Константность

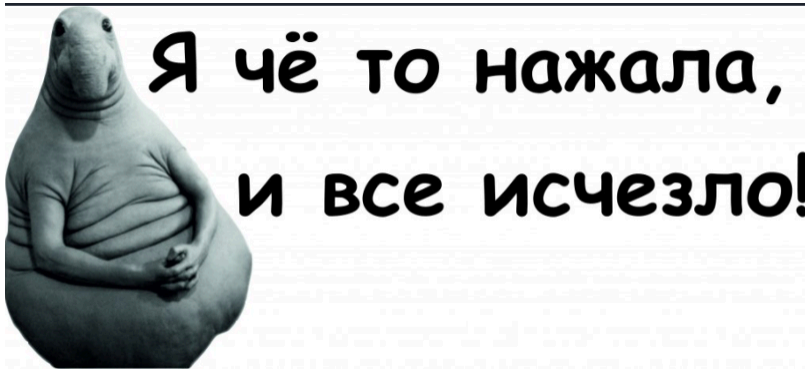
Проблемы

1. Как гарантировать неизменяемость объекта?

```
double pi = 3.14159265358979323846;
pi += 1;  // ?!
```

2. Как гарантировать неизменяемость массива при передаче в функцию?

```
bool BinarySearch(int* arr, int size, int key) {
  // ...
  arr[0] = 42;  // Ooops...
  // ...
}
```



Квалификатор `const`

Константный объект - объект, к которому применимы только те операции, которые не меняют его логического состояния.

Чтобы объявить константную переменную необходимо к названию типа добавить слово `const`:

```
// так
const int x = 0;

// или так
int const x = 0;
```

Теперь попытка изменения `x` будет приводить к ошибке компиляции:

```
x = 1;           // CE
std::cin >> x;   // CE
```

Квалификатор `const`

Константные переменные не обязаны инициализироваться значением известным на этапе компиляции:

```
int n = 0;
std::cin >> n;

const int square = n * n;  // OK
```

А есть ли способ указать, что переменная должна быть инициализирована константным значением (известным на этапе компиляции)?

Спецификатор `constinit`

`constinit` - спецификатор, который указывает, что переменная должна быть инициализирована константным значением (известным на этапе компиляции).

```
constinit int x = 0;  // OK
```

```
int n = 0;
std::cin >> n;

constinit int square = n * n;  // CE: n * n - не константное выражение
```

При этом `constinit` не обязывает переменную быть константной:

```
constinit int x = 0;
x = 1;  // Ok
```

Спецификатор `constexpr`

`constexpr` - спецификатор, который указывает, что переменная должна быть инициализирована константным значением (известным на этапе компиляции) и не должна изменяться.

Иными словами, `constexpr` = `constinit` + `const`.

```
constexpr int x = 0;  // OK
x = 1;                // CE
```

```
int n = 0;
std::cin >> n;

constexpr int square = n * n;  // CE: n * n - не константное выражение
```

Указатели и `const`

Указатели и `const`

Указатели могут быть константными с двух точек зрения.

1. *Указатель на константу*. Константным является объект, на который указывают:

```
int x = 0;
const int y = 0;

const int* px = &x;  // либо int const*
const int* py = &y;  // либо int const*

std::cin >> *px;   // CE: объект считается неизменяемым
std::cout << *py;  // OK: объект доступен для чтения
```

Указатели и `const`

Указатели могут быть константными с двух точек зрения.

2. *Константный указатель*. Константным является сам указатель:

```
int x;
int y;

int* const px = &x;

std::cin >> *px;   // OK
std::cout << *py;  // OK

px = &y;  // CE: указатель неизменяемый
++px;     // CE
```

Указатели и `const`

Константности можно комбинировать:

```
const int* px = ...;  // указатель на константу
int const* py = ...;  // указатель на константу
int* const pz = ...;  // константный указатель
const int* const pa = ....;  // константный указатель на константу
int const* const pa = ...;   // константный указатель на константу
```

Лайфхак: читайте типы указателей справа налево.

`int const* const - константный указатель на константу int`

Ссылки и `const`

Ссылки и `const`

Ссылка на константу - ссылка, которая не позволяет изменять объект, на который она ссылается.

```
int x = 0;

const int& ref = x;  // дает возможность только для чтения
std::cout << ref;    // OK
std::cin >> ref;     // CE
```

На константную переменную можно создать только ссылку на константу:

```
const int y = 0;

const int& ref = y;
int& ref2 = y;  // CE
```

Ссылки и `const`

Исключение: ссылка на константу может быть создана на временный объект (результат rvalue-выражения):

```
int& ref = 42;  // CE

const int& ref = 42;  // OK
```

Константный ссылка "продлевает жизнь" временного объекта до конца жизни ссылки.