|  |  |
| --- | --- |
|  | Министерство науки и высшего образования Российской Федерации  Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»  (МГТУ им. Н.Э. Баумана) |
| ФАКУЛЬТЕТ «ИНЖЕНЕРНЫЙ БИЗНЕС И МЕНЕДЖМЕНТ»  КАФЕДРА «ПРОМЫШЛЕННАЯ ЛОГИСТИКА» (ИБМ-3)  Домашнее задание  «Парадигмы и конструкции языков программирования»  38.03.05 Бизнес-информатика  Новые функции в С++  Студенты ИБМ3-34Б Д.Ю. Барабаш  (Подпись, дата)    Руководитель В.И. Артемьев  (Подпись, дата)  2024 г. | |

**Содержание:**

[Введение 3](#_Toc183874814)

[Глава 1 4](#_Toc183874815)

[1.1 Особенности С++ 4](#_Toc183874816)

[1.2 Новые функции С++ 7](#_Toc183874817)

[Заключение 12](#_Toc183874818)

[Список использованных источников 13](#_Toc183874819)

# **Введение**

C++ — один из самых популярных и универсальных языков программирования, разработанный Бьёрном Страуструпом в 1980-х годах как расширение языка C. Основная идея C++ заключалась в добавлении объектно-ориентированных возможностей к уже существующей мощной и эффективной структуре C. На протяжении десятилетий язык продолжал эволюционировать, оставаясь востребованным в различных областях: от системного и прикладного программирования до разработки игр и высокопроизводительных приложений.

C++ сочетает в себе низкоуровневую эффективность и высокоуровневые абстракции, что делает его подходящим для создания как операционных систем и драйверов, так и сложных программных комплексов. Благодаря поддержке различных парадигм программирования, C++ предоставляет возможности для процедурного, объектно-ориентированного и даже функционального стилей.

Главной причиной популярности C++ является его универсальность. Язык предоставляет возможности для работы на низком уровне — управляя памятью и аппаратными ресурсами, что делает его подходящим для разработки системного программного обеспечения, например, операционных систем и драйверов устройств. В то же время он поддерживает высокоуровневые абстракции, что позволяет использовать его для создания сложных приложений с множеством слоев логики.

Благодаря своей гибкости и производительности, C++ часто используется в критически важных областях:

* Системное программирование: C++ подходит для разработки операционных систем, компиляторов и драйверов.
* Игровая индустрия: Мощные игровые движки, такие как Unreal Engine, написаны на C++, что позволяет разрабатывать высокопроизводительные игры.
* Финансовые приложения: В сфере финансов используется для создания торговых платформ и высокочастотных алгоритмических стратегий.
* Встроенные системы: C++ широко используется в разработке программного обеспечения для устройств с ограниченными ресурсами

# **Глава 1**

* 1. **Особенности С++**

C++ — это язык, который заслужил свою популярность благодаря сочетанию гибкости и производительности. Он одинаково хорошо подходит как для низкоуровневого программирования, где требуется полный контроль над памятью и ресурсами, так и для высокоуровневой разработки сложных приложений с использованием мощных абстракций. Это делает его уникальным инструментом, способным адаптироваться под самые разные задачи и сферы применения.

На протяжении десятилетий язык претерпел множество изменений и улучшений. Каждый новый стандарт не только добавлял полезные возможности, но и упрощал жизнь разработчикам, повышая удобство работы с кодом. Чтобы лучше понять, что делает C++ столь универсальным и эффективным, рассмотрим его ключевые особенности, благодаря которым он продолжает оставаться одним из лидеров среди языков программирования.

1. Условные операторы

Большая часть разработчиков, работающих с С++, знают и используют условные операторы.

x = (y < 0) ? 10 : 20;

Но не все программисты понимают, что их также можно использовать в качестве передаваемого значения:

(a == 0 ? a : b) = 1;

Что является сокращением следующего:

if (a == 0) a = 1; else b = 1;

2. URI в С++

У вас есть возможность поместить URI в исходники C++ без получения ошибок.

Например:

void foo() {

http://stackoverflow.com/

int bar = 4;

...

}

3. Арифметика указателей

Использование указателей при написании кода чревато появлением огромного количества ошибок. По этой причине программисты на С++ стараются их избегать. Альтернативой могут послужить числовые литералы.

4. Псевдоним для namespace alias

До некоторого времени это не было известно никому, кроме тех, кто читал дополнительную документацию. Теперь информацию об этом можно найти в каждой ссылке по С++.

namespace fs = boost::filesystem;

fs::path myPath( strPath, fs::native );

5. Объявление классов и функций

В init части цикла for можно объявлять не одни лишь переменные. Здесь также можно разместить классы и функции.

for(struct { int a; float b; } loop = { 1, 2 }; ...; ...) { ... }

Благодаря этому можно использовать множество переменных с разными типами.

6. Ассоциативность оператора массива

A[8] - это то же самое, что и \*(A + 8). Так как операция сложения ассоциативна, она может быть переписана как \*(8 + A), что синонимично ..... 8[A]  
Никто не говорил, что это должно быть обязательно полезно... :-)

7. Объединения тоже могут быть шаблонами

Еще одна вещь, о которой знают немногие, это то, что объединения тоже могут быть шаблонами:

template

union union\_cast {

From from;

To to;

union\_cast(From from)

:from(from) { }

To getTo() const { return to; }

};

8. Метапрограммирование

C++ - это язык программирования, в котором существует несколько парадигм. Они позволяют расширить функционал языка. Так, например, в нем существует метапрограммирование шаблонов. Вероятно, никто не ожидал, что оно позволит стать языку Тьюринг-полным подъязыком, способным исполняться во время компиляции.

9. Унарный оператор +

Многие не подозревают о возможностях унарного оператора +, функциональность которого недоступна при разработке на С. Он может быть использован во многих случаях:

Перевод Enumeration в integer  
+AnEnumeratorValue

И значение вашего перечислителя, которое имело тип перечисления сейчас стало типом integer с соответствующим ему значением. Это пригодится в случае, если вы захотите реализовать перегруженный оператор для своего перечисления.

Получение значения переменной  
Вам нужно использовать класс, который использует встроенный статический инициализатор без внешнего объявления, но сделать это не удается? С помощью оператора можно создать временный класс, который не будет зависим от типа

struct Foo {

static int const value = 42;

};

// Пытается делать что-то интересное...

template

void f(T const&);

int main() {

// не получилось связать- пробуем получить адрес

f(Foo::value);

// работает - передает временное значение

f(+Foo::value);

}

Разложение массива на указатели  
Вам необходимо передать два указателя какой-либо функции, однако вы наперед знаете, что ничего из этого не выйдет? В этом случае вам очень пригодится оператор:

// Это делает кое-что интересное...

template void f(T const& a, T const& b);

int main() {

int a[2];

int b[3];

f(a, b); // не работает! разные значения для "T"!

f(+a, +b); // работает! T это "int\*" в обоих случаях }

* 1. **Новые функции С++**

За десятилетия своего существования C++ постоянно эволюционировал, адаптируясь к новым требованиям разработчиков и технологиям. С каждым новым стандартом язык становился не только более мощным, но и удобным для повседневной разработки. Нововведения направлены на повышение читаемости кода, упрощение сложных конструкций и улучшение производительности.

Современные версии C++ — это не просто обновления, а серьезные шаги вперед, которые позволяют создавать более безопасные, гибкие и эффективные приложения. Рассмотрим ключевые функции и изменения, которые появились в последних стандартах C++ и значительно расширили возможности языка.

**Лямбда-функции**

Лямбда-функции позволяют создавать анонимные функции прямо в месте вызова, что делает код компактным и удобным.

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> numbers = {1, 2, 3, 4, 5};

std::for\_each(numbers.begin(), numbers.end(), [](int n) {

std::cout << n \* 2 << " ";

});

return 0;

}

Результат:

2 4 6 8 10

**Умные указатели (std::shared\_ptr, std::unique\_ptr)**

Умные указатели автоматически управляют памятью, освобождая ресурсы, когда они больше не нужны.

#include <iostream>

#include <memory>

class Example {

public:

Example() { std::cout << "Constructor\n"; }

~Example() { std::cout << "Destructor\n"; }

};

int main() {

std::unique\_ptr<Example> ptr = std::make\_unique<Example>();

return 0;

}

Результат:

Constructor

Destructor

**Автоматическое определение типов (auto)**

Компилятор автоматически выводит тип переменной на основе инициализатора.

#include <iostream>

int main() {

auto x = 42; // int

auto y = 3.14; // double

auto z = "Hello"; // const char\*

std::cout << x << ", " << y << ", " << z << std::endl;

return 0;

}

**Корутины (Coroutines)**

Позволяют создавать функции, которые можно приостанавливать и возобновлять.

#include <iostream>

#include <coroutine>

struct Generator {

struct promise\_type {

int current\_value;

auto get\_return\_object() { return Generator{this}; }

auto initial\_suspend() { return std::suspend\_always{}; }

auto final\_suspend() noexcept { return std::suspend\_always{}; }

auto yield\_value(int value) {

current\_value = value;

return std::suspend\_always{};

}

void return\_void() {}

void unhandled\_exception() { std::terminate(); }

};

struct iterator {

std::coroutine\_handle<promise\_type> handle;

bool operator!=(std::default\_sentinel\_t) const { return !handle.done(); }

void operator++() { handle.resume(); }

int operator\*() const { return handle.promise().current\_value; }

};

std::coroutine\_handle<promise\_type> handle;

Generator(promise\_type\* p) : handle(std::coroutine\_handle<promise\_type>::from\_promise(\*p)) {}

iterator begin() { handle.resume(); return iterator{handle}; }

std::default\_sentinel\_t end() { return {}; }

};

Generator counter(int max) {

for (int i = 1; i <= max; ++i) {

co\_yield i;

}

}

int main() {

for (int value : counter(5)) {

std::cout << value << " ";

}

return 0;

}

**Концепции (Concepts)**

Позволяют накладывать ограничения на шаблоны, делая код более понятным и безопасным.

#include <iostream>

#include <concepts>

template <typename T>

concept Integral = std::is\_integral<T>::value;

template <Integral T>

T add(T a, T b) {

return a + b;

}

int main() {

std::cout << add(3, 5) << std::endl; // Работает

// std::cout << add(3.14, 5.0) << std::endl; // Ошибка компиляции

return 0;

}

**Структурные привязки**

Позволяют распаковывать элементы структур и кортежей в отдельные переменные.

#include <iostream>

#include <tuple>

int main() {

std::tuple<int, double, std::string> data(1, 3.14, "C++17");

auto [id, value, text] = data; // Распаковка кортежа

std::cout << id << ", " << value << ", " << text << std::endl;

return 0;

}

Эволюция C++ — это не просто история обновлений, а последовательное движение к более выразительному, удобному и безопасному программированию. Новые функции, такие как лямбда-выражения, умные указатели, концепции и корутины, дают разработчикам мощные инструменты для решения самых сложных задач. При этом C++ сохраняет свою главную силу — высокую производительность и гибкость.

Каждое нововведение не только облегчает написание кода, но и открывает новые горизонты для творчества. C++, оставаясь верным своим корням, продолжает развиваться, отвечая на вызовы современного мира программирования. В этом и заключается его уникальность: он остается актуальным инструментом для создания технологий будущего.

4o

# **Заключение**

В ходе исследования языка программирования C++ было рассмотрено его значение, особенности и ключевые нововведения, которые сделали его одним из самых популярных и мощных инструментов в мире разработки. Язык сочетает в себе гибкость, высокую производительность и поддержку различных парадигм программирования, что позволяет решать широкий спектр задач — от низкоуровневого программирования до разработки сложных прикладных приложений.

C++ продолжает развиваться, и новые стандарты, значительно улучшили возможности языка. Лямбда-функции, умные указатели, концепции и корутины — все эти нововведения сделали код более чистым, безопасным и удобным для разработки. В то же время C++ продолжает сохранять свою высокую производительность и контроль над ресурсами, что особенно важно для системного программирования и приложений с высокой нагрузкой.

Таким образом, язык C++ остается важным инструментом для создания программных решений в самых различных областях, и его развитие в будущем будет продолжать открывать новые возможности для программистов. C++ доказал свою актуальность и мощь, оставаясь на переднем крае технологий и обеспечивая разработчиков всем необходимым для реализации сложных и высокопроизводительных проектов.

# **Список использованных источников**

1. 10 особеностей С++, о которых знают лишь продвинутые программисты [Электронный ресурс] // Режим доступа: <https://proglib.io/p/hiddencpp> // (Дата обращения: 30.11.24)
2. Современные возможности С++, о которых надо знать всем програмистам [Электронный ресурс] // Режим доступа: <https://habr.com/ru/companies/ruvds/articles/451870/> // (Дата обращения: 15.05.24)
3. 21 фича современного С++ [Электронный ресурс] // Режим доступа: https://habr.com/ru/companies/otus/articles/741428/ (Дата обращения: 30.11.24)