```cpp
// declaration
// (D,F,K,sigma,tau)->Black(D,F,K,sigma,tau)

template <typename D, typename F, typename K, typename V, typename T>
struct BlackTraits;

template <typename D, typename F, typename K, typename V, typename T> inline
typename BlackTraits<D, F, K, V, T>::type black(const FuncExpr<D>& d,
    const FuncExpr<F>& f, const FuncExpr<K>& k, const FuncExpr<V>& v,
    const funcExpr<T>& t)
{
    return BlackTraits<D, F, K, V, T>::apply(d(), f(), k(), v(), t());
}

// (D,F,K,sigma,tau) -> D((FÎ¸(d_1)-KÎ¸(d_2))
template <typename D, typename F, typename K, typename V, typename T>
struct BlackTraits {
    typedef ForwardBlackTraits<F, K, V, T> FwdB1;
    typedef FuncBinatyTraits<D, typename FwdB1::type, scalar_mult> Disc;
    typedef typename Disc::type type;

    static type apply(const D& d, const F& f, const K& k, const V& v, const T& t)
    {
        return Disc::apply(d, FwdB1::apply(f,k,v,t));
    }


};

// (F, K, sigma, tau)->FÎ¸(d_1)-KÎ¸(d_2)
template <typename D, typename F, typename K, typename V, typename T>
struct ForwardBlackTraits {
    typedef AoNTraits<F, K, V, T> AoN;
    typedef CoNTraits<F, K, V, T> CoN;
    typedef FuncBinaryTraits<typename AoN::type, typename CoN::type,
        scalar_minus> Minus;
    typedef typename Minus::type type;

    static type apply(const D& d, const F& f, const K& k, const V& v, const T& t)
    {
        return Minus::apply(AoN::apply(f, k, v, t), CoN::apply(f, k, v, t));
    }
};

// (F, K, sigma, tau)->FÎ¸(d_1)
template <typename D, typename F, typename K, typename V, typename T>
struct AoNTraits {
    typedef D1Traits<F, K, V, T> D1;
    typedef FuncUnaryTraits<typename D1::type, scalar_normdist> N1;
    typedef FuncBinaryTraits<F, typename N1::type, scalar_mult> Mult;
    typedef typename Mult::type type;

    static type apply(const F& f, const K& k, const V& v, const T& t)
    {
        return Mult::apply(f, N1::apply(D1::apply(f, k, v, t)));
    }
};

// (F, K, sigma, tau)->KÎ¸(d_2)
template <typename F, typename K, typename V, typename T>
struct CoNTraits {
    typedef D2Traits<F, K, V, T> D2;
    typedef FuncUnaryTraits<typename D2::type, scalar_normdist> N2; // 2é \205æ¼\224ç®\227traits
    typedef FuncBinaryTraits<F, typename N2::type, scalar_mult> Mult; // 3é \205æ¼\224ç®\227traits
    typedef typename Mult::type type;

    static type apply(const F& f, const K& k, const V& v, const T& t)
    {
        return Mult::apply(f, N2::apply(D2::apply(f, k, v, t)));
    }

};

// (F, K, sigma, tau)->d_2 + sigma * sqrt(tau)
template <typename F, typename K, typename V, typename T>
struct D1Traits {
    typedef D2Traits<F, K, V, T> D2;
    typedef StdevTraits<V, T> Sd;
    typedef FuncBinaryTraits<typename D2::type, typename Sd::type, scalar_plus> Plus;
    typedef typename Plus::type type;

    static type apply(const F& f, const K& k, const V& v, const T& t)
    {
        return Plus::apply(D2::apply(f, k, v, t), Sd::apply(v, t));
    }
};

// (F, K, sigma, tau)->log(F/K)/(sigma * sqrt(tau) + 0.5 * sigma * sqrt(tau)
template <typename F, typename K, typename V, typename T>
struct D2Traits {
    typedef LogMoneynessTraits<F, K> LnFK;
    typedef StdefTraits<V, T> Sd;
    typedef Rational<double, 1, 2> Half;
    typedef FuncBinaryTraits<typename LnFK::type, typename Sd::type, scalar_div> X1;
    typedef FuncBinaryTraits<typename X1::type, typename X2::type, scalar_mult> X2;
    typedef FuncBinaryTraits<typename X1::type, typename X2::type, scalar_plus> Plus;
    typedef typename Plus::type type;

    static type apply(const F& f, const K& k, const V& v, const T& t)
    {
        return Plus::apply(X1::apply(LnFK::aply(f, k), Sd::apply(v, t)),
            X2::apply(Half(), Sd::apply(v, t)));
    }
};

// (F,K)->log(F/K)
template <typename F, typename K>
struct LogMoneynessTraits {
    typedef FuncBinaryTraits<F, K, scalar_div> FK;
    typedef FuncUnaryTraits<typename FK::type scalar_log> Ln;
    typedef typename Ln::type type;

    static type apply(const F& f, const K& k)
    {
        return Ln::apply(FK::apply(f, k));
    }
};

// (sigma, tau) -> sigma * sqrt(tau)
template <typename V, typename T>
struct StdevTraits {
    typedef FuncUnaryTraits<T, scalar_sqrt> Sqrt;
    typedef FuncBinaryTraits<V, typename Sqrt::type, scalar_mult> Mult;
    typedef typename Mult::type type;

    static type apply(const V& v, const T& t)
    {
        return Mult::apply(v, Sqrt::apply(t));
    }
};

int main()
{

    return 0;
}
```