ОБРАБОТКА ИСКЛЮЧЕНИЙ

- Понятие исключительной ситуации.
- Проблемы при описании и обработке исключительных ситуаций обычными средствами
- Идеальный механизм обработки исключений
- Обработка исключений в C++
- Восстановление процесса вычислений

ШАБЛОН ДЛЯ КОНТЕЙНЕРА «СТЕК»

```
template<class T> class Stack{
private:
    struct DataElement{
            T Value;
            struct DataElement *next;
    };
    DataElement *first;
public:
    Stack(){ first = NULL;}
    ~Stack(){ while(first != NULL) Pop();}
    void Push(T value){
      DataElement *d = new DataElement();
      d->Value = value;
      d->next = first;
      first = d;
    }
    T Pop(){
      if(first == NULL)
            Тут нужно что-то сделать!!!!;
      DataElement *d = first;
      first = first->next;
      T tmp = d->value;
      delete d;
      return tmp;
    }
};
```

ПОНЯТИЕ ИСКЛЮЧИТЕЛЬНОЙ СИТУАЦИИ

Под исключительной ситуацией (исключением) понимается состояние вычислительного процесса с относительно малой вероятностью его возникновения.

- **К исключительным относят некоторые ошибочные или крайние ситуации:**
  - сбои в аппаратуре,
  - ошибки пользователей в задании исходных данных для программы,
  - переполнение или потеря значимости в арифметических операциях,
  - несоответствие типа переменной типу присваиваемого этой переменной значения,
  - попытки получения доступа к защищенным данным или другим ресурсам,
  - нехватка данных во входном файле,
  - внешние воздействия на программу со стороны оператора ЭВМ или других программ
  - и т.п.
- Для обработки исключений, как правило, необходимо существенного изменить общий ход вычислительного процесса.
- Чтобы обеспечить продолжение работы программы или по крайней мере ограничить последствия исключительных ситуаций, программист должен предусматривать их специальную обработку.

СИНХРОННЫЕ И АСИНХРОННЫЕ ИСКЛЮЧЕНИЯ

- **Синхронные исключения – возникают при выполнении конкретного оператора**
  - **например, деление на 0**
- **Асинхронные исключения – могут возникнуть в любой момент времени**
  - **например, прерывание работы программы от пользователя, выход из строя аппаратуры**

- обычные операторы управления, обеспечивают соответствие статической и динамической структур любой программы:
  - выполнение любого законченного блока программы обычно начинается с первого оператора и заканчивается последним, расположенным в блоке;
  - если внутри блока описаны другие блоки, то они и выполняются во время выполнения исходного.
  - Обработка исключительной ситуации требует резкого изменения логики выполнения программы
- Для обработки исключений, если обычные управляющие конструкции и применимы, то они существенно усложняют статическую структуру программы, затемняют логику алгоритма.

ОБЫЧНЫЕ УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ ПЛОХО ПРИМЕНИМЫ ДЛЯ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Не все исключительные ситуации можно описать и проверить с помощью стандартных логических выражений с объектами, над которыми выполняются действия, приводящие к исключительным ситуациям.

К подобным ситуациям относятся:

- переполнение и потеря значимости в арифметических операциях,
- обнаружение нечислового символа при попытке ввода числа из входного файла
- и многие другие.

Такие ситуации называют **исключениями уровня выполнения,** поскольку они могут быть обнаружены только при выполнении соответствующего "опасного" оператора.

Все остальные исключительные ситуации называют ***исключениями программного уровня***, поскольку они могут быть обнаружены, обработаны и предотвращены обычными средствами языка программирования.

Исключения уровня выполнения могут быть обнаружены, обработаны и предотвращены и обычными средствами языка программирования, если операцию, которая может вызвать эту ситуацию, а также ее аргументы реализовать на более низком уровне.

НЕ ВСЕ ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ МОЖНО ПРЕДУПРЕДИТЬ С ПОМОЩЬЮ СТАНДАРТНЫХ ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ

Возникновение ряда исключительных ситуаций может быть не связанным с текущим состоянием вычислений или связанным лишь частично.

Например, в любой момент выполнения программы ей может быть передано указание о завершении работы.

На этот случай необходимо предусмотреть вариант ускоренного окончания работы программы с сохранением промежуточных результатов.

Обычными языковыми средствами такого рода ситуации запрограммировать невозможно.

ДЛЯ ОБРАБОТКИ АСИНХРОННЫХ ИСКЛЮЧЕНИЙ СТАНДАРТНЫЕ СРЕДСТВА УПРАВЛЕНИЯ НЕ ПРИГОДНЫ

Характер обработки исключительной ситуации, возникшей при выполнении процедуры, определяется не функцией этой процедуры, а конкретной задачей, решаемой посредством обращения к этой процедуре.

Например, возникновение ситуации потери значимости при решении системы уравнений в одних программах (для которых не требуется высокая степень надежности) должна приводить к завершению вычислений с предоставлением соответствующих диагностических сообщений, а в других (для которых требуется повышенная надежность) — к попытке решить систему уравнений другим методом посредством обращения к другой процедуре.

РАЗДЕЛЕНИЕ ОБНАРУЖЕНИЯ ПРОБЛЕМЫ И ЕЕ ОБРАБОТКА

Контроль исключительных ситуаций обычными средствами значительно увеличивает время работы программы даже в тех случаях, когда ситуации не возникают.

ПРОБЛЕМЫ ПРИ ОТСУТСТВИИ ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

ТРЕБОВАНИЯ К ИДЕАЛЬНОМУ МЕХАНИЗМУ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Механизм обработки исключительных ситуаций должен:

- обеспечивать обнаружение всех видов исключений, а также их обработку в соответствии с заданным алгоритмом.
- быть простым и удобным для программиста, предоставлять единообразное описание всех видов исключений, в том числе и уровня выполнения. Сами описания исключений и алгоритмов их обработки не должны затемнять описание основной части алгоритма.
- позволять описывать динамическую иерархию алгоритмов обработки исключений, предоставляющих возможность восстановления нормальной работы программы или выполнения других подобных действий в зависимости от контекста.
- не приводить к существенному увеличению времени работы программы, когда исключительные ситуации, предусмотренные в программе, не возникают.

ОБРАБОТКА ИСКЛЮЧЕНИЙ В РАЗЛИЧНЫХ ЯЗЫКАХ

Пересылка вспомогательного параметра, который используется в качестве переменной состояния.

- Используется в библиотечных функциях С

Передача в подпрограмму метки в качестве параметра.

- можно сделать только в языках, допускающих использование меток в качестве параметра.
- Передача метки позволяет вызываемому модулю передать управление в другую точку вызывающего модуля, если возникает исключительная ситуация.
- Такое использование меток является общепринятым в языке FORTRAN.

Третий способ — создать обработчик в виде отдельной подпрограммы, передаваемой в качестве параметра в вызываемый модуль.

- Одна из проблем, связанных с таким подходом, заключается в том, что передачу подпрограммы-обработчика следует выполнять при *каждом* вызове, независимо от того, желательно это или нет.
- Более того, чтобы обрабатывать несколько видов исключительных ситуаций, необходимо было бы передавать несколько разных модулей-обработчиков, усложняя код.

ОБРАБОТКА ИСКЛЮЧЕНИЙ В C++

Исключения C++ не поддерживают обработку асинхронных событий, таких, как ошибки оборудования или обработку прерываний, например, нажатие клавиш Ctrl+C.

Механизм исключений предназначен только для событий, которые происходят в результате работы самой программы и указываются явным образом.

Исключения возникают тогда, когда некоторая часть программы не смогла сделать то, что от нее требовалось. При этом другая часть программы может попытаться сделать что-нибудь иное.

Исключения позволяют логически разделить вычислительный процесс на две части — обнаружение аварийной ситуации и ее обработка.

Главной причиной является то, что функция, обнаружившая ошибку, может не знать, что предпринимать для ее исправления, а использующий эту функцию код может знать, что делать, но не уметь определить место возникновения.

Это особенно актуально при использовании библиотечных функций и программ, состоящих из многих модулей.

Для передачи информации об ошибке в вызывающую функцию не требуется применять возвращаемое значение, параметры или глобальные переменные, поэтому интерфейс функций не раздувается.

Место, в котором может произойти ошибка, должно входить в контролируемый блок — составной оператор, перед которым записано ключевое слово try.

Этапы обработки исключительных ситуаций:

Обработка исключения начинается с появления ошибки. Функция, в которой она возникла, генерирует исключение. Для этого используется ключевое слово throw с параметром, определяющим тип исключения. Параметр может быть константой, переменной или объектом и используется для передачи информации об исключении его обработчику.

Отыскивается соответствующий обработчик исключения и ему передается управление.

Если обработчик исключения не найден, вызывается стандартная функция terminate, которая вызывает функцию abort, аварийно-завершающую текущий процесс. Можно установить собственную функцию завершения процесса.

ОБЩИЙ МЕХАНИЗМ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

КОНТРОЛИРУЕМЫЙ БЛОК

Ключевое слово try служит для обозначения ***контролируемого блока*** — кода, в котором может генерироваться исключение. Блок заключается в фигурные скобки:

```
try{
    ...
}
```

Все функции, прямо или косвенно вызываемые из try-блока, также считаются ему принадлежащими.

ПОРОЖДЕНИЕ ИСКЛЮЧЕНИЯ

Генерация (порождение) исключения происходит по ключевому слову **throw**, которое употребляется либо с параметром, либо без него:

throw [выражение];

Тип выражения, стоящего после throw, определяет **тип порождаемого исключения**.

При генерации исключения **выполнение текущего блока прекращается**, и происходит поиск соответствующего обработчика и передача ему управления.

Не всегда исключение, возникшее во внутреннем блоке, может быть сразу правильно обработано. В этом случае используются вложенные контролируемые блоки, и исключение передается на более высокий уровень с помощью ключевого слова **throw** без параметров.

ПРИМЕР ПОРОЖДЕНИЯ ИСКЛЮЧЕНИЯ

```
template<class T> class Stack{
private:
    struct DataElement{
            T Value;
            struct DataElement *next;
    };
    DataElement *first;
public:
    Stack(){ first = NULL;}
    ~Stack(){ while(first != NULL) Pop();}
    void Push(T value){
      DataElement *d = new DataElement();
      d->Value = value;
      d->next = first;
      first = d;
    }
    T Pop(){
      if(first == NULL)
            throw «Стек пуст»;
      DataElement *d = first;
      first = first->next;
      T tmp = d->value;
      delete d;
      return tmp;
    }
};
```

ОБРАБОТЧИКИ ИСКЛЮЧЕНИЙ

Обработчики исключений начинаются с ключевого слова catch, за которым в скобках следует тип обрабатываемого исключения. Они должны располагаться непосредственно за try-блоком.

Можно записать один или несколько обработчиков в соответствии с типами обрабатываемых исключений. Синтаксис обработчиков напоминает определение функции с одним параметром — типом исключения.

Существует три формы записи:

catch(тип имя){ ... /* тело обработчика */ }

имя параметра используется в теле обработчика для выполнения каких-либо действий — например, вывода информации об исключении.

catch(Тип){ ... /* тело обработчика */ }

не предполагает использования информации об исключении, играет роль только его тип

catch(...){ ... /* тело обработчика */ }

Многоточие вместо параметра обозначает, что обработчик перехватывает все исключения.

Так как обработчики просматриваются в том порядке, в котором они записаны, обработчик третьего типа следует помещать после всех остальных.

ЗАВЕРШЕНИЕ ОБРАБОТКИ ИСКЛЮЧЕНИЯ

- ***После обработки исключения*** **управление передается первому оператору, находящемуся непосредственно за обработчиками исключений.**
- **Туда же, минуя код всех обработчиков, передается управление, если исключение в try-блоке не было сгенерировано.**

ПЕРЕХВАТ ИСКЛЮЧЕНИЙ

Термин стек вызовов обозначает последовательность вызванных, но еще не завершившихся функций.

Раскручиванием стека называется процесс освобождения памяти из-под локальных переменных и возврата управления вызывающей функции.

Когда функция завершается, происходит естественное раскручивание стека. Тот же самый механизм используется и при обработке исключений.

Когда с помощью throw генерируется исключение, функции исполнительной библиотеки C++ выполняют следующие действия:

1) **создают копию параметра throw в виде статического объекта, который существует до тех пор, пока исключение не будет обработано;**
2) **в поисках подходящего обработчика раскручивают стек, вызывая деструкторы локальных объектов, выходящих из области действия;**
3) **передают объект и управление обработчику, имеющему параметр, совместимый по типу с этим объектом.**

Механизм исключений позволяет корректно уничтожать объекты при возникновении ошибочных ситуаций.

Исключение может быть как стандартного, так и определенного пользователем типа. При этом нет необходимости определять это тип глобально — достаточно, чтобы он был известен в точке порождения исключения и в точке его обработки.

Класс для представления исключения можно объявить внутри класса, при работе с которым оно может возникать.

Конструктор копирования этого класса должен быть объявлен как public, поскольку иначе будет невозможно создать копию объекта при генерации исключения.

СВОЙСТВА МЕХАНИЗМА ОБРАБОТКИ ИСКЛЮЧЕНИЙ C++

Язык C++ не позволяет возвращать значение из конструктора и деструктора.

Механизм исключений дает возможность сообщить об ошибке, возникшей в конструкторе или деструкторе объекта.

Этот же механизм можно использовать в функциях, которые не возвращают значений или должны возвращать значения определенного типа.

ИСКЛЮЧЕНИЯ В КОНСТРУКТОРАХ И ДЕСТРУКТОРАХ

СВОЙСТВА ИСКЛЮЧЕНИЙ В КОНСТРУКТОРЕ

Если в конструкторе объекта генерируется исключение, автоматически вызываются деструкторы для полностью созданных в этом блоке к текущему моменту объектов, а также для полей данных текущего объекта, являющихся объектами, и для его базовых классов.

Например, если исключение возникло при создании массива объектов, деструкторы будут вызваны только для успешно созданных элементов.

Если объект создается в динамической памяти с помощью операции new и в конструкторе возникнет исключение, память из-под объекта корректно освобождается.

Использование собственных классов исключений предпочтительнее применения стандартных типов данных.

С помощью классов можно более гибко организовать передачу информации об исключении, легче дифференцировать обработку исключений, а кроме того, появляется возможность использовать иерархии классов.

Поскольку механизм управления исключениями позволяет создать обработчик для базового класса, родственные исключения часто можно представить в виде иерархии.

Производя исключения от общего базового класса, можно в обработчике перехватывать ссылку или указатель на базовый класс, используя полиморфизм.

В зависимости от обстоятельств можно использовать либо обработчик исключений базового класса, который будет перехватывать и производные исключения, либо собственные обработчики производных классов.

ИЕРАРХИИ ИСКЛЮЧЕНИЙ

```
class Matherr{};
class Overflow: public Matherr{}; // Переполнение
class Underflow: public Matherr{}; // Исчезновение порядка
class ZeroDivide: public Matherr{}; // Деление на ноль
class IOerr{};
class Readerr:   public IOerr{}; // Ошибка чтения
class Writerr:   public IOerr{}; // Ошибка записи
class Seekerr:   public IOerr{}; // Ошибка поиска
int main(){
    try{
        …
    }
    catch(IOErr){
        …
    }
    catch(Overflow){
        …
    }
}
```

ПРИМЕР ИЕРАРХИИ ИСКЛЮЧЕНИЙ

ВИДЕО УРОК

https://youtu.be/gtKUIbceN5o