

Объектно-ориентированное программирование

Лекция 5. Обработка исключений

П. А. Макаров



СЫКТЫВКАРСКИЙ
ЛЕСНОЙ
ИНСТИТУТ

03 ноября 2023 г.

1. Ошибочные ситуации и их обработка
2. Общая идея механизма обработки исключений
3. Возбуждение исключений
4. Обработка исключений
5. Обработчики с многоточием
6. Автоматическая очистка
7. Преобразования типов исключений
8. Обработка исключений с помощью специального класса
9. Контрольные вопросы и задания

Ошибочные ситуации и их обработка

Пример возникновения ошибочных ситуаций

```
1 #include <stdio.h>
2
3 int main(void) {
4     int x, y, z, status;
5
6     printf("Input 3 space-separated integers x, y, z: ");
7     status = scanf("%d %d %d", &x, &y, &z);
8
9     printf("Number of correctly processed arguments: %d\n",
10           status);
11    printf("The variables x, y, z have been assigned values:
12           %d, %d, %d\n", x, y, z);
13    return 0;
14 }
```

Листинг 1: Анализ ошибок, возможных при работе функции `scanf`

Ошибочные ситуации и их обработка

Типичные действия при возникновении ошибок

- прерывание выполнения программы (функция `exit`, прототип которой содержится в заголовочном файле `stdlib.h`);
- вывод значения, означающего ошибку (глобальная переменная `errno`, функции `strerror`, `perror` и `error`);
- вывод сообщения об ошибке в стандартный поток ошибок `stderr` и передача программе приемлемого значения, которое позволит ей продолжить работу.

Определение 1

Исключение в общем смысле слова — это возникновение аномальных условий, требующих специальной обработки во время выполнения программы. Исключение нарушает нормальный ход выполнения программы и приводит к выполнению предварительно сформулированного обработчика исключений.

Определение 2

*В более специальном смысле слова, **исключение** — это особый способ завершения функции, т. е. в языках программирования, поддерживающих исключения, функция может либо вернуть значение, либо возбудить исключение.*

Общая идея механизма обработки исключений

Некоторые замечания об исключениях

Замечание 1

Механизм исключений не является неотъемлемой частью ООП.

Преимущества механизма обработки исключений

- исключения невозможно проигнорировать;
- исключения выделяют обработку ошибок и восстановление после них из основного потока управления;
- при использовании в программах классов и объектов исключения иногда оказываются единственной возможностью обработки ошибок.

Замечание 2

Никогда не применяйте механизм обработки исключений для штатного выхода из вложенных управляющих конструкций или глубоких цепочек вызовов функций.

Общая идея механизма обработки исключений

Этапы обработки исключений

1. выделение контролируемого блока, т. е. блока, в котором может возникнуть исключительная ситуация — блок `try`;
2. генерация одного или нескольких исключений с помощью оператора `throw` внутри блока `try` или внутри функций, которые вызываются из этого блока;
3. размещение сразу за блоком `try` одного или нескольких обработчиков исключений `catch`.

Оператор

```
1 throw <expression>;
```

Листинг 2: Синтаксис оператора throw

возбуждает исключение, тип и значение которого определяются выражением <expression>.

В простейшем случае <expression> имеет один из стандартных типов, таких как целое число (`int`), С-строка (`const char*`) и т. д., но чаще всего в роли исключения используется объект специального класса.

Обработка исключений

Простейший пример

```
1 while(true) {
2     int x, y;
3     cout << "Input x and y: " << endl;
4     cin >> x >> y;
5     try {
6         if(y == 0)
7             throw 0;
8         cout << " x/y = " << (x / y) << endl;
9         break;
10    }
11    catch(int e) {
12        cout << "Error! Dividing by 0." << endl;
13    }
14 }
```

Листинг 3: Пример обработки исключения деления на 0

Обработка исключений

Правила обработки множественных исключений

1. Обработчики (catch-блоки) рассматриваются по порядку, один за другим, причём сработать может только один из них или ни одного.
2. Обработчик может поймать только исключение, возникшее во время работы соответствующего try-блока.
3. Если исключение не поймано ни одним из обработчиков, оно “выбрасывается” дальше, как если бы фрагмент кода, в котором исключение возникло, вовсе не был обрамлён никаким try-блоком.

Обработка исключений

Пример возбуждения множественных исключений

```
1 double** InverseMatrix(double** a, int m, int n) {
2     if(m != n)
3         throw "The matrix must be square!";
4     double det = Determinant (a, n);
5     if(det == 0.0)
6         throw 0;
7     // code for calculating the inverse matrix
8     // ...
9 }
```

Листинг 4: Возбуждение исключений в функции, вычисляющей обратную матрицу

Обработка исключений

Пример обработки множественных исключений

```
1 double** a;
2 int m, n;
3 // ...
4 try {
5     double** b = InverseMatrix(a, m, n);
6 }
7 catch(char* str) {
8     cout << str << endl;
9 }
10 catch(int e) {
11     cout << "The matrix is degenerate!" << endl;
12 }
```

Листинг 5: Обработка возможных исключений при вычислении обратной матрицы

Обработка исключений

Связанные исключения — 1

```
1 double** a, **b;
2 int m, n, k, l;
3 // ...
4 try {
5     double** c = MultiplyMatrix(a, m, n, b, k, l);
6     double** d = InverseMatrix(a, m, n);
7 }
8 catch(char* s) {
9     cout << s << endl;
10 }
11 catch(int i) {
12     if(i == 0)
13         cout << "The matrix is degenerate!" << endl;
14     if(i == 1)
15         cout << "Matrices of this size cannot be multiplied!" <<
16             endl;
17 }
```

Листинг 6: Первый вариант обработки связанных исключений

```
1 double** a, **b;
2 int m, n, k, l;
3 // ...
4 try {
5     double** c = MultiplyMatrix(a, m, n, b, k, l);
6 }
7 catch(int i) {
8     cout << "Matrices of this size cannot be multiplied!" <<
9         endl;
10 }
11 try {
12     double** d = InverseMatrix (a, m, n);
13 }
14 catch(char* s) {
15     cout << s << endl;
16 }
17 catch(int i) {
18     cout << "The matrix is degenerate!" << endl;
19 }
```

Листинг 7: Второй вариант обработки связанных исключений

```
1 void f(int n) {  
2     int *p = new int[n];  
3     // ...  
4     delete [] p;  
5 }
```

Листинг 8: Проблема обработки исключений при работе с динамической памятью

```
1 void f(int n) {
2     int *p = new int[n];
3     // ...
4     delete [] p;
5 }
```

Листинг 8: Проблема обработки исключений при работе с динамической памятью

```
1 void f(int n) {
2     int *p = new int[n];
3     try {
4         // ...
5     }
6     catch(...) {
7         delete [] p;
8         throw;
9     }
10    delete [] p;
11 }
```

Листинг 9: Решение данной проблемы

В Листинге 9 мы были вынуждены перехватывать, а после этого заново генерировать исключения произвольного вида, чтобы обеспечить корректное освобождение локально выделенной динамической памяти. Так приходится делать не всегда, так как работа с исключениями может быть упрощена за счёт использования механизма автоматической очистки, который состоит в следующем.

Замечание 3

Компилятор гарантирует вызов деструкторов для всех локальных объектов, у которых имеются деструкторы, прежде чем текущая функция завершится штатно либо по исключению.

Замечание 4

Целочисленные типы при обработке исключений не преобразуются друг к другу. Таким образом, обработчик `catch(int x)` не поймает исключение типа `short` или `long`.

Допустимые преобразования

- `throw any_type` \rightarrow `catch(any_type &);`
- `throw any_type *` \rightarrow `catch(const any_type *)`
не `throw const any_type *` \nrightarrow `catch(any_type *)!`;
- `throw any_type &` \rightarrow `catch(const any_type &)`
не `throw const any_type &` \nrightarrow `catch(any_type &)!.`

Пример 1

При открытии файлов может быть полезным передавать обработчику не только строку типа «open file error», но и имя соответствующего файла, а также значение переменной errno сразу после выполнения fopen для анализа возможных проблем.

Замечание 5

В классе, используемом для исключений, практически всегда обязан присутствовать конструктор копирования.

Обработка исключений с помощью специального класса

Пример — заголовок класса

```
1 class FileException {
2     int err_code;
3     char *filename;
4     char *comment;
5 public:
6     FileException(const char *fn, const char *cmt);
7     FileException(const FileException& other);
8     ~FileException();
9     const char *GetName() const { return filename; }
10    const char *GetComment() const { return comment; }
11    int GetErrno() const { return err_code; }
12 private:
13    static char *strdup(const char *str);
14 };
```

Листинг 10: Заголовок класса FileException

Обработка исключений с помощью специального класса

Пример — реализация класса

```
1 FileException::FileException(const char *fn, const char *cmt
    ) {
2     err_code = errno;
3     filename = strdup(fn);
4     comment  = strdup(cmt);
5 }
6 FileException::FileException(const FileException& other) {
7     err_code = other.errno;
8     filename = strdup(other.filename);
9     comment  = strdup(other.comment);
10 }
11 FileException::~FileException() {
12     delete[] filename;
13     delete[] comment;
14 }
15 char* FileException::strdup(const char *str) {
16     char *res = new char[strlen(str) + 1];
17     strcpy(res, str);
18     return res;
19 }
```

Листинг 11: Реализация класса FileException

Обработка исключений с помощью специального класса

Пример — использование класса

```
1 // ...
2 try {
3     // ...
4     FILE *f = fopen(file_name, "r");
5     if(!f)
6         throw FileException(file_name, "file open error");
7     // ...
8 }
9 // ...
10 catch(const FileException &ex) {
11     fprintf(stderr, "File exception: %s (%s): %s\n", ex.
12         GetName(), ex.GetComment(), strerror(ex.GetErrno()))
13     ;
14     return 1;
15 }
16 // ...
```

Листинг 12: Использование класса FileException

1. Изучите самостоятельно возможные методы обнаружения ошибок ввода при использовании объекта `std::cin`.
2. Почему не следует применять механизм обработки исключений для штатного выхода из вложенных управляющих конструкций или глубоких цепочек вызовов функций в программах на C++?
3. Приведите пример ситуации, когда функция `main` не сможет обработать возбуждённое исключение.
4. Что такое динамическая идентификация типов (RTTI — Run-Time Type Identification)? Какое это имеет отношение к обработке исключений?
5. Изучите самостоятельно особенности обработки исключений в языке Python.