Ответы_Вариант 1_2013

1. **Ответ:**

```
void N1::N2::B::g() {
                        N1::N2::B::x = 20.13;
                        N1::A::f ();
                        N1::N2::B::f (2013);
                        N1::f (13, 20132013); }

int main (){            N1::N2::B b;
                        b.N1::A::f ();
                        b.N1::N2::B::f (13);
                        ::x =N1::f ('2', 013);  }
```

Отмечены исправляющие (обязательные) изменения.

Критерии: За отсутствие каждого **необходимого (из 6)** расширения имени (без которого ошибка) **-2** .
За каждое другое не отмеченное уточнение области видимости (из 6) **-1**.
При указании хотя бы одного правильного расширения имени (где это необходимо) за отсутствие вариантов расширений не наказывать.
Для остальных расширений имен считать достаточным указание ближайшей области видимости, за отсутствие полной цепочки вложенных областей видимости – не наказывать.

2. Неверные вызовы:

		Будет выдано в поток:
`pc -> f()`	(в классе *C* нет функции с именем *f*, поле *f* не доступно),	`A::h,65`
`pa -> f(2013)`	(в классе *A* нет функции *f()* с параметром),	`A::f`
`pb -> g()`	(в классе *B* нет функции *g()* без параметров).	`C::h,13`
`pc -> h()`	(в классе *C* нет функции *h()* без параметров)	`B::g,13`
		`A::h,65`
		`A::f`
		`C::h,13`
		`B::g,13`
		`C::h,0`
		`C::h,67`
		`C::g`
		`C::h,0`

Критерии: За каждую не найденную\лишнюю ошибку или неправильно вызванную функцию **– 1**.

3. Функция преобразования – метод класса специального вида (***operator*** <*mun*> *()*), которые преобразуют объект класса к типу <*mun*>, например, "***operator int()***". Алгоритм преобразования кодируется в теле функций преобразования.
1). Пользовательские преобразования автоматически применяются только, если они однозначны. 2). Допускается не более одного пользовательского преобразования для обработки одного параметра одного вызова.

```
struct S { long s;
           S(){ s = 0L; }                    // конструктор умолчания
           operator long (){ return s; } }; //   S --> long

long f(long l) { return l; }
```

Теперь разрешено: `S s; long nl = f (s);`

Критерии: За неправильное объяснение **–5**.
За каждое из двух неправильно описанных или пропущенных ограничений **–3**.
За отсутствие примера **–2**.

4. Ответ: `A(2013), A(13), A(13), B(), C(), ~C(), ~B(), ~A(), ~A(), ~A().`

Критерии: За каждую ошибку **–3**.
За указание `A()` вместо `A(2013)` снижать на **1** балл.

5.

```
template<class T> fraction<T> fraction<T>::operator/ (const fraction<T> & x)
          { fraction<T> w;  w.n = n * x.d;     w.d = d * x.n;    return w; }
template<class T> fraction<T> operator/ (const fraction<T>& x, const fraction<T>& y)
          { fraction<T> w;  w.n = x.n * y.d;    w.d = x.d * y.n;    return w; }
```

Критерий: За отсутствие\ошибку каждого примера (за однотипные ошибки наказывать 1 раз): **-5.**

6. При исследовании возможности использования имени проводится (1) проверка однозначности, (2) выбор перегруженной функции, (3) контроль доступа.

```
class A { int g(int); public: double x;... };
class B {             public: double x; void f (short); void f (char); ... };
class C: public A, public B { public: ... };
C c; C * p = & c; p -> x = 1.0;   // неоднозначность наследования поля A::x
p -> f (1);                       // неоднозначность выбора функции f()
p -> g (2);                       // запрет доступа к функции A::g(int)
```

Критерий: За пропуск одного из трёх шагов: **-3**.
За указание их неправильного порядка: **-3**.
За отсутствие каждого из трёх примеров: **-1**.

7. **Ответ:** `XcftXkfcftf0fimc`

Критерии: За каждую неверную (лишнюю или пропущенную) печать **-2**.

8. Ответ:

1. Заменить слово *"class"* словом *"struct"*.
2. Перед определением типа поля *"i"* вставить слово *"static"*.
3. В определениях методов *"f()"* и *"g()"* вставить после пустого списка формальных параметров слово *"const"*.
4. В определениях методов *"f()"* и *"g()"* вставить перед типом формального параметра слово *"const"*.

Критерий: За пропуск ошибок доступа к закрытым членам класса: **-5**.
За пропуск указания статичности поля *"i"*: **-8.**
За пропуск каждой из двух ошибок вызова константных методов: **-4.**
За пропуск каждой из двух ошибок в заголовках функций: **-4.**
За указание ошибки там, где её нет: **-5.**

9. **Ответ:**

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

Критерий: За утверждение о невозможности безопасного преобразования ставить полный балл: **10**.
За безошибочное решение с использованием динамического приведения (как здесь написано, как будто класс полиморфен): **-1**.
За любую другую ошибку: **-10** (включая использование статического приведения).

10. **Решение:**

```
float g (const V & vect, const L & lst)
{ L::const_iterator lp = lst.begin ();
  V::size_type t, n = 0; float vw;
  float w = 0.0f;
  while (lp != lst.end ())
     {      t = (* lp).Index;
            vw = (* lp).Weight; ++ lp;
            if (t < 0 || t >= vect.size ()) continue;
            cout << vect [t] << ' ' << vw << endl;
            w += vect [t] * vw;
            ++ n;
     }
  return n ? w / n : 0.0f;
}
```

Критерий: За каждую серьёзную ошибку: **-3**.
За использование обычного итератора вместо константного, отсутствие проверок на пустой список и выход за пределы диапазона индексов не наказывать.