# Изключения

Любомир Чорбаджиев
Технологическо училище "Електронни системи"
Технически университет, София
lchorbadjiev@elsys-bg.org
$Revision$ : 1.1

27 февруари 2005 г.

## Обработка на грешки

Има различни "философски" подходи към обработката на грешки.

- *"Това не може да се случи"* — нека не се тревожим за ситуации, които не могат да се случат.
- *"Ненамеса"* — нека другите се оправят с грешките. Обикновено води до ненормално състояние на програмата, поради което тя "умира" на друго място.
- *"Грешките са навсякъде"* — вграждане на код за обработка на грешките навсякъде в програмата.
- *"Обработка на изключения"* — грешките са изключения. Позволява да се оптимизира "нормалното" поведение на програмата.

# Обработка на грешки

```
1  #include <cstdio>
2  using namespace std;
3
4  int main(void) {
5    FILE* infile=fopen("temp1.txt","r");
6    if(infile==NULL) {
7      printf("error␣opening␣infile␣'temp1.txt'\n");
8      return 1;
9    }
10   FILE* outfile=fopen("temp2.txt","w");
11   if(outfile==NULL) {
12     printf("error␣opening␣outfile␣'temp2.txt'\n");
13     fclose(infile);
14     return 1;
15   }
```



```
16   int ch=-1;
17   while((ch=fgetc(infile))!=EOF){
18     fputc(ch,outfile);
19
20   }
21   fclose(infile);
22   fclose(outfile);
23   return 0;
24 }
```

# Обработка на изключения

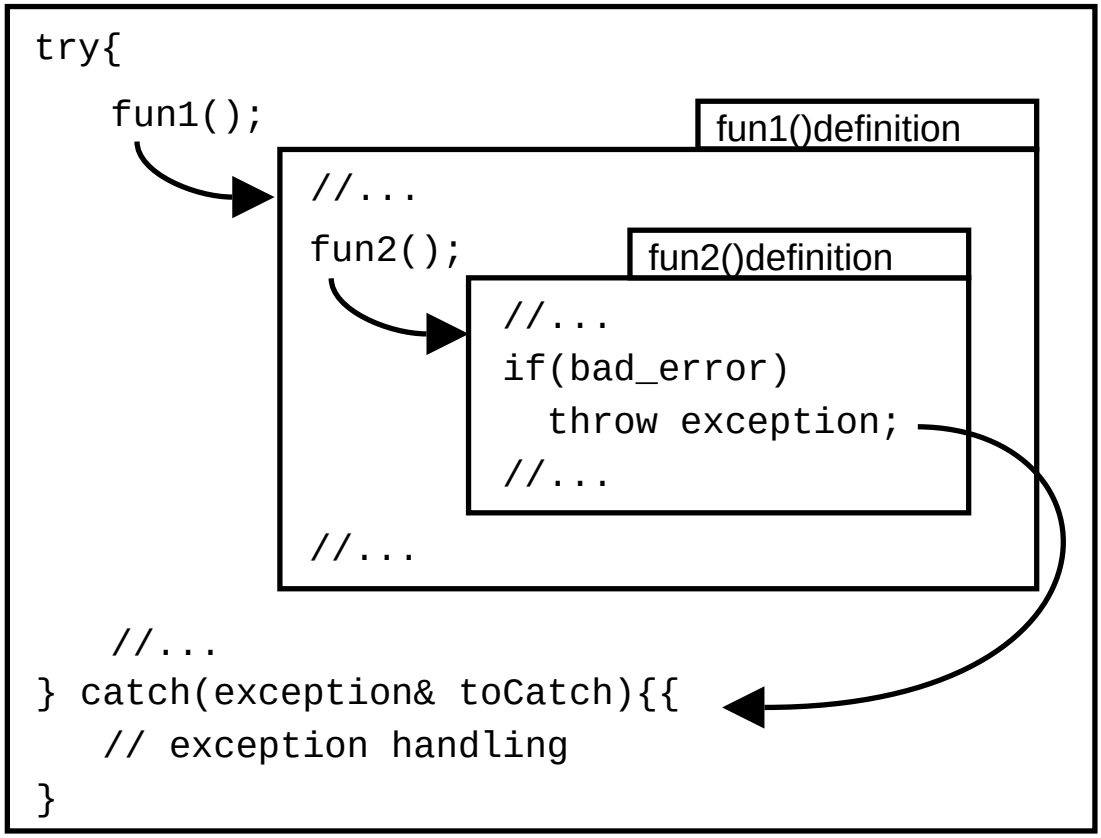
- Механизмът на изключенията в C++ е:
  - ◇ разработен за обработка на грешки и други изключителни ситуации;
  - ◇ предназначен за обработка на синхронни изключения — грешки при вход/изход, излизане извън границите на масив, и т.н.;
  - ◇ не е предназначен за обработка на асинхронни събития — движение на мишката, обработка на прекъсванията и т.н.;
- Изключенията не трябва да се използват като управляващи структури (не са оптимизарани за такава употреба);



# Обработка на изключения

- Обработката на изключенията в C++ е разделена на две части:
  - ◇ Функцията, която открие грешка по време на изпълнението си, но не знае как да се справи с нея (как да я обработи) **генерира изключение** (**throw**);
  - ◇ Функцията, която може да се справи с проблем от даден тип **прихваща** (**catch**) тези изключения и ги обработва;

# Развиване на стека (Stack Unwind)






# Генериране на изключения

```
1 void fun(void) {
2   //...
3   if(bad_error)
4     throw exception();
5   //...
6 }
```

# Обработка на изключения

```
try {
  //...
  fun();
  //...
}
catch(exception){
  // recovery/clean up
}
catch(other_exception){
  // ...
}
```



# Пример: Генериране и обработка на изключения

```
#include <iostream>
#include <string>
using namespace std;

class DivisionByZeroError{
  string message_;
public:
  DivisionByZeroError(void)
    : message_("division␣by␣zero")
  {}
  const string& what(void) const {
    return message_;
  }
};
double quotient(int n, int d){
  if(d==0)
    throw DivisionByZeroError();
  return static_cast<double>(n)/d;
};
```

```
20 int main(int argc, char* argv[]){
21   int n, d;
22   double result;
23
24   cout<<"Enter␣two␣ingegers␣(EOF␣for␣end)"<<endl;
25   while(cin >> n >> d){
26     try {
27       result=quotient(n,d);
28       cout<<"quotient:"<<result<<endl;
29     }
30     catch(const DivisionByZeroError& ex){
31       cout<<"exception␣catched:␣"<<ex.what()<<endl;
32     }
33     cout<<"Enter␣two␣ingegers␣(EOF␣for␣end)"<<endl;
34   }
35   return 0;
36 }
```

```
Enter two ingegers (EOF for end)
5 7
quotient:0.714286
Enter two ingegers (EOF for end)
7 0
exception catched: division by zero
Enter two ingegers (EOF for end)
8 2
quotient:4
Enter two ingegers (EOF for end)
```

# Генериране на изключения

- За генериране на изключение в C++ се използва ключовата дума **throw**;
  - ◇ Обикновено **throw** има един операнд; в по-редки ситуации **throw** може да се използва без операнди;
  - ◇ Най-често за операнди на **throw** се използват обекти;
- Генерираното изключение се прихваща от най-близкия **catch** блок;
- Изпълнението на блока, където е генерирано изключението, спира;
- Управлението на програмата се предава на най-близкият **catch** блок, който успее да прихване изключението;



# Обработка на изключение

- Обработката на изключение се извършва в **catch** блок;
- Кодът, който може да генерира изключение, трябва да бъде разположен в **try** блок;
- Синтаксиса на **try–catch** блока е:

```
try {
  // код, генериращ изключения;
}
catch( ExceptionType parameter ){
  // код, обработващ изключенията;
}
```

# Обработка на изключение

- Изключението се прихваща, само ако неговият тип съответства на дефинирания в **catch** блока тип на изключението (ExceptionType);
- Когато някое изключение не се прихване от нито един **catch** блок се извиква функцията `terminate()`; по подразбиране тази функция вика системната функция `abort()`;



# Обработка на изключение

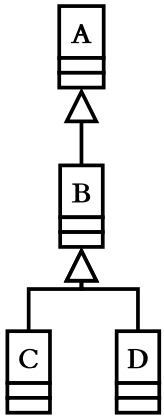
- Изключенията, които се прихващат от даден **catch** блок:

```
catch( ExceptionType parameter ){
  // exception handling code
}
```

  са:

  - ◇ Типът на изключението съвпада напълно с типа на параметъра <ExceptionType> в **catch** блока;
  - ◇ Типът на параметъра в **catch** блока <ExceptionType> е базов клас за типа на изключението;

# Пример: Обработка на изключение




```
1 try {
2   // код, генериращ
3   // изключения от типовете
4   // A, B, C и D
5 } catch(const B& toCatch) {
6   //...
7 } catch(const A& toCatch) {
8   //...
9 }
```



# Повторно генериране на изключение

- Използва се, когато прихванатото изключение не може да бъде обработено;
- Синтаксис:

```
catch(ExceptionType parameter){
  // ...
  throw;
}
```

- Повторно генериране на изключение може да се изпълни само в рамките на **catch**-блок;
- Повторно генерираното изключение се обработва от следващият **catch**-блок;

# Пример: Повторно генериране на изключение

```
1  #include <iostream>
2  #include <exception>
3  using namespace std;
4
5  void fun(void) {
6    try {
7      cout<<"Exception␣thrown␣in␣fun()"<<endl;
8      throw exception();
9      cout<<"This␣should␣not␣be␣printed"<<endl;
10   }
11   catch(exception& ex){
12     cout<<"Exception␣handled␣in␣fun()"<<endl;
13     throw;
14   }
15   cout<<"This␣should␣not␣be␣printed"<<endl;
16 }
```



```
17 int main(int argc,char* argv[]){
18
19   try {
20     fun();
21     cout<<"This␣should␣not␣be␣printed"<<endl;
22   }
23   catch(const exception& ex){
24     cout<<"Exception␣handled␣in␣main()"<<endl;
25   }
26
27   cout<<"Program␣can␣continue"<<endl;
28
29   return 0;
30 }
```

```
Exception thrown in function fun()
Exception handled in function fun()
Exception handled in main()
Program can continue
```

# Спецификация на изключенията

- Дефинира какви изключения могат да бъдат генерирани от дадена функция.
- Синтаксис:

```
void fun(void) throw (exception1, exception2) {
  /*...*/
}
```

- Функцията може да генерира изключения само от изброените в спецификацията типове или изключения, чийто тип е наследник на някой от специфицираните.



# Спецификация на изключенията

- Ако функцията генерира друг тип изключение (такъв който не е в спецификацията), се извиква функцията `unexpected()`.
- Ако спецификацията е **throw**(), то това означава, че функцията не трябва да генерира изключения; ако се генерира какво да е изключение се извиква функцията `unexpected()`.
- Ако функцията няма **throw** спецификация, то това означава, че тя може да генерира всякакви изключения.

# Обработка на неочаквани изключения

- Функцията `std::unexpected()`:
  - ◇ Вика се когато се генерира изключение, чийто тип не е включен в спецификацията на изключенията.
  - ◇ По подразбиране, поведението на `std::unexpected()` е да извика `std::terminate()`.
  - ◇ Поведението на `std::unxpected()` може да се промени чрез `std::set_unexpected()`.



# Обработка на неочаквани изключения

- Функцията `std::terminate()`:
  - Вика се когато някое генерирано изключение не бъде обработено от нито един **catch** блок.
  - По подразбиране поведението на `std::terminate()` е да извика `abort()`.
  - Поведението на `std::terminate()` може да се промени чрез `std::set_terminate()`.

# Изключения в конструкторите

- Генерирането на изключения дава възможност да се предаде информация за грешка, при работата на конструктора.
- Ако в конструктора се генерира изключение, то автоматично се викат:
  - ◇ деструкторите на всички член-променливи, които са инициализирани преди генерирането на изключение;
  - ◇ деструкторите на базовите класове, чиито конструктори са завършили работата си преди генериране на изключението.
- Генерирането на изключения в копиращите конструктори и операторите за присвояване трябва да се избягва.



# Изключения в деструкторите

- Когато деструкторът се вика в процеса на обработка на някакво изключение, т.е. при развиване на стека, то деструкторът не трябва да генерира изключения.
- Ако в такъв случай деструкторът все пак генерира изключение, то автоматично се вика `std::terminate()`.

# Изключения и управление на ресурсите

Пример:

```
1  FILE* f=fopen(filename,"w");
2  // code, throwing exceptions
3  fclose(f);
```

Пример:

```
1  FILE* f=fopen(filename,"w");
2  try {
3    // code, throwing exceptions
4  }
5  catch(...){
6    fclose(f);
7    throw;
8  }
9  //...
10 fclose(f);
```



**Решение:** в конструкторът заемаме ресурса, в деструктора го освобождаваме:

```
1  class File_ptr {
2    FILE* pf_;
3  public:
4    File_ptr(const char* name, const char* a){
5      pf_=fopen(name,a);
6    }
7    ~File_ptr(void){
8      fclose(pf_);
9    }
10   operator FILE*(){return pf_;}
11 };
12 File_ptr f(filename,"w");
13 // code, throwing exceptions
```

# Изключения и управление на ресурсите

Пример:

```
1  int* ptr=new[100];
2  // code, throwing exceptions
3  delete [] ptr;
```

Пример:

```
1  int* ptr=new[100];
2  try {
3    // code, throwing exceptions
4  }
5  catch(...){
6    delete [] ptr;
7    throw;
8  }
9  //...
10 delete [] ptr;
```

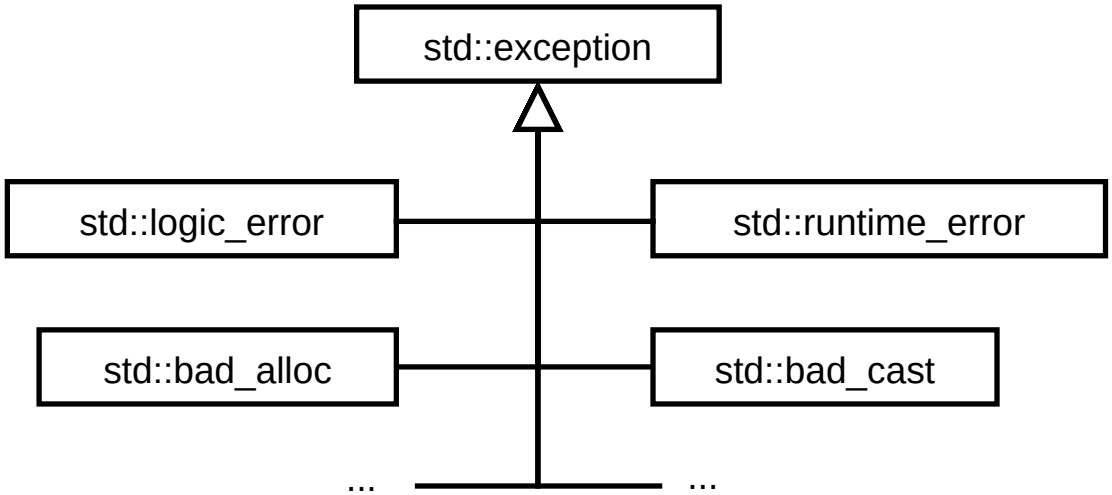


**Решение:** в конструкторът заемаме ресурса, в деструктора го освобождаваме:

```
1  class Mem_ptr {
2    int* ptr_;
3  public:
4    Mem_ptr(unsigned size){
5      ptr_=new[size];
6    }
7    ~Mem_ptr(void){
8      delete [] ptr_;
9    }
10   operator int*(){return ptr_;}
11 };
12 Mem_ptr mptr(100);
13 // code, throwing exceptions
```

# Стандартни изключения

- Дефинирани са в `<exception>`;




# Използване на изкключения: маєив с проверка на границите



```
1  #include <iostream>
2  using namespace std;
3
4  class IndexOutOfBounds {};
5
6  class Array {
7    unsigned int size_;
8    int* data_;
9  public:
10   Array(unsigned int size=10)
11     : size_(size), data_(new int[size_])
12   {}
13   Array(const Array& other)
14     : size_(other.size_), data_(new int[size_])
15   {
16     for(unsigned int i=0; i< size_; i++)
17       data_[i]=other.data_[i];
18   }
19   ~Array(void) {
20     delete [] data_;
21   }
```

```
22    unsigned size() const {
23      return size_;
24    }
25    Array& operator=(const Array& other) {
26      if(this!=&other) {
27        delete [] data_;
28        size_=other.size_;
29        data_=new int[size_];
30        for(unsigned i=0;i<size_;i++)
31          data_[i]=other.data_[i];
32      }
33      return *this;
34    }
35    int& operator[](unsigned int index)
36      throw (IndexOutOfBounds)
37    {
38      if(index>=size_) {
39        throw IndexOutOfBounds();
40      }
41      return data_[index];
42    }
43  };
```

```
44  int main(void) {
45    Array a1(3), a2;
46    for(int i=0;i<3;++i) {
47      a1[i]=i;
48    }
49    a2=a1;
50    for(int i=0;i<3;i++) {
51      cout << "a2[" << i << "]=" << a2[i] << endl;
52    }
53    try {
54      cout << "a2[" << 3 << "]=" << a2[3] << endl;
55    } catch(IndexOutOfBounds toCatch) {
56      cerr << "IndexOutOfBounds␣exception␣catched..."
57           << endl;
58    }
59    return 0;
60  }
```

```
lubo@dobby:~/school/cpp/notes> ./a.out
a2[0]=0
a2[1]=1
a2[2]=2
IndexOutOfBounds exception catched...
```