

Авторское оформление кода сохранено. Выявленные во время написания коллоквиума студентами опечатки исправлены.

1. Если есть ошибки в операторах функции main(), исправьте их с помощью операции указания области видимости «::». Вычеркните операторы, которые не удалось исправить с помощью «::». Для оставшихся операторов с помощью операции «::» обозначьте ту область видимости, к которой относится вызываемый метод.

```
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); };
A * p; B b; C c;
int main() { 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; }
```

2. Есть ли ошибки в интерфейсах классов C и D программы на C++? Если есть, то **объясните**, в чём 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; }
```

3. Для каждого вызова перегруженной функции с одним параметром укажите шаг алгоритма, на котором будет выбрана наиболее подходящая функция или выявлена неоднозначность. Для выбранной функции укажите её прототип.

```
int f(double a=1.0){return a;} int f(long double a=5.0){return a;}
int main () { float f = 1.0f; double d = 2.0; long double ld = 3.01;
            f ();
            f (4);
            f (f);
            f (d);
            f (ld);
}
```

4. Для объектов из задания 1 определить, какие конструкторы и деструкторы и в каком порядке будут выполняться при работе следующего фрагмента программы:

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

5. В открытой части класса `c1`, предназначенного для работы со сложными объектами, имеются конструктор и функция преобразования. Для облегчения работы с объектами этого класса создаётся дополнительный класс `ObjPtr`, в закрытую часть которого включается поле указателя на класс `c1`.

```
template <class T> class c1 { /* ... */
public: c1 (const T s = 0);
       operator T ();
};
template <class T> class ObjPtr { /* ... */ c1 <T> * c; public: /* ... */
};
```

Для класса «ObjPtr» написать реализацию операций сравнения на равенство «==» таким образом, чтобы правильными оказались выражения (обязательно использовать и методы, и внешние функции):

```
ObjPtr<int> t1, t2;
/* ... */ (t1 == 3) || (5 == t1) && (t1 == t2) /* ... */
```

6. Что такое наследование и для чего оно используется в Си++? Какой длины и какого вида цепочки можно строить в Си++? Как синтаксически описывается наследственная иерархия разных видов в Си++?
7. 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;
}
```

8. Если есть ошибки в следующем фрагменте, то в чём 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; }
```

9. Что такое «дружественная функция»? Описать основное отличие функций-друзей от методов классов. Привести пример использования дружественной функции.
10. Написать функцию `g()` с параметром — контейнером-списком указателей на элементы длинного целого типа. Функция просматривает контейнер от конца к началу и меняет знак значения, на которое указывает текущий указатель, если следующее за ним значение отрицательно. Например, список $(-1, -2, 3, 4)$ превратится в $(-1, 2, -3, 4)$. Затем функция печатает значения, на которые указывают элементы контейнера, в прямом порядке. Функция возвращает число изменённых элементов.