# Лабораторная работа № 6

## Урок № 1

#include <QCoreApplication>

#include <iostream>

int main(int argc, char \*argv[])

{

    QCoreApplication a(argc, argv);

    std::cout << "Qt version: " << qVersion() << std::endl;

    return a.exec();

}

## Урок № 2

#include <QTextStream>

int main() {

   QTextStream out(stdout);

   QString field1 = "Name: ";

   QString field2 = "Occupation: ";

   QString field3 = "Residence: ";

   QString field4 = "Marital status: ";

   // Вычисляем размер самой широкой строки

   int width = field4.size();

   out << field1.rightJustified(width, ' ') << "Robert\n";

   out << field2.rightJustified(width, ' ') << "programmer\n";

   out << field3.rightJustified(width, ' ') << "New York\n";

   out << field4.rightJustified(width, ' ') << "single\n";

   return 0;

}

## Урок № 3

#include <QTextStream>

#include <QDate>

#include <ctime> // подключаем необходимый заголовочный файл

int main() {

   QTextStream out(stdout);

   out.setCodec("CP866"); // устанавливаем CP866-кодировку для консоли

   time\_t t = time(0); // получаем Unix-время

   out << t << endl;

   QDateTime dt;

   dt.setTime\_t(t); // конвертируем Unix-время в обычный формат даты

   out << dt.toString() << endl;

   QDateTime cd = QDateTime::currentDateTime();

   out << cd.toTime\_t() << endl; // метод toTime\_t() также может быть использован для получения Unix-времени

   return 0;

}

## Урок № 4

#include <QTextStream>

#include <QMap>

int main(void) {

    QTextStream out(stdout);

    // Создаем QMap, содержащий 2 пары элементов

    QMap<QString, int> items = { {"coins", 5}, {"books", 3} };

    // С помощью метода insert() добавляем новую пару значений

    items.insert("bottles", 7);

    // Получаем все значения словаря и выводим их на экран

    QList<int> values = items.values(); // метод values() возвращает список значений словаря

    out << "Values:" << endl;

    for (int val : values) {

        out << val << endl;

    }

    // Аналогично выводим все ключи словаря

    QList<QString> keys = items.keys(); // метод keys() возвращает список, содержащий все ключи в словаре

    out << "Keys:" << endl;

    for (QString key : keys) {

        out << key << endl;

    }

    // Создаем итератор для QMap в Java-стиле

    QMapIterator<QString, int> it(items); // этот итератор может использоваться для итерации по элементам QMap

    out << "Pairs:" << endl;

    // С помощью итератора перебираем все элементы QMap

    while (it.hasNext()) {

        it.next();

        out << it.key() << ": " << it.value() << endl; // метод key() возвращает текущий ключ, а метод value() возвращает текущее значение

    }

}

## Урок № 5

#include <QTextStream>

#include <QFileInfo>

#include <QDir>

#include <conio.h>

int main(int argc, char \*argv[]) {

  QTextStream out(stdout);

  out.setCodec("cp-866");

  if (argc != 2) {

      qWarning("Usage: list\_dir directory");

      return 1;

  }

  QString directory = argv[1];

  // Создаем объект класса QDir с заданным именем каталога

  QDir dir(directory);

  if (!dir.exists()) {

      qWarning("The directory does not exist");

      return 1;

  }

  // Метод setFilter() определяет тип файлов, которые должны быть возвращены методом entryInfoList()

  dir.setFilter(QDir::Files | QDir::AllDirs);

  // Метод setSorting() задает порядок сортировки, используемый методом entryInfoList()

  dir.setSorting(QDir::Size | QDir::Reversed);

  // Метод entryInfoList() возвращает список объектов QFileInfo для всех файлов и папок в каталоге, отфильтрованных и упорядоченных соответствующими методами

  QFileInfoList list = dir.entryInfoList();

  int max\_size = 0;

  // Проходимся по списку и определяем максимальный размер имени файла

  foreach (QFileInfo finfo, list) {

      QString name = finfo.fileName();

      int size = name.size();

      if (size > max\_size) {

          max\_size = size;

      }

  }

  // Добавляем дополнительные два пробела к длине столбца

  int len = max\_size + 2;

  // Выводим имена столбцов

  out << QString("Filename").leftJustified(len).append("Bytes") << endl; // метод leftJustified() возвращает строку заданного размера, выравненную по левому краю и дополненную символом заполнения (по умолчанию пробел) справа

  // Просматриваем список файлов и выводим их названия и размеры. Первый столбец выравнивается по левому краю и заполняется пробелами по мере необходимости, затем добавляется второй столбец

  for (int i = 0; i < list.size(); ++i) {

    QFileInfo fileInfo = list.at(i);

    QString str = fileInfo.fileName().leftJustified(len);

    str.append(QString("%1").arg(fileInfo.size()));

    out << str << endl;

  }

  return 0;

}