Указатели и ссылки

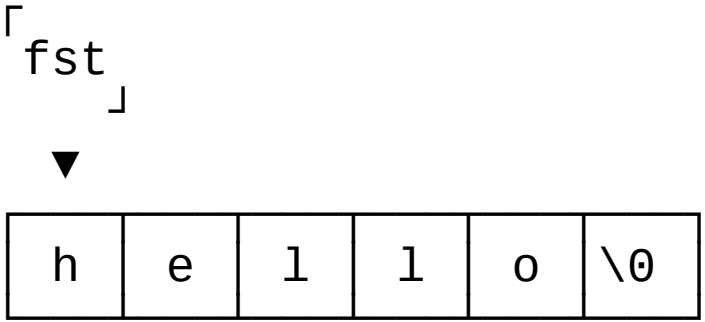
Массив VS указатель

```
const char         fst[] = “hello”;
const char* const snd   = “world”;
```

В чем разница?

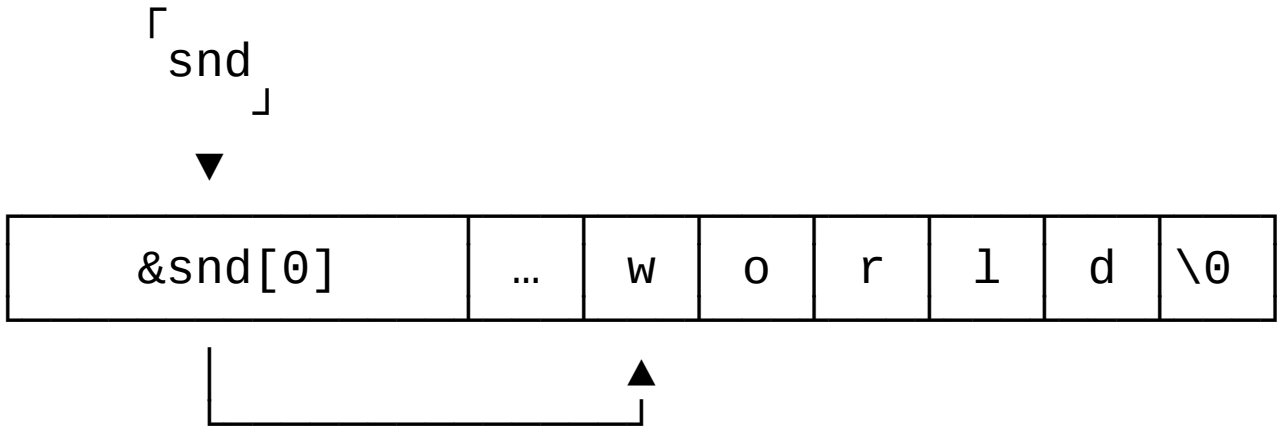
Имя массива — не объект

```
const char fst[] = “hello”;
```



Указатель — это объект

```
const char* const snd = “world”;
```



Имя массива — не объект

```
const char fst[] = “hello”;
const char* const snd = “world”;

&fst    = 0x7ffc085ec43a

&fst[0] = 0x7ffc085ec43a

&snd    = 0x7ffc085ec430

&snd[0] = 0x402004
```

https://godbolt.org/z/T6javz54b

Применение указателей

Применение указателей

1. Массивы и адресная арифметика
2. Передача тяжелых значений в функции без копирования
3. Out и in-out параметры функций
4. Создание динамических структур данных
5. Семантика отсутствия объекта

Массивы и адресная арифметика

Пусть

```
int pi[] = {3, 14, 15, 92, 65};
```

Тогда следующие выражения эквивалентны:

```
pi[2]   *(pi + 2)

2[pi]   *(2 + pi)
```


https://godbolt.org/z/9f1rd3v6q

Передача тяжелых значений без копирования

Массивы:

```
int strcmp(const char *s1, const char *s2);

char s1[] = “...”, s2[] = “...”;
strcmp(s1, s2); // array to pointer decay
```

Тяжелые объекты:

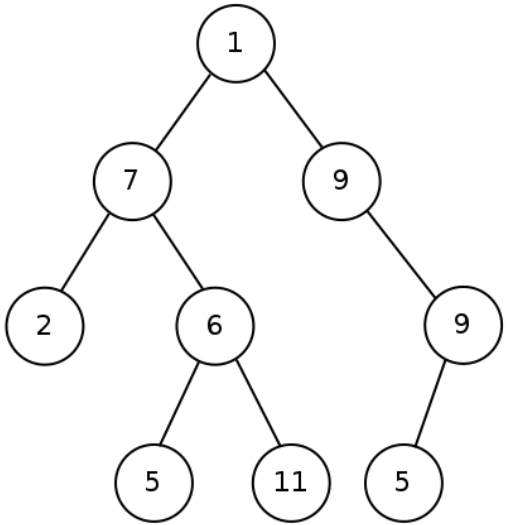
```
print(const Matrix* matrix);
```

Out и in-out параметры функций

```
int SDL_GetRendererOutputSize(
  SDL_Renderer* renderer,
  /* Out */ int* w, int* h);


// In-out
int pthread_mutex_lock(pthread_mutex_t* mutex);
int pthread_mutex_unlock(pthread_mutex_t* mutex);
```

Динамические структуры данных

```
struct TreeNode {
  TreeNode* left_;
  TreeNode* right_;
};
```



Семантика отсутствия объекта

```
FILE *fopen(const char *pathname, const char *mode);
```

RETURN VALUE
Upon successful completion […] return a FILE pointer.
Otherwise, **NULL** is returned

Какого типа NULL?

MSVC: Определение NULL

```
// vcruntime.h
#ifndef NULL
    #ifdef __cplusplus
        #define NULL 0
    #else
        #define NULL ((void *)0)
    #endif
#endif
```

В C++ NULL не может быть void*

Почему в C++ NULL == 0

Хотим написать:

```
char* str = NULL;
```

C++ строже относится к преобразованию указателей

Корректно в C, Некорректно в C++:

```
char* str = (void*)0;
```

Корректно в C и в C++:

```
char* str = 0;
```

nullptr

В C++ рекомендуется использовать **`nullptr`**:

```
// gcc
typedef decltype(nullptr) nullptr_t;
```

ES.47: Use nullptr rather than 0 or NULL

Мотивация nullptr: разрешение перегрузки

```
const char* f(double*) { return "f(double*)\n"; }

const char* f(long)    { return "f(long)\n"; }

int main() {
  std::cout
    << "f(NULL) -> "    << f(NULL)    // f(long)
    << "f(nullptr) -> " << f(nullptr); // f(double*)
}
```


https://godbolt.org/z/dMWKT5M3s

Применение указателей

1. Массивы и адресная арифметика
2. Передача тяжелых значений в функции без копирования
3. Out и in-out параметры функций
4. Создание динамических структур данных
5. Семантика отсутствия объекта

Это очень много для единственной абстракции

Объясните различия в определениях

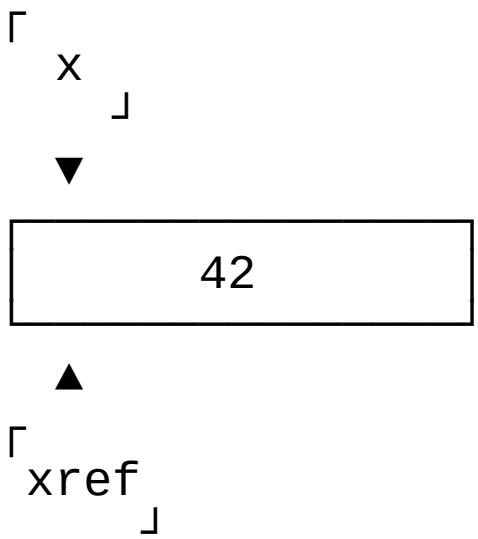
```
char* s1;

char const* s2;

const char* s3;

char* const s4;

const char* const s5;
```

Объясните различия в определениях

```
char* s1;

char const* s2;  // east const

const char* s3;  // const west

char* const s4;

const char* const s5;
```

Ссылка — это синоним объекта

```
int x = 42;
int& xref  = x;
```



Ссылки прозрачны для операций

```
int range[] = {3, 5};
auto* p = &range[0];
auto& r = range[0];

++p;
++r;
```

***p == ? r == ?**

Ссылки прозрачны для операций

```
int range[] = {3, 5};
auto* p = &range[0];
auto& r = range[0];

++p;
++r;
```

`*p == 5  r == 4`

Некоторые свойства ссылок

1. Не поддерживают адресную арифметику.

```
int& r = x;
++r; // то же, что и ++x
```

2. Обязаны быть инициализированы (связаны с объектом)

```
int& r; // Ошибка компиляции
```

3. Не обладают семантикой отсутствия значения (**нет `nullref`**)

Константность указателей и ссылок

```
char* s1;                       char& r1 = …;
char const* s2;                 N/A
const char* s3;                 N/A
char* const s4 = …;             char const& r2 = …;
const char* const s5 = …;       const char& r3 = …;
```

Константность ссылок

const int& — ссылка на константу (reference to const).

Обычно для простоты говорят «константная ссылка» (constant reference, const ref), но мы понимаем, что как таковых «константных» ссылок не бывает.

В чем разница?

```
void fp(int* p) {
    *p = 0xAB;
}

void fr(int& r) {
    r = 0xAB;
}
```

```
int main() {
    int g = 0;
    fp(&g);
    fr(g);
}
```


https://godbolt.org/z/rqdv8Mqc9

Применение ссылок

1. Передача аргументов в функцию
2. Локальные синонимы
3. Короткий синоним для глубоко вложенного объекта
4. Передача массивов в функцию (крайне редко)

Передача аргументов в функцию

In параметр:

```
void f(const std::vector<Widget>& widgets);
```

Out/In-out параметр:

```
void f(std::vector<Widget>& widgets);
```

Сравните эти подходы

```
void get_screen_size_p(int* x, int* y);
void get_screen_size_r(int& x, int& y);


get_screen_size(&x, &y);
get_screen_size(x, y);
```

Передача параметров в функцию

	Cheap or impossible to copy (e.g., int, unique_ptr)	**Cheap to move** (e.g., vector<T>, string) or **Moderate cost to move** (e.g., array<vector>, BigPOD) or **Don't know** (e.g., unfamiliar type, template)	**Expensive to move** (e.g., BigPOD**[]**, array<BigPOD>)
Out	X f()		
In/Out	f(X&)		
In	f(X)	f(const X&)	
In & retain "copy"			

"Cheap" ≈ a handful of hot int copies
"Moderate cost" ≈ memcpy hot/contiguous ~1KB and no allocation

** or return unique_ptr<X>/make_shared_<X> at the cost of a dynamic allocation*

CppCoreGuidelines

Ограничение ссылок

Нельзя создать массив или контейнер ссылок.

Нельзя создать указатель на ссылку.

Все это — ошибка компиляции:

```
int& rarr[10];
std::vector<int&> refs;

int&* ptr_to_ref = …;
```

Как вы думаете, почему?

Про this

```
struct Point {
  double x_, y_;

  void move(/*Point* this, */ const Vec2d* vec) {
    /*this->*/ x_ += vec->x_;
    /*this->*/ y_ += vec->y_;
  }
};
```

Почему this – указатель, а не ссылка?

Про this

- Для this вам не понадобится адресная арифметика
- Обычно он не бывает nullptr
- Всегда инициализирован силами компилятора

this появился в языке раньше ссылок

Объясните, что здесь могло бы происходить

```
void f(int& x) {
  ++x;
}

int main() {
  double x = 42;
  f(x);
}
```

Это не скомпилируется

Временные объекты связываются с const ref

```
void f(const int& x) {
  // ++x;
}

int main() {
  double x = 42;
  f(x);
}
```

Это ~~так~~ special case

Кажется, здесь нужен специальный механизм