Санкт-Петербургский государственный университет

Вопросы

1. Что такое перегрузка функций? Какие у неё есть проблемы.
2. Что такое шаблоны в C++?
3. Что такое инстанциация шаблона?
4. Какие параметры бывают у шаблона?
5. Что такое специализация шаблона? Приведите пример использования.
6. Расскажите об имплементации шаблонных классов в cpp-файле.
7. Что такое структура данных? Какие основные СД можно выделить?
8. Что такое STL и из чего состоит?
9. Статический / динамический массив. Одномерный / двумерный.
10. Что такое std::size_t?

Санкт-Петербургский государственный университет

Вопросы

11. Какой контейнер является статическим массивом в STL?
12. Приведите пример реализации контейнера статического массива.

Санкт-Петербургский государственный университет

Ответы

1 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);
```

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

2 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

3 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.

4 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

5 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.

```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

6 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.

```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

6 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

7 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

8 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

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

9 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

10 Что такое std::size_t?

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

11 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

12 Приведите пример реализации контейнера статического массива.

```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++;
            }
        }

        int get_size() const noexcept { return size; }
        T& operator [] (const int i) { return arr[i]; }
};
```

```cpp
int main() {
    const int data[4] = {1, 2, 3, 4};

    try {
        {
            int tmp[] = {5, 5, 5, 5, 5};
        }
        StaticArray<int, 5> arr(data, data + sizeof(data) / sizeof(data[0]));
        for (int i = 0; i < arr.get_size(); i++) {
            std::cout << arr[i] << ' ';
        }
    } catch(const std::out_of_range& e) {
        std::cerr << e.what() << std::endl;
    }

    return 0;
}
```

Санкт-Петербургский государственный университет

Ответы

12 Приведите пример реализации контейнера статического массива.

... продолжение

```cpp
StaticArray(std::initializer_list<T> arr) {
    if (arr.size() > size) {
        throw std::out_of_range("Выход за пределы массива");
    }
    std::copy(arr.begin(), arr.end(), this->arr);
    if (arr.size() < size) {
        for (int i = arr.size(); i < size; i++) {
            this->arr[i] = {};
        }
    }
}
```

```cpp
int main() {
    try {
        StaticArray<int, 5> arr = {1, 2, 3, 4};
        for (int i = 0; i < arr.get_size(); i++) {
            std::cout << arr[i] << ' ';
        }
    } catch(const std::out_of_range& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```