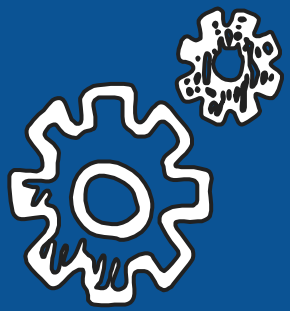
Алгоритмы и Алгоритмические Языки

Семинар #20:

1. Коды возврата и исключения.
2. Механизм размотки стека.
3. Гарантии безопасности исключений.
4. Выдача ДЗ№1.

28.01.2025

Коды возврата и исключения



Обработка ошибок: коды возврата

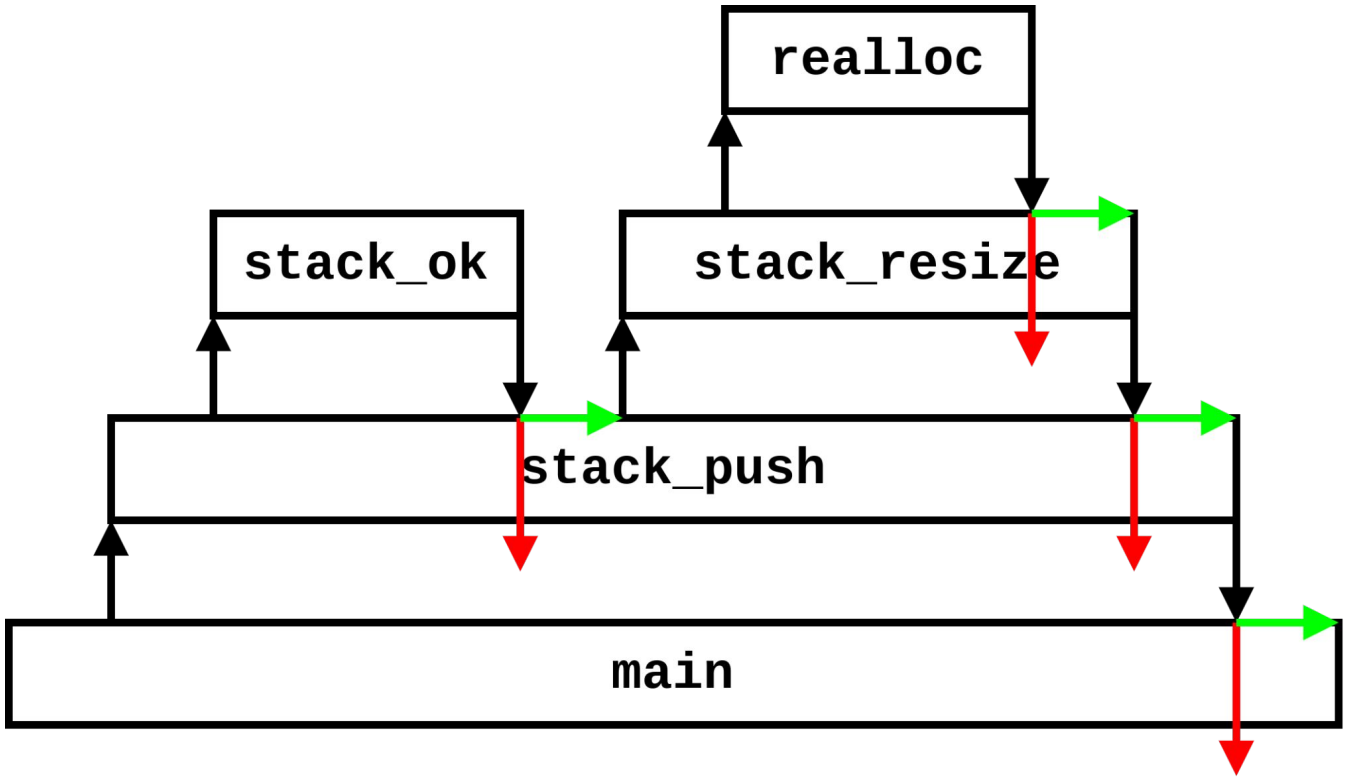
```
StackRetCode stack_resize(struct Stack* stack, size_t new_capacity)
{
    // Перевыделяем память под элементы стека
    data_t* new_array = realloc(stack->array, new_capacity * sizeof(data_t));
    if (new_array == NULL)
    {
        return STACK_NOMEM;
    }
```

```
StackRetCode stack_push(struct Stack* stack, data_t element)
{
    // Проверяем состояние стека
    StackRetCode ret = stack_ok(stack);
    if (ret != STACK_OK)
    {
        return ret;
    }
```

```
// Производим перевыделение памяти
ret = stack_resize(stack, new_capacity);
if (ret != STACK_OK)
{
    return ret;
}
```

```
ret = stack_push(&stack, push_i);
if (ret != STACK_OK)
{
    printf("[ERROR] Unable to push
    return EXIT_FAILURE;
}
```

Обработка ошибок: коды возврата



Обработка ошибок в конструкторах

```
Stack::Stack(StackResizePolicy policy = RESIZE_SUM)
    array_    (new Data_t[1]),
    size_     (0),
    capacity_ (1),
    policy_   (policy)
{}
```

Ошибка выделения памяти в конструкторе:

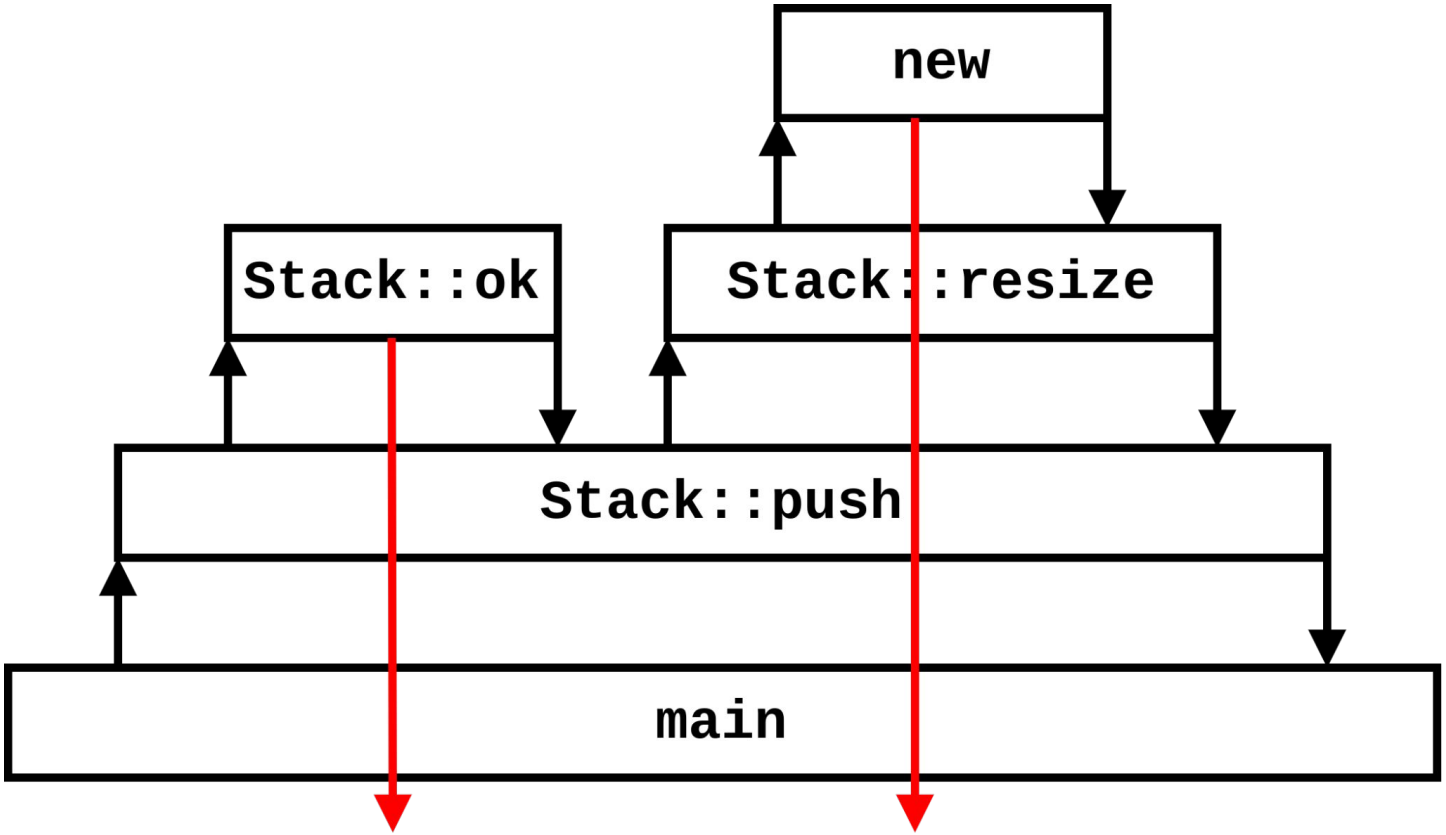
1. Возвращать код ошибки из конструктора – некуда и некому.
2. Продолжать исполнение – десятки часов отладки.

Обработка ошибок: исключения

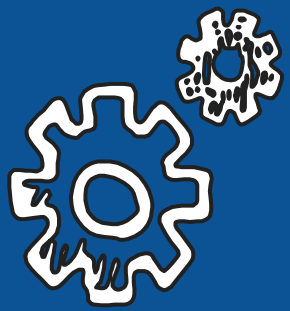
```cpp
// Проверяем стек на пустоту
if (size_ == 0U)
{
    // Из пустого стека извлечение элемента невозможно
    throw std::runtime_error("Unable to pop from empty stack");
}
```

```cpp
try
{
    Data_t popped = stack.pop();
    if (popped != pop_i - 1U)
    {
        printf("[ERROR] Popped invalid element (got %lu
        return EXIT_FAILURE;
    }
}
catch (std::runtime_error& exc)
{
    printf("[ERROR] Runtime error: %s\n", exc.what());
    return EXIT_FAILURE;
}
```

Обработка ошибок: исключения



Механизм размотки стека



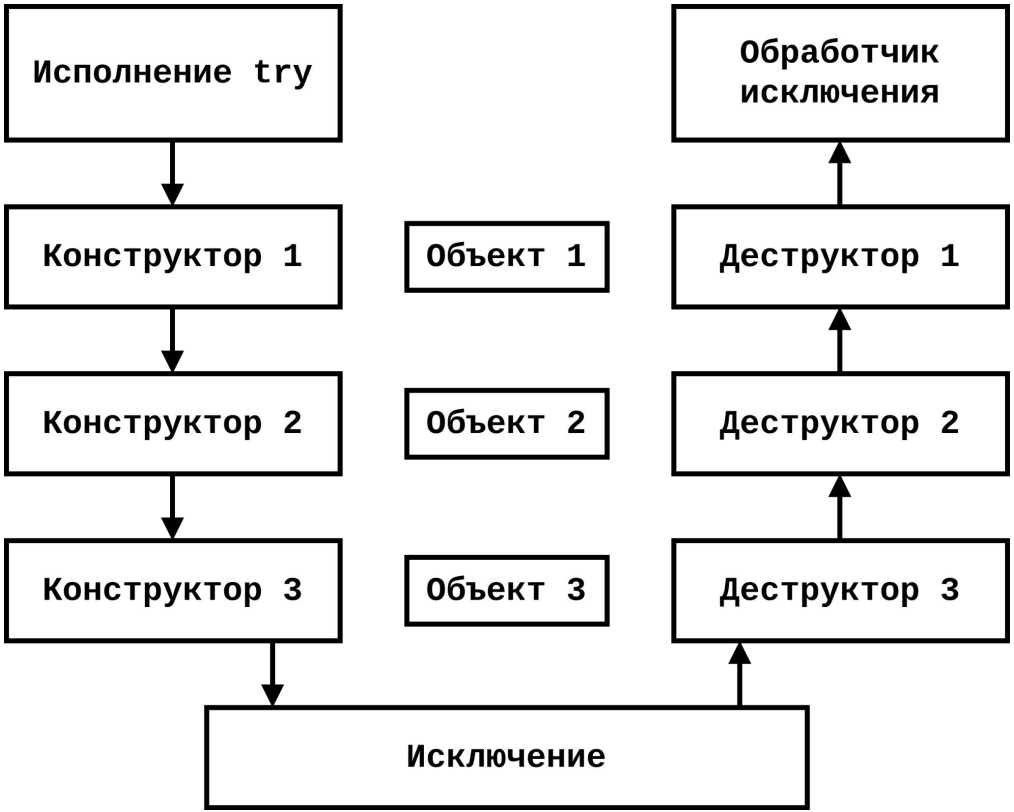
Отправка исключения

```cpp
// Проверяем стек на пустоту
if (size_ == 0U)
{
    // Из пустого стека извлечение элемента невозможно
    throw std::runtime_error("Unable to pop from empty stack");
}
```

```
int64_t Stack::pop(int64_t* arg1)

{
    Stack::check(arg1);
    if (arg1[1] == 0)
    {
        void* thrown_exception = __cxa_allocate_exception(0x10);
        std::runtime_error::runtime_error(thrown_exception, "Unable to pop from empty stack");
        __cxa_throw(thrown_exception, _typeinfo_for_std::runtime_error, std::runtime_error::~runtime_error);
        /* no return */
    }
```

Размотка стека



Обработка исключения

```
// Тестирование на безопасность исключений.
Stack stack = {0, 1, 2};

// Извлекаем элементы стека.
stack.pop();
stack.pop();
stack.pop();

try
{
    stack.pop();
}
catch (const std::runtime_error& exc)
{
    // Функция pop предоставляет строгую гарантию без
    // Поэтому после выброса исключения объект будет
    // что и до выброса исключения.
    printf("Catch exception: \'%s\'\n", exc.what());
}
```

```
int64_t var_48 = 0;
int64_t var_40_1 = 1;
int64_t var_38_1 = 2;
Stack::Stack(&var_88, &var_48);
Stack::pop(&var_88);
Stack::pop(&var_88);
Stack::pop(&var_88);
Stack::pop(&var_88);
if (Stack::empty(&var_88) != 1)
{
    puts("[ERROR] Stack state changed");
    r14 = 1;
}
```

```
*(uint64_t*)((char*)arg5 - 0xd8) = __cxa_begin_catch(arg4);
printf("Catch exception: '%s'\n", *(uint64_t*)(**(uint64_t**)
__cxa_end_catch();
```

Безопасность относительно исключений

Гарантии безопасности исключений

- **Базовая гарантия**

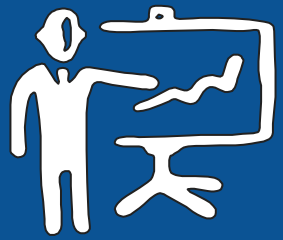
  После исключения классом возможно пользоваться.
  Из-за исключения не теряются ресуры (нет утечек памяти).

- **Строгая гарантия**

  Транзакционная семантика: либо операция прошла успешно, либо состояние объекта не изменилось.

- **Гарантия бессбойности** (noexcept)
  Исключительные ситуации невозможны.

Вопросы?



Красивые иконки взяты с сайта handdrawngoods.com