О структуре программы по моделированию и статистике

(простейший вариант)

В Doc нужно хранить все те данные, которые потом будут использоваться.

Давайте их перечислим:

• параметры распределения, с которыми моделируется распределение

• параметры распределения для нулевой гипотезы

• результат моделирования (выборка: массив плюс размер)

• частоты теоретического распределения

• частоты эмпирического распределения

• результаты применения хи-квадрат.

(Еще – массив из p-levels и массив для третьего пункта задания, но их пока не будем рассматривать. Их разрешается хранить в виде массивов, не упаковывая в классы.)

Все эти данные должны храниться в Doc в структурированном виде, т.е.

в виде классов (объектов).

Параметры распределений можно хранить отдельно, объединив их в класс или структуру `Distribution`. Этот объект должен уметь считать теоретические вероятности, число состояний (если распределение бесконечное), возвращать название распределения. Получится что-то вроде `Distribution d0, d1;`

Выборка должна храниться в абстрактном базовом классе `Sample` (можно подобрать более удачное название, например, `BinSampleGenerator`), наследники которого обеспечивают полиморфизм с виртуальной функцией `Simulate()` для моделирования разными методами. В `Doc` храним на него указатель `Sample *s;`
При инициализации моделирования (когда будет нужно) используем параметры из `d1`.

Теперь нужно решить, как будут храниться частоты. Можно сделать класс `Chi2Histogram` и завести поле `Chi2Histogram h;`. В класс `Chi2Histogram` на вход (в конструкторе и в `SetData(…)`) будут поступать выборка и `d0`. Он внутри организует два массива частот – теоретические и эмпирические. В нем есть функция `CalcChi()`, которая умеет на основе массивов частот считать статистику критерия Chi2, число степеней свободы и P-level. Это все нужно сделать полями класса. Функция вычисления chi2 не должна портить массивы частот, хранящиеся в классе. Итак, мы все рассортировали по классам и наметили, как классы между собой взаимодействуют. Возможно, понадобится знать, какой именно метод в данный момент выбран и в каком режиме программа работает (режиме Гистограмма или P-levels, например).

В конструкторе Doc вы записываете в поля класса какие-то значения по умолчанию. Указатель s либо инициализируете нулем, либо пишете что-то конкретное вроде
`s=new SampleTable(…);`

Правило: *все поля класса, которые являются указателями, в конструкторе должны инициализироваться либо nullptr, либо new. В деструкторе должны быть вызваны delete. Это относится и к Doc. Если понадобился new для поля класса не в конструкторе, а в какой-то другой функции, то действия следующие:*

```
delete s; /*или delete[]s;*/ s=new …
```

Это правило позволит избежать утечек памяти.

Теперь поймем, что происходит при выборе пунктов меню. Пусть мы выбрали пункт Гистограмма. Тогда создается диалог и в него из того, что хранится в Doc, заполняются поля. Например,

`Dial d; d.m_size=s->size;` и т.д. (можно обработать и ситуацию, когда s==nullptr)

Затем вызывается диалог `if(d.DoModal()==IDOK)` …

Внутри if из диалога записывается информация обратно в поля класса Doc:

```
delete s;
isBernulli=d.m_method;
if(isBernulli) s=new SampleBernulli(d.m_size, Distribution(d.m_n1,d.m_p1) ,…);
/*или d1.n=d.m_n1; d1.p=d.m_p1; s=new SampleBernulli(d.m_size,d1,…);*/
else s=new SampleTable(d.m_size,…);
… // более универсально и правильно – использовать switch
s -> Simulate();
h.SetData(s, d);
h.Chi2();
```

Теперь частоты и p-level сосчитаны и можно вызвать `UpdateAllViews();`

Класс View берет нужные данные из Doc и рисует то, что нужно. Заметим, что данная система позволяет при следующем запуске диалога помнить те значения его полей, которые были установлены при предыдущем запуске.

Теперь о рисовании p-levels для определения истинных ошибок первого рода и для определения мощностей. Если мы считаем ошибки первого рода, то нам нужно задать только одно распределение, т.е. d0 задается через диалог, а потом присваивается `d1=d0;`

Для мощностей через диалог задаются d0 и d1; Проще вообще не делать различие: если пользователь задал d1 и d0 одинаковыми, то получится первый случай, если разными – то второй.

Теперь в цикле массив P-levels может генерироваться так:

```
for(…){
s -> Simulate();
h.SetData(s, d);
h.Chi2();
plevels[i]=h.plevel;
}
```

Сам цикл не зависит от того, что считается, ошибки или мощности. Дублирования кода (кусков почти одинакового кода) быть не должно!

Прототип класса организуется в виде заголовочного файла (см. текст по поводу их организации), а реализации функций класса – в виде файла cpp.