МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М.В. ЛОМОНОСОВА

Факультет
вычислительной математики
и кибернетики

И. А. Волкова, А. А. Вылиток, Л. Е. Карпов

Сборник задач и упражнений по языку Си++

(учебное пособие для студентов II курса)

Москва – 2013

УДК 004.43(075.8)
ББК 32.973-018.1я73
В67

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

Рецензенты:

И.В. Машечкин — д.ф.-м.н.;
С.Ю. Соловьев — д.ф.-м.н.

Волкова И. А., Вылиток А.А., Карпов Л. Е.

В 67 **Сборник задач и упражнений по языку Си++:** Учебное пособие для студентов II курса. — М.: Издательский отдел факультета ВМК МГУ имени М. В. Ломоносова (лицензия ИД № 05899 от 24.09.2001 г.); МАКС Пресс, 2013 — 64 с.

ISBN 978-5-317-04595-1

В сборнике представлены задачи и упражнения по языку Си++, рекомендуемые студентам 2 курса для подготовки к коллоквиуму по основам объектно-ориентированного программирования в рамках курса «Системы программирования». Рассматриваемая версия языка Си++ соответствует стандарту ISO/IEC (1998).

В сборнике собраны задачи на использование основных механизмов объектно-ориентированных языков программирования: инкапсуляции, наследования, полиморфизма. Значительная часть задач предлагалась студентам в письменных проверочных работах в 2008—2013 гг.

Для студентов факультета ВМК в поддержку основного лекционного курса «Системы программирования», а также для преподавателей, ведущих практические занятия по этому курсу.

Авторы выражают благодарность преподавателям кафедры алгоритмических языков за участие в обсуждении и составлении задач.

УДК 004.43(075.8)
ББК 32.973-018.1я73

Учебное издание
ВОЛКОВА Ирина Анатольевна
ВЫЛИТОК Алексей Александрович
КАРПОВ Леонид Евгеньевич
СБОРНИК ЗАДАЧ И УПРАЖНЕНИЙ ПО ЯЗЫКУ СИ++
Учебное пособие для студентов II курса

19992, ГСП-2, Москва, Ленинские Горы, МГУ им. М.В. Ломоносова, 2-й учебный корпус

ISBN 978-5-317-04595-1

I. Задачи и упражнения

1. Абстрактные типы данных (АТД). Классы. Конструкторы и деструкторы

1.1. Есть ли ошибки в приведенном фрагменте программы? Если есть, то объясните, в чем они заключаются.[1] Как изменить описание класса *A*, не вводя новые методы и не меняя *f()*, чтобы в *f()* не было ошибок? Что будет напечатано в результате работы функции *f()*?

```
class A {
    int a, b;
public:
    A (A & x) {
        a = x.a;
        b = x.b;
        cout << 1;
    }
    A (int  a) {
        this -> a = a;
        b = a;
        cout << 2;
    }
};
void f () {
    A x (1);
    A y;
    A z = A (2.5, 4);
    A s = 6;
    A w = z;
    x = z = w;
}
```

1.2. Описать конструктор для некоторого класса *A* таким образом, чтобы были выполнены следующие условия:

а) это единственный явно описанный конструктор класса *A*,

б) справедливы следующие описания объектов класса *A*:

```
A a;
A b(1);
A c(1, 2);
A d('1', 1);
```

1.3. Если есть ошибки в приведенном фрагменте программы, то объясните**,** в чем они заключаются, и вычеркните ошибочные конструкции.

Что будет выдано в стандартный канал вывода при вызове функции *main ()*?

```
a)  class X {
        int i;
        double t;
        X(int k) {
             i = k;
             t = 0;
             cout << 1;
        }
```

[1] Везде, где это необходимо, предполагается наличие *#include<iostream>* и ***using namespace*** *std* или *#include<cstdio>*.

```
    public:
        X(int k, double r = 0) {
            i = k;
            t = r;
            cout << 2;
        }
        X & operator= (X & a) {
            i = a.i;
            t = a.t;
            cout << 3;
            return * this;
        }
        X(const X & a) {
            i = a.i;
            t = a.t;
            cout << 4;
        }
    };

    int main() {
        X a;
        X b(1);
        X c (2, 3.5);
        X d = c;
        X e (6.5, 3);
        c = d = e;
        return 0;
    }
```

b)
```
class X {
        int i;
        double t;
        X( ) {
            i = 0;
            t = 1.0;
            cout << 1;
        }
    public:
        X(int k = 0, double r = 1.5) {
            i = k;
            t = r;
            cout << 2;
        }
        X(const X & a) {
            i = a.i;
            t = a.t;
            cout << 3;
        }
    };

    int main() {
        X a;
        X b(1);
        X c (1.5, 2);
        X d = b;
        X e = 3;
        b = c = e;
        return 0;
    }
```

1.4. Описать класс *A* таким образом, чтобы все конструкции функции *main ()* были верными, а на экран выдалось **100 300**.

```
int main () {
    A a1 (5),  a2 = 3;
    a1  *=  10;
    a2  *=   a1  *=  2;
    cout << a1.get() << a2.get() << endl;
    return 0;
}
```

1.5. Описать класс B таким образом, чтобы все конструкции функции *main* были верными, а на экран выдалось **10 20 30.**

```
int main () {
    B b1,  b2 = b1, b3 (b2);
    cout << b1.get() << b2.get() <<  b3.get () << endl;
    return 0;
}
```

1.6. Описать класс C таким образом, чтобы все конструкции функции *main* были верными, а на экран выдалось **14 10 48.**

```
int main () {
    C c1 (7),  c2 = 5, c3 (c1 + c2);
    cout << c1.get() << c2.get() << c3.get () << endl;
    return 0;
}
```

1.7. Описать класс *A* так, чтобы:

- все конструкции функции main были верными,
- явно в классе *A* можно описать не более одного конструктора,
- на экран выдалось **15 60 7**.

Нельзя использовать исключения и любые функции досрочного завершения программы.

```
int main () {
    A a1(5), a2 = 4, a3;
    a2 *= a1 *= 3;
    cout << a1.get( ) << ' ' << a2.get() << ' ' <<  a3.get( ) << endl;
    return 0;
}
```

1.8. Описать класс *B* так, чтобы:

- все конструкции функции *main* были верными,
- класс *B* содержал только один явно описанный конструктор,
- на экран выдалось **17 11 6**.

Нельзя использовать исключения и любые функции досрочного завершения программы.

```
int main () {
    B b1 (1), b2(2,3), b3 (b1);
    b1 += b2 += b3;
    cout << b1.get() << ' ' << b2.get() << ' ' << b3.get () << endl;
    return 0;
}
```

1.9. Описать класс *C* так, чтобы:

- в main ошибочным было только описание объекта *c2*,
- класс *C* содержал только один явно описанный конструктор,
- после удаления описания *c2* на экран выдалось **14 56**.

Нельзя использовать исключения и любые функции досрочного завершения программы.

```
int main () {
    C c1(7), c2 = 5, c3(c1 + c1);
    cout << c1.get ( ) << ' ' << c3.get ( ) << endl;
    return 0;
}
```

1.10. Что напечатает следующая программа?

```
class I {
    int i;
public:
    I() : i(9) { cout << "sun" <<endl; }
    I(int a) : i(a) { cout << "venus " << i << endl; }
    I(const I & other) : i(other.i) { cout << "earth " << i << endl; }
    ~I() { cout << "moon" << endl; }
    int Get() { return i; }
    void operator+= (const I & op) { i+=op.i; }
};

void f(I & x, I y) {
    y += 1000;
    x += y;
}

int main() {
    I i1;
    I i2(20);
    i2 += 400;
    f(i1, i2);
    cout << i1.Get() << i2.Get() << endl;
    return 0;
}
```

1.11. Что напечатает следующая программа?

```
class I {
    int i;
public:
    I() : i(6) { cout << "owl" << endl; }
    I(int a) : i(a) { cout << "sheep " << i << endl; }
    I(const I & other) : i(other.i) { cout << "horse " << i << endl; }
    ~I() { cout << "wolf" << endl; }
    int Get() { return i; }
    void operator*=(const I & op) { i*=op.i; }
};

void f(I x, I & y) {
    x *= 1;
    y *= x;
}
```

```
int main() {
    I i1;
    I i2(3);
    i1 *= 7;
    f(i1, i2);
    cout << i1.Get() << ' ' << i2.Get()<< endl;
    return 0;
}
```

1.12. Что напечатает следующая программа?

```
class I {
    int i;
public:
    I() : i(5) { cout << "fist" << endl; }
    I(int a) : i(a) { cout << "lance " << i << endl; }
    I(const I & other) : i(other.i) { cout << "dagger " << i << endl; }
    ~I() { cout << "pistole" << endl; }
    int Get() { return i; }
    void operator+=(const I & op) { i+=op.i; }
};

void f(I & x, I y) {
    y += 1000;
    x += y;
}

int main() {
    I i1;
    I i2(30);
    i2 += 700;
    f(i1, i2);
    cout <<  i1.Get() << ' ' << i2.Get() << endl;
    return 0;
}
```

1.13. Даны описания структуры, переменной и функции:

```
struct mystr {
    int a, b;
};

int i = sizeof(mystr);

int f(mystr s) {
    return 0;
}
```

Дополните описание структуры *mystr* (не изменяя описание функции *f*) так, чтобы только описание *f* стало ошибочным.

1.14. Опишите структуру с именем *smartstr*, удовлетворяющую двум условиям:
(1) можно создать объект типа *smartstr*;
(2) нельзя создать массив элементов типа *smartstr* в динамической памяти.

1.15. Какие конструкторы и деструкторы и в каком порядке будут выполняться при работе следующего фрагмента программы:

a)
```
class A {};
class B : public A {};
class C : public B {};
int main(){
    C c;
    A a = c;
    struct D {
        B b;
        D(): b(5){}
    } d;
}
```

b)
```
class A {};
class B : public A {};
class C : public B {};
int main(){
    class D {
        B b;
        A a;
    public:
        D (): a(b){ }
    } d;
    C c;
}
```

c)
```
class A {};
class B {};
class C : public A, public B {};
int main(){
    C c;
    class D {
        C c;
        B b;
    public: D(): b(c){}
    } d;
}
```

1.16. Что будет выдано на печать при работе следующей программы?

```
struct S {
    int x;
    S (int n) { x = n; printf (" Cons    "); }
    S (const S & a) {  x = a.x;   printf (" Copy    "); }
    ~S ( ) {  printf ("Des   "); }
};

S f( S  y ) {
    y = S(3);
    return y;
}
int main () {
    S s (1);
    f (s);
    printf ("%d   ",  s.x);
    return 0;
}
```

1.17. Что будет выдано на печать при работе следующей программы?

```
struct S {
    int x;
    S(int n) {  x = n;  printf(" Cons    "); }
    S(const S & a) { x = a.x;  printf(" Copy    "); }
    ~S() {  printf("Des   "); }
};

S  f( S & y ) {
    y = S(3);
    return y;
}

int main () {
    S s(1);
    f (s);
    printf("%d   ",  s.x);
    return 0;
}
```

1.18. Что будет выдано на печать при работе следующей программы?

```
struct S {
    int x;
    S( int n ) { x = n;  printf( " Cons    " ); }
    S( const S &  a ) {  x = a.x;   printf( " Copy    " ); }
    ~S( ) {  printf( "Des   " ); }
};

S & f( S y ,  S & z) {
    y =  S (3);
    return z;
}

int main ( ) {
    S s(1);
    f( s,  s );
    printf( "%d   ",  s.x );
    return 0;
}
```

1.19. Внести добавления в описания заданных методов (**не меняя вывод на экран!**) структур *B* и *D* так, чтобы все конструкции *main ()* были правильными, а на печать выдалось **5535324242**.

```
struct B {
    float x;
    B (float a) {   x = a;   cout << 5; }
    ~B( ) {  cout << 2; }
};
struct D : B {
    D( )  { cout << 3; }
    ~D( ) {  cout << 4; }
};
```

```
int main ( ) {
    B  * p1 = new B (1),  * p2 = new D[2];
    delete p1;
    delete [ ] p2;
    return 0;
}
```

1.20. Внести добавления в описания заданных методов (**не меняя вывод на экран!**) структур *B* и *D* так, чтобы все конструкции *main ()* были правильными, а на печать выдалось **11163343**.

```
struct B {
    int x;
    B (int a) {   x = a;   cout << 1; }
    ~B () {  cout << 3; }
};

struct D : B {
    D (int d ) : B (d) { cout << 6; }
    ~D () {  cout << 4; }
};

int main () {
    B  * p1 = new B [2],  * p2 = new D (1);
    delete  [ ] p1;
    delete  p2;
    return 0;
}
```

1.21. Внести добавления в описания заданных методов (**не меняя вывод на экран!**) структур *B* и *D* так, чтобы все конструкции *main ()* были правильными, а на печать выдалось **776898**.

```
struct B {
    int x;
    B() {   x = 7;   cout << 7; }
    ~B() {  cout << 8; }
};

struct D : B {
    D( int d) { x  = d ;  cout << 6; }
    ~D() {  cout << 9; }
};

int main () {
    B * p1 = new B [1],  * p2 = new D[1];
    delete [ ] p1;
    delete [ ] p2;
    return 0;
}
```

1.22. Есть ли ошибки в приведённом ниже фрагменте? Если да, объясните, в чём они заключаются.

```
a)  int n;
    float f(float a, int t = 3, int d = n) {
        return a * (float) (t % d);
    }
```

```
b)  float f(float a = 2.7, int t, int d = 8) {
        return sa * (float) (t % d);
    }

c)  enum { myparam = 18 };
    float f(float a, int t = myparam + 5, int d = t + 8) {
        return a * (float) (t % d);
    }
```

2. Перегрузка операций. Перегрузка функций

2.1. Даны описание класса и функции:

```
class Cls {
        int i;
public:
        Cls() { i = 1; }
};

void f(Cls * p, Cls * q) {
        *p = *q;
}
```

Дополните описание класса *Cls* (не изменяя описание функции *f*)так, чтобы только описание *f* стало ошибочным.

2.2. Описать прототипы двух перегруженных функций *f* из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
a)  f (1);
    f ('+', '+');
    f (2.3);
    f (3, “str”);

b)  f ();
    f (“abc”);
    f (2);
    f ('+', 3);

c)  f (0, 1);
    f (1, 0);
    f (0, "m");
    f ("n", 0);

d)  f ("p");          // где       struct X {
    f (x, 0);         //                 X(int);
    f (0, 0);         //                 operator int();
    f (x, "q");       //           } x;
    f (1, "r");
```

e)
```
f (1000000000000);
f (1);
f ();
f (0, 0);
f ("t");
f (1, "u");
```

2.3. Для каждого вызова перегруженной функции с одним параметром укажите, какая функция и на каком шаге алгоритма будет выбрана.

a)
```
int f(int a = 0) { return a; }
int f(double a) { return a; }
int main() {
    short int s;
    int i;
    bool b;
    enum e {A, B, C};
    float f1 = 1.0f;
    f();
    f(s);
    f(f1);
    f(b);
    f(A);
}
```

b)
```
int f(double a = 1.0){return a;}
int f(long double a = 5.0){return a;}
int main() {
    float f1 = 1.0f;
    double d = 2.0;
    long double ld = 3.0l;
    f();
    f(4);
    f(f1);
    f(d);
    f(ld);
}
```

c)
```
int f(int a = 1){return a;}
int f(long double a = 5.0){return a;}
int main () {
    short int s;
    int i;
    bool b;
    float f1 = 1.0f;
    double d = 2.0;
    f(s);
    f(i);
    f(b);
    f(f1);
    f(d);
}
```

3. Наследование. Видимость и доступность имен

3.1. Если есть ошибки в реализации функций *B::g ()* и *main ()*, объясните, в чем они заключаются.

Для всех правильных операторов этих функций с помощью операции разрешения области видимости «**::**» укажите, из какой области видимости выбираются участвующие в их записи имена.

Какие конструкторы и деструкторы и в каком порядке будут вызываться при работе данной программы?

a)
```
int x = 0;
void f (int a, int b){x = a+b;}

class A {
    int x;
public:
    void f (){x = 2;}
};

class B: public A {
public:
    void f (int a){::x = a;}
    void g ();
};

void B::g() {
    f();
    f(1);
    f(5 , 1);
    x = 2;
}

B  ret (B & x, B & y) {
    return  x;
}

int main () {
    B b;
    f(5);
    f('+', 6);
    b  =  ret  (b,  b);
    return 0;
}
```

b)
```
double a = 0;
void f (double x){a = x;}

struct A {
    double a;
    void f (){a = 2;}
};
```

```
class B : A {
public:
    void f (int a){::a = a;}
    void g ();
};

void B::g() {
    f(1.2);
    f();
    a = 2;
}

void  empty (B & a, B  b) { }


int main () {
    B d;
    f();
    f(6);
    empty  (d, d);
    return 0;
}
```

c)
```
int x = 0;
void f(int a, int b) { x = a+b; }
class A {
    int x;
public:
    void f() { x = 2; }
    void f(char a1, char b1) { x = a1-b1; }
};

class B: public A {
public:
    void f(int a) { ::x = a; }
    void g () {
         f();
         f(0);
         f(5.3 , 1);
         x = 1;
    }
};

int main () {
    B b;
    f(2);
    f(3, 'a');
    return 0;
}
```

d)
```
double a = 0;
void f(double x = 2) {
    a = x;
}

void f() {
    a = 1;
}
```

```
struct BBase {
    double a;
    void f(){
        a = 2;
    }
};

class B: BBase {
public:
    void f(int a) { ::a = a; }
    void g() {
        f('r');
        f();
        a = 2;
    }
};

int main () {
    B d;
   f();
   f(6);
   return 0;
}
```

e)

```
float y = 0;
void f(float a) {
     y = a;
}

class T {
     int y;
public:
    void f() {
         y = 2;
    }
};

class B : public T {
public:
    void f(float n, float m) { ::y = n * m; }
    void f(char c1, char c2) { ::y = c1 + c2; }
    void g () {
         f();
         f(1);
         f(-1 , 1);
         y = 2;
    }
};

int main () {
    B b;
    f(5);
    f('+', 6);
    return 0;
}
```

3.2. Если есть ошибки в реализации методов заданных классов и функции *main (),* исправьте их, используя операцию разрешения области видимости «**::**».

Какие конструкторы и деструкторы и в каком порядке будут вызываться при работе данной программы?

a)
```
int x = 0;
int f (int a, int b) { return x = a + b; }

class A {
    int x;
public:
    A (int n = 1) { x = n; }
    int f() { return ::x = x; }
};

class B {
    int x;
public:
    B (int n = 2) { x = n; }
};

class C: public A, public B  {
    int x;
public:
    int f(int a) { return ::x = x; }
     void g ();
};

void C::g() {
    x = f ();
    f (3);
    x = f (4, 5);
    x = 6;
}

int main () {
    C c;
    B b = c;
    A a = c;
    c.f ();
    c.f (7);
    x = f ('8', 9);
    return -1;
}
```

b)
```
int x = 0;
int f() { return x = 1; }

class A {
    int x;
public:
    A( int n = 2) { x = n; }
    int f() { return x = 3; }
    int f(int a, int b) { return x = a % b; }
};
```

```
class B: public A {
    int x;
public:
    int f(int a) { return x = a; }
    int f(int a, int b) { return x = a + b; }
    int g(A * a, B * b);
};

int  B::g (A * pa, B * pb) {
    x = f ();
    x = f (5)
    x = f (6, 6);
    x = A::f (5);
    return -1;
}

int main () {
    B a;
    class C {
        B b;
        A a;
    public:
        C(): b(), a (b) { }
    };
    C c;
    x =  a.f ();
    x =  a.f (7);
    return  a.g (& a, & a);
}
```

c)

```
int x = 0;
int f (int a, int b) { return x = a + b; }
class A {
    int x;
public:
    A(int n = 1) { x = n; }
    int f() { return ::x = x; }
};

class B {
     int x;
public:
     B (int n = 2) { x = n; }
};

class C: public A, public B {
    int x;
public:
    int f(int a) { return ::x = x; }
    void g ();
};

void C::g() {
    x = f ();
    f (3);
    x = f (4, 5);
    x = 6;
}
```

```
int main () {
    C c;
    B b;
    A a;
    c.f();
    c.f(7);
    x = f('8', 9);
    return -1;
}
```

d)

```
int x = 0;
int f() {return x = 1; }
class A {
    int x;
public:
    A( int n = 2) { x = n; }
    int f() { return x = 3; }
    int f(int a, int b) { return x = a % b; }
};

class B: public A {
    int x;
public:
    int f(int a) { return x = a; }
    int f (int a, int b) { return x = a + b; }
    int g (A * a, B * b);
};

int  B::g (A * pa, B * pb) {
    x = f();
    x = f(5);
    x = f(6, 6);
    x = A::f(5);
    return -1;
}

int main () {
    B a;
    x = a.f();
    x = a.f(7);
    class C {
          A a1;
    public:
          C(): a1(b) {}
    };
    C c;
    return  a.g(& a, & a);
}
```

e)

```
int x = 4;
class A {
    int x;
public:
    A(int n = 1);
    int f(int a = 0, int b = 0);
};
```

```
class B: public A {
public:
   int x;
    B(int n = 2);
    int f(int a);
};

class C: public B {
    int x;
public:
    C(int n = 3);
    int f(int a, int b);
    int g(A * p);
};

int main () {
    A * p;
    B b;
    C c;
    A a = c;
    struct D {
        B b;
        D(): b(5) {}
    } d;
    p = & b;
    x = c.g(& c);
    x = c.f();
    x = c.f(x);
    x = c.f(x, 1);
    x = p -> f();
    x = p -> f(x);
    x = p -> f(x, 1);
    return 1;
}
```

f)

```
int x = 4;
class A {
    int x;
public:
    A(int n = 3);
    int f(int a = 0, int b = 0);
};

class B: public A {
    int x;
public:
    B(int n = 1);
    int f(int a = 0);
};

class C: public  B {
    int x;
public:
    C(int n = 2);
    int f(int a, int b);
    int g(A * p);
};
```

```
int main () {
    A * p;
    B b;
    C c;
    class D {
        B b;
        A a;
    public:
        D(): a(b) {}
    } d;
    p = & b;
    x = c.g(& b);
    x = c.f();
    x = c.f(x);
    x = c.f(x, 1);
    x = p -> f();
    x = p -> f(x);
    x = p -> f(x, 1);
    return 2;
}
```

3.3. Если есть ошибки в следующем фрагменте, то в чем они заключаются? Исправьте ошибки, ничего не удаляя, добавив в общей сложности не более 14 символов.

```
class A {
public:
    int * n;
    int m;
};

class B: public A {
public:
    int * p;
};

class C: public A {
public:
    int * c;
};

class D: public B, public C {
public:
    int * e;
};

int main () {
    D fA, * f = new D;
    fA.m =0;
    return *((* f).e = & fA.m);
}
```

3.4. Если есть ошибки в следующем фрагменте, то в чем они заключаются? Исправьте ошибки, ничего не удаляя, добавив в общей сложности не более 12 символов.

```
class S {
public:
    int s;
    void sp(int si) { s = si; }
};
```

```
class T: S {
public:
    int t;
    void tp(int ti) { t = ti; s = ti; }
};

class U: T {
public:
    int u;
    void up(int ui) { u = ui; t = ui; s = ui; }
};

void g()  {
    U * pu = new U;
    T * pt = pu;
    S * ps = pu;
}
```

3.5. Если есть ошибки в следующем фрагменте, то в чем они заключаются? Исправьте ошибки, ничего не удаляя, добавив в общей сложности не более 29 символов.

```
class W {
public:
    int * w;
    void wf (int * wp) { w = wp; }
};

class X: W {
public:
    int * x;
    void xf (int * xp) { x = xp; w = xp; }
};

class Y: W {
public:
    int * y;
    void yf (int * yp) { y = yp; w = yp; }
};

class Z: X, Y {
public:
    int * z;
    void zf (int * zp) { z = zp; x = zp; y = zp; }
};
void h () {
    int hi;
    W * pw;
    X * px;
    Y * py;
    Z * pz;
    pz = new W;
    (*pz).w = & hi;
    pz -> xf ((*pz).X::w);
}
```

4. Виртуальные функции. Абстрактные классы

4.1. Описать условия включения механизма виртуальности для метода класса. Привести пример записи виртуальной функции и обращения к ней.

4.2. Есть ли ошибки в приведенном фрагменте программы? Если есть, то объясните, в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный канал вывода при работе программы?

```
class X {
public:
    virtual int g (double x) {
        h ();   cout << "X::g" << endl;
        return 1;
    }
    void h () { t ();   cout << "X::h" << endl;}
    virtual void t ()       { cout << "X::t" << endl;}
};

class Z: public X {
public:
    int g (double y) {
        h ();   cout << "Z::g" << endl;
        return 3;
    }
    virtual void h () { t (1);   cout  << "Z::h" << endl;  }
    virtual void t (int k)   { cout  << "Z::t" << endl;  }
};

int main(){
    X a;    Z b;    X * p = &b;
    p -> g(1.5);
    p -> h();
    p -> t(5);
}
```

4.3. Есть ли ошибки в приведенном фрагменте программы? Если есть, то объясните, в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный канал вывода при работе программы?

```
class T {
public:
    virtual  int f (int x) {
        cout  << "T::f" << endl;
        return 0;
    }
    void g () {
        f (1);
        cout  << "T::g" << endl;
    }
    virtual void h ()     {
        g ();
        cout  << "T::h" << endl;
    }
};
```

```
class S: public T {
public:
    int f (double y){
        cout  << "S::f" << endl;
        return 2;
    }
    virtual void g () {
        f (1);
        cout  << "S::g" << endl;
    }
    virtual void h () {
        g();
        cout  << "S::h" << endl;
    }
};
int main(){
    T t;  S s;  T *p = &s;
    p -> f (1.5);
    p -> g ();
    p -> h ();
}
```

4.4. Есть ли ошибки в приведенном фрагменте программы? Если есть, то объясните, в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный канал вывода при работе программы?

```
class K {
public:
    virtual  int f (int x){
       cout  << "K::f" << endl;
       return 0;
    }
    void g () {
        f (1);
        cout  << "K::g" << endl;
    }
    virtual void h () {
        g ();
        cout  << "K::h" << endl;
    }
};

class P: public K {
public:
    int f (double y) {
        cout  << "P::f" << endl;
        return 2;
    }
    virtual void g () {
        f (1);
        cout  << "P::g" << endl;
    }
    virtual void h () {
        g();
        cout  << "P::h" << endl;
    }
};
```

```
int main(){
    K k;  P p;  K *t = &p;
    t -> f (0.7);
    t -> g ();
    t -> h ();
}
```

4.5. Есть ли ошибки в приведенном фрагменте программы? Если есть, то объясните, в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный поток вывода при работе программы?

```
class A {
public:
    virtual void f (int x) {
        h (x);
        cout << "A::f," << x << endl;
    }


    void g () {
        h (0); cout << "A::g"    << endl;
    }
    virtual void h (int k) {
        cout << "A::h," << k << endl;
    }
};

class B: virtual public A {
public:
    void f (int y) {
        h (y); cout << "B::f," << y << endl;
    }
    void g () {
        h (1); cout << "B::g"       << endl;
    }
    void h (int k) {
        cout << "B::h," << k << endl;
    }
};

int main(){
    A a; B b; A * p = & b;
    p -> f (2);
    p -> g ();
    p -> h ();
    p -> h (3);
}
```

4.6. Есть ли ошибки в приведенном фрагменте программы? Если есть, то объясните, в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный поток вывода при работе программы?

```
class C {
public:
    virtual void f (int x) {
         h (x); cout << "C::f," << x << endl;
    }
```

```
    virtual void g () {
         h (0); cout << "C::g"  << endl;
    }
    virtual void h () {
         cout << "C::h"   << endl;
    }
    virtual void h (int k) {
        h ();  cout << "C::h," << k << endl;
    }
};

class D: public C {
public:
    virtual void f (int y) {
         h (y); cout << "D::f," << y << endl;
    }
    virtual void g ()  {
         h (1); cout << "D::g"  << endl;
    }

    virtual void h () {
         cout << "D::h"  << endl;
    }
    virtual void h (int k) { h ();  cout << "D::h," << k << endl; }
};

int main(){
    C c; D d; C * p = & d;
    p -> f (2);  p -> g ();
    p -> h ();  p -> h (3);
}
```

4.7. Есть ли ошибки в приведенном фрагменте программы ? Если есть, то объясните, в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный поток вывода при работе программы?

```
class T {
public:
    virtual void f (int x) {
        h ();  cout << "T::f," << x << endl;
    }
    void g ()  {
        h ();  cout << "T::g"  << endl;
    }
    virtual void h () {
        cout << "T::h"       << endl;
    }
};

class U: virtual public T {
public:
    void f (int y) {
        h (y); cout << "U::f," << y << endl;
    }
    virtual void g () {
        h (0); cout << "U::g"    << endl;
    }
```

```
    void h (int k) {
       cout << "U::h," << k << endl;
    }
};
int main(){
    T t; U u; T * p = & u;
    p -> f (1);  p -> g ();
    p -> h ();  p -> h (2);
}
```

4.8. Что напечатает следующая программа?

```
class A {
    int i;
public:
    A(int x) { i = x; cout << "first" << endl; }
    virtual ~A() { cout << "second" << endl; }
    int f() const { return i + g() + h(); }
    virtual int g() const { return i; }
    int h() const { return 39; }
};

class B : public A {
public:
    B() : A(70) { cout <<"third" << endl; }
    ~B() { cout << "fourth" << endl; }
    int f() const { return g() - 2; }
    virtual int g() const { return 4; }
    int h() const { return 6; }
};

int main() {
    B b;
    A* p = &b;
    Cout << "result = (“ << p->f() <<';'<< b.f() << ')' << endl;
    return 0;
}
```

4.9. Что напечатает следующая программа?

```
class A {
    int i;
public:
    A(int x) { i = x; cout << "mercury" << endl; }
    virtual ~A() { cout << "venus" << endl; }
    int f() const { return 96; }
    virtual int g() const { return i; }
    int h() const { return i - f() - g(); }
};

class B : public A {
public:
    B(int x) : A(x+20) { cout << "earth" << endl; }
    ~B() { cout << "mars" << endl; }
    int f() const { return 8; }
    virtual int g() const { return 3; }
    int h() const { return f() + g(); }
};
```

```
int main() {
    B b(17);
    A* p = &b;
    Cout << "result = (" << p->h() << ';'<< b.h() <<')' << endl;
    return 0;
}
```

4.10. Что напечатает следующая программа?

```
class A {
    int i;
public:
    A(int x) { i = x; cout << "dog" << endl; }
    virtual ~A() { cout << "cat" << endl; }
    int f() const { return i + g() + h(); }
    virtual int g() const { return i; }
    int h() const { return 5; }
};



class B : public A {
public:
    B() : A(21) { cout << "sheep" << endl; }
    ~B() { cout << "horse" << endl; }
    int f() const { return g() - 3; }
    virtual int g() const { return 7; }
    int h() const { return 9; }
};

int main() {
    B b;
    A* p = &b;
    Cout << "result = (" <<  p->f() << ';' << b.f() <<')' << endl;
    return 0;
}
```

4.11. Дан фрагмент программы:

```
struct A {
    int i;
    virtual void f() = 0;
    virtual ~A() {}
};

int g(A a) { return a.i * 5; }
```

Есть ли в этом фрагменте ошибки? Если да, то в чем они заключаются?

4.12. Дан фрагмент программы:

```
struct S {
    virtual void f() const = 0;
    virtual ~S() {}
};
```

```
struct A {
    S s;
    int i;
};
```

Есть ли в этом фрагменте ошибки? Если да, то в чем они заключаются?

4.13. Дан фрагмент программы:

```
class B {
public:
    virtual int f() = 0;
    int g() { return f() * 10; }
    virtual ~B() {}
};
int h(B b) { return b.g() + 2; }
```

Есть ли в этом фрагменте ошибки? Если да, то в чем они заключаются?

4.14. Что напечатает следующая программа?

```
struct B {
    virtual void f (int n) { cout << "f (int) from B" << endl; }
    static int i;
};

struct D:  B {
    virtual void f (char n) { cout << "f (char) from D" << endl; }
};

int B::i = 1;

int main () {
    D  d;
    B   b1,  b2,  *pb = &d;
    pb -> f ( 'a');
    b1.i += 2;
    b2.i += 3; d.i += 4;
    cout << b1.i << ' ' << b2.i << ' ' << d.i << ' ' <<  B::i << endl;
    return 0;
}
```

4.15. Что напечатает следующая программа?

```
struct K {
    virtual void add_st ( K * n ) {
         st ++;    cout << "add_st (K*) from K" << endl;
     }
    static int st;
};

struct L:  K {
    virtual void add_st ( L * a ) {
        st++;    cout << "add_st (L*) from L" << endl;
     }
};
```

```
int K::st = 2;
int main () {
    L   ob,  ob2;
    K   k,  *pl = &ob;
    pl  -> add_st (& ob2);
    k.st ++;      ++ob.st ;
    cout << k.st << ' ' << ob.st << ' ' <<  K::st << endl;
    return 0;
}
```

4.16. Что напечатает следующая программа?

```
struct S {
    static double d;
    virtual  S &  g () { cout << "g ( ) from S" << endl; }
};

struct T: S {
    virtual  T & g ( ) { cout << "g ( ) from T" << endl; }

};

double S::d = 1.5;

int main () {
    T   t;   S   s ,  *ps = &t;
    ps  -> g ();
    s.d = 5;  t.d = 7;
    cout << s.d << ' ' << t.d << ' ' <<  S::d << endl;
    return 0;
}
```

5. Аппарат исключений

5.1. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
class X;
void F(X & x, int n);
class X {
public:
    X() { try { F(*this, -2);     cout << 1 << endl; }
          catch (X)  { cout << 2 << endl; }
          catch (int) { cout << 3 << endl; }
    }
    X(X &) { cout << 12 << endl; }
};

class Y: public X {
public:  Y () {cout << 4 << endl;}
         Y (Y & a)  {cout << 5 << endl;}
         ~Y ()      {cout << 6 << endl;}
};
```

```
void F(X & x, int n) {
     try { if (n < 0) throw x;
           if (n > 10) throw 1;
           cout << 7 << endl;
     }
     catch (int) { cout << 8 << endl; }
     catch (X&)  { cout << 9 << endl;   throw; }
}

int main() { try { Y a; }
             catch (...)  {  cout << 10 << endl;  }
             cout << 11 << endl;
}
```

5.2. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
class A {
public:
    A () { cout << 1 << endl;}
};
class B: public A {
public:
    B (int n) {
        try  { if (n == 0) throw *this;
               if (n > 11) throw 11;
             }
        catch (int){ cout << 2 << endl; }
        catch (B&)  { cout << 3 << endl;   throw; }
        cout << 4 << endl;
    }
    B (B&)   {cout << 5 << endl;}
     ~B ()   {cout << 6 << endl;}
};

int main() {
    try {
        B b(0);   B c (3);
    }
    catch (...)  {  cout << 7 << endl;  }
         cout << 8 << endl;
}
```

5.3. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
class X;
void F(X & x, int n);
class X {
public:
    X() {   try    {
                F(*this, -2);
                cout << 1 << endl;
            }
            catch (X){ cout << 2 << endl; }
            catch (int)   { cout << 3 << endl; }
    }
    X (X &) { cout << 12 << endl; }
};
```

```
class Y: public X {
public:
    Y () {cout << 4 << endl;}
    Y (Y & a) {cout << 5 << endl;}
    ~Y () {cout << 6 << endl;}
};

void F(X & x, int n) {
    try { if (n < 0) throw x;
        if (n > 10) throw 1;
        cout << 7 << endl;
    }
    catch (int)  { cout << 8 << endl; }
    catch (X&)   { cout << 9 << endl;   throw; }
}

int main() { try { Y a; }
         catch (...){  cout << 10 << endl;  }
         cout << 11 << endl;
}
```

5.4. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct X;
void f(X & x, int n);
int const P = 1; int const Q = 1; int const R = 1;
struct X {
    X() {   try { f(*this, -1);cout <<  1 << endl; }
            catch (X)    {     cout <<  2 << endl; }
            catch (int)  {     cout <<  3 << endl; }
    }
    X (X &)  {   cout <<  4 << endl; }
    ~X ()    {   cout <<  5 << endl; }
};

struct Y: X {
    Y (){   f(*this, -1);
                cout <<  6 << endl; }
    Y (Y &)  {   cout <<  7 << endl; }
    ~Y ()    {   cout <<  8 << endl; }
};

void f(X & x, int n) {
     try {   if (n < 0) throw x;
             if (n > 0) throw 1;
             cout <<  9 << endl;
    }
    catch (int)    {      cout << 10 << endl; }
    catch (X& a) {
        cout << 11 << endl;
        f(a, 1);
        cout << 12 << endl;
        throw;
    }
}
```

```
int main() {
    try { Y a; }
    catch (...)  {
         cout << 13 << endl;
          return 0;
        }
    cout << 14 << endl;
    return 0;
}
```

5.5. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct X;
void f(X & x, int n);
int const P = 1; int const Q = 1; int const R = 1;
struct X {
    X(){
        try  {   f(*this, 0);
                 cout <<  1 << endl;
         }
        catch (X){ cout <<  2 << endl; }
        catch (int){      cout <<  3 << endl; }
    }
    X (X &)  {   cout <<  4 << endl; }
    ~X ()    {   cout <<  5 << endl; }
};
struct Y: X {
    Y () {    f(*this, -1);
              cout <<  6 << endl;
    }
    Y (Y &)  {   cout <<  7 << endl; }
    ~Y ()    {   cout <<  8 << endl; }
};

void f(X & x, int n) {
    try {
        if (n < 0) throw x;
        if (n > 0) throw 1;
        cout <<  9 << endl;
    }
    catch (int)    {      cout << 10 << endl; }
    catch (X& a) {
        cout << 11 << endl;
         f(a, 1);
        cout << 12 << endl;
         throw;
    }
}

int main() {
    try { Y a;
    }
    catch (...)  {
        cout << 13 << endl;
         return 0;
    }
    cout << 14 << endl;
    return 0;
}
```

5.6. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct X;
void f(X & x, int n);
int const P = 1; int const Q = 1; int const R = 1;
struct X {
    X(){
         try {   f(*this, -1);
                 cout <<  1 << endl;
         }
         catch (X)  { cout <<  2 << endl; }
         catch (int) {    cout <<  3 << endl; }
    }
    X (X &)  {   cout <<  4 << endl; }
    ~X ()    {   cout <<  5 << endl; }
};

struct Y: X {
    Y () {    f(*this, 1);
              cout <<  6 << endl;
    }
    Y (Y &)  {   cout <<  7 << endl; }
    ~Y ()    {   cout <<  8 << endl; }
};

void f(X & x, int n) {
    try {
        if (n < 0) throw x;
        if (n > 0) throw 1;
        cout <<  9 << endl;
    }
    catch (int)     { cout << 10 << endl; }
    catch (X& a) {
        cout << 11 << endl;
        f(a, 0);
        cout << 12 << endl;
        throw;
    }
}

int main() {
    try { Y a; }
    catch (...)  {
        cout << 13 << endl;
        return 0;
    }
    cout << 14 << endl;
    return 0;
}
```

5.7. Что напечатает следующая программа?

```
class Ex {
    int code;
public:
    Ex(int i) : code(i) {}
```

```
    Ex(const Ex& ex) : code(ex.code) {}
    int Get() const { return code; }
};

struct Ex90 : Ex {
    Ex90() : Ex(90) {}
};

void f() {
    throw Ex90();
    cout << "dog" << endl;
}

void t() {
    try { f(); }
    catch(Ex90 &x) {
        cout<< "cat" << endl;
        throw Ex(x.Get() + 1);
        cout << "sheep" << endl;
    }
    catch(Ex &) { cout << "horse" << endl; }
    cout <<"cow" << endl;
}

int main() {
    try { t(); }
    catch(Ex &x) { cout << "elephant " << x.Get() << endl; }
    catch(...) { cout << "wolf" << endl; }
    return 0;
}
```

5.8. Что напечатает следующая программа?

```
class Ex {
    int code;
public:
    Ex(int i) : code(i) {}
    Ex(const Ex& ex) : code(ex.code) {}
    int Get() const { return code; }
};

struct Ex60 : Ex {
    Ex60() : Ex(60) {}
};

void f() {
    throw Ex60();
    cout << "sword" << endl;
}

void t() {
    try { f(); }
    catch(Ex60 &x) {
        cout << "lance" << endl;
        throw Ex(x.Get() + 1);
        cout << "dagger" << endl;
    }
    catch(Ex &) { cout << "knife" << endl; }
    cout << "hammer" << endl;
}
```

```
int main() {
    try { t(); }
    catch(Ex &x) { cout << "arche " << x.Get() << endl; }
    catch(...) { cout << "pistole" << endl; }
    return 0;
}
```

5.9. Что напечатает следующая программа?

```
class Ex {
   int code;
public:
   Ex(int i) : code(i) {}
   Ex(const Ex& ex) : code(ex.code) {}
   int Get() const { return code; }
};



struct Ex51 : Ex {
    Ex51() : Ex(51) {}
};

void f() {
    throw Ex51();
    cout << "train" << endl;
}

void t() {
    try { f(); }
    catch(Ex51 &x) {
        cout << "plane" << endl;
        throw Ex(x.Get() + 1);
        cout << "helicopter" << endl;
    }
    catch(Ex &) { cout << "car" << endl; }
    cout << "truck" << endl;
}

int main() {
    try { t(); }
    catch(Ex &x) {
        cout << "boat " << x.Get() << endl;
    }
    catch(...) { cout << "rocket" << endl; }
    return 0;
}
```

5.10. Что будет выдано в стандартный поток вывода при работе следующей программы?

```
void f(X & x, int n);
struct X {
    X () {   try  { f(*this, -1);
                    cout << "a"; }
             catch (X){ cout << "b"; }
             catch (int){ cout << "c"; }
    }
    X (X &) { cout << "d"; }
```

```
    virtual ~X () { cout << "e"; }
};

struct Y: X {
    Y () {  try {
                  f(*this, 0);
                   cout << "f"; }
             catch (Y) { cout << "g"; }
             catch (int){ cout << "h"; }
             cout << "i";
          }
    Y (Y &){ cout << "j"; }
    ~Y (){ cout << "k"; }
};

void f(X & x, int n) { try  { if (n < 0) throw -n;
                              else if (n == 0) throw x;
                              else throw n;}
                       catch (int){ cout << "l"; }
}

int main() { try { Y a; }
             catch (...){ cout << "m"; return 1; }
             cout << "n"; return 0;
           }
```

5.11. Что будет выдано в стандартный поток вывода при работе следующей программы?

```
void f(X & x, int n);
struct X {
    X () { try  { f(*this, 0); cout << "a"; }
           catch (X){ cout << "b"; }
           catch (int){ cout << "c"; }
    }
    X (X &) { cout << "d"; }
    virtual ~X () { cout << "e"; }
};

struct Y: X {
    Y () {  try { f (*this, 0); cout << "f"; }
            catch (Y) { cout << "g"; }
            catch (int) { cout << "h"; }
            cout << "i";
    }
    Y (Y &) { cout << "j"; }
    ~Y () { cout << "k"; }
};

void f(X & x, int n) {
    try  { if (n < 0) throw -n;
           else if (n == 0) throw x;
           else throw n;
         }
    catch (int) { cout << "l"; }
}
```

```
int main() { try { Y a; }
             catch (...) { cout << "m"; return 1; }
             cout << "n";
             return 0;
           }
```

5.12. Что будет выдано в стандартный поток вывода при работе следующей программы?

```
void f(X & x, int n);
struct X {
    X () { try { f(*this, 1);  cout << "a"; }
           catch (X){ cout << "b"; }
           catch (int){ cout << "c"; }
         }
    X (X &){ cout << "d"; }
    virtual ~X () { cout << "e"; }
};




struct Y: X {
    Y () { try { f (*this, 1);    cout << "f"; }
           catch (Y){ cout << "g"; }
           catch (int){ cout << "h"; }
           cout << "i";
    }
    Y (Y &){ cout << "j"; }
    ~Y (){ cout << "k"; }
};

void f(X & x, int n) {
    try  { if (n < 0) throw -n;
           else if (n == 0) throw x;
           else throw n; }
    catch (int){ cout << "l"; }
}

int main() { try { Y a; }
             catch (...){ cout << "m"; return 1; }
             cout << "n"; return 0;
}
```

5.13. Что напечатает следующая программа?

```
struct S {
    S ( int a) {
         try  { if (a > 0)  throw  *this;
              else if (a < 0)  throw 0;
         }
         catch ( S & ) { cout << "SCatch_S&" << endl; }
         catch (int) {  throw; }
         cout << "SConstr" << endl;
    }
    S (const S & a) { cout << "Copy" << endl; }
    ~S ( ) { cout << "Destr" << endl; }
};
```

```
int  main ( ) {     try  { S   s1(1),  s2(-2);
                           cout << "Main" << endl;
                    }
                    catch (S  &) {   cout << "MainCatch_S&" << endl;   }
                    catch  ( ... )  {  cout << "MainCatch_..." << endl; }
                    return 0;
}
```

5.14. Что напечатает следующая программа?

```
struct S {
    S ( int a) {
        try { if (a > 0)   throw  *this;
              else
              if (a < 0) throw 0;
        }
        catch ( S & ) {
            cout << "SCatch_S&" << endl; }
            catch (int) {  throw; }
            cout << "SConstr" << endl;
    }
    S (const S & a) { cout << "Copy" << endl; }
            ~S ( ) { cout << "Destr" << endl; }
    };

int  main ( ) {
    try  {   S   s1( 0 ),      s2 ( 5 );
             cout << "Main" << endl;
    }
    catch (S  &) {   cout << "MainCatch_S&" << endl;   }
    catch ( ... ) {   cout << "MainCatch_..." << endl;   }
    return 0;
}
```

5.15. Что напечатает следующая программа?

```
struct S {
    S (int a) {    try  {
                      if (a > 0)  throw  *this;
                       else  if (a < 0)  throw 0;
                    }
                    catch ( S & ) { cout << "SCatch_S&" << endl;  throw;}
                    catch (int) {  cout << "SCatch_int" << endl; }
                   cout << "SConstr" << endl;
    }
    S (const S & a) { cout << "Copy" << endl; }
    ~S ( ) { cout << "Destr" << endl; }
};

int  main ( ) {
    try  {   S   s1(-3),      s2(25);
             cout << "Main" << endl;
    }
    catch (S  &) {   cout << "MainCatch_S&" << endl;   }
    catch ( ... ) {   cout << "MainCatch_..." << endl;   }
    return 0;
}
```

6. Константные и статические члены класса.

6.1. Есть ли ошибки в тексте приведенной программы? Можно ли исправить описание класса, не вводя дополнительных членов, чтобы программа стала верной? Если да, то как?

```
class A {
public:
    int y;
    void f() {cout << "f" << endl;}
};

int A::y;
int main () {
    A::y = 1;
    const A a;
    a.f();
    return 0;
}
```

6.2. Есть ли ошибки в тексте приведенной программы? Можно ли исправить описание класса, не вводя дополнительных членов, чтобы программа стала верной? Если да, то как?

```
class X {
public:
    void g () {cout << "g" << endl;}
    int h (int n) {cout << "f" << endl; return n}
};

int main () {
    int k;
    const X x;
    X::g();
    k = x.h(5);
    return 0;
}
```

6.3. Есть ли синтаксические ошибки в тексте приведенной программы? Можно ли **исправить описание класса**, не вводя дополнительных членов и не убирая имеющиеся, чтобы программа стала верной? Если да, то как?

а)
```
class A {
    static int i;
    static void f() {
        g();
        cout << “f()” << endl;
    }
    void g() {
        if (i >= 0)
            i = -1, f();
        cout << “g ()” << endl;
    }
};

int A::i = 1;
```

```
int main () {
    A::i = 1;
    A a;
    a.f();
    a.i = 0;
    return 0;
}
```

b)
```
class A {
    static int i;
    void f() {
        if (i >= 0)
            i = -1, g();
        cout << "f()" << endl;
    }

    void g() {
        f();
        cout << "g()" << endl;
    }
};

int A::i = 1;
int main () {
    A::i = 1;
    const A a;
    a.f();
    a.i = 0;
    return 0;
}
```

c)
```
class A {
    static int i;
    void f() const {
        if (i < 0)
            g(i);
        cout << "f ()" << endl;
    }
    void g(int & n) {
        i = n;
        f();
        cout << "g ()" << endl;
    }
};

int A::i = 1;
int main () {
    const A a;
    a.g(2);
    return 0;
}
```

6.4. Опишите класс *A* таким образом, чтобы были верными все конструкции следующего фрагмента программы:

a)
```
int A::x;
int main () {
    const A a;
```

```
        a.x = 1;
        a.get_0();
        return 0;
    }

b)  const char A::a = '+';
    int main () {
        A ob;
        A::f();
        return 0;
    }

c)  int main () {
        const A x;
        A::g();
        x.h();
        return 0;
    }
```

6.5. Если есть ошибки в приведенной программе, то объясните**,** в чем они заключаются. Ошибочные операторы или ключевые слова вычеркните (допускается не более двух вычеркиваний). Что будет выдано в стандартный поток вывода при работе получившейся программы?

```
class A {
public:
    static void f(int x) {
        h (x);
        cout << "A::f," << x << endl;
    }
    void g() {
        cout << "A::g" << endl;
    }
    void h(int x) {
        g ();
        cout << "A::h," << x << endl;
    }
};
class B: virtual public A {
public:
    static void f(int x) {
        h (x);
        cout << "B::f," << x << endl;
    }
    void g() {
        cout << "B::g" << endl;
    }
    void h (int x) {
        g ();
        cout << "B::h," << x << endl;
    }
};

int main() {
    B::f(0);
    B b;
    A * p = & b;
    p -> f(1);
    p -> g();
```

```
    p -> h(2);
    A::f(3);
    return 0;
}
```

6.6. Есть ли ошибки в интерфейсах классов *C* и *D* программы? Если есть, то объясните, в чем они заключаются и внесите нужные исправления, оставив без изменения реализацию классов и функции *main ()*. Что будет выдано в стандартный поток вывода при работе получившейся программы?

```
class C {
public:
    C(int x = 0) {}
    virtual int f(int x) {
        cout << "C::f," << x << endl;
        return h (x);
    }



    virtual int g() {
        cout << "C::g" << endl;
        return 1;
    }
    virtual int h (int x) {
        cout << "C::h," << x << endl;
        return x;
    }
    virtual operator int () { return 99; }
};

class D: public C {
public:
    int f(int x) {
        cout << "D::f," << x << endl;
        return h (x);
    }
    int g(int x) {
        cout << "D::g"       << endl;
        return 1;
    }
    int h(int x) {
        cout << "D::h," << x << endl;
        return x;
    }
    D(int x = 0) {}
    operator int () { return 100; }
};

int main() {
    const D d;
    C const * const t = & d;
    t -> f(3);
    t -> f(d);
    t -> g();
    t -> h(5);
    return 0;
}
```

6.7. Добавить (если нужно) в класс *A* служебные слова ***const***, так, чтобы заданный фрагмент программы был верным.

a)
```
class A {
    int i;
public:
    A(int x) { i = x; }
    A(A & y) { i = y.i; }
    const A f(const A & z) {
         cout << endl;
         return *this;
    }
};

const A  t1( ) {
    const A a = 5;
    return a.f( a );
}
```

b)
```
class A {
    int i;
public:
    A(int x) { i = x; }
    A(A & y) { i = y.i; }
    const A f(A & c) const {
         cout << c. i << endl;
         return *this;
    }
};
const A t1(const A a) {
    A b  = A(5);
    return b.f( a );
}
```

c)
```
class A {
    int i;
public:
    A(int x) { i = x; }
    A(A & y) { i = y.i; }
    const A f( const A c) {
         cout << c.i << endl;
         return *this;
    }
};

 const A t1(const A * a) {
    A b = A(3);
    return  a -> f( b );
 }
```

6.8. В приведённой программе возможно наличие синтаксических ошибок в определении класса *A*. Если ошибки есть, исправьте их заменой, исключением или добавлением нужных служебных слов языка Си++. Обоснуйте сделанные исправления.

a)
```
class A {
        int i;
        int f(int & x) { return g(x); }
        int g(int & x) {
                if (x >= 0) f(-i);
                return i;
        }
};

int A::i = 2013;

int main () {
        const A a;
        A::i = 201;
        a.f(20);
        return a.i = 1;
}
```

b)
```
class A {
        int x;
        int y;
        int p()  {
                return y >= 0 ? y = -1 :
                q();
        }
        int q() const { return p(); }
        int r()  { return x = y; }
        A(int z) { x = y > 0 ? z % y : -y; }
};

int A::y = 13;

int main() {
        A::y = 1;
        A::p();
        const A b1(2013), b2(b1);
        b1.q();
        return b2.x = 2;
}
```

c)
```
class A {
        int m;
        void m1() { if (m < 0) m2(m); }
        int m2(int & n) const { return m1(),n; }
        void m3(int & n) { m = m2(n); }
};

int A::m = 1;

int main() {
        A::m3 (2013);
        A mm;
        return mm.m2 (3);
}
```

7. Динамическая идентификация и приведение типов

7.1. Укажите лишние и ошибочные операции динамического приведения типа, если таковые имеются в функции *main ()*. Дайте необходимые пояснения своим исправлениям.

```
class K           { public:         void g (){ cout << "K::g"; } };
class L: public K { public:         void f (){ cout << "L::f"; } };
class M: public K { public:virtual  void h (){ cout << "M::h"; } };
class P: public L { public:         void f (){ cout << "P::f"; } };
class Q: public M { public:virtual  void h (){ cout << "Q::h"; } };
class R: public P { public:virtual  void f (){ cout << "R::f"; }
                           virtual  void h (){ cout << "R::h"; } };
class S: public Q { public:virtual  void f (){ cout << "S::f"; }
                           virtual  void h (){ cout << "S::h"; } };
int main (){
    S os, * s = & os; K * k; L * l; M * m; P * p; Q * q; R * r;
    int a, b;
    k = dynamic_cast <K *>(s);   s = dynamic_cast <S *>(k);
    l = dynamic_cast <L *>(k);   m = dynamic_cast <M *>(s);
    p = dynamic_cast <P *>(l);   q = dynamic_cast <Q *>(m);
    r = dynamic_cast <R *>(q);   s = dynamic_cast <S *>(p);
    return 0;
}
```

7.2. Добавить в функцию *f9 ()* использование механизма приведения типов так, чтобы ее выполнение всегда завершалось нормально.

```
struct  B    { virtual void g () {} };
struct  D: B { char y [100];  };
void f9 (B & b, D & d, int n) {
    D * pd = (n > 0) ? & d : (D *) & b;
    strcpy (pd -> y, “one_variant\n”);
}
```

7.3. Добавить в функцию *putnull ()* использование механизма приведения типов так, чтобы ее выполнение всегда завершалось нормально.

```
struct  B    { virtual void empty () {} };
struct  D: B { int mas [30]; };
void putnull (B * pb){
    D  * pd = (D *) pb;
    if (! pb) return;
    for (int i = 0; i < 30; i++) pd -> mas [i] = 0;
}
```

7.4. Добавить в функцию *puthi ()* использование механизма приведения типов, так, чтобы ее выполнение всегда завершалось нормально.

```
struct  B    { virtual void hi () { cout << “Hi!” << endl; } };
struct  D: B { char txt [10] [4]; };
void puthi (B * pb, D * pd) {
    int i = 10;
    if (! pb) return;
    pd = (D *) pb;
    while (i) strcpy ((pd -> txt) [--i], ”Hi!”);
}
```

7.5. Какие виды операций преобразования типов имеются в языке Си++? Укажите назначение каждого вида, приведите пример записи каждой из перечисленных операций.

7.6. Для приведённой ниже программы описать функцию *f ()*, которая, получая в качестве параметра указатель типа *A**, возвращает его значение, наиболее безопасным образом преобразованное к типу *B**, а в случае невозможности преобразования корректно завершает работу программы.

```
struct  A    { virtual void z () {} };
struct  B: A { int x; B (int y = 5) { x = y; } };
B * f (A * pa);
int main (){
    try {
         B b,  * pb = f (& b);
         cout << pb -> x << endl;
         A a;    pb = f (& a);
         cout << pb -> x << endl;
    }
    catch (...) { }
    return 0;
}
```

7.7. Для приведённой ниже программы описать функцию *f ()*, которая, получая в качестве параметра ссылку на объект базового класса *A*, возвращает ссылку на объект производного класса *C*, полученную наиболее безопасным образом, а в случае невозможности приведения типов корректно завершает программу.

```
struct  A    { virtual void z () {} };
struct  B: A { };
struct  C: B { int x; C (int n = 3) { x = n; } };
C & f (A & ra);
int main () {
    C c, & pc = f (& c);
    cout << pc.x << endl;
    return 0;
}
```

7.8. Для приведённой ниже программы описать функцию *f ()*, которая, получая в качестве параметра ссылку на объект типа ***void*** ***, возвращает ссылку на объект производного класса *B*, полученную наиболее безопасным образом, а в случае невозможности приведения типов корректно завершает программу.

```
struct  A    { virtual void z () {} };
struct  B: A { int x; B (int n = 7) { x = n; } };
B * f (void * p);
int main () {
    B b, * pb = f (& b);
    cout << pb -> x << endl;
    return 0;
}
```

7.9. Для приведённой ниже программы описать функцию *f ()*, которая, получая в качестве параметра ссылку на объект класса *B*, возвращает указатель на объект класса *C*, полученный наиболее безопасным образом, а в случае невозможности приведения типов корректно завершает программу.

```
struct  A       { int x; virtual void z () {} };
struct  B: A    { int x; };
struct  C: B    { int x; C (int n = 4) { x = n; } };
C * f (B & rb);
int main () {
    C c, * pc = f (c);
    cout << pc -> x << endl;
    return 0;
}
```

7.10. Можно ли получить информацию о типе объекта во время работы программы на Си++?
Какие операции языка Си++ используются в подобных случаях?
Для объектов каких типов эти операции имеют смысл?
Какие стандартные исключения генерируются в результате выполнения этих операций?
Привести пример программы, в которой используются эти операции.

8. Шаблоны

8.1. Привести пример использования параметрического полиморфизма в Си++.

8.2. Есть ли ошибки в следующих заголовках шаблонов?
Если есть, поясните, в чём они заключаются.

```
template <double f> void funcA (double d = f) { /*...*/ }
template <float f> class A { /*...*/ };
template <int n> class B { /*...*/ };
template <int n> void funcB (int t = n) { /*...*/ }
template <class Cs> class  C { /*...*/ };
template <class Cs> struct D { /*...*/ };
template <class Cs> void funcC (const Cs& ref) { /*...*/ }
class myclass { public: myclass() {} };
template <myclass c> class E { /*...*/ };
struct mystruct { int a, b; };
template <class mystruct> void funcD (mystruct *p = 0) { /*...*/ }
template <mystruct a> void funcE (mystruct *p = &a) { /*...*/ }
template <struct mystruct> void funcF (mystruct *p = 0) { /*...*/ }
```

8.3. Есть ли ошибки в приведенном фрагменте программы на Си++? Если есть, то объясните, в чем они заключаются. Ошибочные конструкции вычеркните из текста программы. Что будет напечатано при вызове функции *main ()*?

```
class complex {
    double re, im;
public:
    complex (double r = 0, double i = 0) {
        re = r;
        im = i;
        cout << “constr” << endl;
    }
    operator double () {
        cout << “operator double ” << endl;
        return re;
    }
    double get_re () { return re; }
    void print() const  {
        cout << "re=" << re << "  im=" << im << endl;
    }
};

template <class T>
T f (T& x, T& y) {
    cout << "template f" << endl;
    return x > y ? x : y;
}

double f (double x, double y)
{
    cout << "ordinary f" << endl;
    return x > y ? -x : -y;
}




int main ()
{   complex a(2, 5), b(2, 7), c;
    double x = 3.5, y = 1.1;
    int i, j = 8, k = 10;
    c = f (a, b);
    cout << "c = ";
    c.print ();
    x = f (a, y);
    cout << "x = " << x <<endl;
    i = f (j, k);
    cout << "i = " << i <<endl;
    cout << "Выбор сделан!" << endl;
    return 0;
}
```

8.4. Есть ли ошибки в приведенном фрагменте программы? Если есть, то объясните, в чем они заключаются. Ошибочные конструкции вычеркните из текста программы. Что будет напечатано при вызове функции *main ()*? Перегрузите шаблонную функцию *max ()* так, чтобы сравнение строк осуществлялось лексикографически (то есть в соответствии с кодировкой символов).

```
template <class T>
T max (T& x, T& y) {
    return x > y ? x : y;
}
```

```
int main () {
    double x = 1.5, y = 2.8, z;
    int i = 5, j = 12, k;
    char * s1 = "abft";
    char * s2 = "abxde", * s3;
    z = max (x, y);
    cout << "z = "<< z << endl;
    k = max <int>(i, j);
    cout << "k = "<< k << endl;
    z = max (x, (double &) i);
    cout << "z = "<< z << endl;
    z = max (y, (double &) j);
    cout << "z = "<< z << endl;
    s3 = max (s1, s2);
    cout << "s3 = "<< s3 << endl;
    cout << "Выбор сделан!" << endl;   return 0;
}
```

8.5. Какие из следующих утверждений являются верными, а какие — ошибочными? Объясните, в чем заключаются эти ошибки.

- в одной зоне описания не может быть описано несколько шаблонов класса с одинаковыми именами
- шаблонная функция и обычная функция в одной зоне описания не могут иметь одинаковые имена

8.6. Для класса рациональных дробей с числителями и знаменателями некоторого интегрального типа

```
template<class T> class fr { T n; T d; ... };
```

описать два варианта (методом класса и функцией-другом этого класса) реализации вне класса операций, либо объяснить, почему какой-либо из вариантов невозможен:

- '+', выполняющей сокращение числителя и знаменателя рациональной дроби, если они имеют общие множители (унарный «+»).
- '+', выполняющей сложение двух рациональных дробей, либо рациональной дроби и значения соответствующего интегрального типа.
- '-', выполняющей изменение знака рациональной дроби (унарный «–»).
- '-', выполняющей вычитание двух рациональных дробей, либо рациональной дроби и значения соответствующего интегрального типа.
- '*', выполняющей умножение одной рациональной дроби на другую, либо рациональной дроби на значение соответствующего интегрального типа.
- '/', выполняющей деление одной рациональной дроби на другую, либо рациональной дроби на значение соответствующего интегрального типа.
- '=', выполняющей присваивание рациональной дроби значения другой рациональной дроби, либо значения соответствующего интегрального типа.
- '+=', выполняющей увеличение рациональной дроби на рациональное значение, либо на значение соответствующего интегрального типа.

- '-=', выполняющей уменьшение рациональной дроби на рациональное значение, либо на значение соответствующего интегрального типа.
- '*=', выполняющей присваивание с умножением рациональных дробей, либо с умножением рациональной дроби на значение соответствующего интегрального типа.
- '/=', выполняющей присваивание с делением рациональных дробей, либо с делением рациональной дроби на значение соответствующего интегрального типа.
- '==', выполняющей сравнение двух рациональных дробей, либо сравнение рациональной дроби со значением соответствующего интегрального типа.
- '!= ', выполняющей сравнение двух рациональных дробей, либо сравнение рациональной дроби со значением соответствующего интегрального типа.
- '<', выполняющей сравнение двух рациональных дробей, либо сравнение рациональной дроби со значением соответствующего интегрального типа.
- '>', выполняющей сравнение двух рациональных дробей, либо сравнение рациональной дроби со значением соответствующего интегрального типа.
- '<=', выполняющей сравнение двух рациональных дробей, либо сравнение рациональной дроби со значением соответствующего интегрального типа.
- '>=', выполняющей сравнение двух рациональных дробей, либо сравнение рациональной дроби со значением соответствующего интегрального типа.
- увеличения '++', выполняющей увеличение на 1 значения рациональной дроби.
- уменьшения '--', выполняющей уменьшение на 1 значения рациональной дроби.
- '<<', выполняющей вывод в текстовый поток значения рациональной дроби в виде «числитель/знаменатель».
- других операций, представляющихся полезными при работе с рациональными дробями.

9. STL

9.1. Дать определение контейнера. Каково назначение контейнеров?

9.2. Перечислить основные контейнеры библиотеки STL.

9.3. Какие виды итераторов допускают контейнеры *vector* и *list*?

9.4. Какие виды операций сравнения итераторов допустимы для этих контейнеров?

9.5. Дать определение итератора.

9.6. Какие категории итераторов определены в STL?
Чем отличаются друг от друга итераторы разных категорий?

9.7. Чем различаются прямые и обратные итераторы?
Привести пример использования обратного итератора.
Какие виды итераторов допускают обратные итераторы?

9.8. Какие виды операций сравнения итераторов допустимы для двунаправленных итераторов?
Привести пример ошибочного использования двунаправленного итератора.

9.9. Сравнить возможности, предоставляемые двунаправленным итератором и итератором произвольного доступа.

9.10. Перечислить типы итераторов библиотеки STL, допускающих использование в многопроходных алгоритмах.
Сравнить наборы операций, предоставляемые одно- и двунаправленными итераторами.
Привести по одному примеру ошибочного использования одно- и двунаправленных итераторов.

9.11. Верно ли решена задача: «Описать функцию, суммирующую значения элементов списка, стоящих на нечетных местах, считая, что элементы списка нумеруются с 1»? Если есть ошибки, объясните, в чем они заключаются и как их исправить.

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

int main ()
int g (list <int> & lst){
    int S=0;
    list<int> :: iterator p = lst.begin();
    while (p != lst.end())
        { S += * p; p += 2;
        }
    return S;
};
```

9.12. Описать функцию, которая добавляет после каждого элемента заданного контейнера-списка *list* <***int***> еще один такой же элемент, но с обратным знаком, а затем исключает из списка все отрицательные элементы и распечатывает результат.

9.13. Описать функцию, которая считает количество положительных элементов заданного контейнера-списка *list* <***int***>, а затем распечатывает это значение (выдает в стандартный поток *cout*).

9.14. Описать функцию, которая, не возвращая никакого значения, по заданному контейнеру *vector* <***bool***> считает количество истинных и ложных элементов в нем, а затем выдаёт эти значения в стандартный поток *cout*.

9.15. Описать функцию, которая печатает *«Yes»* или *«No»* в зависимости от того, содержится ли заданное целое число *x* в заданном контейнере-списке *list* <***int***>.

9.16. Описать функцию, которая удаляет каждый второй элемент заданного контейнера-вектора *vector* <***char***>, а затем распечатывает его элементы в обратном порядке.

9.17. Описать функцию, которая удваивает (добавляет еще один такой же) каждый элемент заданного контейнера-списка *list* <***int***>, а затем распечатывает его элементы в обратном порядке.

9.18. Описать функцию *g ()* с тремя параметрами: непустой и неизменяемый контейнер-вектор типа *vector* <***float***>, непустой контейнер-список типа *list* <***float***>, целое число — шаг по первому контейнеру. Функция должна исследовать элементы списка, выбираемые от его конца с шагом, равным 1, и элементы вектора, выбираемые от его начала с шагом, равным третьему параметру. Если обнаруживаются пары элементы разных знаков, то у текущего элемента списка должен меняться знак. Изменённый список распечатывается в прямом порядке. Функция возвращает общее количество неотрицательных элементов списка.

9.19. Описать функцию *g ()* с тремя параметрами: непустой контейнер-вектор типа *vector* <***int***>, непустой контейнер-список типа *list* <***int***>, целое число – шаг по первому контейнеру. Функция должна, последовательно проходя по списку от начала к концу, перезаписывать на место очередного его элемента соответствующий очередному шагу элемент вектора (сам вектор при этом не изменяется), а затем распечатывать элементы списка в обратном порядке. Функция возвращает количество изменённых элементов списка.

9.20. Описать функцию *g ()* с тремя параметрами: непустой и неизменяемый контейнер-список типа *list* <***long int***>, непустой контейнер-вектор типа *vector* <***long int***>, целое число — шаг по второму контейнеру. Функция должна копировать отрицательные элементы списка с шагом, равным 1, в уже имеющийся контейнер-вектор, от его начала к концу с шагом, равным третьему параметру, а затем распечатывать элементы вектора в прямом порядке. Функция возвращает количество измененных элементов вектора.

9.21. Описать функцию *g ()* с тремя параметрами: непустой контейнер-вектор типа *vector* <***int***>, непустой контейнер-список типа *list* <***int***>, целое число — шаг по первому контейнеру. Функция должна, последовательно проходя по списку от начала к концу, перезаписывать на место очередного его элемента соответствующий очередному шагу элемент вектора (сам вектор при этом не изменяется), а затем распечатывать элементы списка в обратном порядке. Функция возвращает количество изменённых элементов списка.

9.22. Описать функцию *g ()* с тремя параметрами: непустой и неизменяемый контейнер-вектор типа *vector* <***double***>, непустой контейнер-список типа *list* <***double***>, целое число – шаг по первому контейнеру. Функция должна сравнивать элементы списка, выбираемыми от его начала с шагом, равным 1, с элементами вектора, выбираемыми от начала с шагом, равным третьему параметру. Если обнаруживается несовпадение очередной выбранной пары, то в список в текущем месте вставляется отсутствующий элемент. Изменённый список распечатывается в обратном порядке. Функция возвращает количество элементов, вставленных в список.

9.23. Описать функцию *g ()* с параметром, представляющим собой контейнер-вектор элементов целого типа. Функция должна менять местами значения элементов вектора, одинаково удалённых от начала и конца вектора (первого с последним, вто-

рого с предпоследним и т. д.). Функция возвращает число сделанных перестановок.

9.24. Описать функцию *g ()* с параметром, представляющим собой контейнер-вектор указателей на элементы вещественного типа. Считая от начала контейнера, функция должна обнулять значения, на которое указывают указатели с четными номерами, если значения, на которые указывают указатели с нечетными номерами, отрицательны, а затем распечатывать значения, на которые указывают элементы контейнера в обратном порядке. Функция возвращает число измененных значений.

9.25. Описать функцию *g ()* с параметром, представляющим собой контейнер-список указателей на элементы длинного целого типа. Функция, просматривая контейнер от конца к началу, меняет знак значения, на которое указывает указатель с четным номером, если значение, на которое указывает указатель с нечетным номером, отрицательно, а затем распечатывать значения, на которые указывают элементы контейнера в прямом порядке. Функция возвращает число измененных элементов.

9.26. Описать функцию *g ()* с параметрами, представляющими собой контейнер-список целых элементов и контейнер-вектор указателей на элементы такого же типа. Функция должна, последовательно проходя по элементам контейнеров от начала к концу вектора и от конца к началу списка, менять местами элементы контейнеров, а затем распечатывать целые значения элементов контейнеров в прямом порядке (сначала весь список, затем вектор). Функция возвращает количество переставленных элементов контейнеров.

9.27. Описать функцию *g ()* с параметром, представляющим собой контейнер-вектор элементов целого типа. Функция должна менять местами значения соседних элементов с четным и нечетным номерами, считая от конца контейнера, если четный элемент меньше нечетного, а затем распечатывать значения элементов контейнера в прямом порядке. Функция возвращает число измененных значений.

9.28. Описать функцию *g ()* с параметром, представляющим собой контейнер-вектор элементов целого типа. Функция должна считать число элементов, значения которых превосходят среднее значение элементов вектора, и распечатывать элементы контейнера в обратном порядке. Функция возвращает число измененных значений.

9.29. Описать функцию-шаблон (от одного параметра, который может быть контейнером STL, например, вектором целых чисел), которая для любого последовательного контейнера STL распечатывает его предпоследний элемент, если таковой имеется, а также функцию *main()*, которая формирует контейнер-список из 5 целых чисел и применяет к нему описанную функцию-шаблон.

9.30. Описать функцию-шаблон (от одного параметра, который может быть контейнером STL, например, вектором целых чисел), которая для любого последовательного контейнера STL распечатывает сумму его трёх последних элементов, если таковые имеются, а также функцию *main()*, которая формирует контейнер-вектор из 5 целых чисел и применяет к нему описанную функцию-шаблон.

9.31. Описать функцию-шаблон (от одного параметра, который может быть контейнером STL, например, списком целых чисел), которая для любого последовательного контейнера STL распечатывает каждый второй элемент, начиная с конца, а также функцию *main ()*, которая формирует контейнер-список из 5 целых чисел и применяет к нему описанную функцию-шаблон.

9.32. Даны описания:

```
typedef vector<int> V;
struct Weight_t {   V::size_type Index;   // индекс элемента вектора
                    float Weight;         // вес элемента вектора
};
typedef list<Weight_t> L;
```

Описать функцию *g ()*, которая по заданному вектору типа *V* и соответствующему ему списку типа *L*, просматривая список от начала к концу, вычисляет средневзвешенное значение обнаруженных элементов вектора (средний результат умножения элементов на их веса), выдавая в выходной поток значения и веса элементов.

9.33. Даны описания:

```
typedef vector<double> V;
struct Signif_t {   V::size_type Index;   // индекс элемента вектора
                    bool Signif; // значимость элемента (true – да)
};
typedef list<Signif_t> S;
```

Описать функцию *g ()*, которая по заданному вектору типа *V* и соответствующему ему списку типа *S*, просматривая список от начала к концу, вычисляет сумму обнаруженных значащих элементов вектора, выдавая в выходной поток индексы и значения суммируемых элементов.

9.34. Даны описания:

```
typedef vector<bool> B;
struct Value_t {  B::size_type Index;     // индекс элемента вектора
                  int Value;              // значение элемента
};
typedef list<Value_t> T;
```

Описать функцию *g (),* которая по заданному вектору значимости типа *B* и соответствующему ему списку типа *T*, просматривая список от конца к началу, выдаёт в выходной поток значения целочисленных полей элементов списка, сопровождаемое их значимостью, вычисляет сумму значимых целочисленных полей списка и возвращает это значение.

9.35. Даны описания:

```
struct Value_t {  bool Signif;     // значимость элемента (true – да)
                  int Value;       // значение элемента вектора
};
typedef vector< Value_t> B;
typedef list< B::size_type> T;
```

Описать функцию *g ()*, которая по заданному вектору типа *B* и соответствующему ему списку типа *T*, просматривая список от конца к началу, вычисляет сумму значимых целочисленных полей элементов вектора, и возвращает это значение, одновременно выдавая в выходной поток значения всех целочисленных полей элементов вектора, сопровождаемые их значимостью.

II. Ответы и решения

1. Абстрактные типы данных (АТД). Классы. Конструкторы и деструкторы

1.1. Ошибочны 2, 3 и 4 описания объектов в функции *f ()*. Исправить можно так:

```
class A {
    int a, b;
public:
    A (const A & x) {
        a = x.a;
        b = x.b;
        cout << 1;
    }
    A (int  a = 0, int b = o) {
        this -> a = a;
        b = a;
        cout << 2;
    }
};
```

На печать будет выдано:

без оптимизации:	2221211
с оптимизацией:	22221

1.2.
```
A ( int a = 0,  int b = 1 ) {…}
```

1.4.
```
class A {
    int x;
public:
    A ( int y ) { x = y; }
    int get () { return x; }
    int  operator *=  ( int y ) { return  x =  x*y; }
};
```

1.16. Cons Copy Cons Des Copy Des Des 1 Des

1.17. Cons Cons Des Copy Des 3 Des

1.18. Cons Copy Cons Des Des 1 Des

1.19. 1) перед деструктором базового класса добавить ***virtual*.**
2) добавить либо *B* (***float a* = 0**), либо *D* ( ) : ***B* (0)** {...}

1.20. 1) перед деструктором базового класса добавить ***virtual*.**
2) добавить *B* (***int a* = 0**) {...}

1.21. 1) перед деструктором базового класса добавить ***virtual*.**
2) добавить *D* (***int d* = 0**) {...}

2. Перегрузка операций. Перегрузка функций

2.1. Необходимо убрать в приватную часть операцию присваивания от аргумента типа «ссылка на *Cls*», например:

```
class Cls {
        void operator= (const Cls &) {}
public:
        Cls () {}
};
```

2.2

a)
```
void g (double a, int b = 0 );
void g (int, const char * = 0);
```
Возможны варианты с ***double***, ***char***...

b)
```
void f (const char * a = 0 );
void f (int a, int b = 0);
```
Возможны варианты с char...

e)
```
void f (double d = 0, const char * s = 0);
void f (const char *);    либо   void f (const char *, int i = 0);
```

2.3.

a)
```
f();  // f(int),   (а) на этапе выбора функций,
      //              подходящих по кол-ву параметров;
f(s); // f(int),   (б) целочисленное расширение
f(f); // f(double),(б) вещественное расширение
f(b); // f(int),   (б) целочисленное расширение
f(A); // f(int),   (б) целочисленное расширение
```

b)
```
f();  //    (а) неоднозначность определения числа параметров
f(4); //    (в) неоднозначность при стандартном преобразовании
f(f); // f(double),(б) вещественное расширение
f(d); // f(double),(а) точное отождествление
f(ld);// f(long double),(а) точное отождествление
```

c)
```
f(s);  // f(int),   (б) целочисленное расширение
f(i); // f(int),   (а) точное отождествление
f(b); // f(int),   (б) целочисленное расширение
f(f); // (в) неоднозначность при стандартном преобразовании
f(d); // (в) неоднозначность при стандартном преобразовании
```

3. Наследование. Видимость и доступность имен

3.1.

a)
```
void B::g() {
        f();          // ошибка, эта f не видна
        f(1);         // B::f(1);
        f(5 , 1);     // ошибка эта f не видна
        x = 2;        // ошибка, так как int x private в базовом классе
}

int main () {
        B b;   // A(), B()
        f(5);  //ошибка, эта f не видна
        f('+', 6);    //::f('+', 6);
        b = ret (b, b); // A(const A &), B(const B &)
        return 0;
}
```
Будут вызваны следующие конструкторы и деструкторы:
`A(), B(), A(const A &), B(const B &), ~B(), ~A(), ~B(), ~A().`

b)
```
void B::g() {
        f(1.2);  // B::f
        f();     // ошибка эта f не видна
        a = 2;   // A::a = 2
}

int main () {
        B d;
        f();    //ошибка
        f(6);   //::f
        empty  (d, d);
        return 0;
}
```
Будут вызваны следующие конструкторы и деструкторы:

`A(), B(), A(const A &), B(const B &), ~B(), ~A(), ~B(), ~A().`

3.2.

a)
```
C::x = A::f ();
C::f (3);
C::x = ::f (4, 5);
C::x = 6;

c.A::f ();
c.C::f (7);
::x =::f ('8', 9);
```
Подчеркнуты исправляющие (обязательные) расширения.
Будут вызваны следующие конструкторы и деструкторы:
`A(), B(), C(), B(const B &), A(const A &), ~A(), ~B(), ~C(), ~B(), ~A()`

b)
```
x =  A:: f ();
x =  B:: f (5);
x =  B:: f (6, 6);
```

```
x =   B:: f (5);
x = a.A:: f ();
x = a.B:: f (7);
return  a.B::g (& a, & a);
```

Подчеркнуты исправляющие (обязательные) расширения.

Будут вызваны следующие конструкторы и деструкторы:

```
A(), B(), A(), B(), A(const A &), C(), ~C(), ~A(), ~B(), ~A(), ~B(), ~A()
```

c)

```
C::x = A::f ();
C::f (3);
C::x = ::f (4, 5);
C::x = 6;
//main()
c.A::f ();
c.C::f (7);
::x =::f ('8', 9);
```

Подчеркнуты исправляющие (обязательные) расширения.

Будут вызваны следующие конструкторы и деструкторы:

```
A(), B(), C(), B(const B &), A(const A &), ~A(), ~B(), ~C(), ~B(), ~A().
```

3.3. Требуется уточнить обращения к полю *m* с помощью операций разрешения области видимости «*::*» для класса *B* либо для *C* , например, так:

```
fA.C::m = 0;
return *((* f).e = & fA.C::m);
```

Либо во всех описаниях производных классов вставить указание виртуального наследования (всего 4 поправки, из которых исправления в классах *B* и *C* при таком подходе — обязательны):

```
class B: virtual public A { public: int * p; };
class C: virtual public A { public: int * c; };
```

3.5.

1. Чтобы преобразование из класса *W* в функции *h ()* в класс *Z* стало возможным, это преобразование надо исправить:

```
pz = (Z *)(X *)(new W);
```

2. Дополнительно необходимо сделать открытым вид наследования класса *W* классом *X* и класса *X* классом *Z*:

```
class X: public W   { ... };  class Z: public X,Y {  ... };
```

3. Необходимо уточнить область видимости поля *w* в теле функции *h ()*:

```
(*pz).X::w=& hi;
```

4. Открытости наследования можно добиться, заменяя в наследующем классе слово ***class*** словом ***struct***.

4. Виртуальные функции. Абстрактные классы

4.1.

1. Имеется иерархия классов, хотя бы из двух классов — базового и производного.
2. В базовом классе функция объявлена с ключевым словом ***virtual.***
3. В производном классе есть функция с таким же именем, с таким же списком параметров (количество, типы и порядок параметров совпадают) и с таким же типом возвращаемого значения.
4. Вызов функции производного класса осуществляется через указатель на объект базового класса (или с помощью ссылки на объект базового класса) без указания самого объекта и уточнения с помощью операции разрешения области видимости.

```
class B          { public: virtual void print (); };
class P: public B { public:        void print (); };
void B::print () { ... }
void P::print () { ... }

B * ps = new P; ...  ps -> print ();
```

4.2.

```
int main( ){
    X a;    Z b;    X * p = &b;
    p -> g(1.5); // печатается   Z::t
                 //              Z::h
                 //              Z::g
    p -> h();   //печатается           X::t
                //                     X::h
    p -> t(5);  // ошибка, X::t – без параметров
  }
```

4.4.

```
int main( ){
    K k;  P p;  K *t = &p;

    t -> f (0.7); // печатается   K::f
     t -> g ();    // печатается   K::f
                   //             K::g
         t -> h();  //печатается    P::f
                    //             P::g
                    //             P::h
}
```

4.6. Ошибок нет.

Будет выдано в поток: `D::h  D::h,2  D::f,2  D::h  D::h,1  D::g  D::h  D::h  D::h,3`

4.7. Ошибочным является вызов *p->h (2)* (Функция *h **(int)*** из производного класса не замещает *h ()* базового класса, так как имеет отличающийся набор параметров).

Будет выдано в поток: `U::h,1  U::f,1  T::h  T::g  T::h`

4.10.

```
dog
sheep
result = (33 ; 4)
horse
cat
```

4.11. При передаче переменной типа *A* в функцию *g ()* по значению должен быть создан объект типа *A*, а это невозможно, поскольку структура содержит чистую виртуальную функцию.

4.12. Полем структуры *A* является объект типа *S*, а это невозможно, поскольку структура *S* содержит чистую виртуальную функцию.

4.13. При передаче переменной типа *B* в функцию *g ()* по значению должен быть создан объект типа *B*, а это невозможно, поскольку структура содержит чистую виртуальную функцию.

4.14. `f (int) from B`

`10 10 10 10`

4.15. `add_st (K*) from K`

`5 5 5`

4.16. `g ( ) from T`

`7 7 7`

5. Аппарат исключений

5.1. Будет напечатано:

12 ¬ 9 ¬ 2 ¬ 4 ¬ 6 ¬ 11

5.6.
```
4 ¬ 11 ¬ 9 ¬ 12 ¬  2 ¬ 5 ¬ 10 ¬ 6 ¬ 8 ¬ 5 ¬ 14
```

5.7.
```
cat
elephant 91
```

5.8.
```
lance
arche 61
```

5.9.
```
plane
boat 52
```

6. Константные и статические члены класса.

6.1. Исправления:

```
class A {
public:
    static int y;
    void f() const {cout << "f" << endl;}
};
```

6.3. а)

1. Заменить слово ***class*** словом ***struct***, либо вставить после открывающей фигурной скобки определения класса слово ***public:***.

2. Сделать метод *g ()* статическим.

b)

1. Заменить слово ***class*** словом ***struct***, либо вставить после открывающей фигурной скобки определения класса слово ***public***:.

2. В определениях методов *f ()* и *g ()* вставить после пустого списка формальных параметров слово ***const***.

6.4. **a)**

```
class A {
public:
    static int x;
    void get_0() const {return;}
};
```

b)

```
class A {
public:
    static const char a;
    static void f(){}
};
```

6.5. Ошибочными являются указания на то, что статические методы *f ()* вызывают нестатические методы. Надо исключить в методах *f ()* вызовы методов *h ()*:

На печать будет выдано:

```
B::f,0   A::f,1   A::g   A::g   A::h,2   A::f,3
```

7. Динамическая идентификация и приведение типов

7.1. Ошибочные конструкции переведены в примечания. Лишние операции динамического приведения типа заменены операциями присваивания.

```
k = s;                  //  Приведение производного указателя к
базовому
// s = dynamic_cast <S *>(k);       // Тип K – не полиморфный
// l = dynamic_cast <L *>(k);       // Тип K – не полиморфный
m = s;                  //  Приведение производного указателя к
базовому
// p = dynamic_cast <P *>(l);       // Тип L – не полиморфный
q = dynamic_cast <Q *>(m);
r = dynamic_cast <R *>(q);
// s = dynamic_cast <S *>(p);       // Тип P – не полиморфный
```

7.2. Вариант исправления функции *f9 ()* с использованием операции ***typeid*** (возможен также вариант исправления с использованием операции ***dynamic_cast***):

```
void f9 (B & b, D & d, int n)
{   D * pd = (n > 0) ? & d : (D *) & b;
    if (typeid (* pd) == typeid (d))
    strcpy (pd -> y, “one_variant\n”);
}

}
```

7.6. Вариант написания функции *f ()*:

```
B * f (A * p)
{   B* pb = dynamic_cast<B *> (p);
    if (pb) return pb;
    else exit (0);
}
```

8. Шаблоны

8.2. В качестве параметра шаблона можно использовать либо тип (что показывается служебными словами ***class***, ***typename***, но не ***struct***), либо параметр целочисленного, перечислимого, указательного, ссылочного типа, либо типа «указатель на функцию-член». Типы ***double***, ***float***, *myclass*, *mystruct*, ***struct*** к таковым не относятся. Ошибки в строках:

```
template <double f> void funcA (double d = f) { /*...*/ }
template <float f> class A { /*...*/ };
template <myclass c> class E { /*...*/ };
template <mystruct a> void funcE (mystruct *p = &a) { /*...*/ }
template <struct mystruct> void funcF (mystruct *p = 0) { /*...*/ }
```

8.4. Будет напечатано:

```
z = 2.8
k = 12
z = 1.5
z = 2.8
s3 = abft
Выбор сделан!
```

9. STL

9.15.

```
void f (const int x, const list <int> & l)
{   list<int>::const_iterator p = l.begin ();
    while (p != l.end ())
        if (* p == x)
        {   printf ("Yes\n");
            return;
        }
    printf ("No\n");
}
```

III. Литература

1. Standard for the C++ Programming Language ISO/IEC 14882, 1998.
2. Страуструп Б. Язык программирования C++. Специальное изд./Пер. с англ. — М.: «Бином», 2005.
3. Волкова И.А, Иванов А.В., Карпов Л.Е. Основы объектно-ориентированного программирования. Язык программирования Си++. — М.: МГУ, МАКС Пресс, 2011.–112с.

СОДЕРЖАНИЕ