МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ОТЧЁТ ПО ВЫПОЛНЕНИЮ ЗАДАНИЯ

По дисциплине «Современные языки программирования»

Выполнил:

студент группы М43-401Бк-19

Козлов Д. К.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

Дубовский А. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва

2022

**ОГЛАВЛЕНИЕ**

[ЗАДАНИЕ 3](#__RefHeading___1)

[ТЕОРИЯ 4](#__RefHeading___2)

[ДИАГРАММА КЛАССОВ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ. 5](#__RefHeading___3)

[РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ. 5](#__RefHeading___4)

[КОД ПРОГРАММЫ 6](#__RefHeading___5)

# ЗАДАНИЕ

**«Генератор гармонического сигнала»**

При помощи механизма шаблонов языка C++, создать шаблонный класс-хранилище реализованного по типу vector. Класс (например, называемый, TSignal) должен быть статичного размера (кладется в конструктор). Добавление новой точки в сигнал должно быть реализовано не через увеличение размера хранилища, а сдвигом предыдущих сохраненных значений.

Создать шаблонный класс HarmonicGenerator, который принимает на вход значения амплитуды, частоты, фазы, частоты дискретизации. Данный класс должен обладать методом generatePoint, который генерирует значение точки по формуле гармонического сигнала и возвращает значение. Также он хранит внутри себя индекс последней сгенерированной точки для использования в функции времени при генерации точки.

Требования на оценку «ОТЛИЧНО»:

Создать приложение при помощи фреймворка Qt, которое отображает график гармоники и позволяет настраивать ее, используя разработанные классы.

# **ТЕОРИЯ**

**Шабло́ны** (template)— средство языка [C++](https://ru.wikipedia.org/wiki/C%2B%2B), предназначенное для кодирования [обобщённых алгоритмов](https://ru.wikipedia.org/wiki/Обобщённое_программирование), без привязки к некоторым параметрам (например, [типам данных](https://ru.wikipedia.org/wiki/Тип_данных), размерам буферов, значениям по умолчанию).

В [C++](https://ru.wikipedia.org/wiki/C%2B%2B) возможно создание шаблонов [функций](https://ru.wikipedia.org/wiki/Функция_(программирование)) и [классов](https://ru.wikipedia.org/wiki/Класс_(программирование)).

Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой тип или значение одного из допустимых типов (целое число, enum, указатель на любой объект с глобально доступным именем, ссылка).

**Model-View-Controller** (**MVC**, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

* ***Модель*** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
* ***Представление*** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* ***Контроллер*** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

**Наблюдатель** (*Observer*) — [поведенческий](https://ru.wikipedia.org/wiki/Поведенческие_шаблоны_проектирования) [шаблон проектирования](https://ru.wikipedia.org/wiki/Шаблон_проектирования). Также известен как «подчинённые». Реализует у класса механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними.

Классы, на события которых другие классы подписываются, называются субъектами , а подписывающиеся классы называются наблюдателями.

**Cmake** (**C**ross-platform [**M**ake](https://ru.wikipedia.org/wiki/Make)) — это [кроссплатформенная](https://ru.wikipedia.org/wiki/Кроссплатформенность) [утилита](https://ru.wikipedia.org/wiki/Утилита), обладающая возможностями [автоматизации сборки](https://ru.wikipedia.org/wiki/Автоматизация_сборки) программного обеспечения из [исходного кода](https://ru.wikipedia.org/wiki/Исходный_код). Сам CMake не занимается непосредственно сборкой, а лишь генерирует файлы сборки из предварительно написанного [скрипт-файла](https://ru.wikipedia.org/wiki/Скрипт) «CMakeLists.txt» и предоставляет простой единый интерфейс управления. Помимо этого, CMake способен автоматизировать процесс [установки](https://ru.wikipedia.org/wiki/Установка_программного_обеспечения) и [пакетирования](https://ru.wikipedia.org/wiki/Система_управления_пакетами).

**qmake** — устаревшая утилита из состава [Qt](https://ru.wikipedia.org/wiki/Qt), которая помогает облегчить процесс сборки приложения на разных платформах. qmake автоматически генерирует [make-файлы](https://ru.wikipedia.org/wiki/Makefile), основываясь на информации в файлах проекта (\*.pro).

# **ДИАГРАММА КЛАССОВ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ.**

# **РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ.**

##### 

# **КОД ПРОГРАММЫ**

Файл main.cpp:

#include "main.h"  
int main(int argc, char \*argv[]){ QApplication a(*argc*, *argv*); MainWindow w; w.show(); *return* a.exec();}

Файл mainwindow.cpp:

#include "mainwindow.h"  
TSignal<double> signal1(100);HarmonicGenerator<float, int, float, int> generator(10, 1000, 0, 40000);  
MainWindow::MainWindow(QWidget \*parent) : QMainWindow(*parent*) , ui(*new* Ui::MainWindow){ ui->setupUi(*this*); MainWindow::makePlot();}  
MainWindow::~*MainWindow*(){ *delete* ui;}  
  
void MainWindow::on\_pushButton\_clicked(){ QVector<double> y(signal1.getSignalSize()); *for* (int i=0; i<signal1.getSignalSize(); ++i) { y[i] = i; }  
 float val = generator.generatePoint(); char mass[10] = {0}; itoa(val, *mass*, 10); signal1.addPoint(val);  
 ui->customPlot->graph(0)->setData(y, signal1.getVector()); ui->customPlot->replot();  
 QString freqSTR = ui->freq->text(); *if* (freqSTR.toStdString() != "") generator.setFreq(freqSTR.toInt());  
 QString samplingFreqSTR = ui->samplingFreq->text(); *if* (samplingFreqSTR.toStdString() != "") generator.setSamplingFreq(samplingFreqSTR.toInt());  
 QString phaseSTR = ui->phase->text(); *if* (phaseSTR.toStdString() != "") generator.setPhase(phaseSTR.toFloat());}  
  
void MainWindow::makePlot(){ QVector<double> y(signal1.getSignalSize()); y.fill(0, signal1.getSignalSize());  
 ui->customPlot->addGraph(); ui->customPlot->graph(0)->setData(y, signal1.getVector());  
 ui->customPlot->xAxis->setLabel("time"); ui->customPlot->yAxis->setLabel("amplitude");  
 ui->customPlot->xAxis->setRange(0, signal1.getSignalSize()); ui->customPlot->yAxis->setRange(generator.getAmp() \* 1.2, -1 \* generator.getAmp() \* 1.2 ); ui->customPlot->replot();}  
  
void MainWindow::on\_maxGraphPoint\_editingFinished(){ QString maxGraphPointSTR = ui->maxGraphPoint->text(); *if* (maxGraphPointSTR.toStdString() != "") { signal1.setMaxSignalSize(maxGraphPointSTR.toInt()); ui->customPlot->xAxis->setRange(0, signal1.getSignalSize()); }}  
  
void MainWindow::on\_maxAmp\_editingFinished(){ QString maxAmpSTR = ui->maxAmp->text(); *if* (maxAmpSTR.toStdString() != "") { generator.setAmp(maxAmpSTR.toFloat()); ui->customPlot->yAxis->setRange(generator.getAmp() \* 1.2, -1 \* generator.getAmp() \* 1.2 ); }}

Файл harmonicGenerator.h:

#ifndef HARMONICGENERATOR\_H#define HARMONICGENERATOR\_H  
#include "tsignal.h"#include <cmath>  
*extern* TSignal<double> signal1;  
*template* <*class* T1, *class* T2, *class* T3, *class* T4>*class* HarmonicGenerator{*private*: T1 amplitude; *//Амплитуда* T2 frequency; *//Частота* *сигнала* T3 phase; *//Начальная* *фаза* *сигнала* T4 samplingFrequency; *//Частота* *дескретизации* int pointCounter = 0; *//индекс* *последней* *сгенерированной* *точкиpublic*:  
  
 HarmonicGenerator( T1 amplitude, T2 frequency, T3 phase, T4 frequencyOfDescretisation); T1 generatePoint (); T1 getAmp(); void setAmp(T1); void setFreq(T2); void setPhase (T3); void setSamplingFreq (T4);  
};  
  
*template* <*class* T1, *class* T2, *class* T3, *class* T4>HarmonicGenerator<T1, T2, T3, T4>::HarmonicGenerator(T1 amplitude, T2 frequency, T3 phase, T4 samplingFrequency){ *this*->amplitude = amplitude; *this*->frequency = frequency; *this*->phase = phase; *this*->samplingFrequency = samplingFrequency;}  
*template* <*class* T1, *class* T2, *class* T3, *class* T4>T1 HarmonicGenerator<T1, T2, T3, T4>::generatePoint(){ float val = *this*->amplitude \* sin(2 \* M\_PI \* *this*->pointCounter \* *this*->frequency / *this*->samplingFrequency + *this*->phase); *//=* *A* *sin* *(ωt* *+* *φ),* *ω=2* *πf* *this*->pointCounter++; *if* (*this*->pointCounter > (int)signal1.getSignalSize() ) { *this*->pointCounter = 0; signal1.resetSignal(); val = 0; }  
 *return* val;}  
*template* <*class* T1, *class* T2, *class* T3, *class* T4>T1 HarmonicGenerator<T1, T2, T3, T4>::getAmp(){ *return* *this*->amplitude;}  
*template* <*class* T1, *class* T2, *class* T3, *class* T4>void HarmonicGenerator<T1, T2, T3, T4>::setAmp(T1 amp){ *this*->amplitude = amp;  
}  
*template* <*class* T1, *class* T2, *class* T3, *class* T4>void HarmonicGenerator<T1, T2, T3, T4>::setFreq(T2 freq){ *this*->frequency = freq;}  
*template* <*class* T1, *class* T2, *class* T3, *class* T4>void HarmonicGenerator<T1, T2, T3, T4>::setPhase(T3 ph){ *this*->phase = ph;}  
*template* <*class* T1, *class* T2, *class* T3, *class* T4>void HarmonicGenerator<T1, T2, T3, T4>::setSamplingFreq(T4 sFreq){ *this*->samplingFrequency = sFreq;}  
#endif

Файл tsignal.h:

#ifndef TSIGNAL\_H#define TSIGNAL\_H  
#include <vector>#include "qvector.h"#include <iostream>  
*using* *namespace* std;  
*template* <*class* T>*class* TSignal{*private*: QVector<T> signal; int maxSignalSise;*public*: TSignal(int signalSize); void addPoint(T point); QVector<T> getVector(); int getSignalSize(); void setMaxSignalSize(int); void resetSignal();};  
*template* <*class* T>TSignal<T>::TSignal(int signalSize){ signal.fill(0, signalSize); *this*->maxSignalSise = signalSize;}  
*template* <*class* T>void TSignal<T>::addPoint(T point){ signal.removeLast(); signal.insert(0, point);}  
*template* <*class* T>QVector<T> TSignal<T>::getVector(){ *return* signal;}  
*template* <*class* T>int TSignal<T>::getSignalSize(){ *return* *this*->maxSignalSise;}  
*template* <*class* T>void TSignal<T>::setMaxSignalSize(int size){ *this*->maxSignalSise = size; signal.fill(0, *this*->maxSignalSise);}  
*template* <*class* T>void TSignal<T>::resetSignal(){ signal.fill(0, *this*->maxSignalSise);}  
#endif