Лабораторная работа №

Тема: «Контейнеры в Qt5»

Контейнеры — это **классы** общего назначения, которые хранят в себе значения заданного типа. В языке C++ имеется свой набор контейнеров, который называется **STL**.

Стоит отметить, что идеологии контейнеров Qt и STL, при всей своей схожести, отличаются в следующем:

- Фреймворк Qt5 предоставляет максимально богатый интерфейс, а копирование контейнера происходит за константное время, но цена этому — дополнительные проверки при доступе к разделяемым данным (*implicit data sharing*).
- В свою очередь, STL предоставляет ограниченный интерфейс и следует идеологии уменьшения необоснованного расхода процессорного времени, памяти, ненужной нагрузки и пр.

Программируя на Qt, можно использовать как Qt-контейнеры, так и STL. Если вы не знакомы с STL или предпочитаете работать «только с Qt», то можете использовать классы Qt вместо классов STL.

Рассмотрим следующие контейнеры:

- QVector;
- QList;
- QStringList;
- QSet;
- QMap.

Есть 2 типа контейнеров:

- **Последовательные** — элементы хранятся друг за другом (последовательно). Примерами последовательных контейнеров являются QList, QVector, QLinkedList.
- **Ассоциативные** — элементы хранятся в виде пары ключ-значение. Примерами ассоциативных контейнеров являются QMap и QHash.

Контейнер QVector

Контейнерные классы – классы с неявным совместным использованием данных, они оптимизированы для быстрой работы, низкого потребления памяти и минимального увеличения кода (inline), результат в меньшем исполняемом файле. Qt предоставляет конструкцию foreach, которая позволяет очень легко перебрать все элементы, хранящиеся в контейнере.

Вектор – структура данных, очень похожая на обычный массив. Однако использование класса вектора предоставляет некоторые преимущества по сравнению с простым массивом. Например, можно узнать количество элементов внутри вектора (его размер), или динамически расширять его. Еще этот контейнер экономнее, чем другие виды контейнеров.

Добавлять элементы в вектор можно следующими способами:

1 Если нам заранее известно необходимое количество элементов в векторе, мы можем задать начальный размер при его определении и использовать оператор [ ] для заполнения вектора элементами:

Пример 2.1

```
QVector <float> vect1(3);
vect1[0] = 1.0;
vect1[1] = 0.5;
vect1[2] = -0.4;
qDebug() << vect1;
```

Пример 2.2

```
// QVector(3,3,3,3,3)
QVector <int> vect2(5,3);
qDebug()          <<          vect2;
```

2 Для добавления элементов в конец последовательного контейнера необходимо объявить пустой вектор и использовать методы push_back() или append():

Пример 2.3

```
QVector<int> vect3;
vect3.push_back(10);
vect3.push_back(20);
vect3.push_back(30);
qDebug() << vect3;
```

Пример 2.4

```
QVector <float> vect4;
vect4.append(34.0);
vect4.append(0.5899);
vect4.append(-0.678);
qDebug() << vect4;
```

Пример 2.5

```
QVector<QString>vec5;
vect5.append("one");
vect5.append("two");
vect5.append("three");
qDebug()    <<    vect5;
```

Таблица 1 – Часто используемые методы контейнера QVector <T>

Метод	Описание
push_back()	Добавление элемента в конец последовательного контейнера.
clear()	Удаление всех элементов из вектора и освобождение памяти, используемой вектором
remove()	Удаляет элемент в позиции с индексом *i*
replace()	Заменяет элемент в позиции с индексом *i* на *value*
insert()	Вставляет значение *value* в позицию с индексом *i* в векторе
size()	Возвращает количество элементов в векторе

Следующий пример показывает работу с вектором, элементами которого являются целые числа:

```
#include <QVector>
#include <QTextStream>

int main(void) {
    QTextStream out(stdout);
      // Создаем вектор, содержащий целочисленные значения
   QVector<int> vals = {1, 2, 3, 4, 5};
```

```
    // С помощью метода size() возвращаем размер вектора (количество
элементов, содержащихся в нем)
    out << "The size of the vector is: " << vals.size() << endl;

    out << "The first item is: " << vals.first() << endl;
// получаем первый элемент вектора
    out << "The last item is: " << vals.last() << endl;
// получаем последний элемент вектора

    vals.append(6); // вставляем новый элемент в конец вектора
    vals.prepend(0); // вставляем новый элемент в начало вектора

    out << "Elements: ";

    // Перебираем элементы вектора и выводим их на экран
    for (int val : vals) {
        out << val << " ";
    }

    out << endl;
    return 0;
}
```

Задание 1Создайте консольное приложение vector. Подключите все необходимые классы и заполните векторы пятью способами. Сохраните изменения и запустите приложение на выполнение.

Задание 2 Создайте консольное приложение vect_sum, которое позволит находить сумму элементов вектора, а также продемонстрирует применение методов replace() и size().

Пример,

```
QVector<int> vec; // создаем вектор из 11 элементов от 1 до 10
for (int i=0; i<=10; i++ ) {
          vec.push_back(i); }
qDebug() << vec;
for (int i=0; i<=10; i++ ) {
// заменяем элемент вектора «2» на «65»
 vec.replace(2,65); }
          qDebug() << vec;
int sum=0; // находим сумму элементов
for (int i=0; i<vec.size(); i++ ) {
 // от 0 до количества элементов в векторе
           sum += vec[i]; }
```

```
qDebug() <<"Sum = " << sum;
```

Контейнеры QList и QLinkedList

Список — это структура данных, представляющая собой упорядоченный набор связанных друг с другом элементов. Преимущество списков перед векторами и очередями состоит в том, что вставка и удаление элементов в любой позиции происходит эффективнее, так как для выполнения этих операций изменяется только минимальное количество указателей, исключение составляет только вставка элемента в центр списка. Но есть и недостаток — списки плохо приспособлены для поиска определенного элемента списка по индексу, и для этой цели лучше всего использовать вектор.

Списки реализует шаблонный класс QList. В общем виде данный класс представляет собой массив указателей на элементы.

Специфические операции для работы со списками:

move() — Перемещает элемент с одной позиции на другую.

removeFirst() — Выполняет удаление первого элемента списка.

removeLast() — Выполняет удаление последнего элемента списка.

swap() — Меняет местами два элемента списка на указанных позициях.

takeAt() — Возвращает элемент на указанной позиции и удаляет его из списка.

takeFirst() — Возвращает первый элемент и удаляет его из списка.

takeLast() — Возвращает последний элемент и удаляет его из списка.

toSet() — Возвращает контейнер QSet с данными содержащимися в списке.

toStdList() — Возвращает стандартный список STL std::List с элементами из списка.

toVector() — Возвращает вектор QVector с данными содержащимися в списке.

Если вы не собираетесь менять значения элементов, то, из соображения эффективности, не рекомендуется использовать оператор []. Вместо него используйте метод at(), так как он возвращает константную ссылку на элемент.

Одна из самых распространенных операций — обход списка для последовательного получения значений каждого элемента списка. Например:

```
QList<int> list;
list << 10 << 20 << 30;

QValueList<int>::iterator it = list.begin(); // создаем итератор и переводим его в начало списка
while (it != list.end()) {
    qDebug() << "Element:" << *it;
```

```
    ++it;
}
```
Пример,
```
#include <QTextStream>
#include <QList>
#include <algorithm>

int main(void) {

   QTextStream out(stdout);

   // Создаем контейнер QList, в котором будем хранить имена //писателей
   QList<QString> authors = {"Balzac", "Tolstoy",
      "Gulbranssen", "London"};

   // Перебираем каждый элемент массива и выводим на экран
   for (int i=0; i < authors.size(); ++i) {

      out << authors.at(i) << endl; // метод at() возвращает //элемент с
указанным индексом
   }

   // С помощью оператора << вставляем в список 2 новых //элемента
   authors << "Galsworthy" << "Sienkiewicz";

   out << "**********************" << endl;

   // С помощью метода sort() сортируем список в порядке //возрастания
   std::sort(authors.begin(), authors.end());

   // Выводим на экран отсортированный список
   out << "Sorted:" << endl;
   for (QString author : authors) {

      out << author << endl;
   }
}
```
Для перемещения по элементам контейнера предназначен **итератор**. Итераторы позволяют абстрагироваться от структуры данных контейнеров, т.е. если в какой-либо момент вы решите, что применение другого типа контейнера было бы гораздо эффективнее, то все, что вам нужно будет сделать – это просто заменить тип контейнера. На остальном коде, использующем итераторы, это никак не отразится. Qt предоставляет два

стиля итераторов: итераторы в стиле Java; итераторы в стиле STL. Пример вывода элементов контейнера в прямом порядке:

```
QVector<QString> vec;
setlocale(LC_ALL,"");
vec <<"один" << "два" << "три";
QVector<QString>::iterator it = vec.begin();
for (; it != vec.end(); ++it) {
qDebug() << "Элемент:"<< *it ;}
```

Вызов метода begin() из объекта контейнера возвращает итератор, указывающий на первый его элемент, а вызов метода end() возвращает итератор указывающий на воображаемый несуществующий элемент, следующий за последним.

Пример вывода элементов контейнера в обратном порядке:

```
QVector<QString>::iterator id = vec.end();
for (;id != vec.begin();) {
        --id;
        qDebug() << "Элемент:" << *id;}
```

Обратите внимание: прохождение элементов в обратном порядке при помощи оператора - не симметрично с прохождением при помощи оператора ++.

QList<T> – наиболее часто используемый контейнер.

Список – это структура данных, представляющая собой упорядоченный набор связанных друг с другом элементов. Преимущество списков перед векторами и очередями состоит в том, что вставка и удаление элементов в любой позиции происходит эффективнее, так как для выполнения этих операций изменяется только минимальное количество указателей; исключение составляет только вставка элемента в центр списка. Но есть и недостаток: списки плохо приспособлены для поиска определенного элемента списка по индексу, и для этой цели лучше всего использовать вектор. Списки реализует шаблонный класс QList. В общем виде данный класс представляет собой массив указателей на элементы.

Задание 1. Создайте консольное приложение iterators. Добавьте программный код для вывода элементов контейнера QVector в прямом и обратном порядке. Просмотрите результат. Затем измените экземпляр класса QVector на QList и сравните результаты. Добавлять элементы в список можно следующими способами:

1 Если нам заранее известно необходимое количество элементов в списке и их значения:

```
QList <int> list;
list << 10 << 20 << 30;
```

```
qDebug() << list;
```

2 Для добавления элементов в конец последовательного контейнера необходимо объявить пустой список и использовать методы push_back() или append():

```
QList <QString> list2;
list2.append( "one");
list2.append( "two" );
list2.push_back("three");
qDebug() << list2;
```

Задание 2. Создайте консольное приложение list. Подключите все необходимые классы и заполните списки двумя способами (при выводе списка используйте итераторы).

Задание 3. Создайте консольное приложение list2, в котором рассмотрим функции move() и takeFirst(), а также научимся менять значения элементов контейнера с использованием итераторов. Для этого добавьте следующие строки:

```
QList <int > list; list << 3 << 4 << 5 << 6 << 7;
// вносим элементы в список
qDebug() << list;
int firstNumber = list.takeFirst();
//помещаем возвращенное значение в
//переменную и удаляем из списка первый элемент
qDebug() <<"firstNumber: " << firstNumber; qDebug() << list; list.move(0,1); //
перемещаем элемент с нулевой позиции на первую
QList<int>::iterator it = list.begin();
while (it != list.end()) {
        qDebug() << *it; ++it; }
qDebug() << endl;
QList<int>::iterator ip = list.begin()+1; // ставим итератор на первую позицию
while (ip != list.end()-1) { // пока не достигнем последнего элемента
*ip *=(*ip); // возводим в квадрат
++ip; }
ip = list.begin(); // выводим список
while (ip != list.end()) {
qDebug() << *ip; ++ip; }
```

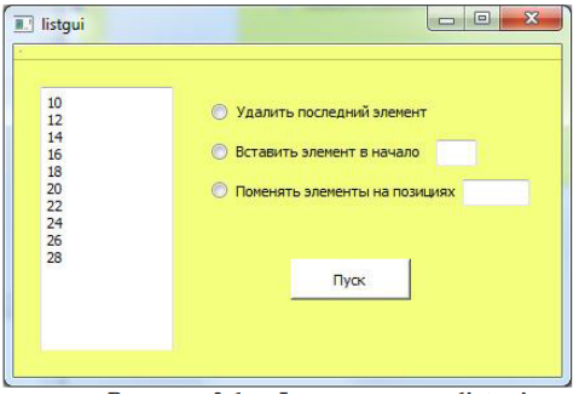
Сохраните изменения и запустите приложение на выполнение.

Задание 4. Создайте приложение, которое позволит удалять и добавлять элементы, а также менять их местами.

1 Создайте приложение Qt Widgets с именем listgui. Имя класса listgui.

2 Разработайте интерфейс пользователя и расположите на форме следующие элементы: textEdit, pushbutton, radioButton (3), lineEdit (2).

3 Переименуйте виджеты radioButton. Дайте названия виджетам: удалить последний элемент; вставить элемент в начало; поменять элементы на позициях.
4 Установите запрет на изменение размеров окна. Пример формы на рисунке 3.1 *(цвет фона, шрифт... установите по своему желанию).*



. Добавим программный код для работы нашего приложения:
1) в заголовочном файле объявите экземпляр класса QList: QList<int> list.
2) добавьте код для заполнения списка и вывода его в textEdit при запуске приложения:

```
for (int i = 0; i < 10; i++) {
        list.append(10+i*2);
         ui→textEdit→append(QString::number(list.at(i)) + " "); }
```

Поскольку мы не собираемся изменять значения элементов, то из соображений эффективности не рекомендуется использовать оператор индексации []. Вместо этого лучше будет воспользоваться методом at(), т.к. этот метод возвращает константную ссылку на элемент.
3) сохраните изменения и запустите для просмотра.
4) перейдите к слоту, который будет вызываться в ответ на событие clicked() и добавьте в него код для работы нашего приложения:

```
if (ui→radioButton→isChecked()) { list.removeLast();
        //если выбран первый radioButton, удаляем последний элемент
        } else if (ui→radioButton_2→isChecked()) {
                int n=ui→lineEdit→text().toInt();
                list.push_front(n);
                 //добавляем введенный в lineEdit элемент в начало
                } else if (ui→radioButton_3→isChecked()) {
                        int a,b;
                        QStringList st = ui→lineEdit_2→text().split(",");
                        a = st[0].toInt(); b = st[1].toInt();
                        list.swap(a,b); }
```

Функция split() разбивает строку на подстроки, как только доходит до запятой, и возвращает список этих строк. ui→textEdit→clear(); // перезаписываем список

```
QList<int>::iterator it = list.begin();
while ( it != list.end()) {
```

```
        ui→textEdit→append(QString::number(*it) + " ");
         it++; }
```
Сохраните изменения в приложении и просмотрите результат.

Задание 5 Создайте консольное приложение linkedlist. В приложении linkedlist мы рассмотрим некоторые стандартные алгоритмы STL, а именно:

а) min_element и max_element − нахождение минимального и максимального элемента ;

б) find (first, last, value) − возвращает итератор, указывающий на первый элемент, равный значению value;

в) remove(first,last,value) − удаление из диапазона всех значений, равных value.

В действительности remove ничего не удаляет, так как ему не передается контейнер. Элементы могут удаляться лишь функциями контейнера, отсюда следует и главное правило: чтобы удалить элементы из контейнера, вызовите erase после remove. Для этого добавьте следующие строки:

```
using namespace std;
QLinkedList<int> list; //заполнение списка
for (int i = 1; i < 10; i++) {
   list << abs(5-i); }
QLinkedList<int>::iterator it;
it = list.begin();
QString str = " ";
while (it != list.end()) {
        str += QString::number(*it) + " ";
        // преобразовываем элементы списка
        // в строку и + к  созданной строке
       it++; }
qDebug() << str; // вывод строки, содержащей элементы списка
//нахождение минимального и максимального элемента
int min = *min_element(list.begin(), list.end());
int max = *max_element(list.begin(), list.end());
qDebug() << "Минимальный элемент:" << min;
qDebug() << "Максимальный элемент:" << max; list.erase(remove(list.begin(),
list.end(),0),list.end()); // удаление 0 str = " ";
it = list.begin();
while (it != list.end()) {
        str += QString::number(*it) + " ";
        it++; }
qDebug() << str;
QLinkedList<int>::iterator iter;
iter = find(list.begin(), list.end(), 4); // находим «4»
*iter *= 10; // умножаем на 10
iter = find(list.begin(), list.end(), 2);
*iter *= 10;
it = list.begin();
```

```
str = " ";
while (it != list.end()) {
        str += QString::number(*it) + " "; it++; }
qDebug() << str;
```

Сохраните изменения и запустите приложение на выполнение.

Задание 6 Создайте консольный проект linkedlist2, объявите экземпляр класса QLinkedList <int> list, заполните его случайными значениями и продемонстрируйте в нем работу других алгоритмов STL:

а) count(first, last, value) – возвращает, сколько раз элемент со значением value входит в последовательность, заданную итераторами;

б) reverse (first, last) – переставляет элементы в обратном порядке;

в) iter_swap (first, last) – меняет местами значения элементов, на которые указывают итераторы.

Задание 7 Создайте графическое приложение, которое позволит пользователю выводить сумму элементов и произведение элементов на четных позициях.

1 Создайте приложение Qt Widgets с именем linkedlistgui.

2 Разработайте интерфейс пользователя, со следующими элементами: textEdit, pushbutton, checkBox (2).

3 Переименовать виджеты checkBox, Дайте названия виджетам: вывод суммы элементов; вывод произведения элементов на четных позициях.

4 Установите запрет на изменение размеров окна. Пример формы на рисунке 3.2 *(цвет фона, шрифт... установите по своему желанию).*

Сохраните изменения и запустите для просмотра.

5 Добавим программный код для работы нашего приложения:

1) в заголовочном файле объявите экземпляр класса QLinkedList:

```
QLinkedList <int> Lili;
```

2) добавьте код для заполнения вектора и вывода его в textEdit при запуске приложения:

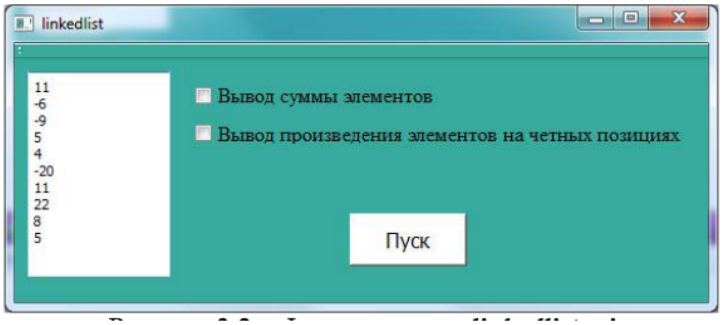


Рисунок 3.2 – Форма проекта linkedlistgui

```
srand(time(0));
for (int i = 0; i < 10; i++) {
        Lili.append(rand() % 50 - 20);
        ui→textEdit→append(QString::number(Lili.last()) + " "); }
```

Функция last() возвращает ссылку на последний элемент в списке. Эта функция предполагает, что список не является пустым. Сохраните изменения и запустите для просмотра;

3) перейдите к слоту, который будет вызываться в ответ на событие clicked(), и добавьте в него код для работы нашего приложения:

```
ui→textEdit→clear();
int sum = 0;
long mul = 1;
if (ui→checkBox→isChecked()) {
        QLinkedList<int>::iterator it = Lili.begin();
        while ( it != Lili.end()) { // находим сумму элементов
                sum += *it;
                it++; }
        ui→textEdit→append("Сумма: " + QString::number(sum)); }
        //выводим сумму элементов в начале списка
        if(ui→checkBox_2→isChecked()) {
                QLinkedList<int>::iterator it = Lili.begin();
                int pos = 0;
                while ( it != Lili.end()) {
                        if (pos % 2 == 0) {
                         // произведение элементов
                        mul *= *it; }
                        pos++;
                        it++; }
        ui→textEdit→append("Умножение:"+QString::number(mul)); }
        QLinkedList<int>::iterator it = Lili.begin();
        while ( it != Lili.end()) {
                //перезаписываем список
                ui→textEdit→append(QString::number(*it) + " ");
                it++; } }
```

Сохраните изменения в приложении и просмотрите результат.

Работа с контейнерами в среде Qt Creator: QStack, QQueue

Стек реализует структуру данных, работающую по принципу Last In First Out: «последним пришел, первым ушел». То есть из стека первым удаляется элемент, который был вставлен позже всех остальных. Класс QStack представляет собой реализацию стековой структуры. Этот класс унаследован от QVector. Процесс помещения элементов в стек обычно называется проталкиванием (pushing), а извлечение из него верхнего элемента — выталкиванием (poping). Каждая операция проталкивания увеличивает размер стека на 1, а каждая операция выталкивания − уменьшает на 1. Для этих операций в классе QStack определены функции push() и pop(). Метод top() возвращает ссылку на верхний элемент. Прежде чем брать элемент необходимо убедиться, что стек не пустой (!stack.isEmpty()).

Задание 1. Создайте консольное приложение stack. Приложение должно увеличивать элементы с четными значениями на 3.

```
QStack<int> stack_1, stack_2; // объявление экземпляров класса QStack
qDebug() << "Задайте количество элементов: "; QTextStream in(stdin);
int count = (in.readLine()).toInt(); //количество элементов стека
qDebug() << "Введите элементы:"; // ввод элементов с клавиатуры
for (int i = 0; i < count; i++) {
        int num = (in.readLine()).toInt();
        stack_1.push(num); } // заносим введенные значения в стек
qDebug() << "Стек 1 : " << stack_1;
while (!stack_1.empty()) {
        int num = stack_1.pop(); // достаем элемент
        if (num % 2 == 0) { num += 3; } // проверяем на четность
        stack_2.push(num); } // помещаем во второй стек
qDebug() << "Стек 2 : " << stack_2;
while (!stack_2.empty()) {
        stack_1.push(stack_2.pop()); } // перекладываем в первый стек
qDebug() << "Стек 1 : " << stack_1;
```

Задание 2 Создайте графическое приложение, для использования стека при решении головоломки «Ханойская башня».

1 Создайте Приложение Qt Widgets с именем tower.

2 Разработайте интерфейс пользователя, используйте следующие элементы: textEdit (3), spinbox (2), pushbutton, label (2). Расположение виджетов представлено на рисунке 4.1.

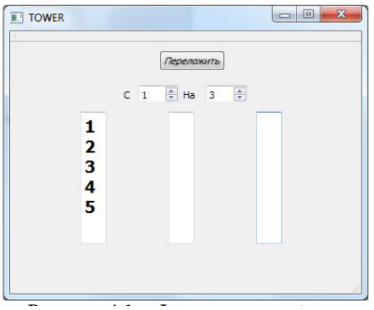


Рисунок 4.1 – Форма проекта tower

3 Для виджетов spinBox и spinBox_2 задайте минимальное значение («1») и максимальное («3»). В виджете spinBox установите текущее значение (value) равным 1, в виджете spinBox_2 равным 3.

4 В label, label_2, spinBox и spinBox_2 и скомпануйте по горизонтали с разделителем.

5 Установите запрет на изменение размеров окна.

6 Виджеты textEdit сделайте доступными только для чтения.

7 Сохраните изменения и запустите для просмотра.

8 Добавим программный код для работы нашего приложения:

1) объявите три объекта класса QStack;

2) добавьте код для заполнения первого стека и вывода его в textEdit при запуске приложения;
3) перейдите к слоту, который будет вызываться в ответ на событие clicked(), и добавьте программный код для перемещения элементов стека:

```
//если значения компонентов spinBox и spinBox_2 совпадают, //то выводим сообщение об ошибке
if (ui→spinBox→value() == ui→spinBox_2→value()) {
        QMessageBox::warning(this,"Предупреждение","Нельзя");}
else { QStack<int> *from_st; //создаем указатели на 2 стека целых чисел
        QStack<int> *to_st;
        switch (ui→spinBox→value()) {
                case 1:{from_st = &stack_1; break;} //в зависимости от выбранных
                case 2:{from_st = &stack_2; break;} //значений спинбоксов,
                case 3:{from_st = &stack_3; break;}} //присваиваем каждому из них
        switch (ui→spinBox_2→value()) { //ссылку на один из трех стеков
                case 1:{to_st = &stack_1; break;}
                case 2:{to_st = &stack_2; break;}
                case 3:{to_st = &stack_3; break;}}
        //проверяем, что стек, из которого мы хотим перенести, не пустой
        if (from_st→isEmpty()) {
          QMessageBox::warning(this,"Предупреждение","Нельзя");}
          else //проверяем, что стек, в который мы хотим записать элемент, пустой
          if ((!(to_st→isEmpty())) ||
          // или его верхнее значение больше, чем то, которое мы хотим поместить
            (from_st→at(from_st→count()-1) >
            to_st→at(to_st→count()-1))) {
            QMessageBox::warning(this,"Предупреждение","Нельзя");}
            else {
             int val;
             val = from_st→pop(); /*значение извлекается из стека, выбранного первым спинбоксом*/
             to_st→push(val); }}
             //и помещается в стек, выбранный вторым спинбоксом
            ui→textEdit→clear();
            ui→textEdit_2→clear();
            ui→textEdit_3→clear();
            for (int i = stack_1.count()-1; i >= 0; i--) {
               ui→textEdit→append(QString::number(stack_1.at(i))); }
            for (int i = stack_2.count()-1; i >= 0; i--) {
               ui→textEdit_2→append(QString::number(stack_2.at(i)));}
            for (int i = stack_3.count()-1; i >= 0; i--) {
```

ui→textEdit_3→append(QString::number(stack_3.at(i)));}}

Сохраните изменения в приложении и просмотрите результат.

Очередь реализует структуру данных, работающую по принципу: «первым пришел, первым ушел». Для очереди определены операции добавления в очередь (enqueue) и извлечения элемента из очереди (dequeue). Реализована очередь в классе QQueue, который унаследован от класса QList.

Задание 3. Создайте консольное приложение queue. Приложение должно увеличивать элементы с четными значениями на 3.

```
QQueue<int> que;
QTextStream in(stdin);
qDebug() << "Количество элементов в очереди: ";
int count = (in.readLine()).toInt();
qDebug() << "Values: ";
for (int i = 0; i < count; i++) {
        int num = (in.readLine()).toInt();
        que.enqueue(num); } // помещаем введенные значения в очередь
qDebug() << "Элементы очереди: " << que;
int len = que.length(); // заносим в переменную значение длины очереди for (int
i = 0; i < len; i++) {
        int num = que.dequeue();// извлекаем элемент из очереди
        if ( num % 2 == 0) {num += 3;}// проверка на четность que.enqueue(num);
        } // записываем обратно в очередь
qDebug() << "Очередь: " << que;
```

Сохраните изменения и запустите проект на выполнение.

Задание 4. Создадим графическое приложение (рисунок 4.2), которое продемонстрирует процесс извлечения и добавления элемента в очередь.

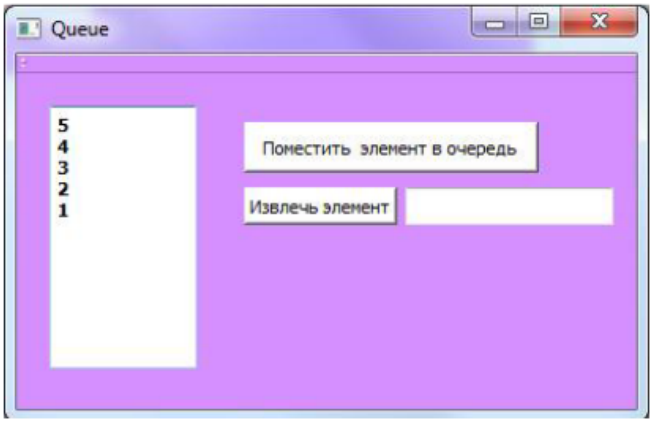


Рисунок 4.2 – Форма проекта queuegui

Установите запрет на изменение размеров окна. Виджеты lineEdit, textEdit сделайте доступными только для чтения. Виджет lineEdit будет выводить текущий извлеченный элемент. В файле queuegui.cpp в конструкторе главного окна добавьте код для заполнения очереди и вывода её в textEdit при запуске приложения:

```
for(int i = 1; i <= 5; i++) {
```

```
 que.enqueue(i);
 ui→textEdit→setText(QString::number(i) +"\n"+ui→textEdit→toPlainText());}
n=5;
```

Функция toPlainText() возвращает текст, который находится в виджете. Перейдите к слоту, который будет вызываться в ответ на событие clicked() и отвечать за добавление элемента в очередь:

```
n++;
que.enqueue(n);
ui→textEdit→clear();
for(int i = 0; i < que.count(); i++) {
ui→textEdit→setText(QString::number(que[i]) +
"\n"+ui→textEdit→toPlainText()); }
```

Перейдите к слоту, который будет вызываться в ответ на событие clicked() и отвечать за извлечение элемента из очереди:

```
ui→lineEdit_2→setText(QString::number(que.dequeue())); ui→textEdit→clear();
for(int i = 0; i < que.count(); i++){
    ui→textEdit→setText(QString::number(que[i]) + "\n"+
    ui→textEdit→toPlainText()); }
```

Сохраните изменения в приложении и просмотрите результат.

Работа с контейнерами в среде Qt Creator: QMap, QSet

QMap<K,T> – ассоциативный массив, отображающий ключи типа K и значения типа T. Достоинство словаря в том, что он позволяет быстро получать значение, ассоциированное с заданным ключом. Необходимо следить за тем, чтобы не было занесено двух разных элементов с одинаковым ключом, ведь тогда не удастся извлечь один из этих элементов. Ключи должны быть уникальными!

Элементы сортируются по ключу, и проход по QMap всегда дает содержимое в отсортированном порядке. Некоторые методы контейнера QMap представлены в таблице 5.1.

Таблица 5.1 − Методы контейнера QMap

Метод	Описание
iowerBound()	Возвращает итератор на первый элемент с заданным ключом. Если такого ключа нет, то возвращается итератор, указывающий на ближайший элемент с большим ключом.
toStdMap()	Возвращает словарь STL с элементами нашего словаря.
upperBound()	Возвращает итератор, указывающий на последний элемент с ключом. Если такого ключа нет, то возвращается итератор, указывающий на ближайший элемент с большим ключом.

Ключ и значение можно получать через методы итератора: key() и value(). Добавлять элементы в словарь можно следующими способами:

Пример 5.1

```
QMap<QString, int> map;
map["ten"] = 10;
map["twenty"] = 20;
```

```
map["thirty"] = 30;
QMap<QString, int>::iterator it = map.begin();
for (;it != map.end(); ++it) {
        qDebug() << "key: " << it.key() << " value: " << it.value();}
```

Пример 5.2

```
QMap<QString, int> map2;
map2.insert("one",1);
map2.insert("two",2);
map2.insert("three",3);
QMap<QString, int>::iterator id = map2.begin();
for (;id != map2.end(); ++id) {
qDebug() << "key: " << id.key() << " value: " << id.value();}
```

Задание 1. Создайте консольное приложение map, подключите необходимые классы и заполните словарь двумя способами.

Задание 2. Создайте консольное приложение «Телефонный справочник», где будем хранить словарь из имен и прикрепленных к ним номеров, а также продемонстрируем варианты поиска записей по ключу или значению, и предусмотрим проверку на наличие записи в словаре с помощью оператора [] и функции contains().

Задание 3. Создайте консольное приложение ***phones***. Для работы приложения добавьте следующие строки: QMap<Qstring,Qstring> phones; // объявляем экземпляр класса Qmap phones[“Anna”] = “79195679845”; // заполнение словаря
phones[“Irina”] = “79128349028”;
phones.insert(“Alex”,”89632675687”);
phones.insert(“Danil”, “89092345681”);
Qmap<Qstring, Qstring>::iterator it = phones.begin();
for (;it != phones.end(); ++it) {
//вывод словаря
qDebug() << “Name: “ << it.key() << “ Phone: “ << it.value(); } qDebug() << endl;
qDebug() << “Phone Alex: “ << phones.value(“Alex”);
//получаем значение
qDebug() << “Phone 89092345681: “ << phones.key(“89092345681”);
if(phones.contains(“Oleg”)) {
// проверяем наличие записи в словаре
qDebug() << “Phone:” << phones[“Oleg”]; }
if(phones.contains(“Irina”)) {
        qDebug() << “Phone:” << phones[“Irina”]; }

Примечание: обратите внимание на использование оператора [], который может использоваться как для вставки, так и для получения значения элемента.

Задание ключа, для которого элемент не существует, приведет к тому, что элемент будет создан. Чтобы избежать этого, нужно проверять существование

элемента, привязанного к ключу. Подобную проверку мы осуществили при помощи метода contains().
Сохраните изменения и запустите приложение на выполнение.
Задание 4. Самостоятельно разработайте графическое приложение mapPhone. Заранее заполните словарь 8 абонентами, при запуске приложения их список должен быть отображен в виджете QTextEdit. Добавьте 2 виджета QRadioButton, которые будут осуществлять переключение между поиском по ключу и поиском по значению. Также следует добавить 2 текстовых поля: для поиска и для вывода результата.
Также предусмотрите удаление и добавление записей в справочник, если введенный ключ уже существует при добавлении записи, выведите информационное сообщение. Контейнер QSet записывает элементы в некотором порядке и предоставляет возможность очень быстрого просмотра значений и выполнения с ними операций, характерных для множеств, таких как объединение, пересечение и разность. Необходимым условием является то, что ключи должны быть разными. Контейнер QSet можно использовать в качестве неупорядоченного списка для быстрого поиска данных. Основные методы контейнера QSet представлены в таблице 5.2. Таблица 5.2 − Методы контейнера QSet

Метод	Описание
unite()	Объединяет элементы множеств, т.е. элементы другого множества, которых нет в данном наборе элементов, вставляются в данное множество.
intersect()	Находит пересечение множеств. Удаляет те элементы из данного множества, которые не содержатся в другом множестве.
subtract()	Находит разность множеств. Удаляет те элементы данного множества, которые находится в другом множестве.

Задание 5. Создайте консольное приложение, где введем два множества: танцоры и певцы, и продемонстрируем работу методов контейнера QSet. Для работы приложения добавьте следующие строки:

```
setlocale(LC_ALL,"");
QSet <QString> singers;
QSet <QString> dancers;
singers << "Иван" << "Евгений" << "Анастасия" << "Александр" << "Ксения" ;
qDebug() << singers << endl;
dancers << "Марина" << "Петр" << "Иван" << "Ксения" ; qDebug() << dancers
<< endl; QSet<QString> result = singers; result.unite(dancers);
qDebug() << "Объединение множеств: ";
qDebug() << result << endl;
result = singers;
result.intersect(dancers);
qDebug() << "Пересечение множеств: ";
qDebug() << result << endl; result = singers; result.subtract(dancers);
qDebug() << "Разность множеств singers - dancers: ";
qDebug() << result << endl;
```

result = dancers;
result.subtract(singers);
qDebug() << "Разность множеств dancers - singers: ";
qDebug() << result << endl;

Задание 6. Самостоятельно создайте графическое приложение setgui.
Добавьте кнопку, 2 виджета QSpinbox и 3 виджета QTextEdit. Пользователь с помощью QSpinbox задает диапазон значений(от 1 до 100), при нажатии на кнопку 3 виджета QTextEdit должны быть заполнены:
а) QTextEdit_1 множеством чисел кратных 3;
б) QTextEdit_2 множеством простых чисел;
в) QTextEdit_3 множеством составных чисел.

Задание 7. Создайте две пустых телефонных книги для хранения записей. Заполните телефонные книги фамилиями абонентов и телефонными номерами. Выведите в textEdit содержимое телефонных книг. Выполните обмен словарей и выведите их на экран. Очистите словари. Дополните телефонную книгу новыми записями и измените один из номеров телефонной книги. Снова выведите в textEdit содержимое телефонной книги.

Работа с файлами в среде Qt Creator

Класс QFile унаследован от класса QIODevice. В нем содержатся методы для работы с файлами (открытие, закрытие, чтение и запись данных). Создать объект можно, передав в конструкторе строку, содержащую имя файла. Можно ничего не передавать в конструкторе, а сделать это после создания объекта вызовом метода setName().
Например:

```
QFile myFile ("D:\\file\\1.txt");
```

или

```
QFile myFile;
myFile.setName("1.txt ");
```

В процессе работы с файлами иногда требуется узнать, открыт файл или нет. Для этого вызывается метод QIODevice::isOpen(), который вернет значение true, если файл открыт, иначе - false.
Чтобы закрыть файл, нужно вызвать метод close(). С закрытием осуществляется запись всех данных буфера. Если требуется выполнить запись данных буфера в файл без его закрытия, то вызывается метод QFile::flush().
Проверить, существует ли нужный вам файл, можно статическим методом QFile::exists(). Этот метод принимает строку, содержащую полный или относительный путь к файлу. Если файл найден, то метод возвратит значение true, в противном случае - false.
Методы QIODevice::read() и QIODevice::write() позволяют считывать и записывать файлы блоками. Если требуется считать или записать данные за один раз, то используют методы QIODevice::writeAll() и QIODevice::readAll().
Для удаления файла класс QFile содержит статический метод remove(). В этот

метод необходимо передать строку, содержащую полный или относительный путь удаляемого файла.

Задание 8. Создадим графическое приложение, в котором продемонстрируем основные методы при работе с файлами.

1 Создайте приложение Qt Widgets с именем file.

2 Разработайте интерфейс пользователя, как показано на рисунке 6.1. Виджеты скомпонованы по горизонтали с разделителем. Установите запрет на изменение размеров окна. Сохраните изменения и запустите для просмотра.

3 Перейдите к слоту, который будет вызываться в ответ на событие clicked() и отвечать за чтение из файла, добавьте в него код:

```
QFile myFile ("D:\\proba\\file\\1.txt");
//прописываем путь к нашему файлу
if (!myFile.exists()) { //файл не найден
        QMessageBox::warning(this,"Ошибка","Файл не найден");
        return; }
if (!myFile.open(QIODevice::ReadOnly)) {
        //файл нельзя открыть для чтения
        QMessageBox::warning(this,"Ошибка","Файл нельзя открыть для
        чтения");
        return; }
QTextStream stream(&myFile);
/*создаем объект класса QTextStream и передаем в конструктор ссылку на
файл, из которого нужно производить чтение*/
QString buffer = stream.readAll(); //считываем в объект класса QString
ui→textEdit→setText(buffer);
//содержимое файла и помещаем в QTextEdit
myFile.close();
```

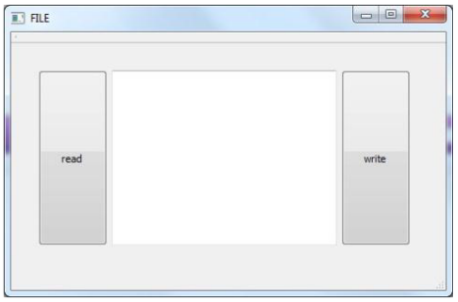


Рисунок 6.1 – Форма проекта file

4 Перейдите к слоту, который будет вызываться в ответ на событие clicked() и отвечать за запись в файл, добавьте в него код:

```
QFile myFile2 ("D:\\proba\\file\\2.txt");
//путь к нашему файлу
if (!myFile2.exists()) {
        //файл не найден    QMessageBox::warning(this,"Ошибка","Файл не
        найден");
        return; }
```

```
if (!myFile2.open(QIODevice::WriteOnly)) {
        //файл нельзя открыть для записи
        QMessageBox::warning(this,"Ошибка","Файл нельзя открыть для
        записи");
        return; }
QTextStream stream(&myFile2);
        /*Преобразуем строку из textedit в массив при помощи метода split,
используя в качестве разделителя между элементами массива пробел.
Константа SkipEmptyParts отвечает за пропуск лишних пробелов*/
QStringList numbers = ui→textEdit→toPlainText().split("
",QString::SkipEmptyParts);
int val;
for (int i = 0; i < numbers.length(); i++) {
val = numbers[i].toInt()*2;
/*преобразуем каждый элемент массива к целому числу и увеличиваем его
вдвое*/
stream << QString::number(val) + " ";
//помещаем во второй файл }
myFile2.close();
```

Сохраните изменения в приложении и просмотрите результат.

Задание 2. Самостоятельно создайте консольное приложение file_console. Дан файл A, компоненты которого являются целыми числами. Найдите: 1) сумму компонент файла A и запишите её в файл B; 2) последний компонент файла A и запишите его в файл C. Предусмотрите проверки на существование и на возможность чтения/записи.

Задания для самостоятельного выполнения:

Задание 1 Самостоятельное решение задач по вариантам. Реализовать приложения с консольным(графическим) интерфейсом пользователя.

Вариант 1 Число, лежащее в диапазоне от -999 до 999, вводится в **lineedit.** Вывести информационное сообщение – словесное описание данного числа вида «отрицательное двузначное число», «нулевое число», «положительное однозначное число» и т.д.

Вариант 2 Значения переменных X, Y, Z (переменные вводятся в **lineedit)** поменять местами так, чтобы они оказались упорядоченными по убыванию.

Вариант 3 Заменить наименьшее из трех чисел (числа вводятся в **lineedit)** суммой двух других чисел и вывести результат в **label.**

Вариант 4 Даны две переменные целого типа: A и B (переменные вводятся в **lineedit).** Если их значения не равны, то присвоить каждой переменной сумму этих значений, а если равны, то присвоить переменным нулевые значения.

Вариант 5 Заменить наибольшее из трех чисел (числа вводятся в **lineedit**) разностью двух других чисел и вывести результат в label.

Вариант 6 Даны две переменные целого типа: A и B (переменные вводятся в **lineedit**). Если их значения не равны, то присвоить каждой переменной максимальное из этих значений, а если равны, то присвоить переменным нулевые значения. **Вариант 7** Из трех данных чисел выбрать наименьшее и наибольшее (числа вводятся в **lineedit**), и заменить третье число их разностью (число должно быть изменено в текущем **lineedit**).

Вариант 8 Даны три переменные: X, Y, Z (переменные вводятся в **lineedit**). Если их значения упорядочены по убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.

Вариант 9 Перераспределить значения переменных X и Y (переменные вводятся в **lineedit**) так, чтобы в X оказалось меньшее из этих значений, а в Y − большее.

Вариант 10 Даны три переменные: X, Y, Z (переменные вводятся в **lineedit**). Если их значения упорядочены по возрастанию или убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.

Задание 2. Самостоятельное решение задач по вариантам. Реализовать для одномерных массивов консольный вариант приложения, для двумерных – с (консольным)графическим интерфейсом. В консольном приложении значения элементов массива вводятся пользователем, в графическом – задаются произвольно.

Вариант 1

1 Дан массив размера N. Вывести его элементы в обратном порядке.

2 Дано число k (0 < k < 11) и матрица размера m x n. Найти сумму и произведение элементов k-го столбца данной матрицы (нахождение суммы или произведения определяется пользователем в виджете comboBox).

Вариант 2

1 Дан массив размера N. Вывести вначале его элементы с четными индексами, а затем − с нечетными.

2 Дана матрица размера m x n. Найти суммы элементов всех ее четных и нечетных столбцов (нахождение суммы четных/нечетных столбцов определяется пользователем в виджете comboBox).

Вариант 3

1 Дан целочисленный массив A. Вывести номер первого из тех его элементов A[i], которые удовлетворяют двойному неравенству: A[1] < A[i] < A[10]. Если таких элементов нет, то вывести 0.

2 Дана матрица размера m x n. Найти минимальное и максимальное значение в каждой строке (нахождение максимального/минимального значения определяется пользователем в виджете comboBox).

Вариант 4

1 Дан целочисленный массив размера N. Преобразовать его, прибавив к четным числам первый элемент. Первый и последний элементы массива не изменять.

2 Дана матрица размера m x n. В каждой строке найти количество элементов, больших среднего арифметического всех элементов этой строки.

Вариант 5

1 Дан целочисленный массив размера N. Вывести вначале все его четные элементы, а затем – нечетные, сохраняя порядок следования элементов.

2 Дана матрица размера m x n. Преобразовать матрицу, поменяв местами минимальный и максимальный элемент в каждой строке.

Задание 3 Самостоятельное решение задач по вариантам. Для работы со связным списком реализовать консольное приложение, со списком – использовать консольный(графический) интерфейс. Перебор элементов осуществлять с помощью итераторов.

Вариант 1

1 Заполнить список случайными элементами. Реализовать добавление элемента в конец списка и удаления с конца (использовать RadioButton для выбора действия).

2 Создать два связных списка. Скопировать элементы первого во второй.

Вариант 2

1 Заполнить список случайными элементами. Реализовать добавление элемента в конец списка и удаления с начала (использовать RadioButton для выбора действия).

2 Создать два связных списка. Реализовать замену одного связного списка на другой.

Вариант 3

1 Заполнить 2 списка случайными элементами. Реализовать добавление введенного элемента в 1 список или второй, или в оба (использовать CheckBox).

2 Заполнить связный список случайными элементами и отсортировать их по возрастанию.

Вариант 4

1 Заполнить список случайными элементами и реализовать удаление элементов с позиций с N по K.

2 Заполнить связный список случайными элементами и отсортировать их по убыванию.

Вариант 5

1 Заполнить 2 списка случайными элементами и заменить все положительные элементы первого списка на значение минимального из второго списка.
2 Заполнить связный список случайными элементами. Удалить из списка все элементы, длина которых больше k.

Задание 4 Самостоятельное решение задач по вариантам. Реализовать приложения с консольным(графическим) интерфейсом.

Вариант 1

1 Заполнить стек 10 случайными числами из интервала
[-10; 20]. Просмотреть содержимое стека. Найти сумму положительных чисел, хранящихся в стеке.
2 Сформировать очередь из 8 чисел. Записать в очередь модуль разности между двумя соседними элементами очереди.

Вариант 2

1 Сформировать стек из 10 случайных целых чисел. Заменить в стеке все положительные значения на 1, а отрицательные - на -1.
2 Сформировать очередь из 10 чисел. Увеличить все значения в очереди на ее максимальный элемент. Результат поместить в очередь.

Вариант 3

1 Заполнить стек 10 случайными числами из интервала
[-10; 80]. Заменить все значения остатками от деления на номер элемента в стеке.
2 Сформировать очередь из 8 чисел. Заменить значение первого элемента очереди суммой первого и последнего, значение второго элемента очереди – суммой второго и предпоследнего и т.д.

Вариант 4

1 Сформировать стек из 8 чисел. Заменить значение первого элемента стека произведением первого и последнего, значение второго элемента стека – произведением второго и предпоследнего и т.д.
2 Заполнить очередь 8 случайными числами из интервала
[-20; 50]. Найти среднее арифметическое значений двух соседних элементов очереди. Результат поместить в очередь.

Вариант 5

1 Сформировать стек из 5 чисел. Поменять местами максимальный и минимальный элементы стека.
2 Заполнить очередь 8 случайными числами из интервала
[0; 50]. Заменить все четные числа их средним арифметическим значением.

Задание 5. Самостоятельное решение задач по вариантам. Реализовать приложения с консольным (графическим) интерфейсом.

Вариант 1 Имеется список класса (все имена различны). Определить, есть ли в классе человек, который побывал в гостях у всех. (Для каждого ученика составить
множество побывавших у него в гостях друзей, сам ученик в это множество не входит.)

Вариант 2 Задан некоторый набор товаров. Определить для каждого товара, какие из них имеются в каждом из n магазинов, какие товары есть хотя бы в одном магазине, каких товаров нет ни в одном магазине.
Вариант 3 Заданы имена девочек. Определить, какие из этих имен встречаются во всех классах данной параллели, какие есть только в некоторых классах, какие из этих имен не встречаются ни в одном классе.
Вариант 4 Известны марки машин, изготовляемых в данной стране и импортируемых за рубеж. Даны некоторые N стран. Определить для каждой из марок, какие из них были доставлены во все страны, доставлены в некоторые из стран, не доставлены ни в одну страну.
Вариант 5 В озере водится несколько видов рыб. Три рыбака поймали рыб, представляющих некоторые из имеющихся видов. Определить, какие виды рыб есть у каждого рыбака, какие рыбы есть в озере, но нет ни у одного из рыбаков

Задание 6Самостоятельное решение задач по вариантам. Реализовать приложения с консольным (графическим) интерфейсом.
Вариант 1 Дан файл f, компоненты которого являются действительными числами. Найдите:
а) наибольший компонент;
б) наименьший компонент с четным номером;
в) наибольший модуль компонента с нечётным номером;
г) разность первого и последнего компонента файла.
Вариант 2 Дан файл f, компоненты которого являются целыми числами. Запишите в файл g наибольшее значение первых пяти компонентов файла f, затем - следующих пяти компонентов и т.д. Если в последней группе окажется менее пяти компонентов, то последний компонент файла g должен быть равен наибольшему из компонентов файла f, образующих последнюю (неполную) группу.
Вариант 3 Даны символьные файлы f1 и f2. Перепишите с сохранением порядка следования компоненты файла f1 в файл f2, а компоненты файла f2 – в файл f1. Используйте вспомогательный файл h.
Вариант 4 Дан файл f, компоненты которого являются целыми числами. Получите в файле g все компоненты файла f: а) являющиеся чётными числами; б) делящиеся на 3 и не делящиеся на 7; в) являющиеся точными квадратами.
Вариант 5 Дан файл f, компоненты которого являются целыми числами. Запишите в файл g все чётные числа файла f, а в файл h - все нечётные. Порядок следования чисел сохранить.

Список литературы
1 Бланшет Ж., Саммерфилд М. Qt 4: Программирование GUI на C++. – Москва : Изд-во «КУДИЦ-ПРЕСС», 2007. - 629 с.

2 Саммерфилд М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++. – Санкт-Петербург : Символ-Плюс, 2011. – 560 с.
3 Шлее М. Qt 4.8. Профессиональное программирование на C++. − Санкт-Петербург : БХВ-Петербург, 2012. − 912 с.