

## 3.3. Исключения в C++ (введение)

Исключение — это нестандартная ситуация, то есть когда код ожидает определенную среду и инварианты, которые не соблюдаются.

Банальный пример: функции, которая суммирует две матрицы, переданы матрицы разных размерностей. В таком случае возникает исключительная ситуация и можно «бросить» исключение.

### 3.3.1. Практический пример: парсинг даты в заданном формате

Допустим необходимо парсить даты

```
struct Date {  
    int year;  
    int month;  
    int day;  
}
```

из входного потока.

Функция ParseDate будет возвращать объект типа Date, принимая на вход строку:

```
Date ParseDate(const string& s){  
    stringstream stream(s);  
    Date date;  
    stream >> date.year;  
    stream.ignore(1);  
    stream >> date.month;  
    stream.ignore(1);  
    stream >> date.day;  
    stream.ignore(1);  
    return date;  
}
```

В этой функции объявляется строковый поток, создается переменная типа Date, в которую из строкового потока считывается вся необходимая информация.

Проверим работоспособность этой функции:

```
string date_str = "2017/01/25";  
Date date = ParseDate(date_str)  
cout << setw(2) << setfill('0') << date.day << '.'  
    << setw(2) << setfill('0') << date.month << '.'  
    << date.year << endl; // OUTPUT: "25.01.2017"
```

Код работает. Но давайте защитимся от ситуации, когда данные на вход приходят не в том формате, который ожидается:

```
2017a01b25
```

Программа выводит ту же дату на экран. В таких случаях желательно, чтобы функция явно сообщала о неправильном формате входных данных. Сейчас функция это не делает.

От этой ситуации можно защититься, изменив возвращаемое значение на `bool`, передавая `Date` в качестве параметра для его изменения, и добавляя внутри функции нужные проверки. В случае ошибки функция возвращает `false`. Такое решение задачи очень неудобное и существенно усложняет код.

### 3.3.2. Выброс исключений в C++

В C++ есть специальный механизм для таких ситуаций, который называется исключения. Что такое исключения, можно понять на следующем примере:

```
Date ParseDate(const string& s){
    stringstream stream(s);
    Date date;
    stream >> date.year;
    if (stream.peek() != '/') {
        throw exception();
    }
    stream.ignore(1);
    stream >> date.month;
    if (stream.peek() != '/') {
        throw exception();
    }
    stream.ignore(1);
    stream >> date.day;
    return date;
}
```

Если формат даты правильный, такой код отработает без ошибок:

```
string date_str = "2017/01/25";
Date date = ParseDate(date_str);
cout << setw(2) << setfill('0') << date.day << '.'
     << setw(2) << setfill('0') << date.month << '.'
     << date.year << endl;
```

Если сделать строчку невалидной, программа упадет:

```

string date_str = "2017a01b25";
Date date = ParseDate(date_str);
cout << setw(2) << setfill('0') << date.day << '.'
      << setw(2) << setfill('0') << date.month << '.'
      << date.year << endl;

```

Чтобы избежать дублирования кода, создадим функцию, которая проверяет следующий символ и кидает исключение, если это необходимо, а затем пропускает его:

```

void EnsureNextSymbolAndSkip(stringstream& stream) {
    if (stream.peek() != '/') {
        throw exception();
    }
    stream.ignore(1);
}

```

Функция ParseDate примет вид:

```

Date ParseDate(const string& s){
    stringstream stream(s);
    Date date;
    stream >> date.year;
    EnsureNextSymbolAndSkip(stream);
    stream >> date.month;
    EnsureNextSymbolAndSkip(stream);
    stream >> date.day;
    return date;
}

```

### 3.3.3. Обработка исключений. Блок try/catch

Ситуация когда программа падает во время работы не очень желательна, поэтому нужно правильно обрабатывать все исключения. Для обработки ошибок в C++ существует специальный синтаксис:

```

try {
    /* ...код, который потенциально
       может дать исключение... */
} catch (exception&) {
    /* Обработчик исключения. */
}

```

Проверим это на практике:

```

string date_str = "2017a01b25";
try {
    Date date = ParseDate(date_str);
    cout << setw(2) << setfill('0') << date.day << '.'
         << setw(2) << setfill('0') << date.month << '.'
         << date.year << endl;
} catch (exception& ex) {
    cout << "exception happens";
}

```

Хорошо бы донести до вызывающего кода, что произошло и где произошла ошибка. Например, если отсутствует какой-то файл, указать, что файл не найден и путь к файлу. Для этого есть класс `runtime_error`:

```

void EnsureNextSymbolAndSkip(stringstream& stream) {
    if (stream.peek() != '/') {
        stringstream ss;
        ss << "expected / , but has: " << char(stream.peek());
        throw runtime_error(ss.str());
    }
    stream.ignore(1);
}

```

Если у исключения есть текст, его можно получить с помощью метода `what` исключения.

```

} catch (exception& ex) {
    cout << "exception happens: " << ex.what();
}

```