Обработка ошибок. Исключения — конспект темы

Применяем класс optional

Опциональные значения — значения, которые могут быть представлены или не представлены. Они позволяют программе работать с переменными, которые в один момент времени хранят значение, а в другой остаются пустыми.

Опциональными значениями управляет шаблонный класс `Optional`. Опциональные значения — простая альтернатива кодам возврата, но их область применения этим не ограничивается.

```
// Функция возвращает пару корней квадратного уравнения либо пустое значение, когда решени
я нет
optional<pair<double, double>> SolveQuadraticEquation(double a, double b, double c) {
    double discriminant = b * b - 4 * a * c;
    if (discriminant < 0) {
        // Возвращаем специальное значение nullopt, означающее отсутствие значения
        return nullopt;
    }

    double x1 = (-b - sqrt(discriminant)) / (2 * a);
    double x2 = (-b + sqrt(discriminant)) / (2 * a);

    // Возвращаем корни уравнения
    return pair{x1, x2};
}
```

Чтобы проверить экземпляр `optional` на наличие в нём значения, используют метод `has_value`. С `optional` можно обращаться как со значением типа `bool`. Пустой объект `optional` будет вести себя как false, а непустой — как true.

```
optional<int> result;
...
if (result.has_value()) // либо просто: if (result)
{
    // result содержит значение
} else {
    // в переменной result пусто
}
```

Чтобы получить доступ к значению, хранящемуся внутри `optional`, используют метод `value` и унарный оператор `*`.

Если внутри `optional` хранится структура или класс, можно получить доступ к его полям напрямую. Для этого есть оператор `->`:

```
int main() {
    cout << "Введите коэффициенты уравнения a*x^2 + b*x + c = 0"s << endl;
    double a, b, c;
    cin >> a >> b >> c;
    // Вместо const optional<pair<double, double>> roots используем auto,
    // это позволит компилятору вывести тип переменной roots самостоятельно
    if (const auto roots = SolveQuadraticEquation(a, b, c);
        roots.has_value())
    {
        cout << "Корни уравнения "s << a << "*x^2 + "s << b << "*x + "s << c << " = 0"s <<
endl;
        // Доступ к значению можно получить методом roots.value() или (*roots)
        // Если внутри хранится структура или класс, доступ к его полям
        // можно получить, используя ->
        cout << "  x1="s << roots.value().first << "; x2="s << roots->second << endl;
    } else {
        cout << "Уравнение не имеет действительных корней"s << endl;
    }
}
```

Введение в исключения

Программа может сигнализировать об исключительной ситуации выбросом исключения — `throw`. У исключений нет недостатков, которые есть у кодов обработки ошибок. Синтаксис:

```
throw выражение
```

При выполнении выражения `throw`:

1. На основе выражения создаётся объект исключительной ситуации;
2. Управление передаётся в ближайший обработчик исключений, способный поймать выброшенное исключение;
3. Если подходящий обработчик исключения не найден, программа аварийно завершает работу.

Эта программа не успеет вывести текст и аварийно завершится, потому что возникающие внутри неё исключения никак не обрабатываются:

```
int main() {
    throw 42; // Выбрасываем значения 42 типа int в качестве объекта исключения
    cout << "Этот текст не будет выведен"s << endl;
}
```

Код, выбрасывающий исключения, должен выполняться внутри блока `try`, за которым следуют один или несколько блоков `catch`:

```
#include <iostream>

using namespace std;

void ThrowSomething() {
    int value;
    cin >> value;
    throw value;
}

int main() {
    // Внутри блока try могут быть выброшены ислючения
    try {
        ThrowSomething();
        cout << "Этот текст не будет выведен"s << endl;
    } catch (int i) {
        // Это обработчик исключений типа int
        cout << "Поймано целое число: "s << i << endl;
    } catch (double d) {
        cout << "Поймано вещественное число: "s << d << endl;
    } catch (...) {
        // В этот обработчик мы попадём, если ни один из предыдущих обработчиков не сработает
        cout << "Поймано исключение неизвестного типа"s << endl;
    }
    cout << "Выход из программы"s << endl;
}
```

В стандартной библиотеке C++ определены несколько классов стандартных исключений. Они объявлены в файле `<stdexcept>`.

Механизм исключений C++ делит выполнение программы на две части:

- работающую, если программа выполняется успешно;
- работающую, когда нормальное выполнение невозможно.

Раскрутка стека

При выбрасывании исключения происходит **раскрутка стека**: программа последовательно покидает вложенные блоки, пока не достигнет начала блока `try`. Если в текущем блоке `try` будет найден подходящий блок `catch`, управление передаётся в него. В противном случае процесс будет продолжаться.

Во время раскрутки стека вызываются деструкторы всех локальных переменных каждого блока в порядке, обратном вызову конструкторов.

В C++ жизненный цикл объекта начинается с вызова конструктора. После окончания работы конструктора инициализация объекта считается завершённой.

При выходе из области видимости объекта происходит его **деинициализация**. Она сопровождается вызовом деструктора.

Если во время работы конструктора выбрасывается исключение, инициализация объекта считается незавершённой. При этом деструктор такого объекта вызван не будет.