Обзор на езика C++ (продължение)
(Rev: 1.1)

Любомир Чорбаджиев[1]
lchorbadjiev@elsys-bg.org

[1]Технологическо училище "Електронни системи"
Технически университет, София

10 януари 2007 г.

Съдържание

Конструктори

- Член-променливите не могат да се инициализират при тяхната дефиниция. Инициализирането на член-променливите трябва да се извърши при създаване на обекти.
- За инициализиране на член-променливите на обектите от даден клас се използва специализирана член-функция, която се нарича *конструктор*.
- При създаването на всеки обект се вика конструктор, който инициализира член-променливите на обекта. Извикването на конструктора се извършва автоматично при създаването на обект.

Конструктори

- Името на конструктора съвпада с името на самият клас.

```
1 class Point {
2   double x_,y_;
3 public:
4   Point(double x, double y); // конструктор
5   //...
6 };
```

- Ако конструкторът има аргументи, то те трябва да се предадат при създаването на обекта. Например:

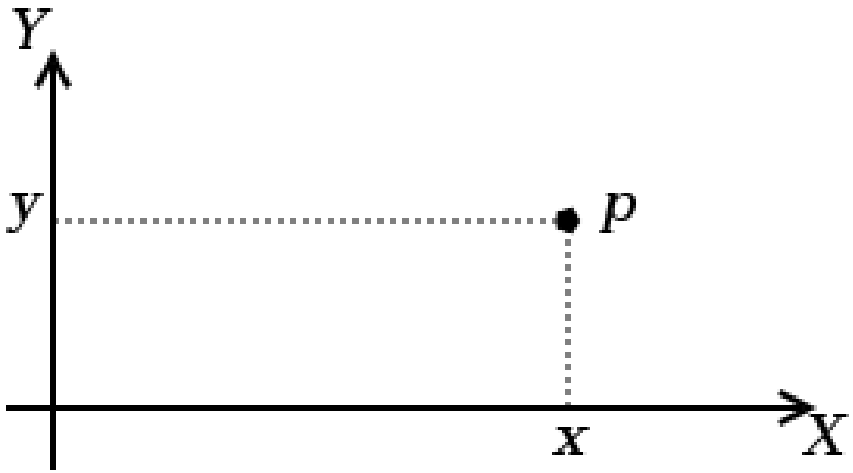
```
1 Point p1 = Point(1.0,1.0);
2 Point p2(2.0,2.0);
3 Point p3; // грешка
4 Point p4(4.0); // грешка
```

Конструктори

- Има възможност за един клас да се дефинират няколко конструктора, които се различават по аргументите, които им се предават.
- Конструктор, който се извиква без аргументи се нарича *конструктор по подразбиране*.

```
1 class Point {
2 public:
3   Point(double x, double y);
4   Point(void);
5 };
6 ...
7 Point p1(1.0,1.0);
8 Point p2;
```

Пример: точка в равнината



Фиг.: Декартови координати на точка в равнината

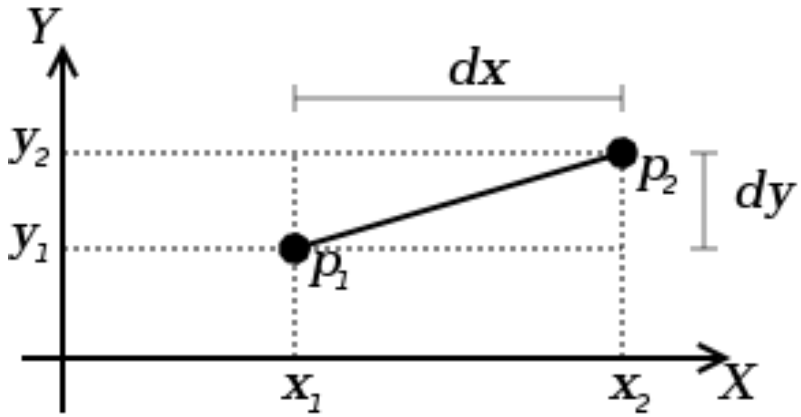
Пример: точка в равнината

```
1 #include <cmath>
2 using namespace std;
3
4 class Point {
5   double x_, y_;
6 public:
7   double get_x() {return x_;}
8   double get_y() {return y_;}
9   void set_x(double x) {x_=x;}
10  void set_y(double y) {y_=y;}
```

Пример: точка в равнината

```
12  Point(double x=0.0, double y=0.0) {
13    x_=x;
14    y_=y;
15  }
16
17  void add(Point other) {
18    x_+=other.x_;
19    y_+=other.y_;
20  }
21
22  void sub(Point other) {
23    x_-=other.x_;
24    y_-=other.y_;
25  }
```

Пример: точка в равнината




```
27   double distance(Point other) {
28     double dx=x_-other.x_;
29     double dy=y_-other.y_;
30     return sqrt(dx*dx+dy*dy);
31   }
32 };
```

Пример: точка в равнината

```
34 Point add(Point p1, Point p2) {
35   Point result(p1.get_x(),p1.get_y());
36   result.add(p2);
37   return result;
38 }
39
40 Point sub(Point p1, Point p2) {
41   Point result(p1.get_x(), p2.get_y());
42   result.sub(p2);
43   return result;
44 }
45
46 double distance(Point p1, Point p2) {
47   return p1.distance(p2);
48 }
```

Основни операции със стек

- Основните операции, които могат да се извършват с един стек са:
  - добавяне на нов елемент в стека — `push()`;
  - изваждане на последния добавен елемент от стека — `pop()`.
- Често стекът се нарича FILO (First In, Last Out) — първи влязъл, последен излязъл.

Реализация на стек

```
1 const int STACK_SIZE=10;
2 class Stack {
3   int data_[STACK_SIZE];
4   int top_;
5 public:
6   Stack() {
7     top_=0;
8   }
```

Реализация на стек

```
9   void push(int val) {
10    if(top_<STACK_SIZE) {
11      data_[top_++]=val;
12    }
13  }
14  int pop(void) {
15    if(top_>0) {
16      return data_[--top_];
17    }
18    return 0;
19  }
```

Реализация на стек

```
20    bool is_empty() {
21      return top_==0;
22    }
23    bool is_full() {
24      return top_==STACK_SIZE;
25    }
26  };
```

Използване на стек

```
28 int main(void) {
29   char* msg="Hello!";
30   char buff[10];
31   Stack st;
32   for(char* p=msg;*p!='\0';p++)
33     st.push(*p);
34   char* p=buff;
35   while(!st.is_empty())
36     *p++=st.pop();
37   *p='\0';
38   return 0;
39 }
```

Пространство от имена

- Пространствата от имена (*namespaces*) са въведени в C++ като поддръжка на така нареченото модулно програмиране.
- По същество пространствата от имена позволяват изграждането на дървовидна структура от имена на идентификаторите в една C++ програма като по този начин намаляват риска от конфликт на имената.
- Пример: файлова система без директории.

Пространство от имена

- Когато размерът на една започне да нараства — конфликтите на имена започват да стават често явление.
- За решаването на този проблем в C++ са въведени пространствата от имена.
- Пространствата от имена могат да бъдат влагани едно в друго и да образуват йерахични структури от имена, подобни на файловата система. Такава йерархична структура от имена може лесно да предпази кода на една програма от конфликти на имената.

Дефиниране на пространство от имена

- За дефиниране на именувано пространство от имена се използва ключовата дума **namespace**. Например:

```
1 namespace elsys {
2   class Student {
3     ...
4   };
5 };
```

- Към едно пространство от имена винаги може да се добавят нови имена.

```
1 namespace elsys {
2   class Teacher {
3     ...
4   };
5 };
```

Използване на пространства от имена

- Идентификаторът може да се квалифицира пълно.

```
elsys::Teacher teacher;
```

- Идентификаторът може да бъде включен в текущата област на видимост, като се използва **using**-дефиниция.

```
using elsys::Teacher;
Teacher teacher;
```

- В текущата област на видимост могат да се включат всички идентификатори, дефинирани в рамките на дадено пространство от имена като се използва **using**-декларация.

```
using namespace elsys;
Teacher teacher;
Student student;
```

Пространство от имена std

- Повечето от типовете, променливите и функциите от стандартната C++ библиотека са дефинирани в пространството от имена std.
- Често срещана практика е, да се използва **using**-декларация за включване на идентификаторите от стандартното пространство от имена в текущата област на видимост.

```
#include <cmath>
#include <cstdlib>
using namespace std;
```

Входно/изходни операции

- Тъй като C++ и C са родствени езици, в една програма на C++ е напълно възможно да се използва стандартната C-библиотека за вход и изход.

```
1 #include <stdio.h>
2
3 int main(int argc, char* argv[]) {
4   printf("Hello world!\n");
5   return 0;
6 }
```

- В C++ входно/изходните операции са организирани като операции с потоци. При разработването на входно/изходната библиотека на C++ специално внимание е обърнато на удобството и лекотата на използване на библиотеката.

Стандартни потоци за вход и изход

- Стандартните потоци за вход и изход са декларирани в заглавния файл `<iostream>`.
- Потоците за вход и изход и операциите с тях са дефинирани в пространството от имена `std`.
- Стандартния поток за изход е `cout`.

```
1 #include <iostream>
2 using namespace std;
3
4 int main(int argc, char* argv[]) {
5   cout << "Hello world!" << endl;
6   return 0;
7 }
```

Стандартен поток за изход cout

- В един израз могат да се комбинират няколко оператора за изход.

```
cout << "Hello" << "␣" << "world!" << endl;
```

- С един оператор за изход могат да се извеждат различни типове данни. Например:

```
cout << "The␣answer␣is␣" << 42 << endl;
```

- За извеждане на край на реда се използва `endl`.

Стандартен поток за вход cin

- Стандартния поток за вход е cin. Операторът за четене от потока е >>.
- Потокът за вход може да обработва последователност от различни по тип променливи.

```
1 #include <iostream>
2 using namespace std;
3
4 int main(int argc, char* argv[]) {
5   int number1, number2;
6   cin >> number1 >> number2;
7   cout << "number1=" << number1 << endl
8        << "number2=" << number2 << endl;
9   return 0;
10 }
```

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.

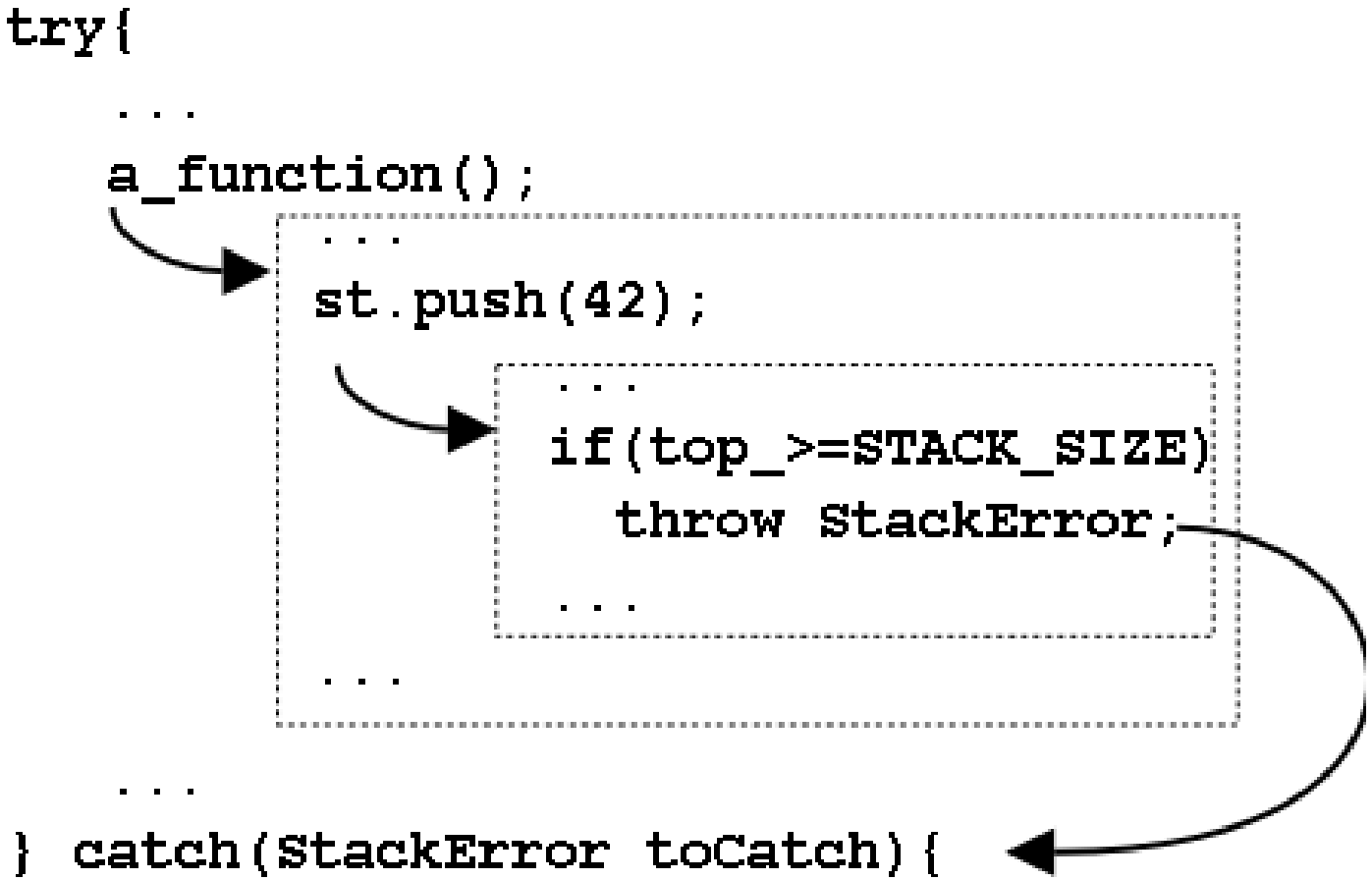
```
1 catch(StackError ex) {
2   log_error(ex);
3   exit(1);
4 }
```

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



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 <iostream>
28 using namespace std;
29 int main(void) {
30   char* msg="Hello␣Cruel␣World!";
31   char buff[10];
```

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

Дефиниция на рационални числа

- Множеството на рационалните числа представлява можеството 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:

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

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

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

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 преставянето във вида $\frac{a'}{b'} = \frac{a}{b}$, за което е изпълнено следното:

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

където $\mathrm{GCD}(a,b)$ 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 ← a mod b
3:     while r ≠ 0 do                        ▷ Ако остатъкът r е 0, то край
4:         a ← b
5:         b ← r
6:         r ← a mod 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:

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

Норамална 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:

$$\frac{A}{B} = \frac{a_1}{b_1} + \frac{a_2}{b_2} = \frac{a_1 b_2 + a_2 b_1}{b_1 b_2}. \tag{3}$$

- 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:

$$\frac{A}{B} = \frac{a_1}{b_1} - \frac{a_2}{b_2} = \frac{a_1 b_2 - a_2 b_1}{b_1 b_2}. \tag{4}$$

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:

$$\frac{A}{B} = \frac{a_1}{b_1} \cdot \frac{a_2}{b_2} = \frac{a_1 a_2}{b_1 b_2}. \tag{5}$$

- 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:

$$\frac{A}{B} = \frac{a_1}{b_1} : \frac{a_2}{b_2} = \frac{a_1 b_2}{b_1 a_2}. \tag{6}$$

```
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 multiplication(Rational r) {
66      num_*=r.num_;
67      den_*=r.den_;
68      standardize();
69    }
70    void division(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.multiplication(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  };
```

Хврърляне 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.

Хврърляне 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;
```

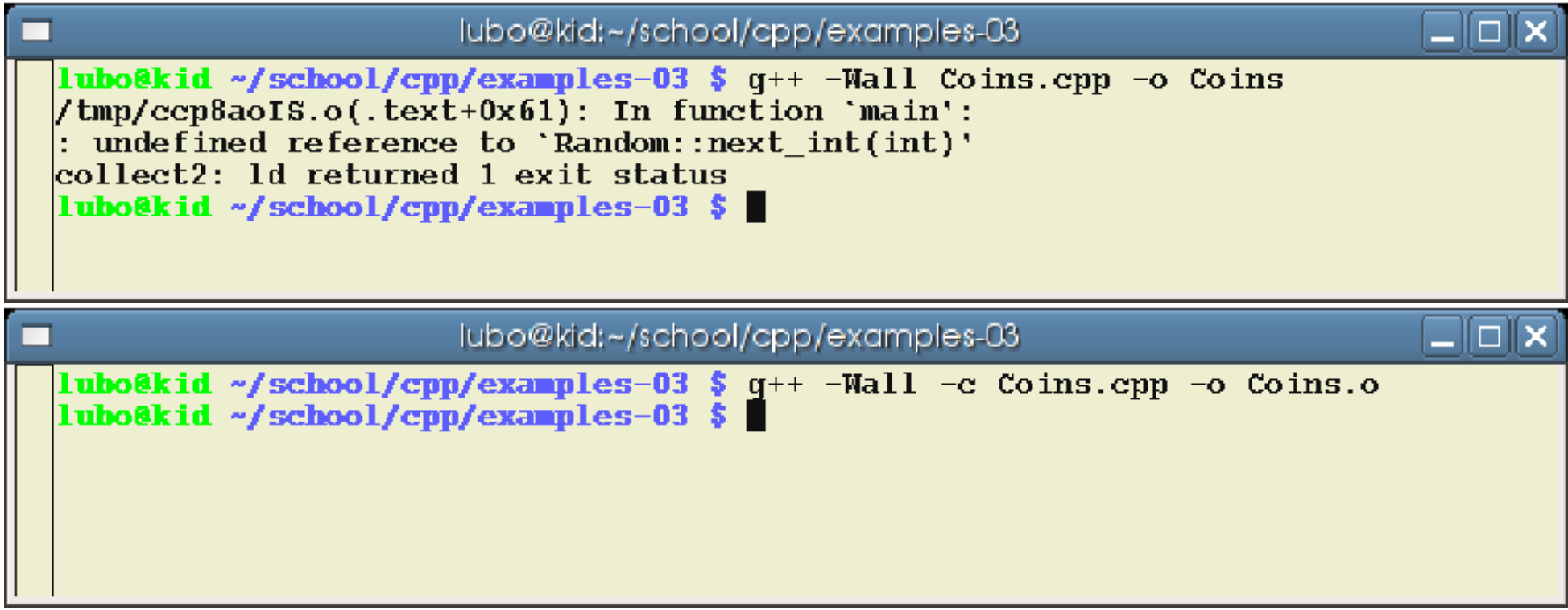
Хвърляне на монета

```
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 резулатата

```
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



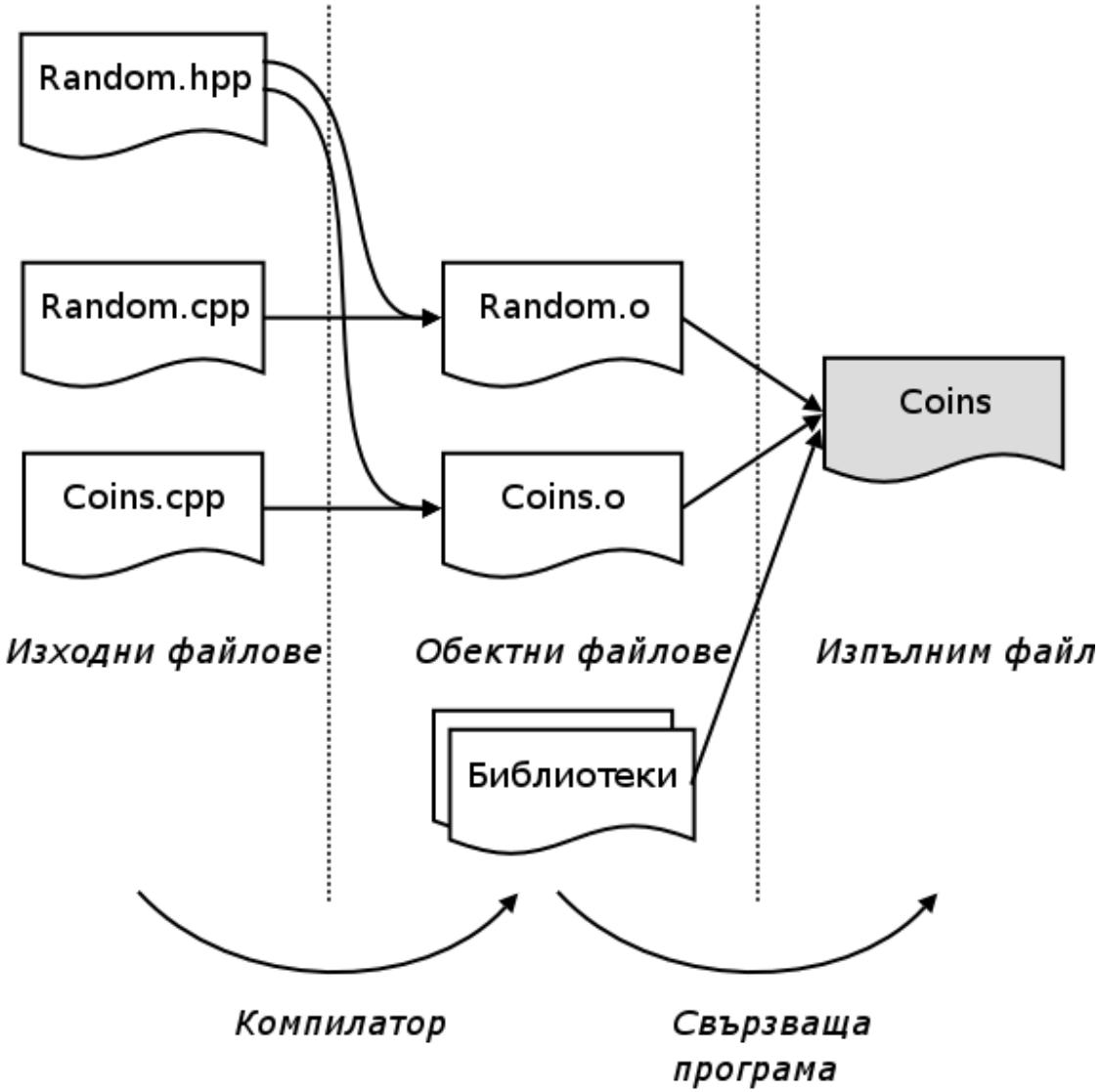
Разделно компилиране

- 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
```


Random.hpp
Random.cpp
Coins.cpp
Random.o
Coins.o
Coins
Библиотеки
Изходни файлове
Обектни файлове
Изпълним файл
Компилатор
Свързваща програма

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 харектеризира 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