Получаем данные из функции — конспект темы

Варианты возврата — случай одного значения

Способы получить из функции данные — результат вычисления или отчёт о выполнении действия:

- возвращение через `return`,
- возвращение через параметр,
- исключение.

`return` позволяет выполнить Copy Elision и NRVO.

Исключения как способ возврата

Код обработчика проще читать и писать в случае исключений.

Исключение действительно должно быть неожиданной, исключительной ситуацией, которая может внезапно произойти и нарушить всю логику выполнения программы. Ожидаемая ошибка при работе одной определённой функции к исключительным ситуациям не относится, и лучше обрабатывать этот случай через распаковку или `variant`.

Не очень чистые функции — побочные эффекты

Назначение функции — вычисление или совершение действия. Результат вычисления возвращается оператором `return`, а результат действия — некоторое изменение во внешнем для функции мире. Оно называется **побочный эффект**.

Функция без побочных эффектов при повторных вызовах с одними и теми же аргументами будет выдавать один и тот же результат.

Функция без побочных эффектов должна принимать все параметры по константной ссылке, константному указателю либо по значению. Если эта функция — метод класса, он должен быть константным и не изменять `mutable` объекты этого класса.

В идеале у функция с побочными эффектами нет `return`. Пример такой функции — сеттер класса:

```
class Person {
public:
    void SetCat(Cat cat) {
        my_cat_ = std::move(cat);
    }

private:
    Cat my_cat_;
};
```

Побочный эффект может касаться параметров, передаваемых по неконстантной ссылке или указателю. Это ещё один способ возврата из функции. Таким способом можно заставить функцию возвращать несколько значений, и он действительно был популярным до появления распаковки, а в некоторых случаях и после.

Менеджер задач Практикума:

```
struct TaskPassResult {
    bool is_passed;
    // ...
};

class Task {
public:
    TaskPassResult RunCode(std::string_view code) const;
};

class TaskManager {
```

```
public:
    void RegisterTaskResult(int student_id, int task_id, TaskPassResult result);
    const Task& GetTask(int task_id) const;
};
```

Метод для прохождения задания и получения оценки за него можно реализовать в классе когорты так:

```
class Cohort {
public:
    void PassTask(TaskManager& task_manager, int student_id, int task_id, std::string_view solution_code) {
        auto result = task_manager.GetTask(task_id).RunCode(solution_code);
        if (result.is_passed) {
            SetTaskPassed(student_id, task_id);
        }
        task_manager.RegisterTaskPass(student_id, task_id, std::move(result));
    }

    void SetTaskPassed(int student_id, int task_id);
};
```

При вызове метода неясно, что он собирается изменять объект `TaskManager`:

```
cohort100.PassTask(task_manager, stident_id, task_id, code);
```

Чтобы сделать побочный эффект более явным, примените **декомпозицию** — разбиение задачи на подзадачи:

```
class Cohort {
public:
    // теперь TaskManager передаётся по константной ссылке
    [[nodiscard]] TaskPassResult PassTask(const TaskManager& task_manager, int student_id, int task_id, std::string_view solution_code) {
        auto result = task_manager.GetTask(task_id).RunCode(solution_code);
        if (result.is_passed) {
            SetTaskPassed(student_id, task_id);
        }
        return result;
    }

    void SetTaskPassed(int student_id, int task_id);
};

// ...
// метод не меняет task_manager, неявного побочного эффекта больше нет
auto result = cohort100.PassTask(task_manager, stident_it, task_id, code);
task_manager.RegisterTaskPass(student_id, task_id, std::move(result));
```

Чем меньше делает функция, тем она понятнее, тем более предсказуемо её поведение и тем лучше она соответствует своему названию.

Не смешивайте два назначения функции и чётко говорите о наличии побочного эффекта названием и сигнатурой. В C++ это можно делать особенно выразительно благодаря константности. Наличие `const` во всех нужных местах показывает отсутствие побочного эффекта.

Желательно, чтобы побочный эффект был виден в месте вызова. Корректность побочного эффекта должна быть проверена юнит-тестами наравне с корректностью возвращаемого значения.