

Примерни програми

Любомир Чорбаджиев¹
lchorbadjiev@elsys-bg.org

¹Технологическо училище “Електронни системи”
Технически университет, София

Revision : 1.4

\$Date: 2005/10/11 11:01:20 \$

Съдържание

- 1 Обработка на изключения
- 2 Рационални числа: `class Rational`
- 3 Псевдо-случайни числа: `namespace Random`
- 4 Примери за използване на `namespace Random`

Обработка на грешки

- По време на изпълнение на програмата дадена функция може да открие възникването на ненормална, грешна ситуация.
- Причината за възникването на такава ситуация може да бъде различна — неправилни входни данни, препълване на диска, изчерпване на наличната динамична памет, невъзможност да се отвори файл и т.н.
- По какъв начин функцията трябва да реагира на такава ситуация?

Обработка на грешки

- C-подход: функцията, открила ненормална ситуация да върне резултат, който сигнализира за наличието на грешка.
- Голяма част от функциите в стандартната C библиотека са организирани точно по този начин.

```
FILE* fopen(const char* filename,  
            const char* mode);  
int fputc(int c, FILE* file);  
int fputs(const char* str, FILE* file);  
int fgetc(FILE* file);
```

Обработка на грешки в класа Stack

Първоначална версия — липсва обработка на грешки.

```
1 class Stack {  
2     ...  
3     public:  
4     ...  
5     void push(int val) {  
6         if(top_ < STACK_SIZE) {  
7             data_[top_++] = val;  
8         }  
9     }  
10    ...  
11 };
```

Обработка на грешки в класа Stack

```
1  int push(int val) {  
2      if(top_ < STACK_SIZE) {  
3          data_[top_++] = val;  
4          return 0;  
5      }  
6      return -1; // Грешка: стека е пълен  
7  }
```

Обработка на грешки в класа Stack

Първоначална версия — липсва обработка на грешки.

```
1 class Stack {
2     ...
3 public:
4     ...
5     int pop(void) {
6         if(top_>0) {
7             return data_[--top_];
8         }
9         return 0;
10    }
11    ...
12 };
```

Обработка на грешки в класа Stack

```
1  int pop(int& val) {  
2      if(top_>0) {  
3          val=data_[--top_];  
4          return 0;  
5      }  
6      return -1; // Грешка: стека е празен  
7  }
```


Обработка на грешки

- Разгледаният подход за обработка на грешки е тежък и тромав.
- При всяко извикване на функция, резултатът от тази функция трябва да се изследва за възможни настъпили грешки. Това прави кода на програмата труден за разбиране и поддържане.
- Друг недостатък на разглеждания подход е, че в него няма стандарти. Това прави трудно еднотипното обработване на грешки.

Генериране и обработка на изключения

- Механизмът за обработката на изключения в C++ предоставя стандартни, вградени в езика средства за реагиране на ненормални, грешни ситуации по време на изпълнение програмата.
- Механизмът на изключенията предоставя еднообразен синтаксис и стил за обработка на грешки в програмата.
- Елиминира нуждата от изрични проверки за грешки и съсредоточава кода за обработка на грешки в отделни части на програмата.

Генериране на изключение

- При възникване на ненормална ситуация в програмата, програмистът сигнализира за настъпването ѝ чрез генерирането на изключение.
- Когато се генерира изключение, нормалното изпълнение на програмата се прекратява докато изключението не бъде обработено.
- В C++ за генериране на изключение се използва ключовата дума **throw**.

Генериране на изключение

```
1 class StackError { ... };
2 class Stack {
3     ...
4     public:
5     ...
6     int pop(void) {
7         if(top_ <= 0)
8             throw StackError;
9         return data_[--top_];
10    }
11    ...
12 }
```

Обработване на изключение

- Най-често изключенията в програмата се генерират и обработват от различни функции.
- След като изключението бъде обработено изпълнението на програмата продължава нормално. Възстановяването на изпълнението на програмата обаче става не от точката на генериране на изключението, а от точката, където изключението е било обработено.
- В C++ обработката на изключенията се изпълнява в **catch**-секции.

```
1 catch (StackError ex) {  
2     log_error(ex);  
3     exit(1);  
4 }
```

Обработване на изключение

- Всяка една **catch**-секция трябва да се асоциира с **try**-блок. В един **try**-блок се групират един или повече оператори, които могат да генерират изключения с една или повече **catch**-секции.

```
1  try {  
2      // Използване на обекти от класа Stack  
3      ...  
4  } catch (StackError ex) {  
5      // Обработка на грешка при използването на стека  
6      ...  
7  } catch (...) {  
8      // Обработка на всички останали грешки  
9      ...  
10 }
```

Обработване на изключение

```
try{  
    ...  
    a_function();  
    ...  
    st.push(42);  
    ...  
    if(top_>=STACK_SIZE)  
        throw StackError;  
    ...  
    ...  
} catch(StackError toCatch){  
    // handling StackError  
} catch(...) {  
    // handling other exceptions  
}
```

The diagram illustrates the flow of an exception. A dashed box encloses the code within the `try` block, specifically the `a_function()` call and the `st.push(42);` statement. Inside this box, another dashed box highlights the `if` statement that throws a `StackError`. An arrow points from the `st.push(42);` line to the `try` block boundary. A second arrow points from the `throw StackError;` line to the `catch(StackError toCatch){` block, indicating that the exception is caught and handled there.

Пример за използване на изключения

```
1  class StackError {};  
2  const int STACK_SIZE=10;  
3  class Stack {  
4      int data_[STACK_SIZE];  
5      int top_;  
6  public:  
7      Stack() {  
8          top_=0;  
9      }  
10     void push(int val) {  
11         if(top_>=STACK_SIZE)  
12             throw StackError();  
13         data_[top_++]=val;  
14     }
```


Пример за използване на изключения

```
15  int pop(void) {
16      if(top_ <=0)
17          throw StackError();
18      return data_[--top_];
19  }
20  bool is_empty() {
21      return top_==0;
22  }
23  bool is_full() {
24      return top_==STACK_SIZE;
25  }
26  };
```

Пример за използване на изключения

```
27 #include <cstdlib>
28 #include <iostream>
29 using namespace std;
30 int main(int arch, char* argv[]) {
31     char* msg="Hello_Cruel_World!";
32     char buff[10];
```

```
33     try {
34         Stack st;
35         for(char* p=msg;*p!='\0';p++)
36             st.push(*p);
37         char* p=buff;
38         while(!st.is_empty())
39             *p+=st.pop();
40         *p='\0';
41     } catch(StackError ex) {
42         cerr<<"StackError caught..."<<endl;
43         exit(1);
44     } catch(...) {
45         cerr<<"Unknown error caught..."<<endl;
46         exit(1);
47     }
48     return 0;
49 }
```

Дефиниция на рационални числа

- Множеството на рационалните числа представлява множеството на частните a/b , където a и b са цели числа и $b \neq 0$. Числото a се нарича *числител*, а числото b – *знаменател*.
- Примери за рационални числа:

$$\frac{1}{2}, \frac{5}{4}, \frac{-6}{2}, \frac{-3}{-4}$$

- Рационалните числа се представят от отношението между числителя и знаменателя. Следователно, всяко рационално число може да бъде представено по различни начини.

Например

$$\frac{3}{4} = \frac{6}{8} = \frac{12}{16} = \frac{15}{20} = \frac{18}{24} \dots$$

са различни представяния на едно и също рационално число.

Редуцирана форма

Нека е дадено рационалното число $\frac{a}{b}$. *Редуцирана форма* на това рационално число се нарича прествянето във вида $\frac{a'}{b'} = \frac{a}{b}$, за което е изпълнено следното:

$$a' = \frac{a}{\text{GCD}(a, b)}, \quad b' = \frac{b}{\text{GCD}(a, b)}, \quad (1)$$

където $\text{GCD}(a, b)$ е най-големият общ делител на a и b .

Алгоритъм на Евклид за намиране на НОД

Има различни алгоритми за намиране на най-голям общ делител (НОД). Един от най-простите и най-ефективни алгоритми е алгоритмът на Евклид.

```
1: procedure EUCLID( $a, b$ )           ▷ Намиране на НОД за  $a$  и  $b$ 
2:    $r \leftarrow a \bmod b$ 
3:   while  $r \neq 0$  do             ▷ Ако остатъкът  $r$  е 0, то край
4:      $a \leftarrow b$ 
5:      $b \leftarrow r$ 
6:      $r \leftarrow a \bmod b$ 
7:   end while
8:   return  $b$                      ▷ НОД е равен на стойността на  $b$ 
9: end procedure
```

Нормална форма

Рационалните числа могат да имат отрицателни числител и знаменател. Например:

$$\frac{-3}{-4} = \frac{3}{4}, \quad \frac{3}{-4} = \frac{-3}{4}. \quad (2)$$

Нормална форма на дадено рационалното число ще наричаме неговата редуцирана форма, в която знаменателят е положителен.

```
1 #include <iostream>
2 using namespace std;
3
4 class RationalError{};
5
6 class Rational {
7     long num_, den_;
8
9     long gcd(long r, long s) {
10         while(s!=0) {
11             long temp=r;
12             r=s;
13             s=temp % s;
14         }
15         return r;
16     }
```



```
18 void reduce(void) {
19     if(num_==0){
20         den_=1;
21     } else {
22         long tempnum=(num_<0)?-num_:num_;
23         long g=gcd(tempnum,den_);
24         if (g>1){
25             num_/=g;
26             den_/=g;
27         }
28     }
29 }
```

```
31 void standardize(void) {  
32     if(den_<0) {  
33         den_=-den_;  
34         num_=-num_;  
35     }  
36     reduce();  
37 }
```

```
38 public:
39     Rational(int num=0, int den=1){
40         num_=num;
41         den_=den;
42
43         if(den_==0)
44             throw RationalError();
45         standardize();
46     }
47
48     long get_numerator() {return num_;}
49     long get_denominator() {return den_;}
50
51     void dump() {
52         cout << "(" << num_ << "/" << den_ << ")";
53     }
```

Събиране и изваждане

- Сумата на две рационални числа $\frac{a_1}{b_1}$ и $\frac{a_2}{b_2}$ се нарича рационалното число $\frac{A}{B}$, което се получава по следния начин:

$$\frac{A}{B} = \frac{a_1}{b_1} + \frac{a_2}{b_2} = \frac{a_1 b_2 + a_2 b_1}{b_1 b_2}. \quad (3)$$

- Разликата на две рационални числа $\frac{a_1}{b_1}$ и $\frac{a_2}{b_2}$ се нарича рационалното число $\frac{A}{B}$, което се получава по следния начин:

$$\frac{A}{B} = \frac{a_1}{b_1} - \frac{a_2}{b_2} = \frac{a_1 b_2 - a_2 b_1}{b_1 b_2}. \quad (4)$$

Умножение и делене

- Произведение на две рационални числа $\frac{a_1}{b_1}$ и $\frac{a_2}{b_2}$ се нарича рационалното число $\frac{A}{B}$, което се получава по следния начин:

$$\frac{A}{B} = \frac{a_1}{b_1} \cdot \frac{a_2}{b_2} = \frac{a_1 a_2}{b_1 b_2}. \quad (5)$$

- Частно на две рационални числа $\frac{a_1}{b_1}$ и $\frac{a_2}{b_2}$ се нарича рационалното число $\frac{A}{B}$, което се получава по следния начин:

$$\frac{A}{B} = \frac{a_1}{b_1} : \frac{a_2}{b_2} = \frac{a_1 b_2}{b_1 a_2}. \quad (6)$$

```
55 void add(Rational r) {  
56     num_ = num_*r.den_ + den_*r.num_ ;  
57     den_ = den_*r.den_ ;  
58     standardize();  
59 }  
60 void sub(Rational r) {  
61     num_ = num_*r.den_ - den_*r.num_ ;  
62     den_ = den_*r.den_ ;  
63     standardize();  
64 }
```

```
65 void multiply(Rational r) {
66     num_*=r.num_;
67     den_*=r.den_;
68     standardize();
69 }
70 void device(Rational r) {
71     num_*=r.den_;
72     den_*=r.num_;
73     standardize();
74 }
75 };
```

```
77 int main(int argc, char* argv[]) {  
78     Rational r(1,2), p(2,3), q(4,2), s(-3,-9);  
79  
80     r.dump(); p.dump(); q.dump(); s.dump();  
81     cout << endl;  
82  
83     r.add(p);  
84     r.dump();  
85     cout << endl;
```



```
87     p.add(s);
88     p.dump();
89     cout << endl;
90
91     r.multiply(q);
92     r.dump();
93     cout << endl;
94
95     return 0;
96 }
```

```
lubo@kid:~/school/cpp/notes
lubo@kid ~/school/cpp/notes $ g++ -Wall Rational.cpp -o Rational
lubo@kid ~/school/cpp/notes $ ./Rational
(1/2) (2/3) (2/1) (1/3)
(7/6)
(1/1)
(7/3)
lubo@kid ~/school/cpp/notes $ █
```

Генериране на псевдо-случайни числа

...random numbers should not be generated with a method choosen at random.

Donald Knuth, The Art of Computer Programming, volume 2.

- **Линеен конгруентен метод:** Същността на метода се заключава с следното — избират четири “магични” числа:
 - m — модул, $m > 0$;
 - a — множител, $0 < a < m$;
 - c — добавка, $0 < c < m$;
 - X_0 — начална стойност, $0 \leq X_0 < m$.

Желаната последователност от псевдо-случайни числа X_n се получава, като се използва формулата:

$$X_n = (aX_{n-1} + c) \bmod m, n > 0. \quad (7)$$

Стандартни функции за генериране на псевдо-случайни числа

- В стандартната C и C++-библиотека са дефинирани набор от функции, които генерират последователности от псевдо-случайни числа — `rand()`, `srand()`, `RAND_MAX`. Тези функции са декларирани в заглавия файл `<stdlib.h>` и `<cstdlib>` съответно.
- `int RAND_MAX` — най-голямото случайно число, което може да се генерира от функцията за генериране на случайни числа.
- `int rand(void)` — следващото псевдо-случайно число. Стойностите, които връща тази функция са между 0 и `RAND_MAX`.

Стандартни функции за генериране на псевдо-случайни числа

- **void srand(int seed)** — установява стартова стойност за серията от псевдо-случайни числа, генерирани от функцията `rand()`.
- Ако функцията `rand()` се извика без предварително да е извикана `srand()`, то функцията `rand()` ще генерира едни и същи последователности от случайни числа при всяко пускане на програмата.
- За да бъдат различни последователностите от псевдо-случайни числа обикновено се вика `srand(time())`.

Заглавен файл Random.hpp

```
1 #ifndef RANDOM_HPP__
2 #define RANDOM_HPP__
3
4 namespace Random {
5     void init(unsigned long seed=0);
6     int next_int();
7     int next_int(int max);
8     int next_int(int min, int max);
9     double next_double();
10 };
11
12 #endif
```

Файл Random.cpp

```
1 #include <cstdlib>
2 #include <ctime>
3
4 #include "Random.hpp"
5 namespace Random {
6     void init(unsigned long seed) {
7         unsigned int s =
8             seed==0?std::time(0):seed;
9         std::srand(s);
10    }
11
12    int next_int() {
13        return std::rand();
14    }
```

Файл Random.cpp

```
16 double next_double() {
17     return static_cast<double>(next_int())/
18         static_cast<double>(RAND_MAX);
19 }
20
21 int next_int(int max) {
22     return next_int() % max;
23 }
24
25 int next_int(int min, int max) {
26     return min + next_int() % (max-min);
27 }
28 };
```


Хвърляне на монета

- Като пример за използване на `namespace Random` нека разгледаме задачата за n хвърляния на монета. Въпросът е колко пъти ще се падне ези?
- За да имитираме хвърляне на монета генерираме случайно цяло число, като използваме `Random::next_int(2)`. Стойностите, които ще връща тази функция са 0 или 1. Приемаме, че ако генерираното псевдо-случайно число е 1, то това означава, че се е паднало ези.

Хвърляне на монета

- Да приемем, че искаме да хвърлим монетата 10 пъти. Тогава следният фрагмент пресмята колко пъти се е паднало ези:

```
1 head_count=0;
2 for(int i=0;i<10;i++) {
3     head_count+=Random::next_int(2);
4 }
```

- Тъй като хвърлянето на монета е случайно събитие, то при всяко пускане на горния фрагмент стойността на `head_count` ще бъде случайно цяло число в интервала $[0, 10]$.
- За да наблюдаваме някакви закономерности в този експеримент трябва да го направим голям брой пъти.

Инициализация

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4
5 #include "Random.hpp"
6 const unsigned int COINS_COUNT=10;
7 const unsigned int TOSS_COUNT=10000;
8
9 int main(int argc, char* argv[]) {
10     Random::init();
11     int head[COINS_COUNT+1];
12     for(unsigned i=0;i<COINS_COUNT+1;i++)
13         head[i]=0;
```

Хвърляне на монета

```
14 for(unsigned i=0;i<TOSS_COUNT;i++) {  
15     int head_count=0;  
16     for(unsigned j=0;j<COINS_COUNT;j++)  
17         head_count+=Random::next_int(2);  
18     head[head_count]++;  
19 }
```

Извеждане на резултатата

```
20     for(unsigned i=0;i<COINS_COUNT+1;i++) {
21         int pos=static_cast<int>(
22             (static_cast<double>(head[i])/
23              TOSS_COUNT)*100.0);
24         cout << setw(2)<< i << " "
25              << setw(7) << head[i];
26         for(int j=0;j<pos;j++) {
27             cout << " ";
28         }
29         cout << "*" << endl;
30     }
31
32     return 0;
33 }
```

Разделно компилиране

```
lubo@kid:~/school/cpp/examples-03
lubo@kid ~/school/cpp/examples-03 $ g++ -Wall Coins.cpp -o Coins
/tmp/ccp8aoIS.o(.text+0x61): In function `main':
: undefined reference to `Random::next_int(int)'
collect2: ld returned 1 exit status
lubo@kid ~/school/cpp/examples-03 $ █

lubo@kid:~/school/cpp/examples-03
lubo@kid ~/school/cpp/examples-03 $ g++ -Wall -c Coins.cpp -o Coins.o
lubo@kid ~/school/cpp/examples-03 $ █
```

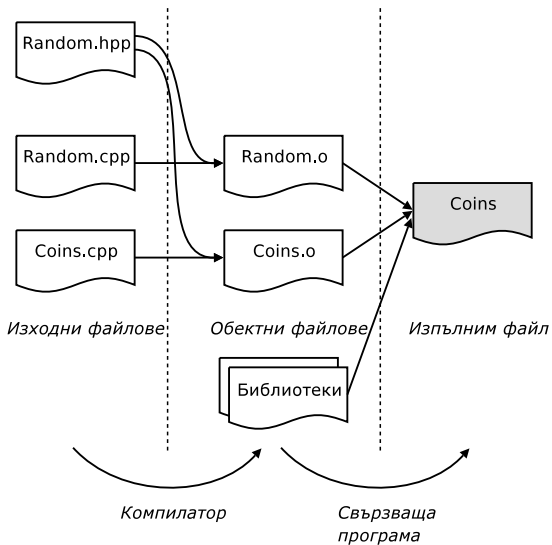
Разделно компилиране

- Следната команда създава обектен файл Coins.o.

```
g++ -Wall -c Coins.cpp
```

- Следната команда се опитва да създаде изпълним файл Coins.

```
g++ -Wall Coins.cpp -o Coins
```



Разделно компилиране

```
lubo@kid:~/school/cpp/examples-03
lubo@kid ~/school/cpp/examples-03 $ g++ -Wall -c Random.cpp
lubo@kid ~/school/cpp/examples-03 $ g++ -Wall -c Coins.cpp
lubo@kid ~/school/cpp/examples-03 $ g++ -Wall Random.o Coins.o -o Coins
lubo@kid ~/school/cpp/examples-03 $ ./Coins
0      9*
1     107 *
2     433  *
3    1151   *
4    2044    *
5    2446     *
6    2101      *
7    1153       *
8     442        *
9     105 *
10     9*
lubo@kid ~/school/cpp/examples-03 $ █
```

Тесте карти

- Като пример за използване на namespace Random нека разработим модел на тесте от 52 карти.
- Всяка една карта принадлежи на определен цвят: спатии (clubs), каро (diamonds), купа (hearts) или пика (spades).
- Всяка карта освен принадлежността ѝ към определен цвят се характеризира и със стойност, която може да бъде: асо (ace), 2, 3, ..., 10, вале (jack), дама (queen), поп (king).
- Целта е да създадем клас CardDeck, който да моделира тесте от 52 карти. Класът трябва да има методи за разместване на картите и за раздаване на карти.

Дефиниция на enum Suit

```
1 #include <iostream>
2 using namespace std;
3 #include "Random.hpp"
4
5 enum Suit {
6     CLUBS=0, DIAMONDS,
7     HEARTS, SPADES
8 };
```

Дефиниция на class Card

```
10 class Card {
11     Suit suit_;
12     int face_;
13 public:
14     void set_card(int card) {
15         suit_ = static_cast<Suit>(card/13);
16         face_ = card%13;
17     }
18     Card(int card=0) {
19         set_card(card);
20     }
```

```
21  Suit get_suit() {
22      return suit_;
23  }
24  int get_face() {
25      return face_;
26  }
27  void print() {
28      static const char FACES[][3]={
29          "A","2","3","4","5","6","7","8","9",
30          "10","J","Q","K"};
31      static const char SUITS[][9]={
32          "Clubs","Diamonds","Hearts","Spades"};
33      cout << FACES[face_]
34          << "(" << SUITS[suit_] << ")";
35  }
36  };
```

Дефиниция на class Deck

```
39 class Deck {  
40     Card cards_[52];  
41     int next_;  
42 public:  
43     Deck(void) {  
44         for(int i=0;i<52;i++) {  
45             cards_[i].set_card(i);  
46         }  
47         next_=0;  
48     }
```

Дефиниция на class Deck

```
50 void shuffle() {
51     for(int i=0;i<52;i++) {
52         int rint=Random::next_int(52);
53         Card temp=cards_[rint];
54         cards_[rint]=cards_[i];
55         cards_[i]=temp;
56     }
57     next_=0;
58 }
59
60 Card deal_one() {
61     return cards_[next_++];
62 }
63 };
```

Главна функция

```
65 int main(int argc, char* argv[]) {
66     Random::init();
67
68     Deck my_deck;
69     my_deck.shuffle();
70     for(int i=0;i<5;i++) {
71         Card c=my_deck.deal_one();
72         c.print();
73         cout << endl;
74     }
75
76     return 0;
77 }
```


Разделно компилиране

```
lubo@kid:~/school/cpp/examples-03
lubo@kid ~/school/cpp/examples-03 $ g++ -Wall -c Random.cpp
lubo@kid ~/school/cpp/examples-03 $ g++ -Wall -c CardDeck.cpp
lubo@kid ~/school/cpp/examples-03 $ g++ -Wall Random.o CardDeck.o -o Cards
lubo@kid ~/school/cpp/examples-03 $ ./Cards
5(Clubs)
9(Clubs)
4(Diamonds)
A(Clubs)
2(Diamonds)
lubo@kid ~/school/cpp/examples-03 $ █
```