Работа с файлами в библиотеке Qt

- Типовая последовательность работы с файлом
- Стандартный диалог выбора файла
- Класс файла
- Потоки данных
- Использование перегруженных операций для работы с потоками

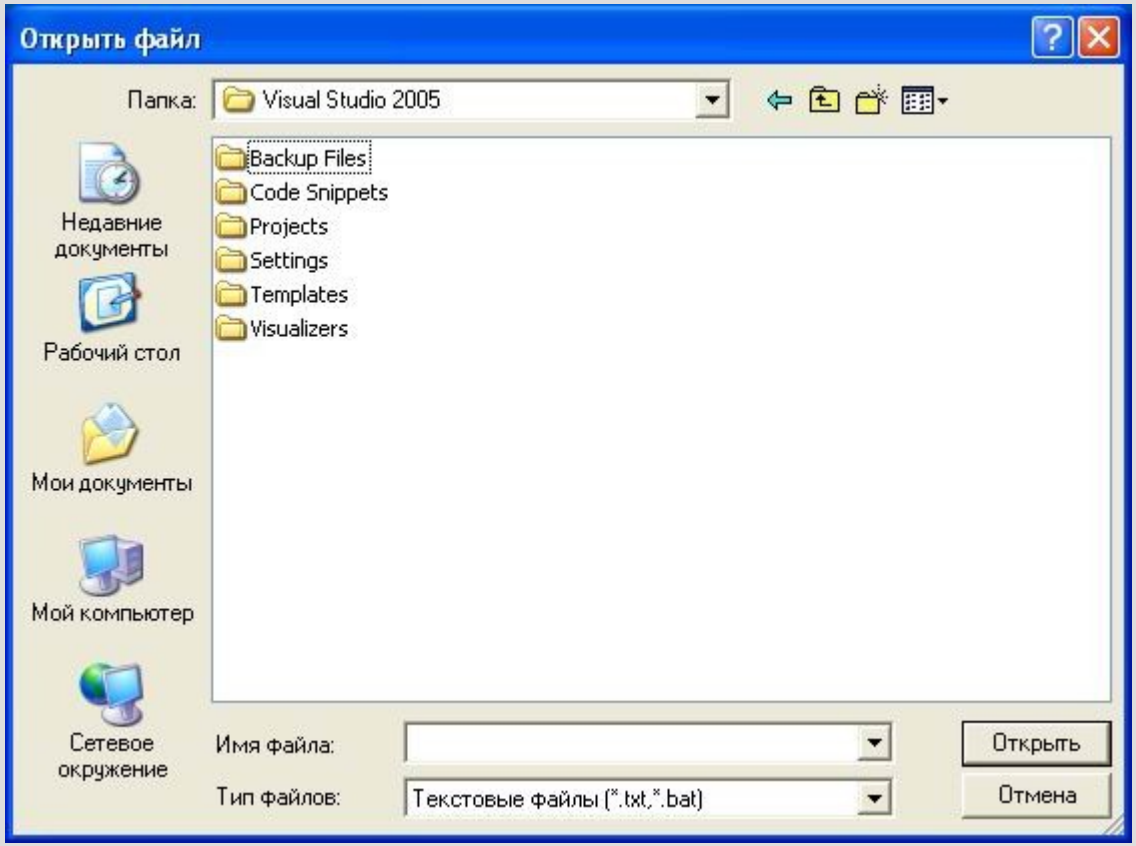
Типовая последовательность работы с файлом

- Класс **QFile** используется для представления файла в программе.
- Типовая последовательность работы с файлом:
  1) узнать имя файла, используя стандартный диалог (**QFileDialog**)
  2) создать объект файла (**QFile**)
  3) открыть файл (**QFile**)
  4) создать поток для ввода/вывода (**QDataStream** или **QTextStream**) и связать его с файлом
  5) считать/записать данные с использованием потока (**QDataStream** или **QTextStream**)
  6) закрыть файл (**QFile**)

Стандартный диалог выбора файла

- Стандартный диалог выбора файла предназначен для того, чтобы дать пользователю возможность выбрать файл (каталог) для открытия или сохранения.

- В библиотеке Qt стандартный диалог выбора файла реализуется классом **`QFileDialog`**



Работа со стандартным диалогом выбора файла

- В большинстве случаев при работе с классом **`QFileDialog`** используется одна из следующих статических функций:
  - **`getOpenFileName(...)`**
  - **`getSaveFileName(...)`**
  - **`getOpenFileNames(...)`**
  - **`getExistingDirectory(...)`**
- Все параметры этих функций имеют значения по умолчанию и могут не указываться при необходимости.

Фильтрация файлов по расширению

- Диалог выбора файлов может отбирать файлы по указанным расширениям, при этом пользователь может выбрать один из предложенных вариантов фильтрации.
- Строка фильтров состоит из описаний фильтров, разделенных двойным знаком точки с запятой.
- Описание фильтра состоит из имени и перечня шаблонов для имен файлов в круглых скобках
- Пример:

```
"Open Office document (*.odt, *.odp);;
 Portable document format (*.pdf)"
```

Диалог открытия файла

  - **QString** — имя выбранного файла, пустая строка в случае отмены
- **QFileDialog::getOpenFileName(**
  - **QWidget *** - указатель на окно-родитель
  - **const QString &** - строка заголовка
  - **const QString &** - начальный каталог (пустая строка если использовать текущий)
  - **const QString &** - фильтр файлов по расширению
  - **QString *** - изначально выбранный фильтр, обычно 0
  - **Options** — опции настройки, обычно 0

)

Задание

- Вызовите стандартный диалог открытия файла с заголовком «Открыть файл», в текущем каталоге, с двумя фильтрами: текстовые файлы (расширения txt, bat) и все файлы.
- Имя открытого файла сохранить в переменную **`Filename`**

Диалог открытия файла

```
Filename =
 QFileDialog::getOpenFileName( this,
 QString("Открыть файл"), QString(),
 QString("Текстовые файлы (*.txt,*.bat);;
          Все файлы (*.*)"));
```

Создание объекта файла

- Один из конструкторов **QFile** принимает имя файла в качестве параметра:

```
QFile::QFile(
  – const QString & - имя файла
)
```

- Созданный объект привязан к файлу, но файл при этом не открывается — он должен быть открыт методом **open()**.

Открытие файла

- **bool** — успешно ли открытие
- **QFile::open(**
  - **OpenMode** — режим доступа:
    - **QIODevice::ReadOnly** — только для чтения;
    - **QIODevice::WriteOnly** — только для записи, имеющиеся данные затираются;
    - **QIODevice::ReadWrite** — для чтения и записи, новые данные добавляются к уже существующим
    - дополнительно может указываться **QIODevice::Text** для взаимодействия с файлом в текстовом режиме (через операцию побитово ИЛИ)

)

Задание

- Создайте объект **`MyFile`** для файла с именем, заданным переменной **`Filename`**, если оно не пустое
- Откройте этот файл в режиме только записи данных

Работа с файлами

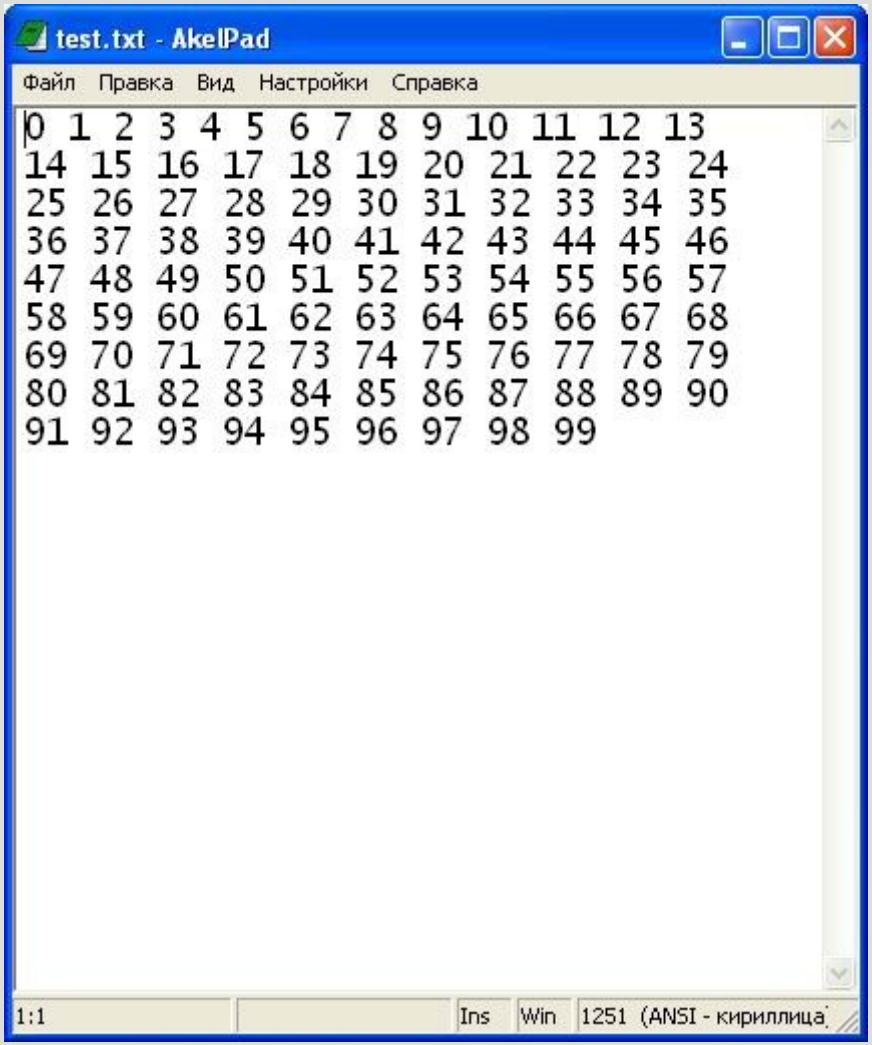
```
if (!Filename.isEmpty()) // если имя файла задан
{
  // Создаем файл
  QFile MyFile(Filename);

  // Открываем файл только для записи
  MyFile.open(QIODevice::WriteOnly);
}
```

Потоки ввода/вывода

- Используя объект файла (**QFile**), можно читать и записывать данные, хранящиеся в файле. Однако чтение/запись выполняется на низком уровне - по-байтово.
- Для высокоуровневой работы с файлом (чтения/записи чисел, строк, дат и т.д.) используются потоки.
- В библиотеке Qt потоки ввода/вывода представлены классами:
  - **QDataStream** — записывает и читает данные в двоичном формате
  - **QTextStream** — записывает и читает данные в текстовом формате

Примеры файлов в двоичном и текстовом формате



оба файла содержат числа от 0 до 99

Связывание потоков ввода/вывода с файлом

- Чтобы поток мог читать и записывать данные в файл он должен быть связан с объектом файла.
- Связь потока с файлом осуществляется при создании потока - в конструктор потока передается указатель на объект файла.

Задание

- Создайте объект **`MyFile`** для файла с именем, заданным переменной **`Filename`**, если оно не пустое
- Откройте этот файл в режиме только записи данных
- В случае успешного открытия файла создайте объект бинарного потока **`output`** для записи данных.

Работа с файлами

```
if (!Filename.isEmpty()) // если имя файла задан
{
  // Создаем файл
  QFile MyFile(Filename);

  // Открываем файл только для записи и создаем
  // поток для записи данных
  if ( MyFile.open(QIODevice::WriteOnly) )
  {
    QDataStream output(&MyFile);
  }
}
```

Работа с потоками ввода/вывода

- Ввод (чтение) из потока осуществляется через перегруженную операцию >>:

```
поток >> переменная_для_ввода;
```

- Несколько операций ввода могут быть записаны в цепочку:

```
поток >> переменная1 >> переменная2;
```

- Вывод (запись) в поток осуществляется аналогичным образом с использованием операции <<
- Операции ввода/вывода в поток сами определяют тип вводимых/выводимых данных и действуют соответственно.

Задание

Выведите в поток **`output`** строку **`str`** и число **`N`**

Вывод в поток

```
output << str << N;
```

Операции ввода/вывода

- В классах **`QDataStream`** и **`TextStream`** определены операции ввода/вывода для стандартных типов данных (числа, массивы символов, логические значения).
- В классах **`QString`**, **`QDate`**, **`QDateTime`** и **`QTime`** определены собственные операции ввода/вывода через поток.
- В контейнерных классах также определены операции ввода/вывода. Однако для хранимых значений и ключей должны быть определены операции ввода/вывода.

Задание

Имеется словарь городов. В словаре хранится название города и кол-во его жителей

```
QMap <QString,int> cities;
```

Запишите в поток **output** содержимое контейнера, если для контейнера определена следующая операция:

```
QDataStream & operator<< ( QDataStream &
    out, const QMap<Key, T> & map )
```

Пример записи контейнера в поток данных

```
// Так тип ключа (QString) поддерживает
// операцию ввода-вывода через поток, а
// сам поток умеет работать с целочисленными
// значениями, то запись контейнера в поток
// выполняется одной операцией
output << cities;
```

Перегрузка операций ввода/вывода в поток

- Возможно создать (перегрузить) операции ввода-вывода через поток для собственных классов.
- Операции потокового ввода/вывода являются бинарными: левым операндом всегда является ссылка на поток, правым — ссылка на объект класса. Для операции записи данных обычно передается ссылка на константный объект.
- Для того чтобы операцию ввода-вывода можно было выполнять последовательно, она возвращает ссылку на поток, с которым работаем.

Задание

Пользуясь знаниями о перегрузке операций дайте ответ на следующие вопросы

1) Каким образом должны быть перегружен операции ввода/вывода в поток — как методы класса или свободные функции? Почему?
2) Сколько аргументов должны принимать операции и каких они должны быть типов? Какие из них могут быть переданы константными ссылками?
3) Каким должен быть тип возвращаемого значения перегруженной операции?

Перегрузка операций ввода/вывода в поток

1) Операции ввода/вывода в поток могут быть перегружены только как свободные функции, т.к. их первым (левым) аргументом является поток, а вы не являетесь автором класса потока и не можете добавлять в него новые методы.
2) Свободная функция, перегружающая операцию, должна принимать два аргумента:
   - ссылку на поток (неконстантную, т.к. поток меняется и при записи и при чтении);
   - ссылку на класс записываемых (считываемых) данных - может быть константной при записи, но не при чтении.

Перегрузка операций ввода/вывода в поток

3)Перегруженная операция должна возвращать ссылку на поток, чтобы операции могли работать в цепочке:

```
output << str << N

    output    << N

       output
```

Вычисляется первым, результат должен равняться output

Задание

Составьте заголовки операций ввода и вывода в поток `QDataStream` класса `FIO`

Синтаксис операций ввода/вывода в поток

- Заголовок операции записи (вывода) данных

```
QDataStream & operator<< (
    QDataStream & stream,
    const FIO & data )
```

Возвращаемое значение: ссылка на поток

Первый аргумент: ссылка на поток

Второй аргумент: ссылка на класс выводимых данных

- Заголовок операции чтения (ввода) данных

```
QDataStream & operator>> ( QDataStream & stream, FIO & data )
```

Перегрузка операций ввода/вывода в поток

- Ввод/вывод класса осуществляется, как правило, последовательным вводом (выводом) всех его полей.
- Следите за тем, чтобы порядок ввода и вывода полей класса в поток был одинаковым.

Задание

- Дан класс:

```
class FIO
 {
 public:
    QString FirstName;
    QString LastName;
    QString SurName;
    ....
 };
```

- Определите для него операцию вывода в поток **QDataStream**

Перегрузка операции вывода в поток

```
QDataStream &operator<< (QDataStream & out,
                         const FIO &data)
{
   out << data.FirstName << data.LastName <<
     data.SurName;

   return out;
}
```

Особенности работы с текстовыми потоками

- Операции ввода и вывода для бинарного потока данных (класса **QDataStream**) являются обратимыми, т.е. если последовательность операций ввода совпадает с последовательностью операций вывода, то данные будут считаны верно.
- Для текстового потока (класса **QTextStream**) операции ввода и вывода не обратимы. Например, сохранив в поток три строки, при чтении мы получим одну общую строку.

Особенности работы с текстовыми потоками

- Для того чтобы операции ввода-вывода с текстовым потоком были обратимы, необходимо записывать строки в поток через разделители.
- В качестве разделителей обычно используются пробелы, знаки табуляции и переводы строк.
- Для вывода в текстовый поток перевода строки используется псевдопеременная **`endl`**.

Задание

- Для класса **FIO**, приведенного выше, перегрузите оператор вывода в поток **QTextStream**, разделяя поля символами перевода строки.
- Перегрузите оператор ввода класса **FIO** из потока **QTextStream**.

Вывод в текстовый поток

```
// Записываем FIO в поток
QTextStream &operator<< (QTextStream &out,
  const FIO &data)
{
   out << data.FirstName << endl
       << data.LastName  << endl
       << data.SurName   << endl;

   return out;
}
```

Ввод из текстового потока

```
// Считываем FIO из потока
QTextStream &operator>> (QTextStream &in, FIO
  &data)
{
   in >> data.FirstName
      >> data.LastName
      >> data.SurName;

   return in;
}
```

Потоковый ввод/вывод с консоли

- Поток **QTextStream** можно использовать для взаимодействия с консолью (текстовым экран), если программа консольная. Для этого используются переменные-псевдофайлы **stdin** (ввод) и **stdout** (вывод).
- Пример создания потока для ввода с консоли:

```
QTextStream conin(stdin);
```

- Пример создания потока для вывода на консоль:

```
QTextStream conout(stdout);
```