

Программирование на современном C++

Память в C++. Работа с файлами и строками



образование

Отметьтесь на портале!

- Посещение необязательное, но тем, кто пришёл, следует отмечаться на портале в начале каждого занятия
- Это позволяет нам анализировать, какие занятия были более или менее интересны студентам, и менять курс в лучшую сторону
- Также это даст возможность вам оставить обратную связь по занятию после его завершения


| пн, 1 ноября | вт, 2 ноября | ср, 3 ноября | чт, 4 ноября | пт, 5 ноября | сб, 6 ноября |
|--------------|--|--------------|--------------|--------------|--------------|
| Нет занятий | <div>18:00 Углубленный C/C++ (w... п</div> <div>Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование</div> <div>А. Халайджи</div> <div>18:00 Углубленный C/C++ (ML) п</div> <div>Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование</div> <div>А. Халайджи</div> | Нет занятий | Нет занятий | Нет занятий | Нет занятий |

Ссылка на Zoom РК сегодня в 18:00

Безопасность интернет-приложений (третий семестр)

Подключиться к конференции Zoom
<https://mailru.zoom.us/j/98586081818?pwd=eWxwSDdCbIhQNnNwQU5iblFvU2dTZz09>

Идентификатор конференции: 985 8608 1818
Код доступа: 785885

 Алексей Набережный 55 минут назад

★ 0 💬 0 ↓ 0 ↑

Запись прошлой лекции (+ что почитать перед РК)

Безопасность интернет-приложений (третий семестр)

Углублённый C/C++

Лекция 5

📍 Онлайн - ML

Отметьтесь, что вы пришли на занятие. Так вы улучшите свою посещаемость и вас увидит преподаватель в своём "Журнале посещений".

Отметиться

Оставьте отзыв о занятии и мы сможем улучшить учебный процесс.

Оставить отзыв

План лекции

- Компьютерная память
- Указатели и ссылки
- Перерыв
- Длительность хранения и связывание
- Встроенные функции и переменные
- Работа с файлами и строками

Компьютерная память

Что такое память?

- Память — часть компьютера, где хранятся программы и данные;
- Память, доступная программе на C++, состоит из одной или нескольких последовательностей байтов;
- Каждый байт имеет уникальный адрес;
- Чаще всего байт состоит из 8 бит;
- Байт - минимальная **адресуемая** единица памяти.

Виртуальная память

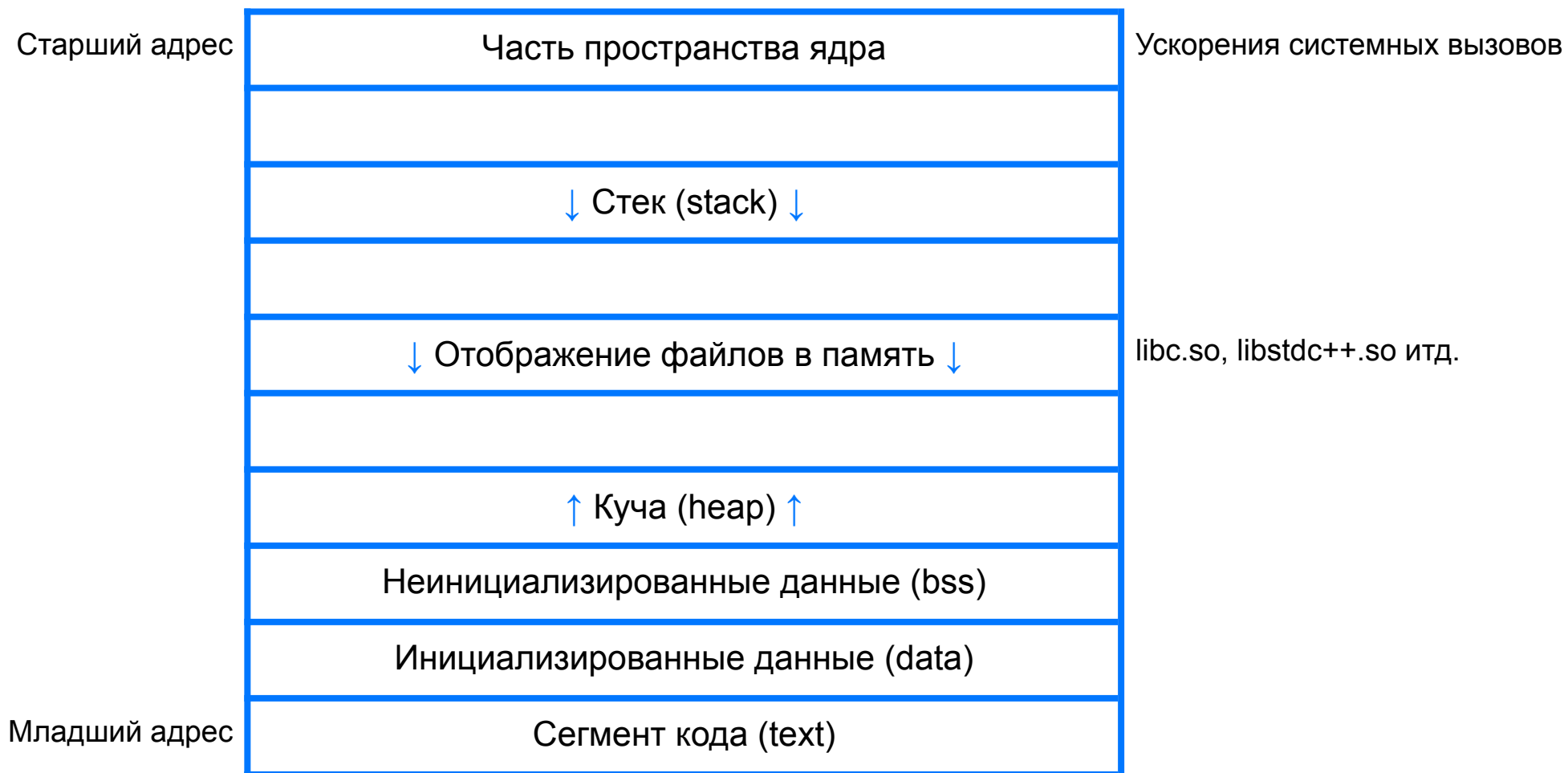
Виртуальная память - метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере.

- Программы оперируют виртуальными адресами, которые транслируются в физические адреса программно-аппаратными средствами компьютера (ОС + MMU)
- Виртуальная память оперируется фиксированными блоками, называемые страницами (4КБ)
- Операционная система может перемещать содержимое сегментов виртуальной памяти (страницы) на вторичные запоминающие устройства (например, диск), позволяя адресовать и использовать больше виртуальной памяти, чем есть физической.
- Избавляет программиста от необходимости согласовывать использование памяти с другими программами
- Позволяет разделять содержимое памяти между несколькими процессами. Например, динамические библиотеки, используемые разными процессами физически расположены в одном месте. (libstdc++)

Типичное размещение сегментов в памяти

- Сегмент кода (**text**), машинные инструкции, которые выполняются центральным процессором;
- Сегмент инициализированных данных (**data**), содержит переменные, которые инициализированы определёнными значениями в тексте программы;
- Сегмент неинициализированных данных (**bss**), сегмент неинициализированных данных;
- Сегмент стека (**stack**), где хранятся переменные с автоматическим классом размещения, а также информация, которая сохраняется при каждом вызове функции;
- Куча (**heap**), или область динамической памяти

Размещение сегментов в памяти



Стек

- Непрерывная область памяти;
- Имеет методы: push и pop;
- Растёт от старших адресов к младшим;
- Имеет конечный размер (обычно 8 MB);
- Содержит локальные переменные из функций;
- Регистр RSP (i386: ESP, ARM: SP, Stack Pointer) указывает адрес вершины стека.
Меняется каждый раз, когда слово или адрес помещаются или удаляются из стека;
- Когда вызывается функция, то создаётся stack frame;
- Stack frame содержит локальные аргументы функции и возвращаемое значение;

Куча (heap)

Область памяти в которой создаются объекты с динамическим временем хранения.

- `new` - пытается выделить память и создать объект или массив объектов с динамическим временем хранения и инициализирует его.
- В случае ошибки, `new` выбросит исключение `std::bad_alloc exception`;
- Если адресуемой памяти в сегменте кучи недостаточно, будет запрошена память у ОС;
- ОС выделяет память страницами от 4 KiB
- Стандартные аллокаторы универсальны: потокобезопасны и одинаково эффективны для блоков разного размера.
- Нельзя совмещать `malloc/free` и `new/delete`
- Объекты созданные с помощью `new` необходимо удалять через `delete` или `delete[]` для массивов

Указатели и ссылки

Указатели

Указатель на объект или функцию — это переменная, содержащая адрес первого байта памяти, занятого объектом или функцией.

Любой указатель может иметь специальное значение нулевого указателя, которое означает, что указатель ни на что не указывает; (nullptr, NULL, 0)

Основной операции

- разыменование — * — получение объекта, на который указывает указатель.
- взятие адреса — & — получение адреса объекта

Указатели примеры

```
int *p1;           // обычный указатель
```

```
const int *pc2;    // указатель на константу
```

```
int *const cp3 = ...; // константный указатель, д.б. инициализирован!
```

```
const int *const cpc4 = ...; // константный указатель на константу, д.б. инициализирован
```

```
int **p2;          // указатель на указатель
```

```
int ***p3;         // указатель на указатель на указатель.
```

```
sizeof(int*)       // размер указателя зависит от архитектуры, на 32х битной системе == 4 байта, на 64х битной 8
```

```
sizeof(p2) == sizeof(p3)
```

Ссылки

Ссылка — это псевдоним уже существующего объекта или функции.

Ссылки были введены как более безопасная альтернатива указателям.

Компилируемый код, использующий ссылки не отличается от кода, использующего указатели

То есть ссылки - это просто указатели с дополнительными ограничениями, диктуемыми правилами языка

Ссылки

- Ссылка обязана быть инициализированной валидным объектом или функцией
- Ссылка не может поменять объект на который ссылается (ссылка связывается единожды)
- Ссылок на void не существует
- Ссылки не являются объектами, поэтому нельзя сделать массив ссылок, указатель на ссылку, ссылку на ссылку

Ссылки примеры использования

- Упрощения доступа ко вложенным полям структур:

```
int& field = input.data[i].extra.info["errors"]
```

- Ссылка позволяет принимать значение «без копирования» и вернуть результат обратно

```
struct Data {
```

```
    int bytes[100];
```

```
};
```

```
void foo(Data data) { /* data копируется при передаче */ }
```

```
void foo(Data &data) { data.bytes[0] = 42; }
```

- Константная ссылка (под этим подразумевается неизменяемость данных, а не самой ссылки) - предпочитаемый способ передачи входных параметров в функции

```
const int &field = ...;
```

```
void foo(const int &data);
```

```
int a = 10;
```

```
foo(a);
```

```
foo(10);
```


Мнемоническое правило спирали

Начиная с имени движемся по спирали по часовой стрелке, подставляя:

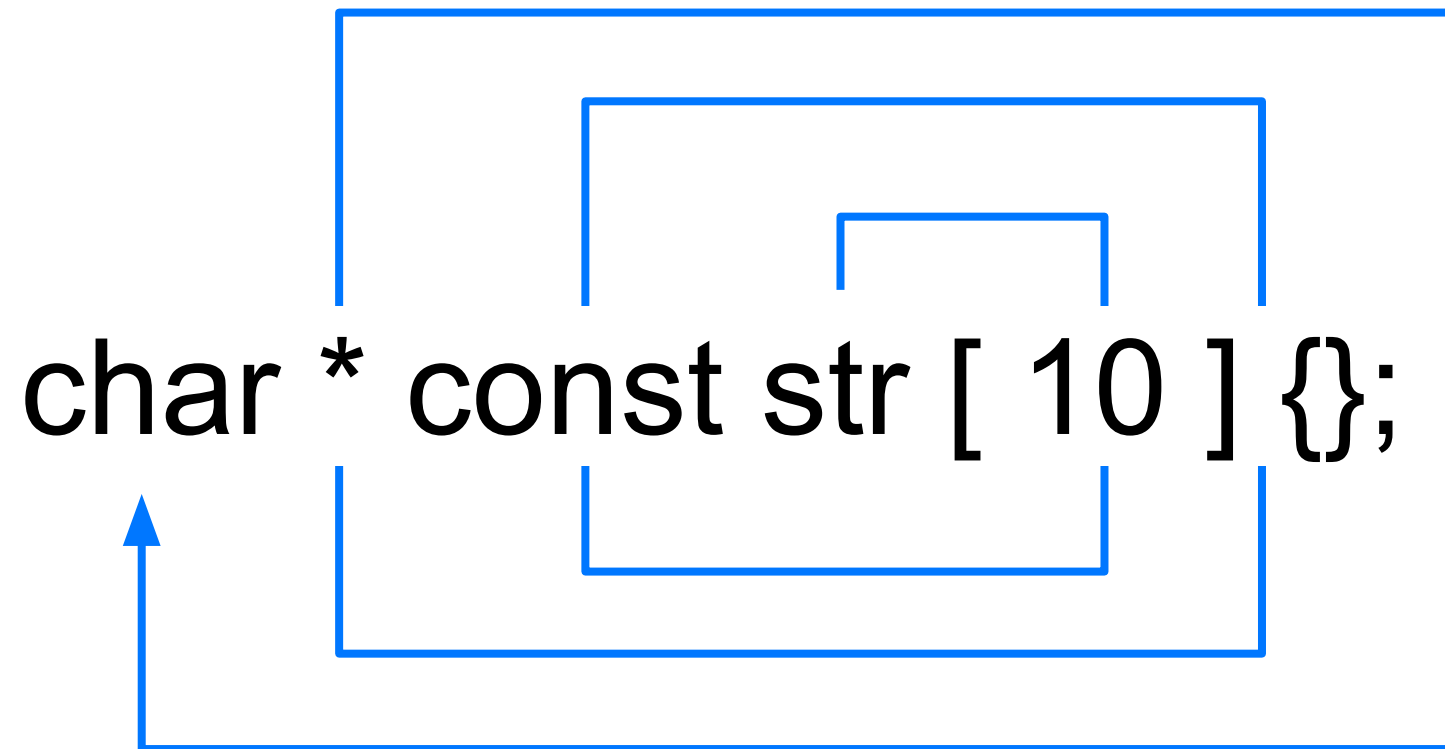
[?] или [] - массив размера ?...

(type1, type2) функция принимающая type1 и type2, возвращающая ...

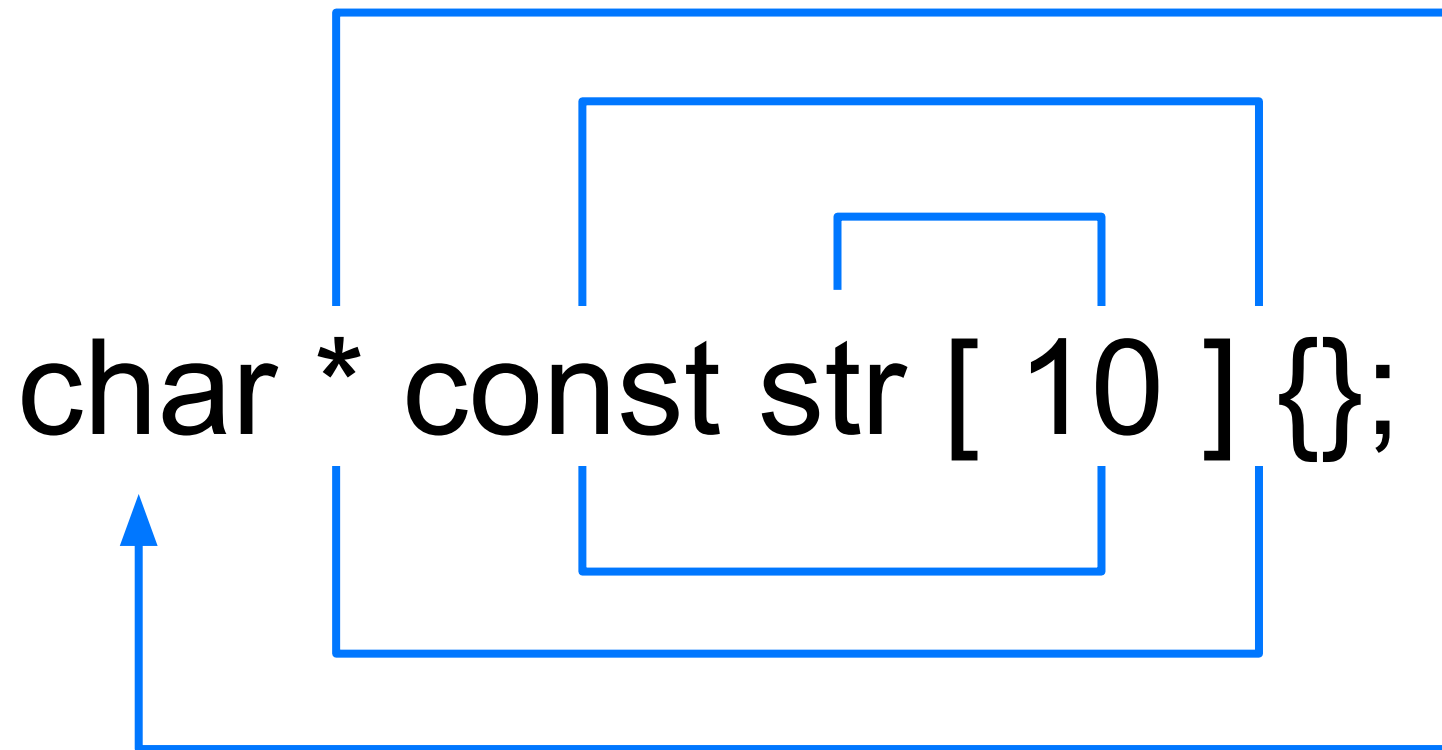
* - указатель/указателя/указателей на ...

& - ссылка/ссылку на ...

Мнемоническое правило спирали

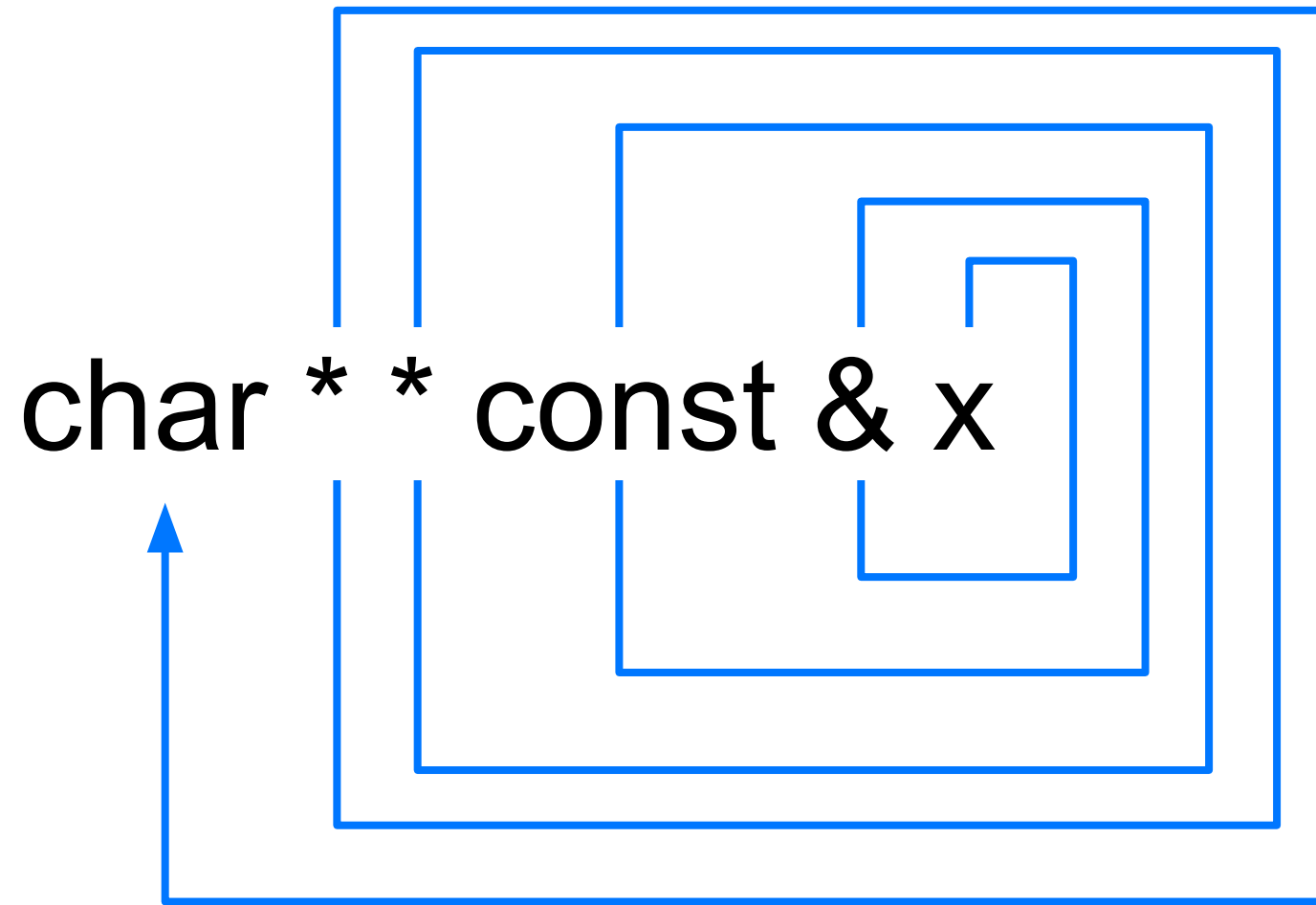


Мнемоническое правило спирали



str - это массив размера 10 константных указателей на char

Мнемоническое правило спирали



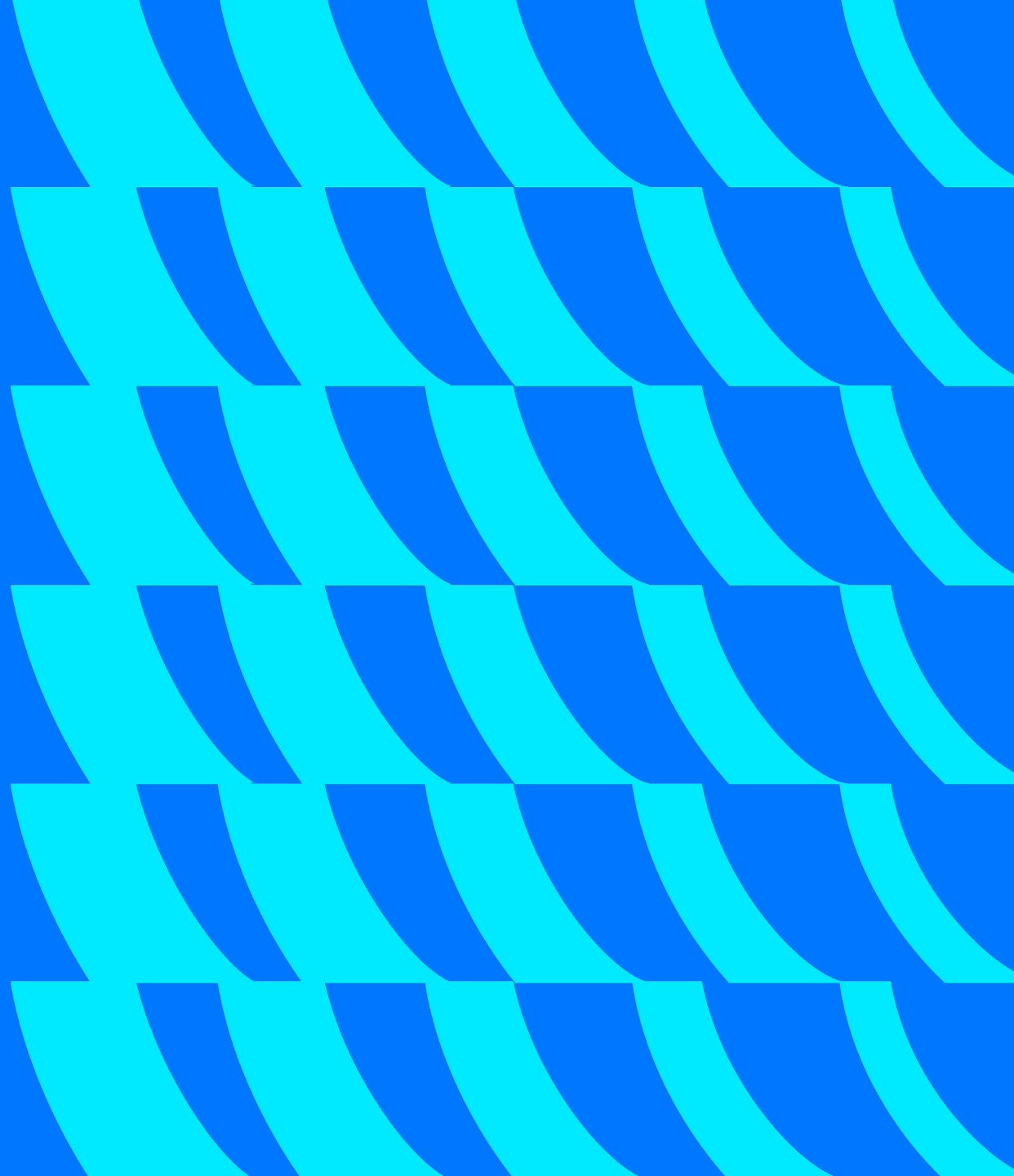
Мнемоническое правило спирали

x - это ссылка на
константный
указатель на
указатель на char

char * * const & x



Перерыв!



Длительность хранения и связывание

Объекты в C++

Программы на C++ создают, уничтожают, ссылаются на, получают доступ и манипулируют объектами.

Объект в C++ имеет:

- размер (sizeof)
- требования по выравниванию (alignof)
- **длительность хранения**
- время жизни
- тип
- значение
- опциональное **имя**

Объекты в C++

Программы на C++ создают, уничтожают, ссылаются на, получают доступ и манипулируют объектами.

Объект в C++ имеет:

- размер (sizeof)
- требования по выравниванию (alignof)
- **длительность хранения**
- время жизни
- тип
- значение
- опциональное **имя**

Имена характеризуются **областью видимости, длительностью хранения и связыванием.**

Объявления, определения

Объявления (Declaration) это то, как вводятся (или повторно вводятся) **имена** в программу на C++.

Определения (Definition) - это объявления, которые полностью определяют сущность, введенную объявлением;

Объявлений может быть несколько.

Попытка использования необъявленного имени в программе приводит к ошибке компиляции.

One Definition Rule

Только одно определение любой переменной, функции, класса, перечисления, концепта (C++20) или шаблона разрешено в любой единице трансляции.

Каким образом можно использовать имена из других единиц трансляции

Чтобы ответить на этот вопрос необходимо:

- Понимать как компилятор обрабатывает единицу трансляции
- Знать **длительность хранения** объекта
- Знать **связывание** имени

Каким образом можно использовать имена из других единиц трансляции

Понимание этого механизма дает:

- Легкое обнаружение ошибок по диагностическим сообщениям компилятора и компоновщика
- Эффективное использование памяти
- Возможность избегать неопределенного поведения (UB) (Например, Static Initialization Order Fiasco)

Работа компоновщика над объектными файлами

```
#include "lib_util.hpp"
#include <iostream>

const int I = 10;

int main() {
    printHello();
    std::cout << I << std::endl;
    return 0;
}
```

Работа компоновщика над объектными файлами


```
#pragma once  
void printHello();
```



```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 10;  
  
int main() {  
    printHello();  
    std::cout << I << std::endl;  
    return 0;  
}
```


Работа компоновщика над объектными файлами

```
#pragma once  
void printHello();
```



```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 10;  
  
int main() {  
    printHello();  
    std::cout << I << std::endl;  
    return 0;  
}
```


```
#pragma once  
void printHello();
```



```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 20;  
  
void printHello() {  
    std::cout << I << std::endl;  
}
```



Работа компоновщика над объектными файлами

```
#pragma once  
void printHello();
```

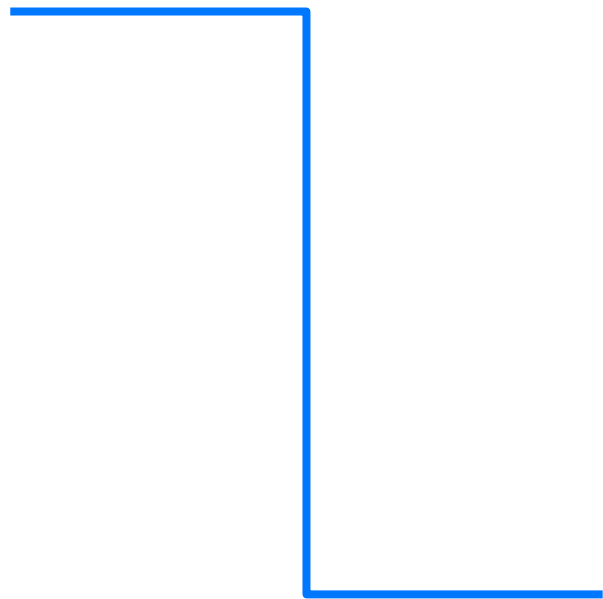


```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 10;  
  
int main() {  
    printHello();  
    std::cout << I << std::endl;  
    return 0;  
}
```

```
#pragma once  
void printHello();
```




```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 20;  
  
void printHello() {  
    std::cout << I << std::endl;  
}
```




Работа компоновщика над объектными файлами

```
#pragma once  
void printHello();
```



```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 10;  
  
int main() {  
    printHello();  
    std::cout << I << std::endl;  
    return 0;  
}
```

```
#pragma once  
void printHello();
```



```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 20;  
  
void printHello() {  
    std::cout << I << std::endl;  
}
```

Работа компоновщика над объектными файлами

```
#pragma once  
void printHello();
```

```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 10;  
  
int main() {  
    printHello();  
    std::cout << I << std::endl;  
    return 0;  
}
```

```
#pragma once  
void printHello();
```

```
#include "lib_util.hpp"  
#include <iostream>  
  
const int I = 20;  
  
void printHello() {  
    std::cout << I << std::endl;  
}
```

**ODR не
нарушается**

Имена объектного файла

При объявлении компилятор фиксирует какие имена необходимы в объектном файле и какие предоставляются объектным файлом.

```
nm -C main.o
```

Длительность хранения

автоматическая - память выделяется в начале включающего блока и освобождается в конце.

статическая - память для объекта выделяется при запуске программы и освобождается при завершении программы. Существует только один экземпляр объекта.

потокковая - память для объекта выделяется, когда поток стартует, и освобождается, когда поток завершается. У каждого потока есть собственный экземпляр объекта.

динамическая - память для объекта выделяется и освобождается с помощью new / delete

Связывание

Если имя имеет связывание, то оно ссылается на ту же сущность, что и точно такое же имя, введенное объявлением в другой области видимости. В противном случае создается несколько экземпляров сущности.

без связывания - на имя можно ссылаться только из той области видимости, в которой оно находится.

внутреннее связывание - на имя можно ссылаться из всех областей видимости в текущей единице трансляции.

внешнее связывание - на имя можно ссылаться из областей видимости в других единицах трансляции.

Связывание

Без дополнительных спецификаторов имена:

Не имеют связывания - имена области видимости блока, локальные классы.

Имеют внутреннее связывание - все имена объявленные в анонимном пространстве имен и константные переменные.

Внешнее связывание - все остальное:

- глобальные функции, классы, переменные
- перечисления
- не константные переменные

Спецификаторы классов памяти

Можем менять связывание и длительность хранения с помощью спецификаторов классов памяти при объявлении:

- **static** - **внутреннее** связывание и статическая или потоковая длительность хранения
- **extern** - **внешнее** связывание и статическая или потоковая длительность хранения
- **thread_local** - потоковая длительность хранения

Спецификаторы классов памяти

Можем менять связывание и длительность хранения с помощью спецификаторов классов памяти при объявлении:

- **static** - **внутреннее** связывание и статическая или потоковая длительность хранения
- **extern** - **внешнее** связывание и статическая или потоковая длительность хранения
- **thread_local** - потоковая длительность хранения

static или анонимное пространство имен можно понимать как “видимо только в сpp файле где объявлено”

extern можно понимать как “определено где-то еще”

Спецификаторы классов памяти

- **static** внутри объявления класса обозначает статический элемент - разберем в другой лекции
- **static** локальные переменные инициализируются в момент, когда до их объявления доходит исполнение программы. В многопоточном коде, компилятор гарантирует, что это будет сделано один раз.

Спецификаторы классов памяти

- **extern** в объявлении переменной без инициализатора не является определением

```
float PI = 3.14f; // definition
```

```
float PI; // definition
```

```
extern float PI = 3.14f; // definition
```

```
extern float PI; // declaration
```

Как пользоваться спецификаторами классов памяти

- Для вспомогательных функции, в том числе шаблонных и констант, необходимых только для реализации (грубо говоря в одном `cpp` файле), лучше использовать внутреннее связывание (через `static` или безымянное пространство имен, связывание не нужно).
- Для глобальных общих переменных (нужен один и тот же экземпляр в нескольких `cpp` файлах) можно использовать внешнее связывание и определение в какой-то одной единице трансляции (например, как `std::cout` через `extern`).
- Инициализация глобальных переменных другими глобальными переменными - очень плохая идея, так как порядок инициализации не определен (Static Initialization Order Fiasco)

Использование спецификаторов классов памяти

| | | | |
|------------|--|-------------|-----------------------------|
| переменная | Внутреннее связывание (static или в анонимном пространстве имен или const) | в сpp файле | Доступна только в сpp файле |
|------------|--|-------------|-----------------------------|

Использование спецификаторов классов памяти

| | | | |
|------------|--|-------------|---|
| переменная | Внутреннее связывание (static или в анонимном пространстве имен или const) | в cpp файле | Доступна только в cpp файле. |
| переменная | Внутреннее связывание (static или в анонимном пространстве имен или const) | в hpp файле | Копия для каждой единицы трансляции с разными адресами. Не подходит для non-const. |

Использование спецификаторов классов памяти

| | | | |
|------------|---|-------------|--|
| переменная | Внутреннее связывание (static или в анонимном пространстве имен или const) | в cpp файле | Доступна только в cpp файле. |
| переменная | Внутреннее связывание (static или в анонимном пространстве имен или const) | в hpp файле | Копия для каждой единицы трансляции с разными адресами. Не подходит для non-const. |
| переменная | Внешнее связывание (в пространстве имен в том числе глобальном, extern const) | в cpp файле | Может нарушить ODR, если где-то появится с таким же именем |

Использование спецификаторов классов памяти

| | | | |
|------------|---|-------------|--|
| переменная | Внутреннее связывание (static или в анонимном пространстве имен или const) | в cpp файле | Доступна только в cpp файле. |
| переменная | Внутреннее связывание (static или в анонимном пространстве имен или const) | в hpp файле | Копия для каждой единицы трансляции с разными адресами. Не подходит для non-const. |
| переменная | Внешнее связывание (в пространстве имен в том числе глобальном, extern const) | в cpp файле | Может нарушить ODR, если где-то появится с таким же именем |
| переменная | Внешнее связывание (в пространстве имен в том числе глобальном, extern const) | в hpp файле | Гарантированное нарушение ODR при включении. Корректная работа только при объявлении как extern и определении в одном cpp |

Использование спецификаторов классов памяти

| | | | |
|---------|--|-------------|---|
| Функция | Внутреннее связывание (static или в анонимном пространстве имен) | в сpp файле | Доступна только в сpp файле |
| Функция | Внутреннее связывание (static или в анонимном пространстве имен) | в hpp файле | Лишняя копия для каждой единицы трансляции. |
| Функция | Внешнее связывание (в пространстве имен в том числе глобальном) | в сpp файле | Может нарушить ODR, если где-то появится определение с таким же именем |
| Функция | Внешнее связывание (в пространстве имен в том числе глобальном) | в hpp файле | Гарантированное нарушение ODR при включении, если определена в заголовочном файле |

Встроенные функции и переменные

Существующие проблемы

- Как определить функцию в заголовочном файле не нарушив ODR?
- Как определить переменную в заголовочном файле, например, для header-only библиотек, у которых нет src для инициализации?
- Как проинициализировать статическую переменную класса (общую для всех экземпляров класса) в заголовочном файле, например, для header-only библиотек, у которых нет src для инициализации?

Спецификатор inline

Изначально был индикатором для оптимизаций компилятора, означающий, что вместо вызова функции предпочтительна подстановка её тела.

Спецификатор inline

Изначально был индикатором для оптимизаций компилятора, означающий, что вместо вызова функции предпочтительна подстановка её тела.

Такая оптимизация увеличивает размер исполняемого файла, но уменьшает накладные расходы на вызов функции.

Спецификатор `inline`

Изначально был индикатором для оптимизаций компилятора, означающий, что вместо вызова функции предпочтительна подстановка её тела.

Такая оптимизация увеличивает размер исполняемого файла, но уменьшает накладные расходы на вызов функции.

При этом стандарт никогда не обязывал компилятор выполнять эту подстановку.

Компилятор может подставлять тело для функций не помеченных как `inline`, и может генерировать инструкцию вызова для функций помеченных как `inline`.

Спецификатор inline

Побочным эффектом такого спецификатора стала необходимость иметь определение функции в каждой единице трансляции, где она использовалась, что вело к нарушению ODR.

Спецификатор inline

Побочным эффектом такого спецификатора стала необходимость иметь определение функции в каждой единице трансляции, где она использовалась, что вело к нарушению ODR.

Таким образом inline из смысла “предпочтительно встраивание” стало означать “разрешено несколько определений”

Спецификатор `inline`

Побочным эффектом такого спецификатора стала необходимость иметь определение функции в каждой единице трансляции, где она использовалась, что вело к нарушению ODR.

Таким образом `inline` из смысла “предпочтительно встраивание” стало означать “разрешено несколько определений”

В C++17 спецификатор `inline` разрешили для использования с переменными в том числе.

Спецификатор inline

Для inline функции и переменной:

- Определение должно быть доступным для каждой единицы трансляции, где они используются
- Для функций и переменных с внешним связыванием может быть несколько определений, при условии что они находятся в разных единицах трансляции и все совпадают (Разные определения UB - компоновщик скорее всего выберет первую)
- Имеют одинаковый адрес во всех единицах трансляции

Спецификатор inline

Примеры

Работа с файлами и строками

Ввод/вывод на основе потоков

`std::fstream` (`ofstream`, `ifstream`) - чтение/запись из / в файл

`std::stringstream` (`ofstream`, `ifstream`) - чтение/запись из / в строковый буфер

`std::cin`, `std::cout`, `std::cerr` - потоки стандартных ввода, вывода, ошибок

Ввод/вывод на основе потоков

| | |
|---------------|--|
| fopen | std::fstream, std::ifstream, std::ofstream |
| fread, fwrite | std::fstream::read, std::fstream::write |
| fgetc | std::fstream::get |
| fgets | std::fstream::get |
| ftell | std::fstream::tellp, std::fstream::tellg |
| fseek | std::fstream::seekp, std::fstream::seekg |

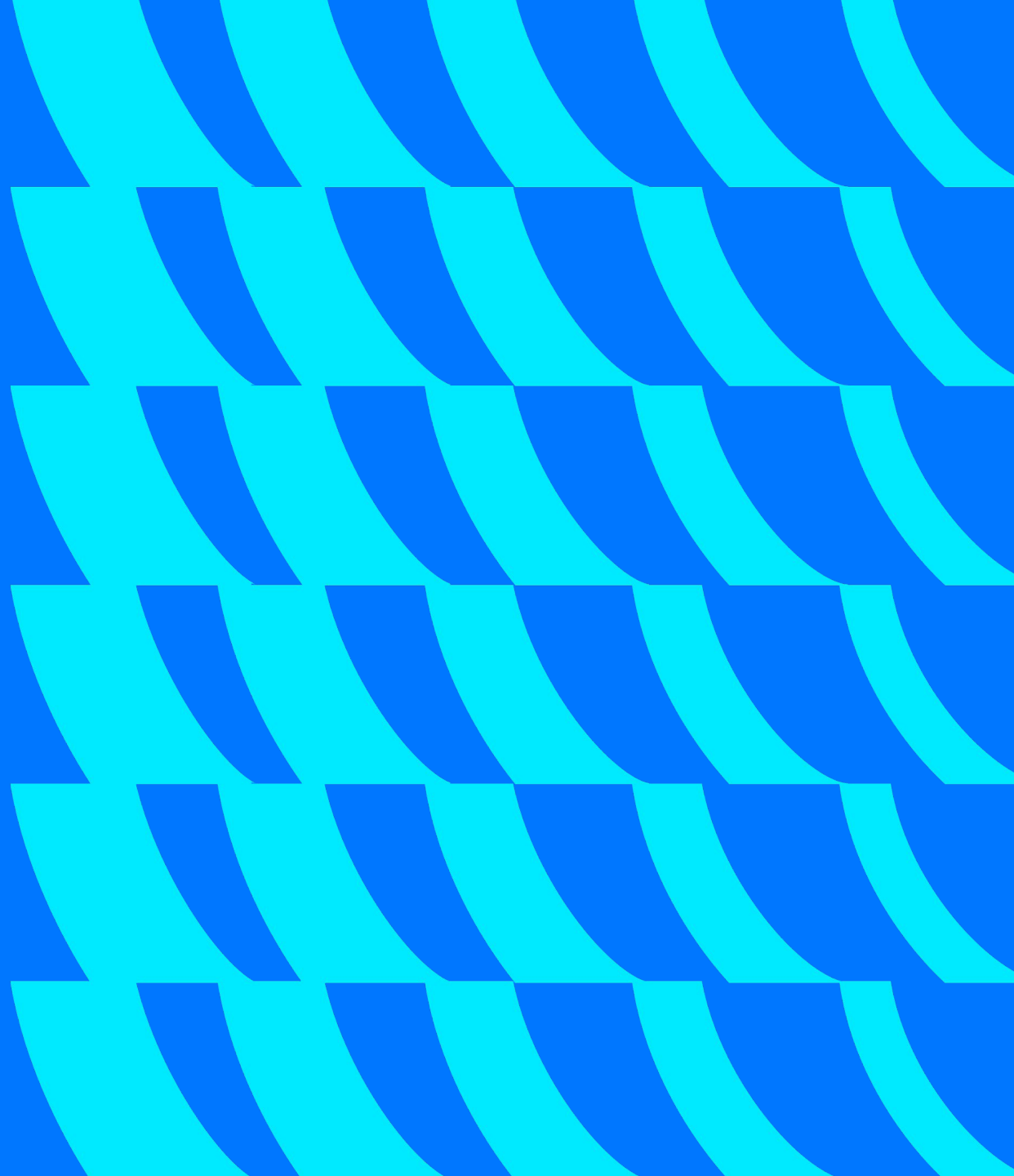
Преобразования

| | |
|---------|-----------------------------------|
| atoi | std::stoi |
| atol | std::stol, std::stoul |
| atoll | std::stoll, std::stoull |
| atof | std::stof, std::stod, std::stold, |
| sprintf | std::to_string |

Работа со строками

| | |
|----------------------|-------------------------------------|
| <code>strlen</code> | <code>std::string::size()</code> |
| <code>strstr</code> | <code>std::string::find()</code> |
| <code>getline</code> | <code>std::string::getline()</code> |

Практика



Спасибо за внимание!

