ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Институт высшей инженерно-технической школы

Курсовая работа

по дисциплине «Специальные вопросы программирования»

Тема: Разработка приложения для обработки списков значений

Студент гр. Е4160 _____ Макаров Д. М.

Преподаватель _____ Шарков И. А.

Санкт-Петербург

2024

Задание на курсовой проект

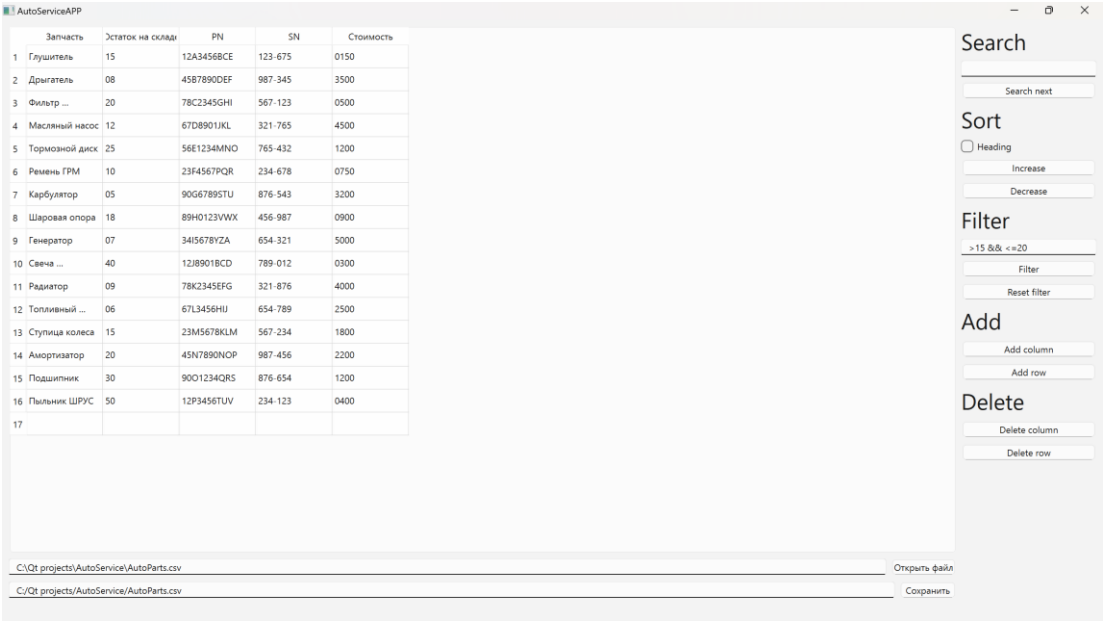
На основе связанного списка разработать приложение, хранящее набор объектов (пользователи, контакты, страны, автомобили, запчасти, оборудование и т.д. на усмотрение студента). Приложение должно обладать следующими характеристиками/возможностями:

- Графический интерфейс (консольное приложение в стиле DOS).
- Загрузка набора объектов из файла.
- Сохранение набора объектов в файл.
- Вывод всех объектов на экран в виде таблицы.
- Добавление новых объектов через графический интерфейс.
- Удаление произвольного объекта.
- Вывод объектов на экран, советующих фильтру. Должна поддерживаться фильтрация как минимум по одному числовому полю и по одному текстовому.
- Вывод объектов на экран, в указанном порядке. Должна поддерживаться сортировка как минимум по одному числовому полю и по одному текстовому.

Описание функционала программы

1. Графический интерфейс и функционал программы

Графический интерфейс программы был разработан в программе Qt creator – это наиболее комфортная среда разработки, в которой у меня был опыт разработки программ на языке C++. Вид графического интерфейса программы представлен на рисунке 1.



	Запчасть	Остаток на складе	PN	SN	Стоимость
1	Глушитель	15	12A3456BCE	123-675	0150
2	Дрыгатель	08	45B7890DEF	987-345	3500
3	Фильтр ...	20	78C2345GHI	567-123	0500
4	Масляный насос	12	67D8901JKL	321-765	4500
5	Тормозной диск	25	56E1234MNO	765-432	1200
6	Ремень ГРМ	10	23F4567PQR	234-678	0750
7	Карбулятор	05	90G6789STU	876-543	3200
8	Шаровая опора	18	89H0123VWX	456-987	0900
9	Генератор	07	34I5678YZA	654-321	5000
10	Свеча ...	40	12J8901BCD	789-012	0300
11	Радиатор	09	78K2345EFG	321-876	4000
12	Топливный ...	06	67L3456HIJ	654-789	2500
13	Ступица колеса	15	23M5678KLM	567-234	1800
14	Амортизатор	20	45N7890NOP	987-456	2200
15	Подшипник	30	90O1234QRS	876-654	1200
16	Пыльник ШРУС	50	12P3456TUV	234-123	0400
17					

Рисунок 1. Вид графического интерфейса программы.

В приложении реализован следующий функционал: загрузка объектов в программу из CSV-файла, сохранение объектов в CSV-файл, последовательный поиск объектов в таблице, установка заголовка (по необходимости, т.е. если он есть в файле), вывод объектов на экран в виде таблице, добавление/удаление объектов в ячейках, добавление/удаление строк и столбцов, вывод объектов на экран в указанном порядке по возрастанию/убыванию, фильтрация числовых ячеек.

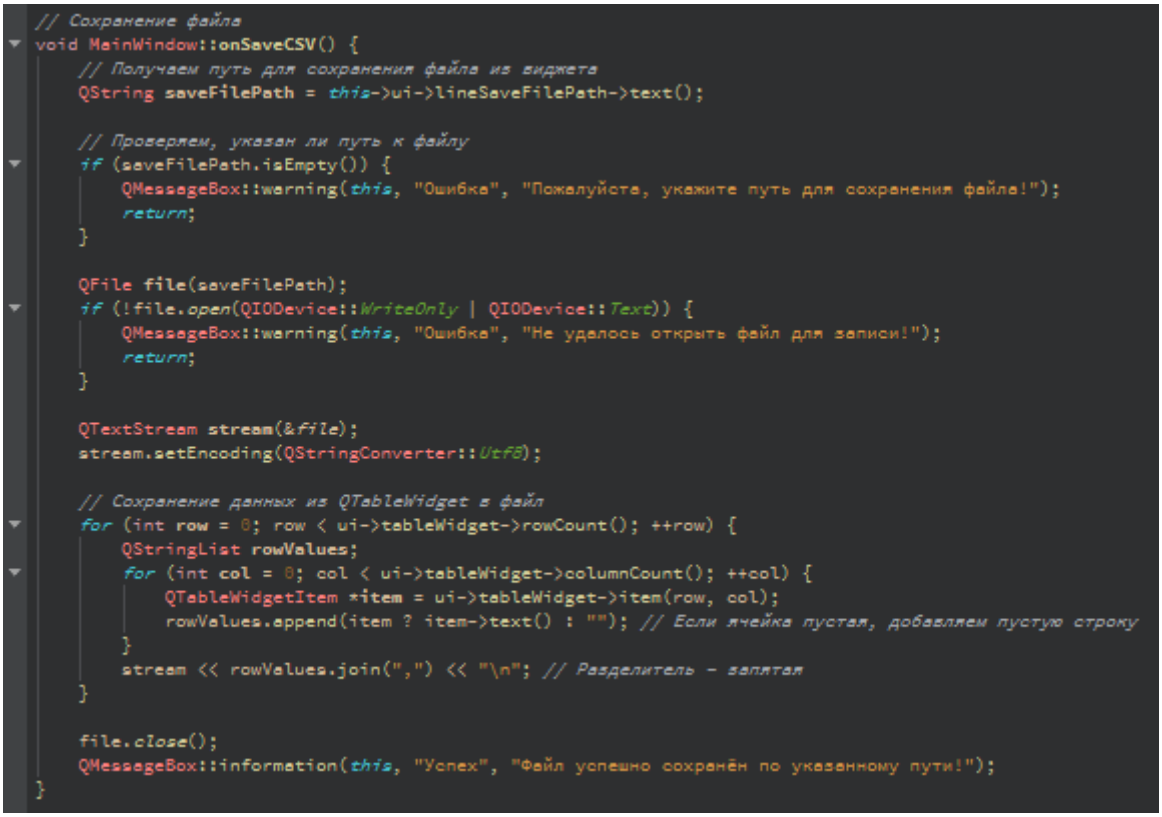
2. Загрузка и сохранение набора объектов в файл

Загрузка объектов в таблицу производится с помощью строки в нижнем краю окна программы, в которую пользователь вводит путь к CSV-файлу. По нажатию кнопки открыть файл – файл выгружается в таблицу. При указании пути и имени файла в строку

сохранения файла и по нажатию кнопки файл будет сохранен. Листинг кода представлен на рисунке 2 и 3.

```
// Слот onLoadCSV
void MainWindow::onLoadCSV() {
    // путь к файлу CSV
    QString openFilePath = this->ui->lineFilePath->text();

    // Проверим, существует ли файл
    if (!QFile::exists(openFilePath)) {
        QMessageBox::warning(this, "Ошибка", "Файл не найден!");
        return;
    }

    QFile file(openFilePath);
    if (!file.open(QIODevice::ReadOnly)) {
        QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл!");
        return;
    }

    QTextStream in(&file);
    QVector<QStringList> data;

    // Чтение CSV файла
    while (!in.atEnd()) {
        QString line = in.readLine();
        QStringList fields = line.split(",");
        data.append(fields);
    }
    file.close();

    // Преобразование строк в числа и поиск максимальной длины
    QVector<int> maxLengths(data.first().size(), 0); // Массив для хранения максимальной длины каждого столбца

    // Находим максимальные длины для каждого столбца
    for (const QStringList &row : data) {
        for (int col = 0; col < row.size(); ++col) {
            bool ok;
            int value = row[col].toInt(&ok); //  преобразование строки в число
            if (ok) { // Если это число
                int length = QString::number(value).length(); // Длина числа
                maxLengths[col] = qMax(maxLengths[col], length); // Обновляем максимальную длину
            }
        }
    }

    // Добавляем нули в числовые элементы
    for (int i = 0; i < data.size(); ++i) {
        for (int col = 0; col < data[i].size(); ++col) {
            bool ok;
            int value = data[i][col].toInt(&ok); // Ппреобразование строки в число
            if (ok) {
                int length = QString::number(value).length();
                // Если длина числа меньше максимальной, добавляем нули
                if (length < maxLengths[col]) {
                    data[i][col] = QString("%1").arg(value, maxLengths[col], 10, QChar('0')); // Форматируем с добавлением нулей
                }
            }
        }
    }

    this->ui->tableWidget->setRowCount(data.size());
    for (int row = 0; row < data.size(); ++row) {
        for (int col = 0; col < data[row].size(); ++col) {
            this->ui->tableWidget->setItem(row, col, new QTableWidgetItem(data[row][col]));
        }
    }
}
```

Рисунок 2. Функция для открытия CSV-файла



```
// Сохранение файла
void MainWindow::onSaveCSV() {
    // Получаем путь для сохранения файла из виджета
    QString saveFilePath = this->ui->lineSaveFilePath->text();

    // Проверяем, указан ли путь к файлу
    if (saveFilePath.isEmpty()) {
        QMessageBox::warning(this, "Ошибка", "Пожалуйста, укажите путь для сохранения файла!");
        return;
    }

    QFile file(saveFilePath);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл для записи!");
        return;
    }

    QTextStream stream(&file);
    stream.setEncoding(QStringConverter::Utf8);

    // Сохранение данных из QTableWidget в файл
    for (int row = 0; row < ui->tableWidget->rowCount(); ++row) {
        QStringList rowValues;
        for (int col = 0; col < ui->tableWidget->columnCount(); ++col) {
            QTableWidgetItem *item = ui->tableWidget->item(row, col);
            rowValues.append(item ? item->text() : ""); // Если ячейка пустая, добавляем пустую строку
        }
        stream << rowValues.join(",") << "\n"; // Разделитель - запятая
    }

    file.close();
    QMessageBox::information(this, "Успех", "Файл успешно сохранён по указанному пути!");
}
```

Рисунок 3. Функция для сохранения таблицы в CSV-файл

3. Вывод всех объектов на экран в виде таблицы

   Чтобы вывести файлы в виде таблицы, был использован виджет QTableWidget – он позволяет комфортно отобразить таблицу и предоставляет небольшой функционал взаимодействия с ней. Вид таблицы также представлен на рисунке 1.

4. Добавление новых элементов в таблицу

   Добавление новых элементов в таблицу реализовано через функционал виджета QTableWidget, в частности добавление/удаление строк и столбцов, изменение содержания ячеек. Вид этой части кода представлен на рисунке 4.

```
// Добавление новой строки
void MainWindow::onAddRow() {
    int currentRowCount = ui->tableWidget->rowCount();
    ui->tableWidget->insertRow(currentRowCount);
}

// Добавление нового столбца
void MainWindow::onAddColumn() {
    int currentColumnCount = ui->tableWidget->columnCount();
    ui->tableWidget->insertColumn(currentColumnCount);
}

void MainWindow::onDelRow() {
    int currentRow = ui->tableWidget->currentRow(); // Получить индекс выбранной строки
    if (currentRow >= 0) { // Проверка, что строка выбрана
        ui->tableWidget->removeRow(currentRow);
    }
}

void MainWindow::onDelColumn() {
    int currentColumn = ui->tableWidget->currentColumn(); // Получить индекс выбранного столбца
    if (currentColumn >= 0) { // Проверка, что столбец выбран
        ui->tableWidget->removeColumn(currentColumn);
    }
}
```

Рисунок 4. Функции добавления и удаления столбцов/строк

5. Вывод объектов на экран, советующих фильтру

Фильтрация таблицы происходит следующим образом – пользователь вводит выражении для фильтрации ячеек по какому-либо признаку, причем фильтрация происходит только для числовых ячеек, для строковых есть сортировка по возрастанию/убыванию. Пользователь вводит выражение вида: «>15 && <=20», и по нажатию кнопки ячейки подсвечиваются желтым цветом. По нажатию кнопки Reset filter – выделение с отфильтрованных ячеек снимается. Листинг этой части кода можно наблюдать на рисунке 5.

```cpp
// Фильтрация таблицы по выражению
void MainWindow::onFilterButtonClicked() {
    static const QRegularExpression regex(R"((<|>|<=|>=|==|!=)(\d+))");

    QString filterExpression = ui->lineFilter->text().trimmed();

    if (filterExpression.isEmpty()) {
        return; // Ничего не делать, если выражение пустое
    }

    // Проверяем на сложное выражение
    if (filterExpression.contains("&&")) {
        QStringList conditions = filterExpression.split("&&");
        QVector<QPair<QString, int>> parsedConditions;

        for (const QString &condition : conditions) {
            QRegularExpressionMatch match = regex.match(condition.trimmed());
            if (!match.hasMatch()) {
                QMessageBox::warning(this, "Некорректное выражение", "Введите выражение в формате >число && <число и т.д.");
                return;
            }
            parsedConditions.append({match.captured(1), match.captured(2).toInt()});
        }

        for (int row = 0; row < ui->tableWidget->rowCount(); ++row) {
            for (int col = 0; col < ui->tableWidget->columnCount(); ++col) {
                QTableWidgetItem *item = ui->tableWidget->item(row, col);
                if (!item) continue;

                bool highlight = true;
                bool validConversion;
                int cellValue = item->text().toInt(&validConversion);

                if (!validConversion) {
                    item->setBackground(Qt::white); // Сбрасываем подсветку для некорректных данных
                    continue;
                }

                for (const auto &condition : parsedConditions) {
                    const QString &operatorStr = condition.first;
                    int value = condition.second;

                    if (operatorStr == ">" && !(cellValue > value)) {
                        highlight = false;
                    } else if (operatorStr == "<" && !(cellValue < value)) {
                        highlight = false;
                    } else if (operatorStr == ">=" && !(cellValue >= value)) {
                        highlight = false;
                    } else if (operatorStr == "<=" && !(cellValue <= value)) {
                        highlight = false;
                    } else if (operatorStr == "==" && !(cellValue == value)) {
                        highlight = false;
                    } else if (operatorStr == "!=" && !(cellValue != value)) {
                        highlight = false;
                    }
                }

                item->setBackground(highlight ? Qt::yellow : Qt::white);
            }
        }
        return;
    }
```

Рисунок 5. Функция для фильтрации ячеек по заданному выражению

6. Сортировка ячеек таблицы

Сортировка ячеек таблицы производится как по возрастанию, так и по убыванию, причем как для строковых ячеек, так и для числовых. Функция сортировки представлена на рисунке 6. Чтобы сортировка числовых выражений происходила корректно, при загрузке CSV-файла в ячейки с числовыми значениями добавляются

нули слева для равенства длины символов с максимальным значением столбца, без добавления нулей числа «15» и «150» программа посчитает равными.

```
// Сортировка по возрастанию
void MainWindow::onSortAscending() {
    int column = ui->tableWidget->currentColumn(); // Получаем текущий выбранный столбец
    if (column == -1) {
        QMessageBox::warning(this, "Ошибка", "Выберите столбец для сортировки!");
        return;
    }
    int headerRow = ui->headingCheckBox->isChecked() ? 0 : 1; // Если включен заголовок, пропускаем первую строку
    ui->tableWidget->sortItems(column, Qt::AscendingOrder);
    if (headerRow == 1) {
        restoreHeaderRow(); // Восстанавливаем заголовок после сортировки
    }
}
// Сортировка по убыванию
void MainWindow::onSortDescending() {
    int column = ui->tableWidget->currentColumn(); // Получаем текущий выбранный столбец
    if (column == -1) {
        QMessageBox::warning(this, "Ошибка", "Выберите столбец для сортировки!");
        return;
    }
    int headerRow = ui->headingCheckBox->isChecked() ? 1 : 0; // Если включен заголовок, пропускаем первую строку
    ui->tableWidget->sortItems(column, Qt::DescendingOrder);
    if (headerRow == 1) {
        restoreHeaderRow(); // Восстанавливаем заголовок после сортировки
    }
}
```

Рисунок 6. Функция для сортировки ячеек

7. Добавление заголовков

В программе предусмотрена функция добавления заголовка. Если в файле присутствует заголовок, то после загрузке файла следует установить чекбокс заголовка в «True», чтобы заменить стандартные имена столбцов на значения ячеек первой строки, в которой и будет находиться заголовок. Функция установки заголовка представлена на рисунке 7

```
// Обработка изменения состояния чекбокса заголовка
void MainWindow::onHeadingStateChanged(int state) {
    if (state == Qt::Checked) {
        QStringList columnLabels;

        // Извлекаем значения из первой строки
        for (int col = 0; col < ui->tableWidget->columnCount(); ++col) {
            QTableWidgetItem *item = ui->tableWidget->takeItem(0, col);
            columnLabels.append(item ? item->text() : QString(""));
        }

        ui->tableWidget->setHorizontalHeaderLabels(columnLabels);
        ui->tableWidget->removeRow(0); // Удаляем первую строку
    } else {
        QStringList columnLabels;
        for (int col = 0; col < ui->tableWidget->columnCount(); ++col) {
            QTableWidgetItem *headerItem = ui->tableWidget->horizontalHeaderItem(col);
            columnLabels.append(headerItem ? headerItem->text() : QString(""));
        }

        // Добавляем значения заголовка в первую строку
        ui->tableWidget->insertRow(0);
        for (int col = 0; col < columnLabels.size(); ++col) {
            ui->tableWidget->setItem(0, col, new QTableWidgetItem(columnLabels[col]));
        }

        // Сбрасываем заголовки столбцов с нумерацией
        for (int col = 0; col < columnLabels.size(); ++col) {
            columnLabels[col] = QString::number(col + 1); // Нумерация столбцов с 1
        }
        ui->tableWidget->setHorizontalHeaderLabels(columnLabels);
    }
}

// Восстановление заголовка после сортировки
void MainWindow::restoreHeaderRow() {
    QList<QTableWidgetItem *> headerItems;

    for (int col = 0; col < ui->tableWidget->columnCount(); ++col) {
        QTableWidgetItem *item = ui->tableWidget->takeItem(0, col);
        if (item) {
            headerItems.append(item);
        }
    }

    ui->tableWidget->removeRow(0);
    ui->tableWidget->insertRow(0);

    for (int col = 0; col < headerItems.size(); ++col) {
        ui->tableWidget->setItem(0, col, headerItems[col]);
    }
}
```

Рисунок 7. Функция для установки заголовка

Выводы и результаты

В результате выполнения работы была разработана программа для загрузки, обработки и сохранения объектов в файл. Такую программу можно использовать, например, для ведения отчетности какого-либо предприятия, по изначальной задумке – автосервиса. В программу можно вносить данные о количестве деталей, их стоимости, остатке на складе, серийных номерах и пр. В целом, весь функционал, который был необходим в задании был успешно реализован, только графический интерфейс был создан не в стиле DOS а в Qt creator, потому что так программа выглядит визуально красивее и она становится удобнее для пользователя.