущества и недостатки.

Singleton — это паттерн программирования, который используется когда надо создать объект единственный на всю программу и доступный в любой части кода.

Примеры применения: логгер работы/ошибок приложения, конфигурация приложения.

Преимущества:
1. Гарантирует наличие только одного экземпляра класса во всём приложении. Это полезно, когда надо обеспечить единую точку контроля в приложении за какой-то функциональностью.
2. Глобальный доступ. Доступ к функциональности из любой части кода и, при этом, без создания глобальных переменных.

Недостатки:
1. Чрезмерное употребления синглтонов ведёт к усложнению логики работы приложения, к увеличению зависимостей частей приложения и теряется такой важный аспект проектирования как слабая связность кода.

Санкт-Петербургский государственный университет

Ответы

11 Что такое перегрузка функций? Какие у неё есть проблемы.

Перегрузка функций — использование нескольких функций с одним и тем же именем, но с различными типами параметров.

Компилятор (статическое связывание) определяет, какую именно функцию требуется вызвать, по типу параметров. Тип возвращаемого значения в разрешении не участвует.

```
1 int max(const int x, const int y);
2 char* max(const char* const x, const char* const y);
3 int max(const int* const arr, const int n);
```

Привести какой-нибудь один пример неоднозначности перегрузки.

При вызове перегруженной функции компилятор выбирает ту функцию, которая наиболее точно соответствует типу фактических параметров. И здесь могут появиться проблемы, связанные с неоднозначностью:

1. Преобразование типа
2. Использование параметров ссылок
3. Использование аргументов по умолчанию

```
1 // Неоднозначность преобразования типа
2 void f(float x) { std::cout << "f(float x)"; }
3 void f(double x) { std::cout << "f(double x)"; }
4
5 float a = 10.1;
6 double b = 5.5;
7 f(a); f(b); f(2);
```

```
1 // Неоднозначность использования параметров ссылок
2 void f(int x) { std::cout << "f(int x)"; }
3 void f(int &x) { std::cout << "f(int &x)"; }
4
5 int a = 5;
6 f(5);
```

```
1 // Неоднозначность аргументов по умолчанию
2 void f(int x) { std::cout << "f(int x)"; }
3 void f(int x, int y = 1) {
4   std::cout << "f(int x, int y = 1)";
5 }
6
7 f(10);
```

Санкт-Петербургский государственный университет

Ответы

12 Что такое шаблоны в C++?

Шаблоны — это механизм параметризации функций и классов, когда их реализация на зависит от типа данных, с которыми производится работа.

В C++ есть шаблоны функций и шаблоны классов.

```
1 template<class|typename T>
2 void sort(T *arr, int n) {...}
3
4 template<class|typename T>
5 class List {...}
```

Между ключевыми словами typename и class разницы нет. Их оба используют для объявления имени типа. Типом может быть и класс, и примитивный тип. Кроме этой ситуации оба этих слова имеют применение и в других ситуациях.

Конкретный тип параметра определяется компилятором автоматически, исходя из типов заданных при вызове функции, либо задаётся явным образом.

```
1 int arr[5] = {5, 4, 3, 2, 1};
2 sort<int>(arr, 5);
3 sort(arr, 5);
4
5 List<int> *l;
6 List<> *l;
```

Санкт-Петербургский государственный университет

Ответы

13 Что такое инстанциация шаблона?

Инстанциация шаблона — это создание компилятором соответствующей версии функции / класса при первом их вызове для конкретных типов данных.

Создание экземпляра функции / класса прост: компилятор копирует шаблон функции / класса и заменяет шаблонный тип (T) фактическим типом, который был указан при первом вызове. Т.о. получается, что написанный код для универсального типа превращается во много реализаций других нужных типов. Важно помнить, что все типы, к которым будет применён шаблон, должны иметь все используемые операции перегруженными.

14 Какие параметры бывают у шаблона?

У шаблонов бывают шаблонные параметры типа и шаблонные параметры, не являющиеся типом.

Шаблонный параметр типа — это тип-заполнитель, который заменяет тип, переданный в качестве аргумента.

Шаблонный параметр, не являющийся типом, — это параметр шаблона, в котором тип параметра предопределен, и он заменяется значением, переданным в качестве аргумента.

```
1 template <typename T, int size>
2 class MyClass {...}
3
4 // Создание экземпляра
5 MyClass<int, 5> myobject;
```

Санкт-Петербургский государственный университет

Ответы

15 Что такое специализация шаблона? Приведите пример использования.

Эффективноть функций и классов шаблонов для различных типов может сильно отличаться. И в этом случае можно предусмотреть специальную реализацию либо полностью переопределить (специализировать) шаблон класса.

Специализация шаблона — это механизм реализации конкретного кода шаблона для конкретного типа его параметра.

Пример использования —

1. объявить класс без описания тела.

2. Написать реализацию метода print для конкретного типа.

```cpp
template <typename T>
class MyClass {
    private:
        T value;
    public:
        MyClass(T value) {
            this.value = value;
        }

        void print() {
            std::cout << value;
        }
};

template <>
void MyClass<double>::print() {
    std::cout << "Специализация print для double";
}

MyClass<double> v(6.7);
```

```cpp
std::string str = "Привет";
MyClass<char*> mystr(str.data());
mystr.print();
str.clear();
mystr.print();

template <>
MyClass<char*>::MyClass(char* value) {
    // Переопределить конструктор
}

template <>
MyClass<char*>::~MyClass() {
    delete[] value;
}
```

Санкт-Петербургский государственный университет

Ответы

16 Расскажите об имплементации шаблонных классов в cpp-файле.

Шаблон — это не класс или функция, это образец, используемый для создания классов или функций. И поэтому стандартное разбиение шаблонов на объявление и определение в .hpp + .cpp работать не будет. Компилятор может использовать шаблон только, если он видит одновременно и объявление, и реализацию и тип.

Объявление класса писать без тела.

```hpp
// myclass.hpp
template <typename T>
class MyClass {
    private:
        T value;
    public:
        MyClass(T value) {
            this.value = value;
        }

        void print();
};
```

```cpp
// myclass.cpp
template <typename T>
void MyClass<T>::print() {
    std::cout << "Реализация print";
}
```

```cpp
// main.cpp
MyClass<int> a(5);
MyClass<double> b(5.5);
a.print();
b.print();
```

Санкт-Петербургский государственный университет

Ответы

16 Расскажите об имплементации шаблонных классов в cpp-файле.

... продолжение

2 решения этой проблемы:
1. Всё включить в .hpp. Реализацию функций написать под классом. У этого решения большой минус: при большом количестве копий шаблона очень сильно увеличится время компиляции и компоновки.
2. Разбить всё не на два файла .hpp + .cpp, а на три: .hpp, .cpp определение шаблона и .cpp определение всех необходимых экземпляров класса.

```cpp
// templates.cpp для классов-шаблонов
#include "myclass.hpp"
#include "myclass.cpp"

template class MyClass<int>;
template class MyClass<double>;
```

Санкт-Петербургский государственный университет

Ответы

17 Что такое структура данных? Какие основные СД можно выделить?

Структура данных — это способ организации данных для более эффективного использования.

Главное свойство СД состоит в том, что у любой единицы данных есть чёткое место, по которому её можно найти. А определение этого места зависит от СД.

Основные СД и их представление в STL:
1. **Массив** (int arr[5])
2. **Динамический массив** (std::vector<T>)
3. **Связный список** (
..... std::list<T> — двусвязный список,
..... std::forward_list<T> — односвязный список

)
4. **Стек** (
..... std::stack<T, container = std::deque<T>>
) // std::vector<T>, std::list<T>
5. **Очередь** (std::deque<T>)
6. **Множество** (std::set<T, compare = std::less<T>>)
std::less<T> — метод проверки уникальности
7. **Карта / словарь** (std::map<P, T, compare = std::less<T>>)
8. **Двоичное дерево посика** (std::set<T, compare = std::less<T>>)

Санкт-Петербургский государственный университет

Ответы

18 Что такое STL и из чего состоит?

STL (Standard Template Library) — это стандартная библиотека шаблонов в C++.
В неё входят шаблоны контейнеров (структуры данных), итераторов и алгоритмов.
Они все находятся в пространстве имён std.

Контейнеры последовательностей (данные находятся в той же последовательности, в которой были записаны):
array, vector, list, forward_list, deque, basic_string

deque по умолчанию реализовывается как динамический массив, т.е. как последовательность элементов.

Но может быть реализован и на основе списка.

Поэтому он в двух местах отмечен.

Ассоциативные контейнеры (данные автоматически сортируются):
set, multiset, map, multimap

Адаптеры контейнеров (контейнеры адаптированные под конкретные цели):
stack, queue, priority_queue

Итераторы реализованы для всех видов контейнеров.

Алгоритмы (#include <algorithm>):
min_element / max_element, find, insert, sort, reverse, swap, ...

Санкт-Петербургский государственный университет

Ответы

19 Статический / динамический массив. Одномерный / двумерный.

Статический массив — это массив, который создаётся при компиляции и находится в стеке.

Динамический массив — это массив, который создаётся во время выполнения программы и находится в куче.

```
// Статический массив
int arr[10];
// или
// Динамический
int *arr;
arr = new int[n];
delete [] arr;
```

Почему индексация массива начинается с 0? (**Ответить**)

Что означает запись 2[arr] и почему? (**Ответить**)

+ статического массива:
1. Скорость доступа (чтение / запись) к произвольному элементу O(1).
2. Просто реализуется, удобен для небольших данных.

− статического массива:
1. Не подходит для больших объёмов данных, т.к. хранится в стеке.
2. Невозможно изменение количества элементов в массиве.
3. Вставка и удаление выполняется за O(n) операций.

Динамический массив подходит для больших данных, т.к. размер кучи может быть намного больше стека.

Многомерный статический и динамический массив.

Каково различие расположения элементов в памяти? (**Ответить**)

Как можно последовательно расположить ВСЕ элементы многомерного динамического массива? И как их потом читать / записывать? (**Ответить**)

```
int matrix[2][2] = {{1, 1}, {1, 1}};

int* *arr = new int*[n];
for (int i = 0; i < n; i++) {
    arr[i] = new int[m];
}
```

Санкт-Петербургский государственный университет

Ответы

20 Что такое std::size_t? Привести пример бесконечного цикла.

std::size_t целочисленный тип без знака, являющийся результатом оператора sizeof.

size_t может хранить максимальный размер теоретически возможного объекта любого типа (включая массивы). std::size_t обычно используется для индексации массивов и счётчиков циклов. Программы, которые используют другие типы, например unsigned int, для индексации массивов, могут неправильно работать на 64-битных системах.

Разрядность std::size_t не меньше 16. Размер size_t совпадает с размером указателя для данной платформы.

```
1 // ВНИМАНИЕ! Какой результат программы?
2 for (std::size_t i = 10; i >= 0; i--) {}
```

Санкт-Петербургский государственный университет

Ответы

21 Какой контейнер является статическим массивом в STL?

В C++11 появился статический массив в виде шаблона std::array<class T, std::size_t n>.

std::array<int, 3> a = {1, 2, 3};

https://ru.cppreference.com/w/cpp/container/array

std::array - это агрегатный тип, т.е. максимально приближен к обычному массиву.

Особенности работы:
1. size() → контейнер можно передавать в функцию, не боясь неявного преобразования к указателю и потери размера.
2. at(size_type pos) возвращает ссылку на элемент по индексу pos. Выполняется проверка выхода индекса за границы диапазона вектора. В случае выхода индекса за пределы диапазона вектора генерирует исключение типа std::out_of_range.
3. Всегда передавайте std::array по ссылке или по константной ссылке.
4. Поскольку длина всегда известна, с std::array работают цикл for-each.
5. std::sort(arr.begin(), arr.end()) — так можно отсортировать массив.
6. Подводный камень контейнера — это его индексация в цикле через переменную индекса: несоответствие типов и размеров индекса и size() (size_type).

```
1 for (unsigned int i = 0; i < arr.size(); i++) {}
2 for (std::size_t i = arr.size(); i >= 0; i--) {}
```

Санкт-Петербургский государственный университет

Ответы

22 Какая иерархия у памяти? Охарактеризуйте. Напишите примерные объёмы и время доступа.

Иерархия памяти компьютера по отношению близости доступа к процессору:

1. Регистры. Находятся в ядре процессора. Являются частью архитектуры. Объём 1 Кб, скорость доступа 1-2 такта.
2. Кэш 1-го уровня. Относится к конкретному ядру. Находится в ЦПУ рядом со своим ядром. Объём 512 Кб, скорость доступа 2-3 такта.
3. Кэш 2-го уровня. Разделяется соседними ядрами. Находится в ЦПУ. Объём 4 Мб. скорость доступа 3-5 тактов.
4. Кэш 3-го уровня. Общий кэш для всех ядер ЦПУ. Находится в ЦПУ. Объём 8 Мб, скорость доступа 30-50 тактов.
5. ОЗУ (Оперативное запоминающее устройство, оперативная память). Находится на отдельной от ЦПУ плате. Объём 16-32 Гб. Скорость доступа 50-200 тактов.
6. Жёсткий диск. Объём > 1 Тб, скорость доступа 50 000 тактов.

23 Что такое виртуальная память? Для решения каких задач она была создана?

Виртуальная память — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем.

Виртуальная память — это модель представления ОЗУ и жёсткого диска в качестве памяти для запускаемой программы.

Программы оперируют с виртуальными адресами данных, которые при необходимости транслируются в физические адреса с помощью MMU.

Причины создания виртуальной памяти:
1. Программист не должен знать, где в памяти будет находиться его программа.
2. Часть работающей программы ввиду нехватки ОЗУ должно иметь возможность быть перемещённым на жесткий диск.
3. Процессы не должны иметь возможности обращения к памяти других процессов.
4. Программист не должен отвечать за физическую организацию памяти и он не может знать, сколько памяти осталось доступно.

Санкт-Петербургский государственный университет

Ответы

24 Как работают inline-функции? Может ли такая функция быть рекурсивной?

Слово inline используется для предложения компилятору вставить код функции непосредственно в точку вызова функции, вместо вызова функции. Это позволяет оптимизировать производительность программы, уменьшая накладные расходы от вызова функции. inline для компилятора является рекомендательным. При этом тело и не inline функции компилятор может вставить вместо её вызова, если оно мало.

В стандарте C слова inline не было. И там для похожих случаев коротких функций могли использовать макросы.

Если функция объявлена как inline, а её подстановка невозможна: рекурсивная функция, тело функции очень большое, - то компилятор поставит вызов функции.

Если реализация методов класса (структуры) написана внутри класса, то такие методы считаются inline.

25 Что такое идиома remove-erase? Написать результат работы remove для {...}

Идиома remove-erase — это оптимизированный алгоритм того, как из последовательности удалить любое количество элементов, удовлетворяющих определённому значению за O(n), а не O(n^2) операций и уменьшение памяти только один раз.

Состоит этот алгоритм из двух фаз:
1. remove. Надо переместить вначало все элементы, которые должны сохраниться. При этом в последовательности должен сохраниться порядок элементов. И сделать это надо за O(n).
2. erase. В хвосте последовательности останется такое количество элементов, которое было удалено ("перезатёрто"). И останется только выделить последовательность меньшего размера памяти и туда записать все элементы до хвоста.

Санкт-Петербургский государственный университет

Ответы

25 Описание алгоритма с примером. В ответ на вопрос не входит. Дополнительно пояснение

Алгоритм.

Идём сначала по последовательности двумя индексами.

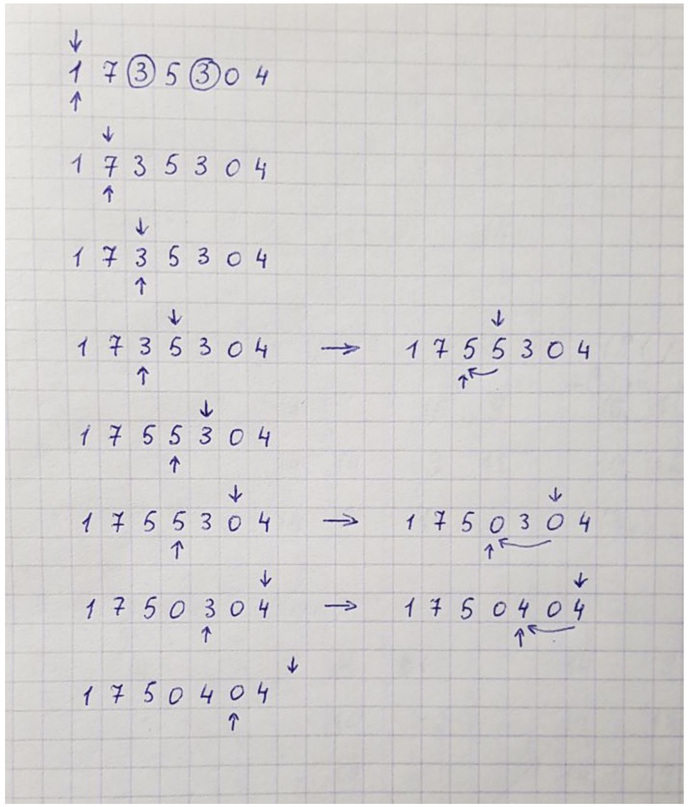
"Верхний" на диаграмме обозначает элемент, значение которого будет сохранено / записано влево. "Нижний" - элемент, на место которого будет записано значение элемента "верхнего" индекса.

Процесс продолжается до тех пор, пока "верхний" индекс не выйдет за пределы последовательности.

На каждом шаге проверяем значение элемента "верхнего" индекса:
1. Если это элемент, который надо удалить, то "верхний" индекс сдвигается на 1 вправо. А "нижний" индекс остаётся на месте.
2. Если это элемент, который должен остаться, то вправо на 1 надо сдвинуть оба индекса. Но перед этим надо проверить: "Если нижний индекс меньше (позади)" верхнего индекса, то на место элемента нижнего индекса надо записать значение элемента верхнего индекса."

В итоге, когда верхний индекс достигнет конца последовательности, то разница между верхний и нижним индексами будет равно количеству удалённых элементов.

Можно посмотреть алгоритм здесь https://www.youtube.com/watch?v=XixADsIJKCo



Санкт-Петербургский государственный университет

Ответы

26 Что такое явное и неявное приведение типов в C++?

Типобезопасная программа — это программа, в которой значения переменных, аргументов функции и возвращаемого значения сохраняют приемлемый тип; операции, включающие значения различных типов, "имеют смысл" и не вызывают потери данных.

Однако иногда требуются небезопасные преобразования, например: double в int, unsigned int в signed int, char* p = "Привет" + 7.

Неявные преобразования — это когда компилятор сам решает каким типом должна быть переменная. Такие ситуации приводят к проблемам неоднозначности, например:
1. При разрешении перегрузки
```
void f(float x) {...}
void f(double x) {...}
f(10);
```
2. При преобразовании любого числового типа и указателя к bool. Это может приводить к трудно обнаруживаемым ошибкам.
3. Преобразования пользовательских родственных типов: повышающие, понижающие, между родственниками.

Всё это приводит и к потере данных, и к непредсказуемым результатам, и к трудновыявляемым ошибкам. Хорошо, если компилятор не разберётся с преобразованием и выкинет ошибку или предупреждение.

Явные преобразования (приведение) из языка C не решают всех проблем, но уменьшают их количество: код становится более читаемым и разработчик всегда видит, где есть потенциальные проблемы; в некоторых случаях компилятор выдаст ошибку компиляции. Синтаксис явных преобразований из языка C:
```
(int) x;
int(x);
```

Санкт-Петербургский государственный университет

Ответы

27 Расскажите о функциях явного приведения типов в C++.

В C++ для большей читабельности кода и частичного контроля преобразования типов ввели операторы: static_cast<тип>(x), dynamic_cast<тип>(x), const_cast<тип>(x), reinterpret_cast<тип>(x).

1. static_cast<тип>(x) используется для преобразования типов на этапе компиляции.

2. dynamic_cast<тип>(x) используется для понижающих преобразований указателя базового класса на производный. Во время выполнения программы производится проверка допустимости преобразования.

3. const_cast<тип>(x) служит для удаления модификатора const.

4. reinterpret_cast<тип>(x) используется для преобразования несвязанных между собой типов, например, указателей в целые или наоборот, указателей типа void* в конкретный тип. При этом внутреннее представление данных остаётся неизменным, а изменяется только точка зрения компилятора на данные.

!!! Результат преобразования всегда будет на совести разработчика.

28 Как выделять и освобождать динамическую память в C? В чём разница с new и delete из C++?

В C динамическую память можно выделить функциями malloc и calloc, а очистить free или realloc.

void *malloc(size_t byte_size); → int* ptr = (int*)malloc(10);

void* calloc(std::size_t count, std::size_t type_size); → int* ptr = (int*)calloc(10, sizeof(int));

Память должна быть освобождена либо void free(void* ptr), либо void* realloc(void* ptr, std::size_t new_size);

В отличие от malloc и calloc new возвращает типизированный указатель, а после выделения памяти вызывается конструктор, который инициализирует память. Если память при new не была выделена, то будет выброшено исключение std::bad_alloc. Результат malloc и calloc надо обязательно проверять на 0.

Санкт-Петербургский государственный университет

Ответы

29 Какая разница между calloc и malloc? Что делает realloc?

void *malloc(size_t byte_size);

void* calloc(std::size_t count, std::size_t type_size)

calloc, в отличие от malloc, возвращает указатель на инициализированный блок памяти, т.е. изначально содержащий нулевые биты. calloc сама занимается вычислением общего размера запрашиваемого блока памяти (перемножением своих аргументов).

void* realloc(void* ptr, std::size_t new_size);

realloc перераспределяет заданную область памяти. Она должна быть предварительно выделена с помощью std::malloc, std::calloc или std::realloc, но не освобождена с помощью std::free. С помощью realloc память может быть как увеличена, так и уменьшена.

realloc возвращает указатель на новый участок памяти. Старый указатель ptr становится не действительным.

Пример использования realloc:

```
1  char* data;
2
3  void resize(std::size_t newSize) {
4      if(newSize == 0) {
5          std::free(data);
6          p = nullptr;
7      } else {
8          if(void* mem = std::realloc(data, newSize)) {
9              data = static_cast<char*>(mem);
10         } else {
11             throw std::bad_alloc();
12         }
13     }
14 }
```

Санкт-Петербургский государственный университет

Ответы

30 Что такое висячий указатель / висячая ссылка? Приведите примеры.

Висячий указатель или висячая ссылка — это указатель, указывающий на область памяти, где ранее хранились данные.

Т.к. ОС может перераспределить ранее освобождённую память (в том числе в другой процесс), то оборванный указатель приводит к неопределённому поведению программы (UB — undefined behavior). Этот вид ошибок очень опасен, и наряду с утечками памяти.

```
1 // Примеры 1.
2 char* f() {
3     char s[100];
4     // Код
5     return s;
6 }
7 char* s = f();
```

```
1 // Пример 2.
2 s = std::malloc(...);
3 // Код
4 std::free(s);
5 // Область видимости s не закончилась
6 // -> могут быть обращения к s.
7
8 // -> Правильный код:
9 s = std::malloc(...);
10 // Код
11 std::free(s);
12 s = nullptr;
13 // Область видимости s продолжается.
```

```
1 // Пример 3.
2 {
3     char* dp = nullptr;
4     // код
5     {
6         char c;
7         dp = &c;
8     } // c выпадает из области видимости
9     // теперь dp является висящим указателем
10    // Код
11 }
```

```
1 // Пример 4.
2 int& f() {
3    int a = 101;
4    int &b = a;
5    return b;
6 }
```

Санкт-Петербургский государственный университет

Ответы

31 Что такое умный указатель? Какие умные указатели есть в стандартной библиотеке?

Все базовые функции выделения и освобождения памяти являются небезопасными. Наиболее часто можно столкнуться со следующими ситуациями:
1. Память была выделена, но не была освобождена (утечка памяти).
2. Память была освобождена, но работа с ней продолжается так, как будто она остаётся выделенной (после free или delete память, на которую указывал указатель считается освобождённой, но указатель при этом не равен nullptr).
3. Память не была выделена, но в неё выполняется запись данных (int* ptr; // ptr имеет значение не nullptr, а произвольный адрес).
4. Попыка несколько раз осовободить одну и ту же память.

В языках java, python, c# для этого разработаны сборщики мусора. В C++11 была создана система умных указателей. Они на примитивном уровне подобны сборщику мусора и сами контролиуют процесс освобождения памяти.

Умный указатель — это класс-шаблон пространства имён std из библиотеки <memory>, который умеет владеть объектом, управлять им через указатель и контролировать особождение выделенной ему памяти.

И, соответственно, разработчику не надо заботиться о вызове delete / delete[] для освобождения памяти.

В <memory> существует 3 умных указателя:
1. unique_ptr — УУ, владеющий данной памятью в единственном числе (на одну память ссылается только один unique_ptr).
2. shared_ptr — УУ, позволяет владеть данной памятью во множественном числе. (на одну память могут ссылаться много shared_ptr).
3. weak_ptr — УУ, являющийся "слабой ссылкой" на объект shared_ptr (временное владение). Другими словами, weak_ptr используется, когда нужен УУ, который имеет доступ к ресурсу, но не считается его владельцем.

С УУ нельзя делать операции адресной арифметики как это было возможно с сырыми указателями.

Санкт-Петербургский государственный университет

Ответы

32 Как работает std::unique_ptr?

unique_ptr — УУ, владеющий данной памятью в единственном числе. (на одну память ссылается только один unique_ptr).

Т.е. ptr_1 = ptr_2 и unique_ptr<int> ptr2 {ptr1}; — ошибка компиляции. Потому, что ОПК и КК у unique_ptr удалены.

unique_ptr полезен, когда нужен указатель на объект, на который НЕ будет других указателей и который будет удален после удаления указателя.

По умолчанию unique_ptr инициализируется nullptr в отличие от сырых указателей: std::unique_ptr<int> ptr;

Выделить память и создать в ней объект можно с помощью функции std::make_unique<int>(1):
std::unique_ptr<int> ptr {std::make_unique<int>(1)};
std::unique_ptr<int> ptr = std::make_unique<int>(1);
std::cout << *ptr1 << std::endl;

unique_ptr могут работать и с массивами: std::unique_ptr<int[]> arr = std::make_unique<int[]>(10);

33 Что произойдёт? Приведите пример с сырыми указателями и объясните.

```
1 std::unique_ptr<int> ptr;
2 ptr = std::make_unique<int>(10);
3 ptr = std::make_unique<int>(11);
```

ОПК в unique_ptr удалён и поэтому ptr_1 = ptr_2 вызовет ошибку компиляции, а в случае ОПП строк 2 и 3 всё будет хорошо: память из-под текущего значения освободится, и будет записано новое значение для указателя.

```
1 int* iptr;
2 iptr = new int(10);
3 iptr = new int(11); // утечка памяти, где лежит 10.
```

Умность указателя заключается в том, что он контролирует ранее выделенную область памяти и когда надо, то сам её освобождает. А освобождение памяти сырым указателям должен контролировать разработчик.

Санкт-Петербургский государственный университет

Ответы

34 Объясните работу кода.

Нет в мире таких умных указателей, которые нельзя было бы поломать, вооружившись хорошо теорией.

```
1 int* p = new int(10);
2 std::unique_ptr<int> ptr_1 {p};
3 std::unique_ptr<int> ptr_2 {p};
```

В данном случае оба умных указателя будут указывать на одну и ту же память.

Важно понимать, что std::unique_ptr<int> ptr {std::make_unique<int>(1)}; — это конструктор перемещения, а в коде в строках 2 и 3 написан конструктор с параметрами (сырого указателя). И в данной ситуации просто копируется значение указателя и вся ответственность за не/единственное владение лежит на разработчике.

И поэтому после освобождения одного из УУ произойдёт через какое-то время опять освобождение этой же памяти, что может привести к неопределённому поведению UB (undefined behavior).

35 Как работает std::shared_ptr?

shared_ptr — УУ, позволяющий владеть данной памятью во множественном числе. (на одну память могут ссылкать много shared_ptr).

Для данных указателей применяется механизм подсчета ссылок. Каждый раз, когда создается объект shared_ptr<T>, увеличивается счетчик объектов shared_ptr<T>, которые содержат определенный адрес. Когда объект shared_ptr<T> удаляется или ему присваивается другой адрес, счетчик ссылок уменьшается на единицу. Когда больше нет объектов shared_ptr<T>, которые ссылаются на определенный адрес, счетчик ссылок сбрасывается в ноль.

Синтаксис инициализации аналогичный unique_ptr:
std::shared_ptr<int> ptr1 {std::make_shared<int>(22)};
std::shared_ptr<int> ptr2 = ptr1; — а здесь КК и ОПК работают.

shared_ptr нельзя проинициализировать через unique_ptr:
std::unique_ptr<int> ptr1 { std::make_unique<int>(1) };
std::shared_ptr<int> ptr2 { ptr1 }; // ошибка компиляции
ptr2 = ptr1; // ошибка компиляции

Санкт-Петербургский государственный университет

Ответы

36 Что такое std::weak_ptr и концепция слабых ссылок?

weak_ptr — УУ, являющийся "слабой ссылкой" на объект shared_ptr (временное владение). Другими словами, weak_ptr используется, когда нужен УУ, который имеет доступ к ресурсу, но не считается его владельцем.

У weak_ptr объект доступен только до тех пор, пока он существует (т.е. пока не была освобождена "сильная ссылка"). Пример использования:

В ответе на вопрос код писать не надо.

```cpp
std::weak_ptr<int> gw;
void observe() {
    std::cout << "gw.use_count() == " << gw.use_count() << "; ";
    if (std::shared_ptr<int> spt = gw.lock()) {
        std::cout << "*spt == " << *spt << '\n';
    } else {
        std::cout << "gw истёк\n";
    }
}
int main() {
    {
        std::shared_ptr<int> sp = std::make_shared<int>(42);
        gw = sp;
        observe();
    }
    observe();
}
// Вывод программы:
// gw.use_count() == 1; *spt == 42
// gw.use_count() == 0; gw истёк
```

gw.lock() проверяет, что "сильная ссылка" не освобождена и если так, то возвращает копию shared_ptr.

Санкт-Петербургский государственный университет

Ответы

36 Что такое std::weak_ptr и концепция слабых ссылок?

продолжение

Концепция слабых ссылок — это особый вид ссылки на объект из динамической памяти, связь которых с объектом, на который они ссылаются, не учитывается сборщиком мусора.

В каких задачах они используются:
1. Циклические ссылки
2. Кэширование
3. Другие

В ответе на вопрос код писать не надо.

Пример циклических ссылок и утечки памяти, т.к. деструкторы объектов вызваны не будут →

```cpp
class Person {
        std::string name;
        std::shared_ptr<Person> myfriend;
    public:
        Person(const std::string &name) : name(name) {
            std::cout << "Конструктор " << name << std::endl;
        }
        ~Person() {
            std::cout << "Деструктор " << name << std::endl;
        }
        friend void set_friends(
          std::shared_ptr<Person> &p1, std::shared_ptr<Person> &p2
        ) {
            p1->myfriend = p2;
            p2->myfriend = p1;
        }
};

int main() {
    std::shared_ptr<Person> masha { std::make_shared<Person>("Маша") };
    std::shared_ptr<Person> misha { std::make_shared<Person>("Миша") };
    set_friends(masha, misha);
    return 0;
}
```

Санкт-Петербургский государственный университет

Ответы

37 Реализовать класс-шаблон, необходимый для выполнения кода.

Код

```cpp
int main() {
    Pair<int> p{ 5, 8 };
    std::cout
        << "Первое значение пары: " << p.get_first()
        << ' '
        << "Второе значение пары: " << p.get_second()
        << std::endl;
}
```

Ответ

```cpp
template<typename T>
class Pair {
    private:
        T first;
        T second;
    public:
        Pair(const T& f, const T& s) : first(f), second(s) {}
        T get_first() const { return first; }
        T get_second() const { return second; }
};
```

Санкт-Петербургский государственный университет

Ответы

38 Напишите класс дроби, выбрасывающий исключение при делении на 0. Исправьте main.

Код

```cpp
int main() {
    Fraction f{ 1, 0 };
    std::cout << f.get_division_result();
}
```

Ответ

```cpp
class Fraction {
    private:
        double numerator;
        double denominator;
    public:
        Fraction(const double n, const double d)
            : numerator(n), denominator(d) {
            if (denominator == 0) {
                throw std::runtime_error("Деление на ноль");
            }
        }
        double get_division_result() const { return numerator / denominator; }
};

int main() {
    try {
        Fraction f{ 1, 0 };
        std::cout << f.get_division_result();
    } catch(std::runtime_error e) {
        std::cout << e.what();
    }
}
```

Санкт-Петербургский государственный университет

Ответы

39 Напишите правильный обработчик исключений функции run_game().

Код

Функция run_game() выбрасывает исключения: int, double, const char* const, std::runtime_error, std::bad_alloc, std::exception.

```
int main() {
    try {
        run_game();
    } catch (std::exception e) {
        // ...
    } catch (double e) {
        // ...
    } catch (char e) {
        // ...
    }
}
```

Ответ

```
int main() {
    try {
        run_game();
    } catch (int e) {
        // ...
    } catch (double e) {
        // ...
    } catch (const char* const e) {
        // ...
    } catch (std::bad_alloc e) {
        // ...
    } catch (std::runtime_error e) {
        // ...
    } catch (std::exception e) {
        // ...
    }
}
```

Санкт-Петербургский государственный университет

Ответы

40 Преобразуйте перегруженные функции в шаблоны.

Код

```
int max(int a, int b) {
    return a >= b ? a : b;
}

double max(double a, double b) {
    return a >= b ? a : b;
}

LongNumber max(LongNumber a, LongNumber b) {
    return a >= b ? a : b;
}

char* max(char* a, char* b) {
    // Написать реализацию
}
```

Ответ

```
template<typename T>
T max(T a, T b) {
    return a >= b ? a : b;
}

template<>
char* max<char*>(char* a, char* b) {
    const char end_str = '\0';
    std::size_t i = 0;
    while (a[i] != end_str && b[i] != end_str) {
        i++;
    }
    return a[i] == end_str ? b : a;
}
```

Санкт-Петербургский государственный университет

Ответы

41 Объясните зачем нужен const_cast<T> и какой и почему будет результат?

Код

```
1 char* func(const char* str) {
2     str++;
3     return const_cast<char*>(str);
4 }
5
6 int main() {
7     const char* hello = "Hello world!";
8     std::cout << func(hello);
9 }
```

Ответ

Результат: ello world!

В func передаётся копия указателя на неизменяемые данные. Но сам указатель изменяем. Оператор ++ сместит значение указателя на один элемент вправо, т.е. на букву e. И вернёт копию указателя, который указывает на строку "Hello world!", но начиная с буквы e.

Функция func принимает копию указателя на неизменяемые данные, а должна вернуть копию указателя на изменяемые данные. Если просто вернуть str, то будет ошибка компиляции. Но можно воспользоваться операцией явного приведения из C++ константных данных к неконстантным const_cast<T>.

Санкт-Петербургский государственный университет

Практика. Ответы

1 Напишите Singleton.

Реализовывать конструктор и деструкторы не надо, т.к. это зависит от того для какого объекта пишется Singleton.

```cpp
class Singleton {
    private:
        static Singleton* instance;

        Singleton();
        ~Singleton();
    public:
        static Singleton& get_instance() {
            if (instance == nullptr) {
                instance = new Singleton();
            }
            return *instance;
        };
        Singleton(const Singleton&) = delete;
        Singleton& operator = (const Singleton&) = delete;
};

Singleton* Singleton::instance = nullptr;
```

Санкт-Петербургский государственный университет

Практика. Ответы

2 Напишите контейнер статический массив с КсПа(*начало, *конец).

```cpp
#include <stdexcept>

template <typename T, int size>
class StaticArray {
    private:
        T arr[size];
    public:
        StaticArray(const T* beg, const T* end) {
            if (end - beg > size) {
                throw std::out_of_range("Выход за пределы массива.");
            }
            int i = 0;
            while (beg != end) {
                arr[i] = *beg++;
                i++;
            }
            while (i < size) {
                arr[i] = {};
                i++;
            }
        }

};
```

Санкт-Петербургский государственный университет

Практика. Ответы

3 Напишите алгоритм "remove_if" для массива из идиомы remove-erase.

```cpp
std::size_t remove_if(
    int* arr,
    const std::size_t n,
    const int remove_value
) {
    std::size_t from = 0;
    std::size_t to = 0;

    while (from < n) {
        if (arr[from] != remove_value) {
            if (to < from) {
                arr[to] = arr[from];
            }
            from++;
            to++;
        } else {
            from++;
        }
    }

    return to;
}
```

Санкт-Петербургский государственный университет

Практика. Ответы

4 Напишие 2 варианта умножение матриц: обычный и дружелюбный к кэшу. Объясните.

```cpp
// Размерность a[l][m], b[m][n], c[l][n].
for (std::size_t i = 0; i < l; i++) {
    for (std::size_t j = 0; j < n; j++) {
        int cij = 0;
        for (std::size_t k = 0; k < m; k++) {
            cij += a[i][k] * b[k][j];
        }
        c[i][j] = cij;
    }
}
```

В коде есть одна неэффективность, это выражение B[k][j] в самом внутреннем цикле. Мы обходим матрицу B по столбцам.

```cpp
// Размерность a[l][m], b[m][n], c[l][n].
for (std::size_t i = 0; i < l; i++) {
    for (std::size_t k = 0; k < m; k++) {
        int aik = a[i][k];
        for (std::size_t j = 0; j < n; j++) {
            c[i][j] += aik * b[k][j];
        }
    }
}
```

Мы не вычисляем каждый элемент матрицы с за раз. Мы вычисляем элементы частично на каждой итерации. Главное — это то, что во внутреннем цикле мы обходим обе матрицы построчно.

Санкт-Петербургский государственный университет

Практика. Ответы

5 Напишите класс, у которого будет утечка памяти из-за циклического std::shared_ptr<T>.

```cpp
class Person {
        std::string name;
        std::shared_ptr<Person> myfriend;
    public:
        Person(const std::string &name) : name(name) {
            std::cout << "Конструктор " << name << std::endl;
        }
        ~Person() {
            std::cout << "Деструктор " << name << std::endl;
        }
        friend void set_friends(
          std::shared_ptr<Person> &p1, std::shared_ptr<Person> &p2
        ) {
            p1->myfriend = p2;
            p2->myfriend = p1;
        }
};

int main() {
    std::shared_ptr<Person> masha { std::make_shared<Person>("Маша") };
    std::shared_ptr<Person> misha { std::make_shared<Person>("Миша") };
    set_friends(masha, misha);
    return 0;
}
```

Санкт-Петербургский государственный университет

Практика. Ответы

6 Напишите удаление элемента из двухсвязного списка.

```
Node* current = begin;
while (current) {
    if (current->value == value) {
        if (current == begin) {
            begin = begin->next;
            if (begin) {
                begin->prev = nullptr;
            }
        } else if (current == end) {
            end = end->prev;
            end->next = nullptr;
        } else {
            current->prev->next = current->next;
            current->next->prev = current->prev;
        }
        delete current;
        return;
    }
    current = current->next;
}
```