# Специализация шаблонов

# Задача

Реализовать функцию возведения в степень.

1. Выбрать правильный алгоритм.
2. Понять, для каких типов он может работать.

# Задача

Реализовать функцию возведения в степень.

1. Выбрать правильный алгоритм.
2. Понять, для каких типов он может работать.

*Предложите алгоритм*

# Попытка 1

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  for (unsigned i = 0; i < n; ++i) {
    acc *= x;
  }
  return acc;
}
```



https://godbolt.org/z/cPTbKrP44

# Попытка 1

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  for (unsigned i = 0; i < n; ++i) {
    acc *= x;
  }
  return acc;
}
```

Покритикуйте этот подход

# Попытка 1

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  for (unsigned i = 0; i < n; ++i) {
    acc *= x;
  }
  return acc;
}
```

*Асимптотическая сложность?*

# Попытка 1

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  for (unsigned i = 0; i < n; ++i) {
    acc *= x;
  }
  return acc;
}
```

O(n), мы явно можем лучше

# Попытка 2

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

*Форматирование ужасно, только для слайда*

# Попытка 2

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

*Асимптотика?*

# Попытка 2

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

Асимптотика? O(log(n))

# Как обобщить?

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Как обобщить?

```
unsigned power(unsigned x, unsigned n) {
  unsigned acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Для каких теперь типов работает функция?

```
template <typename T>
T power(T x, unsigned n) {
  T acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Требования к статическому интерфейсу?

```
template <typename T>
T power(T x, unsigned n) {
  T acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Требования к статическому интерфейсу?

```
template <typename T>
T power(T x, unsigned n) {
  T acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Стат. интерфейс определяется операциями

- Присваивание 1
- Сравнение с 2 оператором <
- Умножение на объект того же типа

Приведите пример типа, с которым алгоритм не будет работать.

# Стат. интерфейс определяется операциями

- Присваивание 1
- Сравнение с 2 оператором <
- Умножение на объект того же типа

Приведите пример типа, с которым алгоритм не будет работать.

Матрица.



https://godbolt.org/z/o7zxYE3Pc

# Предложите способ генералиации алгоритма

```
template <typename T>
T power(T x, unsigned n) {
  T acc = 1;
  if (x < 2 || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Предложите способ генералиации алгоритма

```
template <typename T>
T power(T x, unsigned n) {
  T acc = 1;
  if (x == acc || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Попытка 0. Явная глупость: подмена требования

```
template <typename T>
T power(T x, unsigned n) {
  T acc{}; ++acc;
  if (x == acc || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Попытка 1: уровень косвенности

```
template <typename T>
T power(T x, unsigned n) {
  T acc = id<T>();
  if (x == acc || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# О специализации

- Во время вызова шаблонной функции компилятор создает ее экземпляр для конкретного типа. Этот процесс называется **инстанцированием**.
- Мы можем вмешаться и создать специализацию вручную.

# Синтетический пример

```
template <typename T>
T f(T x) {
  return x;
}

template <>
int f(int x) {
  return x + 1;
}
```



https://godbolt.org/z/cder5K43P

# Возвращаемся к исходной задаче

```
template <typename T>
T id() {
  return 1;
}

template <>
Matrix2x2 id() {
  return Matrix2x2({{1, 0}, {0, 1}});
}
```

# Покритикуйте это решение

```
template <typename T>
T power(T x, unsigned n) {
  T acc = id<T>();
  if (x == acc || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Недостатки

1. Сигнатура функции не выражает зависимость от функции id:
```
template <typename T>
T power(T x, unsigned n);
```
2. Функция id должна быть в глобальном пространстве имен

# Отступление: задача

Пусть нам нужен набор констант: минимальные и максимальные значения для различных типов:

- unsigned
- unsigned char
- …

Как решить эту задачу?

# C-Style

```
#define UCHAR_MIN 0
#define UCHAR_MAX 255
#define UNSIGNED_MIN 0
#define UNSIGNED_MAX 0xffffffff
...
```

*Типизированные константы лучше макросов,*
*но все еще не подходят для обобщенного кода*

# Проблема C-style решения

```
template <typename T>
void f() {
  // Здесь нам нужно максимальное значение для типа T
  T max = ???; // CHAR_MAX, UNSIGNED_MAX, …
}
```

# Решение: трейты

```
template <typename T>
struct Limits;

template <>
struct Limits<unsigned char> {
  constexpr static unsigned min() { return 0; }
  constexpr static unsigned max() { return 0xff; }
};

template <>
struct Limits<unsigned> { ... }
```

# Пример использования

```
template <typename T,
          typename LimitsTrait = Limits<T>>
void f() {
  const auto min = LimitsTrait::min();
  const auto max = LimitsTrait::max();

  std::cout << min << '\n';
  std::cout << max << '\n';
}
```



https://godbolt.org/z/hY1dEK7r8

# Вернемся к задаче

```
template <typename T>
struct IdTrait {
  constexpr static T id() { return 1; }
};

template <>
struct IdTrait<Matrix2x2> {
  constexpr static Matrix2x2 id() {
    return Matrix2x2({{1, 0}, {0, 1}});
  }
};
```

# Использование

```
template <typename T, typename Id = IdTrait<T>>
T power(T x, unsigned n) {
  T acc = Id::id();
  if (x == acc || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```



https://godbolt.org/z/Ev7Psf7qY

# Другие примеры

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>,
    class Allocator = std::allocator<CharT>
> class basic_string;

using string = basic_string<char>;
```

# Регистронезависимые строки

```
struct ci_char_traits : public char_traits<char> {
  static bool eq(char c1, char c2) {
    return toupper(c1) == toupper(c2);
  }
  ...
};

using ci_string = basic_string<char, ci_char_traits>;
```



http://www.gotw.ca/gotw/029.htm

# Промежуточный итог

- Решили проблему с неявными зависимостями
- Решили проблему с пространством имен

```
template <typename T, typename Id = IdTrait<T>>
T power(T x, unsigned n);
```

# Промежуточный итог

- Решили проблему с неявными зависимостями
- Решили проблему с пространством имен

```
template <typename T, typename Id = IdTrait<T>>
T power(T x, unsigned n);
```

*Есть ли еще варианты решения?*

# Шаг 1: переносим в аргументы

```
template <typename T>
T power(T x, T acc, unsigned n) {
  if (x == acc || n == 1) return x;
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}
```

# Шаг 2: отделяем corner cases

```
template <typename T>
T do_power(T x, T acc, unsigned n) {
  while (n > 0)
    if ((n & 1) == 1) { acc *= x; --n; }
    else              { x *= x; n /= 2; }
  return acc;
}

unsigned power(unsigned x, unsigned n) {
  if (x < 2 || n == 1) return x;
  return do_power(x, 1u, n);
}
```

# Пример из стандартной библиотеки

```
template<class InputIt, class T>
T accumulate(InputIt first, InputIt last, T init);

template<class InputIt>
constexpr
typename std::iterator_traits<InputIt>::value_type
reduce(InputIt first, InputIt last);
```

https://en.cppreference.com/w/cpp/algorithm/accumulate

https://en.cppreference.com/w/cpp/algorithm/reduce

# Iterator Traits

https://godbolt.org/z/xheY5997P

https://en.cppreference.com/w/cpp/iterator/iterator_traits

https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/include/bits/stl_iterator_base_types.h