

Код в инструкции “очищен” от лишних и плохопонимаемых вещей, поэтому служит только для объяснения кода, содержащегося внутри “новых” вопросов. Для создания нового генератора лучше всего использовать основу задания 32+, так как они создавались в более позднее время, и эта инструкция написана на основе именно этих генераторов.

Шаг первый: основа

Необходимо создать новый класс, наследованный от `tquest`, в классе должны содержаться:

- `char* strpar1;` -Первая строка настроек
- `char* strpar2;` -Вторая строка настроек

Уже распарсенные настройки (в основном это макс\минимальные значения коэффициентов уравнения):

- `int amin;`
- `int amax;`
- `struct`
- `{`

```
`int point;`
```

```
`int vector;`
```

```
`int plate;`
```

- `} min,max;`
- `questXX(FILE* f);` -Конструктор с чтением из файла

- `questXX(){};` -Конструктор по умолчанию
- `Save(FILE* f);` -Сохранение по текущему дескриптору
- ``copy(questXX* q);` -Метод копирования
- `Edit();` -Метод вызова окна настроек
- `Print(TList* plist);` -Метод печати в контрольную работу
- `Print(TList* plist, class test &t);` -Метод печати в систему тестирования

Шаг второй: реализация

Подключаем хедер класса

- `#include "quest33.h"`

Подключаем хедер формы настроек

- `#include "MQ32Settings.h"`

Подключаем хедер доп. функций

- `#include "UtilsNG.h"`

Конструктор инициализации по ф. дескриптору должен считать строки настроек и присвоить значения необходимым переменным

- `quest33::quest33(FILE* f)`
- `{`
- `Log->Add("Q33 Init..");`
- `char* buf = new char[256];`

- `randomize();`
- `keygen = random (1000) + 1;`
- `reads(f,buf);`
- `strpar1 = strdup(buf);`
- `reads(f,buf);`
- `strpar2 = strdup(buf);`
- `sscanf(strpar1,"%d %d %d %d %d %d",&min.point,&max.point,&min.vector,&max.vector,&min.plate,&max.plate);`
- `qtype = type;`
- `delete buf;`
- `}`

Метод сохранения делает точно-обратный процесс, записывая всё в том же порядке, в котором оно считывалось

- `quest33::Save(FILE* f)`
- `{`
- `Log->Add("Q32 Saving..");`
- `fprintf(f,"head %i\n",type);`
- `fprintf(f,"%s\n",name);`
- `fprintf(f,"%i %i %i\n",itemnumber,subitemnumber,qtype);`
- `fprintf(f,"%s\n",strpar1);`
- `fprintf(f,"%s\n",strpar2);`
- `return 0;`
- `}`

Метод настроек “пропихивает” наш класс внутрь специально-созданного для этого класса формы (подробности ниже). Если генератор настраиваемых параметров не имеет - внутри метода оставить пустой код.

- `quest33::Edit()`
- `{`
- `Log->Add("Q33 Edit settings..");`
- `sscanf(strpar2,"%d %d %d %d %d %d",&min.point,&max.point,&min.vector,&max.vector,&min.plate,&max.plate);`
- `keygen = 0;`
- `nvar = 1;`
- `PLAngle->quest = this;`
- `PLAngle->MinPoint->Position = min.point;`
- `PLAngle->MinVector->Position = min.vector;`
- `PLAngle->MinPlate->Position = min.plate;`
- `PLAngle->MaxPoint->Position = max.point;`
- `PLAngle->MaxVector->Position = max.vector;`
- `PLAngle->MaxPlate->Position = max.plate;`
- `PLAngle->ShowModal();`
- `return 0;`

- }

Первая основная функция нашего класса: генератор задания для контрольной работы. Он должен выдать уравнение для решения, а так же просчитать часть преподавателя.

- `quest33::Print(TList* plist)`

- {

-

```
`Log->Add("Q32 Printing to test..");`
```

```
`struct`
```

```
`{`
```

```
`int X0, Y0, Z0;`
```

```
`int A,B,C,D;`
```

```
`int l,m,n;`
```

```
`} cc;`
```

Необходима генерация коэффициентов. Используются родные функции ментора, а так же “Свежие”, дополнительные функции

```
`cc.X0 = -1*sign()*rgen(keygen,1,min.point,max.point);`
```

```
`cc.Y0 = -1*sign()*rgen(keygen,1,min.point,max.point);`
```

```
`cc.Z0 = -1*sign()*rgen(keygen,1,min.point,max.point);`
```

```
`cc.A = sign()*rgen(keygen,1,min.plate,max.plate);`
```

```
`cc.B = sign()*rgen(keygen,1,min.plate,max.plate);`
```

```
`cc.C = sign()*rgen(keygen,1,min.plate,max.plate);`
```

```
`cc.D = sign()*rgen(keygen,1,min.plate,max.plate);`
```

```
`cc.l = sign()*zero(bZero)*rgen(keygen,1,min.vector,max.vector);`
```

```
`cc.m = sign()*zero(bZero)*rgen(keygen,1,min.vector,max.vector);`
```

```
`cc.n = sign()*zero(bZero)*rgen(keygen,1,min.vector,max.vector);`
```

Далее следует “Системная” часть - шаблонная выдача задания и варианта. Этот код идентичен практически во всех генераторах

```
`if ( !qvar->MZad || ( qvar->MZad && nvar == 1 ) )`
```

```
`{`
```

```
`printf( buf, "String(\"# Тема- %s \")", selecttask->name );`
```

```
`plist->Add( strdup(buf) );`
```

```
`}`
```

```
`else`
```

```
`{`
```

```
`sprintf( buf, "String(#)" );`
```

```
`plist->Add( strdup(buf) );`
```

```
`}`
```

```
`sprintf( buf, "String(Вариант %i, задание %i.)", nvar, nzad );`
```

```
`plist->Add( strdup(buf) );`
```

Для хранения строк используется класс tlist, включенный в ментор. Очередная строчка, после генерации, копируется в контейнер.

```
`sprintf( buf, "String(Формулировка задания)" );`
```

```
`plist->Add( strdup(buf) );`
```

```
`sprintf( buf, "String(Объекты заданы уравнениями:) " );`
```

```
`plist->Add( strdup(buf) );`
```

```
`sprintf( buf, "a" );`
```

```
`plist->Add( strdup(buf) );`
```

```
`sprintf( buf, "((x-(%d))/%d)=((y-(%d))/%d)=((z-(%d))/%d)", cc.X0,cc.l,cc.Y0,cc.m,cc.Z0,cc.n);`
```

```
`plist->Add( strdup(buf) );`
```

В конце следует генерация ответа для преподавателя: каждая строчка, которая не должна уйти студенту, помечается символом @, и при конвертации в изображение будет фигурировать на другой странице.

```
`sprintf( buf, "String(@Часть преподавателя )" );`
```

```
`plist->Add( strdup(buf) );`
```

```
`sprintf( buf, "String(\"Тема - %s \")", selecttask->name );`
```

```
`plist->Add( strdup(buf) );`
```

В менторе “в идеале” обязан присутствовать ключ, согласно которому каждому конкретному варианту должно соответствовать конкретное значение коэффициентов. В процессе “реверса” свидетельств о реальном использовании этого ключа не найдено, однако в целях “Стандартизации” всё же желательно указывать его.

```
`sprintf( buf, "String(ВАРИАНТ %i, Решение задачи %i, ключ %i)", nvar, nzad, keygen );`
```

```
`plist->Add( strdup(buf) );`
```

```
`sprintf( buf, "sin(!(alpha))" );`
```

```
`plist->Add( strdup(buf) );`
```



```
`sprintf( buf, "(%d*sqrt(%d))/%d",abs(cc.A*cc.l+cc.B*cc.m+cc.C*cc.n),  
(cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C),(cc.l*cc.l  
+cc.m*cc.m+cc.n*cc.n)*(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C));`
```

```
`plist->Add( strdup(buf) );`
```

```
`Log->Add(buf);`
```

```
`keygen = 0;`
```

В конце необходимо “Убрать за собой”

```
`delete buf;`
```

```
`delete buf1;`
```

```
`return 0;`
```

- }

Вторая часть генератора - генератор теста для сайта. Основное отличие в том, что вместо части преподавателя идёт генерация верного и неверных ответов и последующее их перемешивание. Часть с ответами можно считать шаблонной.

- `quest33::Print(TList* plist, class test &t)`

- {

- `<..>`

- `Log->Add("Generating..");`

- `pAnswer pAns[4];`
- `//right variant`
- `pAns[0].legit = true;`
- `printf(pAns[0].str, "(%d*sqrt(%d))/%d",abs(cc.A*cc.l+cc.B*cc.m+cc.C*cc.n),
(cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C),
(cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C));`
- `//wrong variant 1`
- `pAns[1].legit = false;`
- `printf(pAns[1].str, "(%d*sqrt(%d))/%d", -
abs(cc.A*cc.l+cc.B*cc.m+cc.C*cc.n), (cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*
(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C), (cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*
(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C));`
- `//wrong variant 2`
- `pAns[2].legit = false;`
- `printf(pAns[2].str, "
(%d*sqrt(%d))/%d",abs(cc.A*cc.l+cc.B*cc.m+cc.C*cc.n)+random(5)+1,
(cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C),
(cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C));`
- `//wrong variant 3`
- `pAns[3].legit = false;`
- `printf(pAns[3].str, "(%d*%d)/sqrt(%d)",abs(cc.A*cc.l+cc.B*cc.m+cc.C*cc.n),
(cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C),
(cc.l*cc.l+cc.m*cc.m+cc.n*cc.n)*(cc.A*cc.A+cc.B*cc.B+cc.C*cc.C));`

Хранение ответов возлагается на структуру pAnswer, а их перемешивание - на доп. функцию shuffleAnswers.

- `shuffleAnswers(pAns);`

Далее мы ищем новую позицию правильного ответа

- `for (int i = 0; i < 4 && Right_Numb == -1; i++)`
- `{`

```
`if (pAns[i].legit)`
```

```
`{`
```

```
`Right_Numb = i+1;`
```

```
`}`
```

- `}`

И попорядку выводим наши варианты

- `plist->Add(strdup("String(Вариант a:)"));`
- `plist->Add(strdup(pAns[0].str));`
- `sprintf(buf, "String()");`
- `plist->Add(strdup(buf));`
- `plist->Add(strdup("String(Вариантò b:)"));`
- `plist->Add(strdup(pAns[1].str));`
- `sprintf(buf, "String()");`

- `plist->Add(strdup(buf));`
- `plist->Add(strdup("String(Вариант c:)"));`
- `plist->Add(strdup(pAns[2].str));`
- `sprintf(buf, "String()");`
- `plist->Add(strdup(buf));`
- `plist->Add(strdup("String(Вариант d:)"));`
- `plist->Add(strdup(pAns[3].str));`
- `sprintf(buf, "String()");`
- `plist->Add(strdup(buf));`

И системная часть (куда мы кстати пишем номер верного ответа).

```
`t.pr_tst = 1;`
```

- `t.ch_ask = 4;`
- `t.right_ask = Right_Numb;`
- `t.msg = "Тест успешно сгенерирован";`
- `keygen = 0;` Убираем мусор.
- `delete buf;`
- `delete buf1;`
- `return 0;`
- `}`
- `//-----`

Шаг третий: размещение внутри парсера

global.h:

Прописать уникальный дефайн для генератора.

utilm.cpp:

Добавить хедер генератора, в функцию qcreate добавить нулевой указатель на него, а так же прописать внутри switch запрос на иницилизацию через дефайн (Целое число).

tkurs.cpp:

Добавить хедер и иницилизацию генератора через дефайн внутри конструктора tkurs.

Шаг четвёртый: размещение внутри MTR файла

- head 3
- Заголовок первой колонки
- head 4
- Первая строка
- head 4
- Вторая строка
- head 4
- Третья строка
- head 2
- %Название нашего генератора%
- %колонка% %строка% %дефайн%
- %параметры1%
- %параметры2%