**ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА  
 (РОСАВИАЦИЯ)**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)**

**Кафедра вычислительных машин, комплексов, систем и сетей**

Курсовая работа

защищена с оценкой

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

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

(подпись преподавателя,

дата)

КУРСОВАЯ РАБОТА

По дисциплине «Операционные системы»

Тема: «Создание Win 32 приложения с использованием функции API»

Задание: «Task List»

Выполнил: студент 2 курса

факультета ПМиВТ

группы ЭВМ2-1

Жердева Валентина

Преподаватель:

доцент кафедры ВМКСС

Черкасова Наталья Ивановна

Москва – 2024

**Содержание**

[Аннотация 3](#_Toc167929217)

[1. Техническое задание 4](#_Toc167929218)

[2. Краткие теоретические сведения 6](#_Toc167929219)

[2.1. Особенности создания приложений для ОС семейства   
Windows 6](#_Toc167929220)

[2.2. Краткая характеристика функций API 8](#_Toc167929221)

[2.3. Краткая характеристика интегрированной среды разработки 12](#_Toc167929222)

[3. Состав и характеристики файлов проекта 14](#_Toc167929223)

[4. Стандартные классы и функции приложения 16](#_Toc167929224)

[5. Пользовательские классы и функции приложения 18](#_Toc167929225)

[6. Структура программы 19](#_Toc167929226)

[7. Системные требования 20](#_Toc167929227)

[8. Руководство пользователя 21](#_Toc167929228)

[8.1. Правила установки и запуска программы 21](#_Toc167929229)

[8.2. Последовательность действий для запуска программы 21](#_Toc167929230)

[8.3. Последовательность действий для выполнения всех  
 требуемых по техническому заданию функций 22](#_Toc167929231)

[9. Список литературы 27](#_Toc167929232)

[10. Приложение 28](#_Toc167929233)

[10.1. Листинги программ 28](#_Toc167929234)

[10.2. Алгоритмы функций 45](#_Toc167929235)

# Аннотация

Курсовая работа выполнена в среде Visual Studio 2022 на языке программирования C++. В рамках курсовой работы разработано приложение, позволяющее выводить заметки на экран, а также создавать, редактировать и удалять их. Отчет по результатам работы состоит из графической части и пояснительной записки. Пояснительная записка включает в себя техническое задание, структуру программы, информацию о системных требованиях и руководство пользователя. Графическая часть включает в себя листинги программ и результаты выполнения программы.

# 1. Техническое задание

Разработать программу, демонстрирующую применение внешних интерфейсов взаимодействия с операционной системой. Требования к программе:

* Программа работает в графическом режиме.
* Программа использует кнопки для ввода данных и графическое меню.
* Программа содержит поле для ввода данных и вывода результата.
* Возможность вывода заметок.
* Возможность редактирования заметок.
* Возможность удаления заметок.
* Возможность перехода между страницами (если создано заметок больше 4).

# 2. Краткие теоретические сведения

## 2.1. Особенности создания приложений для ОС семейства Windows

Операционные системы Windows обладает рядом особенностей. Прежде всего – это графический интерфейс, обеспечивающий пользователю удобство в работе и привлекательное графическое изображение. ОС Windows поддерживает 32/64-битный интерфейс программирования Win32 API - (Application Programming Interface – интерфейс прикладного программирования) [1]. API - это набор похожих на подпрограммы процедур - функций, которые программы вызывают для решения всех задач, связанных с работой ОС. Реализованы они в виде библиотек динамической компоновки .dll, основными из которых являются gdi, user, kernel. Эти библиотеки отображаются в адресное пространство каждого процесса. Windows-приложения выполняются в собственных окнах. Каждое приложение располагает, по крайней мере, одним собственным окном. Через окна приложения выполняется ввод/вывод информации пользователя. Главное окно – это и есть само приложение, но окно – это также и визуальный интерфейс. Работа в Windows ориентирована на события. В Windows приложения выполняются пошагово. После решение одной подзадачи, управление возвращается Windows, которая может вызывать другие программы. Windows переключается между различными приложениями. Программист инициирует событие (вызов команды меню, щелчок мыши на окне), событие обрабатывается, и программное управление передается в соответствующее приложение. Приложение вызывается для обработки события. Таким образом, разработка приложения – это создание окна приложения (создать окно, зарегистрировать его класс, сделать его видимым) и организация обработки сообщений пользователя. В ОС Windows для обеспечения взаимодействия различных процессов и потоков в приложении используется механизм обработки сообщений. Для того чтобы иметь возможность работать с каким-либо устройством, например, с клавиатурой или мышью, программам DOS приходилось отслеживать состояние этих устройств и ожидать их реакции на посланные им сообщения. ОС Windows управляется сообщениями, и уже не программа ожидает реакции от устройства, а сообщение о реакции устройства запускает ту или иную программу. Та часть программы, которая запускается в ответ на конкретное сообщение, называется функцией его обработки. Большую часть работы по передаче сообщений и вызову соответствующих функций обработки берут на себя внутренние процедуры Windows. Приложения для Windows можно разрабатывать различными способами. Первым, безусловно, является непосредственный вызов функций API. Для экономии времени можно использовать подход визуального программирования или генераторы приложений. Типичный генератор создает файлы исходного кода из выбранных команд меню, диалоговых окон, управляющих элементов и др. После того, как оболочка приложения таким образом сконструирована, необходимо заполнить её требуемым кодом. Однако всем генераторам присущи два недостатка. Во-первых, всё равно необходимо писать код для ядра программы, т.к. фактически с использованием генератора создаётся прототип, нечто вроде входного экрана. Во-вторых, автоматически сгенерированные программы не подлежат модернизации, т.е. приложение необходимо переписать заново при внесении изменений. Подход визуального программирования предлагает набор объектов, которые при их активизации интерпретируют некоторые инструкции и могут объединяться с другими объектами или запрограммированными операциями. Однако подобные методы не позволяют создавать эффективно работающих программ, и приложение содержит большое количество избыточного кода. Поэтому системы визуального программирования используются для создания панелей ввода баз данных, календарей, небольших утилит и т.д. Более приемлемым подходом является использования языка C/C++ и библиотеки стандартных классов. Во-первых, классы С++ лучше позволяют моделировать архитектуру системы. Класс окна, например, инкапсулирует данные и функции, т.е. автоматически берет на себя выполнение некоторых внутренних требований, которые необходимо было ранее обеспечить в явном виде. Кроме того, функции-элементы класса могут, при помощи таблицы отклика на сообщение, независимо отвечать на сообщения системы. Это очень важно для управляемой событиями ОС. Во-вторых, библиотека классов представляет каркас прикладных программ, на основе которых можно создавать приложения. Опираясь на механизм наследования, можно создавать новые классы, расширяя функциональные возможности базовых. К недостаткам подобного подхода можно отнести, прежде всего, наличие избыточного кода при статической компоновке приложения или требование наличия в системе соответствующих библиотек при динамической компоновке, хотя визуально код приложения, составленный с помощью библиотеки классов, выглядит существенно более компактным. К данному методу необходимо отнести следующие библиотеки.

WinUser содержит функции, предоставляющие интерфейс доступа к Windows API.

OWL (Object Windows Library) предоставляет каркас прикладных программ, на основе которых строятся приложения (механизм наследования).

MFC (Microsoft Foundation Classes) – библиотека базовых классов, предусматривает использование классов-оболочек, заменяющих функции Windows.

## 2.2. Краткая характеристика функций API

API - это программный интерфейс приложения. Другими словами, это те возможности, которые предоставляет операционная система Windows для использования прикладными программами. Системные функции, которые предоставляет Windows программисту, называются ещё функциями API. Программирование с использованием только этих функций называется API программированием.

Классическая структура API-программы определяется четырьмя компонентами: инициализация; цикл ожидания, или цикл обработки сообщений; функция главного окна; другие функции. В простейшем случае последний компонент может отсутствовать. Два первых компонента располагаются в функции WinMain [2].

Функция WinMain:

int \_\_clrcall WinMain(

[in] HINSTANCE hInstance,

[in] HINSTANCE hPrevInstance,

[in] LPSTR lpCmdLine,

[in] int nShowCmd

);

Функция WinMain вызывается системой, в которую передаются четыре параметра:

1. hInstance - дескриптор текушего экземпляра приложения;

2. hPrevInctance - дескриптор предыдущего экземпляра приложения. Этот параметр всегда имеет значение NULL;

3. lpCmdLine - командная строка для приложения, за исключением имени программы;

4. nCmdShow - управляет способом отображения окна.

\_\_clrcall - указывает, что функцию можно вызвать только из управляемого кода.

Инициализация: здесь производится регистрация класса окна, его создание и вывод на экран. Регистрация класса окна осуществляется функцией ATOM RegisterClass(const WNDCLASS \*lpWndClass).

Единственный параметр функции - указатель на структуру WNDCLASS. После того как класс будет зарегистрирован, окно из данного класса может быть создано функцией CreateWindow.

typedef struct WNDCLASS{

UNIT style;

WNDPROC lpfnWndProc;

int cbClsExtra;

int cbWndExtra;

HINSTANCE hInstance;

HICON hIcon;

HCURSOR hCursor;

HBRUSH hbrBackground;

LPCTSTR lpszMenuName;

LPCTSTR lpszClassName;

} WNDClASS

Перечислим некоторые типичные значения членов структуры:

1. Стили класса окон. Стиль окна определяется комбинацией нескольких предопределённых констант. Довольно часто он полагается нулю, что означает "стиль по умолчанию".

2. Дескриптор иконки окна. Определяется с помощью функции LoadIcon. Первым параметром данной функции является дескриптор приложения, вторым - строка, определяющая имя иконки в ресурсах.

3. Дескриптор курсора. Для определения курсора используется API-функция LoadCursor. Функция похожа на функцию LoadIcon.

4. Имя класса. Название класса - это просто строка, которая потом используется при создании окна.

Создаётся окно функцией CreateWindow [2]:

HWND CreateWindow(

[in, optional] lpClassName, //указывает на имя класса окна

[in, optional] lpWindowName, //имя окна

[in] dwStyle, //стиль создаваемого окна

[in] x, //начальное горизонтальное положение окна

[in] y, //начальное вертикальное положение окна

[in] nWidth, //ширина окна в единицах устройства

[in] nHeight, //высота окна в единицах устройства

[in, optional] hWndParent, //дескриптор для родительского окна

[in, optional] hMenu, //дескриптор меню

[in, optional] hInstance, //дескриптор экземпляра модуля, связанного с окном

[in, optional] lpParam //указатель на значение, передаваемое в окно через структуру CREATESTRUCT

);

Функция возвращает дескриптор созданного окна, при ошибке – 0. Для того чтобы корректно отобразить окно на экране, следует выполнить ещё две функции:

BOOL ShowWindow(HWND hWnd, int nCmdShow) - эта функция отображает окно на экране. Первый параметр - дескриптор окна, второй - режим отображения.

BOOL UpdateWindow(HWND hWnd) - вызов данной функции приводит к немедленной перерисовке окна и посылке функции окна сообщения WM\_PAINT.

Цикл обработки сообщений присутствует во всех приложениях Windows. Правда, не всегда этот цикл представлен явно в программе.

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

В цикле сообщения присутствует три функции. Эти функции есть там всегда, но кроме них в цикле могут быть и другие. Функция GetMessage() выбирает из очереди сообщений приложения очередное приложение.

Во всех трех функциях присутствует указатель на строку MSG:

typedef struct tagMSG[3]{

HWND hwnd; //дескриптор окна

UINT message; //код сообщения

WPARAM wParam; //дополнительный параметр

LPARAM lParam; //дополнительный параметр

DWORD time; //время посылки сообщения

POINT pt; //положение курсора мыши

}

Функция TransleteMessage() преобразует сообщения в единый формат.

Функция DispatchMessage() перенаправляет сообщение оконной процедуре.

Оконная функция – это еще один компонент приложения, отвечающий за обработку сообщений окна. Эта функция вызывается системой и имеет четыре параметра, совпадающих с первыми членами структуры MSG.

Основное назначение оконной функции – обработка сообщений Windows [1]. Каждое приложение получает много сообщений. Их источник может быть разным. Обработка этих сообщений происходит именно в оконной функции. Это означает, что для каждого сообщения необходимо написать свой обработчик.

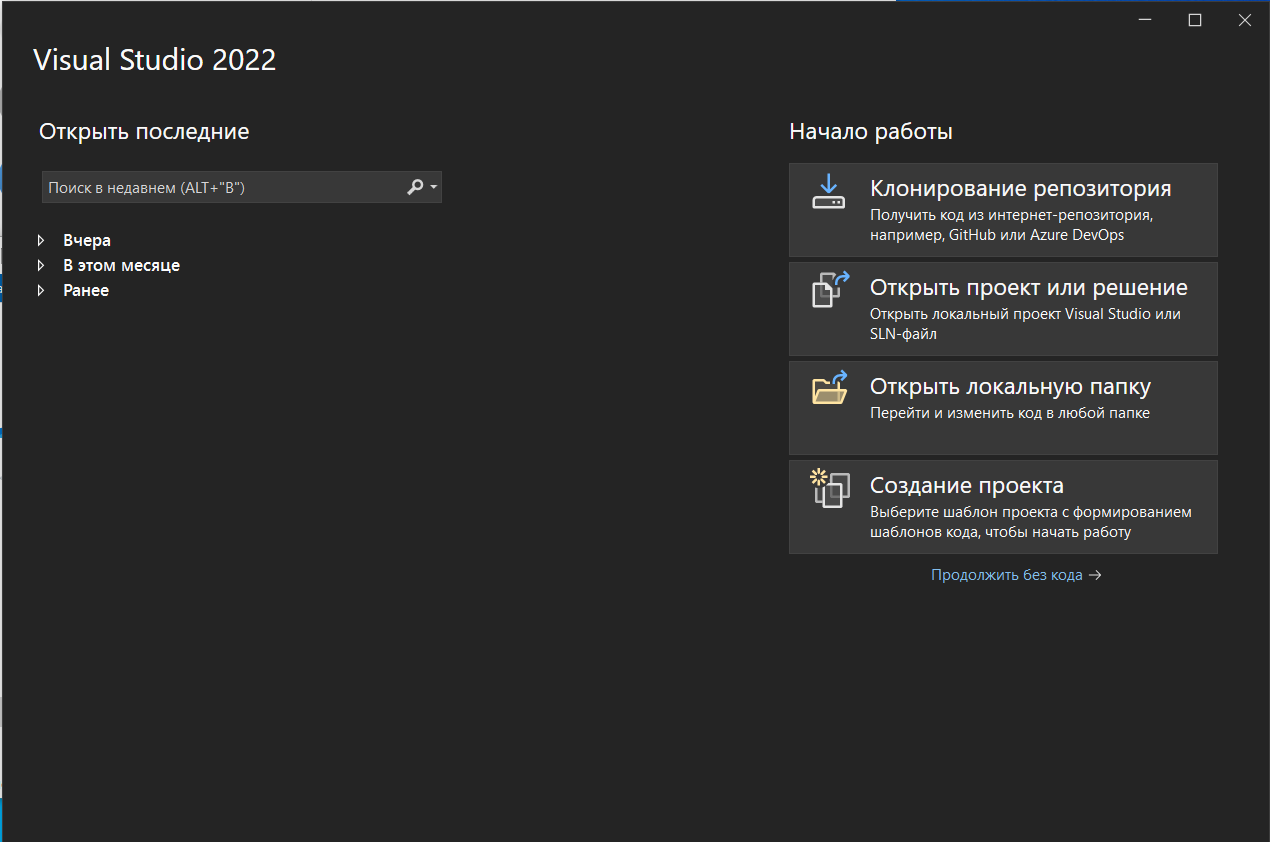
## 2.3. Краткая характеристика интегрированной среды разработки

Наиболее распространенный на данный момент пакет для разработки приложений для Windows является Visual Studio — это полный набор инструментов и служб для создания различных приложений как для платформы Microsoft, так и для других платформ.

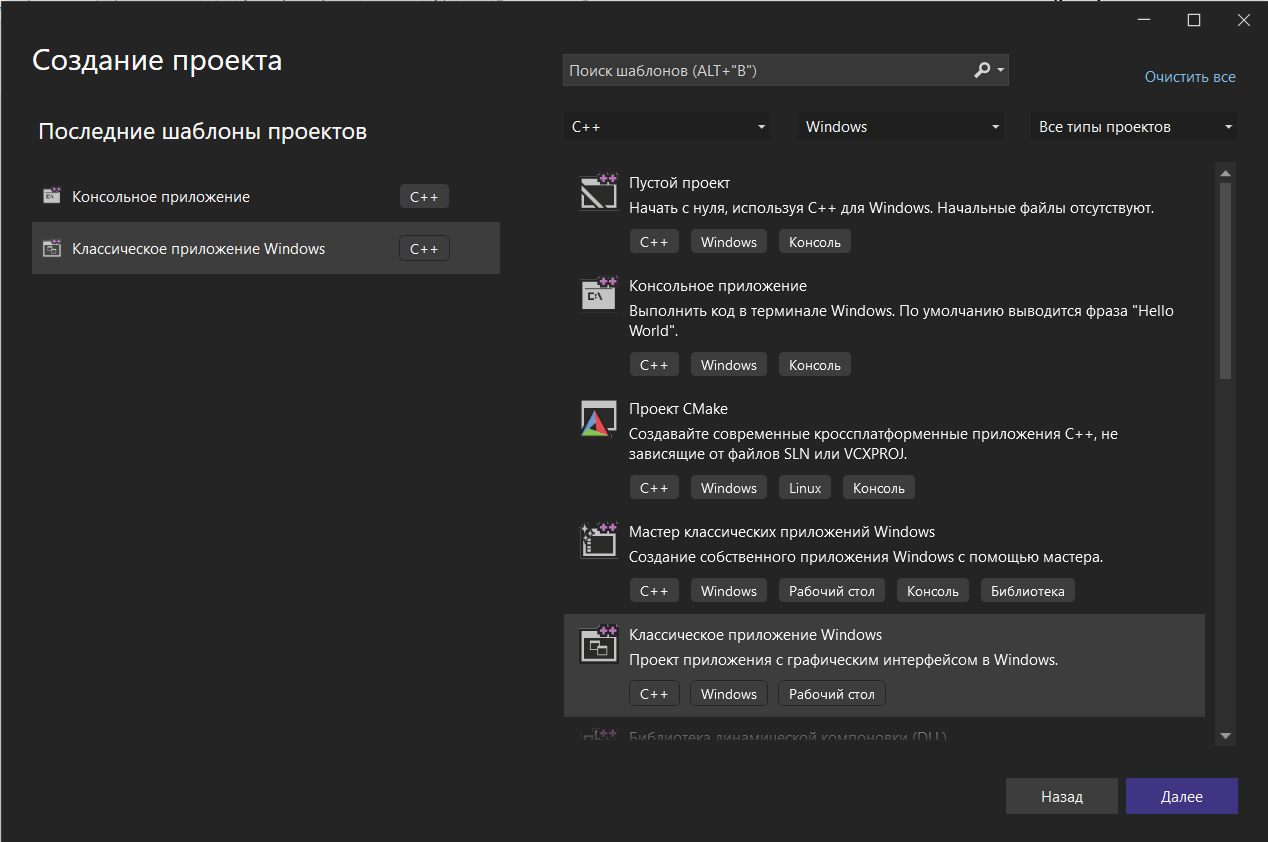
Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Microsoft Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметноориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Внешний вид среды разработки Visual Studio представлен на рисунке 2.1.

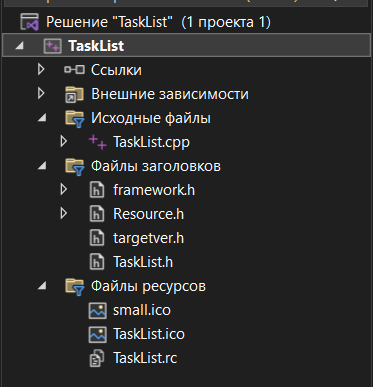
  
Рисунок 2.1. Внешний вид Microsoft Visual Studio 2022

Для реализации проекта был выбран шаблон «Классическое приложение Windows». Выбор шаблона показан на рисунке 2.2.

   
Рисунок 2.2. Выбор шаблона проекта

# 3. Состав и характеристики файлов проекта

Структура проекта, взятая из обозревателя решений Microsoft Visual Studio, представлена на рисунке 3.1.

  
Рисунок 3.1. Структура проекта в обозревателе решений

Характеристика файлов проекта представлена в таблице 3.1.

Таблица 3.1. Характеристика файлов проекта

|  |  |
| --- | --- |
| **Название файла** | **Назначение файла** |
| TaskList.cpp | Исходный код программы, написанный на языке C++, с использованием API-функций |
| Resourse.h | Включаемый файл заголовков, используемый TaskList.rc |
| TaskList.rc | Файл с кодом для загрузки ресурсов |
| TaskList.vcxproj | Основной файл проекта VC++, автоматически создаваемый с помощью мастера приложений. В файле представлены сведения о версии Visual C++, используемой при создании файла, а также о параметрах платформы, конфигурации и проекте, заданных с помощью мастера приложений |

# 4. Стандартные классы и функции приложения

Перечень стандартных классов и функций представлен в таблице 4.1.

Таблица 4.1. Перечень стандартных классов и функций

|  |  |
| --- | --- |
| **Имя** | **Назначение** |
| EndDialog | Функция, которая уничтожает модальное диалоговое окно, в результате чего система завершит обработку диалогового окна. |
| WNDCLASSEX | Структура, которая содержит сведения о классе окна. |
| PAINTSTRUCT | структура, используемая функцией BeginPaint для хранения информации о рисовании. |
| GetStockObject | Функция, которая получает дескриптор для одного из складских перьев, кистей, шрифтов или палитр. |
| LoadCursor | Функция, которая загружает указанный курсор. |
| LoadIcon | Функция, которая загружает указанную иконку. |
| RegisterClassEx | Функция, которая регистрирует класс окна |
| CreateWindow | Функция, которая создает перекрывающееся всплывающее окно или дочернее окно. |
| GetMessage | Функция, которая извлекает сообщение из очереди сообщений вызывающего потока. |
| TranslateMessage | Функция, которая преобразует сообщения виртуального ключа в символьные сообщения. |
| DispatchMessage | Функция, которая отправляет сообщение в оконную процедуру. |
| SetWindowText | Функция, которая изменяет текст строки заголовка указанного окна. |
| DestroyWindow | Функция, которая уничтожает указанное окно. |
| MSG | Структура, которая содержит информацию сообщения из очереди сообщений потока. |
| wWinMain | Главная функция. |
|  |  |
|  | Продолжение таблицы 4.1 |
| UpdateWindow | Функция, которая обновляет клиентскую область окна. |
| MessageBox | Функция, которая отображает сообщение в окне сообщения. |
| BeginPaint | Функция, которая подготавливает окно к рисованию. |
| CreateSolidBrush | Функция, которая создает кисть с заданным цветом. |
| EndPaint | Функция, которая завершает рисование. |
| GetWindowTextLength | Функция, которая возвращает длину текста в окне. |
| GetWindowText | Функция, которая получает текст из окна. |
| GetWindow | Функция, которая получает дескриптор окна, имеющего определенное отношение к указанному окну. |
| RedrawWindow | Функция, которая обновляет (перерисовывает) клиентскую область окна. |
| SetWindowPos | Функция, которая изменяет размер, позицию и Z-порядок указанного окна. |
| DefWindowProc | Функция, которая обработчик сообщений по умолчанию. |
| DestroyWindow | Функция, которая уничтожает указанное окно. |
| DispatchMessage | Функция, которая отправляет сообщение окну. |
| ShowWindow | Функция, которая показывает или скрывает окно. |
| InvalidateRect | Функция, которая добавляет область окна в область обновления. |
| PostQuitMessage | Функция, которая указывает системе, что поток отправил запрос на завершение (quit). |

# 5. Пользовательские классы и функции приложения

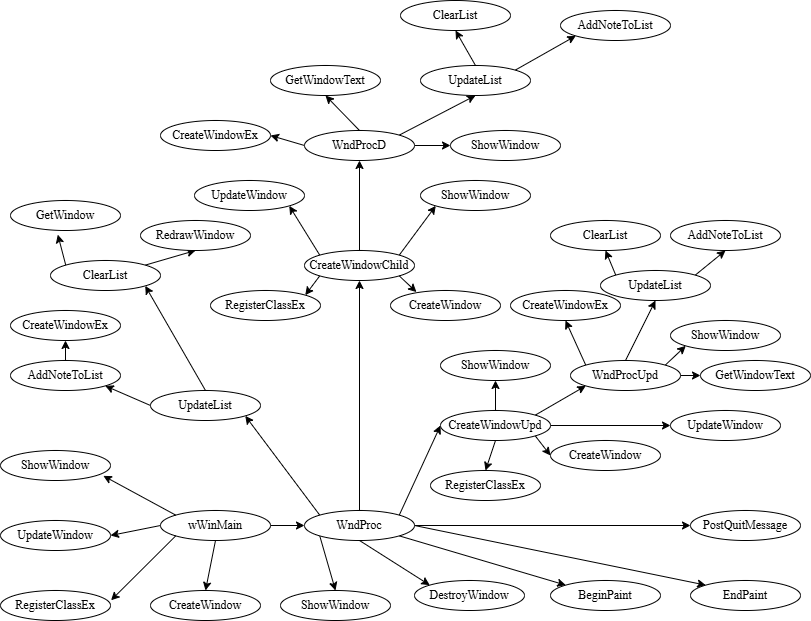
Перечень пользовательских классов и функций представлен в таблице 5.1.

Таблица 5.1. Перечень пользовательских классов и функций

|  |  |
| --- | --- |
| **Имя** | **Назначение** |
| Note | Структура для сохранения данных заметок. |
| AddNoteToList | Функция добавления заметки в родительское окно. |
| ClearList | Функция очистки родительского окна. |
| UpdateList | Функция для прорисовки выбранной страницы. |
| About | Функция обработки сообщения about. |
| WndProc | Функция обработки сообщений родительского окна. |
| WndProcD | Функция обработки дочернего окна для создания заметок. |
| WndProcUpd | Функция обработки дочернего окна для изменений и удалений. |

# 6. Структура программы

Структура программы представлена на рисунке 6.1.

  
Рисунок 6.1. Структура программы

# 7. Системные требования

Операционная система: не ниже Windows NT.

Свободное пространство на диске: не менее 250 Кб.

Оперативная память: не менее 2 Мб.

Необходимые библиотеки dll: • VCRUNTIME140D.dll;

• MSVCP140D.dll;

• VCRUNTIME140\_1D.dll;

• ucrtbased.dll.

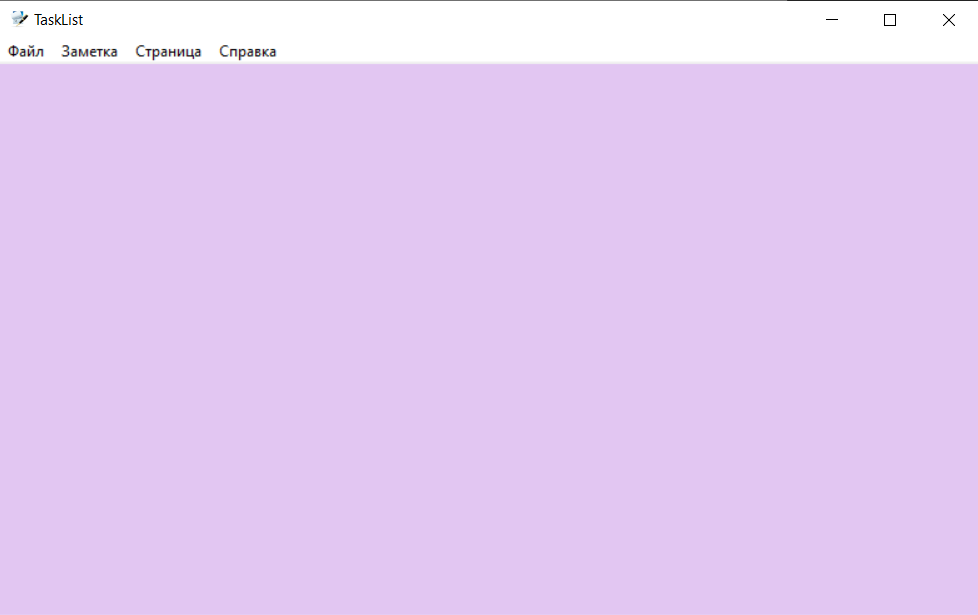
# 8. Руководство пользователя

## 8.1. Правила установки и запуска программы

Для установки программы необходимо сохранить файл TaskList.exe. Далее нужно запустить файл TaskList.exe, щелкнув дважды по иконке приложения левой кнопкой мыши. Запуск программы можно также осуществить из командной строки, указав в качестве параметра путь к открываемому файлу.

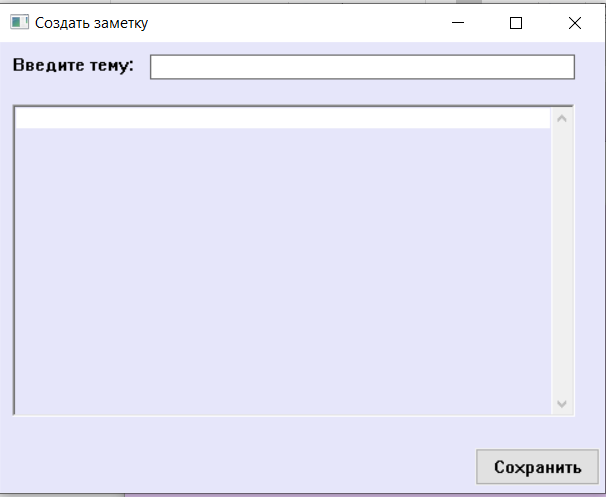
## 8.2. Последовательность действий для запуска программы

Для открытия необходимо запустить приложение TaskList.exe (см. рис.8.1).

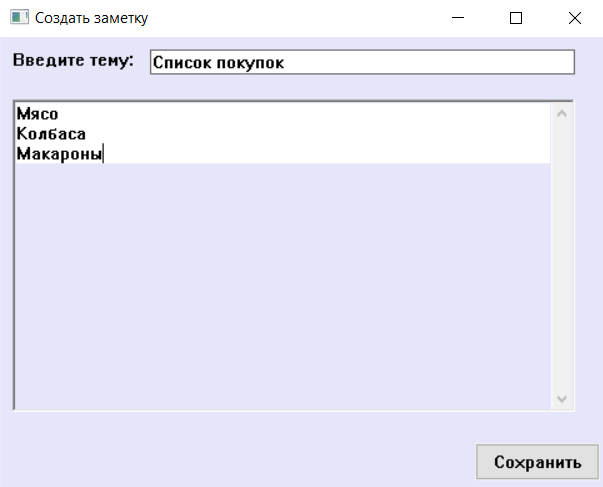
   
Рисунок 8.1. Вид окна при запуске приложения.

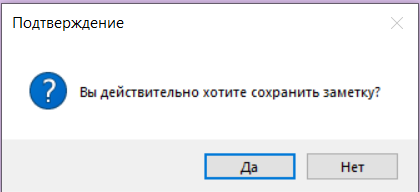
## 8.3. Последовательность действий для выполнения всех требуемых по техническому заданию функций

При выборе в меню пункта «Заметка» и нажатии в подменю кнопки «Создать заметку» появляется окно для создания заметки (см. рис.8.2).

   
Рисунок 8.2. Окно создания заметки.

Вводим текст заметки и ее тему (см. рис.8.3). При нажатии кнопки «Сохранить» выводится диалоговое окно для подтверждения сохранения заметки. После нажатия на кнопку «Да», текст из окна для ввода и дата сохранения ее отображается на главном экране (см. рис.8.4). После нажатия на кнопку «Нет», возвращаемся к редактированию заметки (см. рис.8.5).

   
Рисунок 8.3. Ввод текста в поле для ввода.

  
Рисунок 8.4. Уведомление о сохранении заметки.

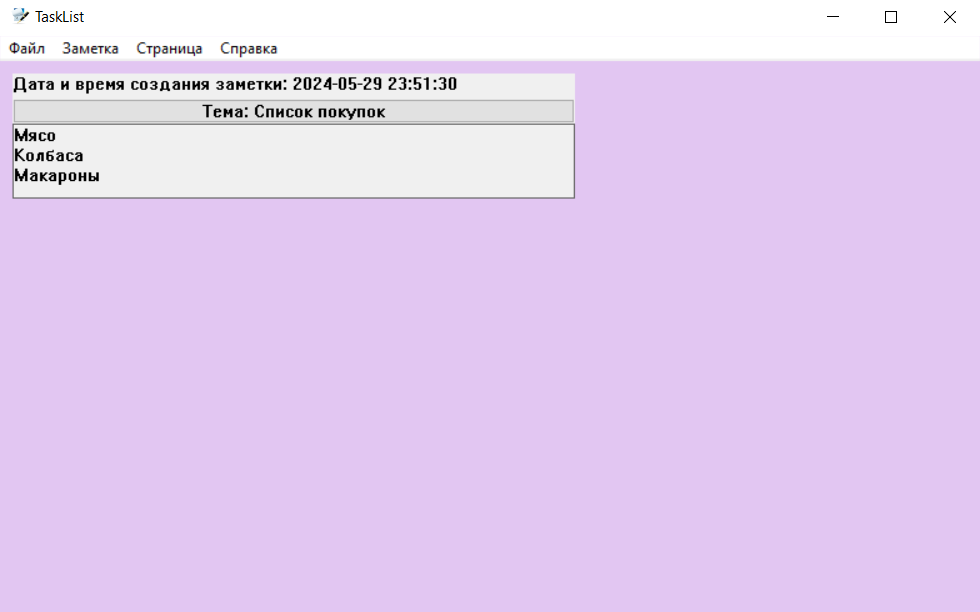


Рисунок 8.5. Отображение заметки на главном экране.

Если текст заметки не введен и нажать на кнопку «Сохранить», выводится уведомление с просьбой ввести его (см. рис. 8.6).

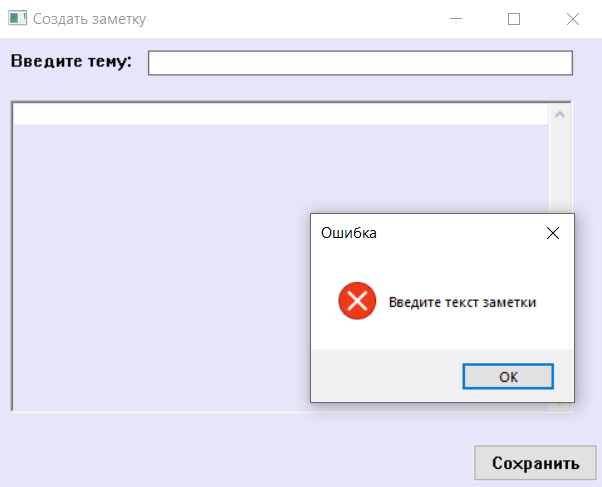
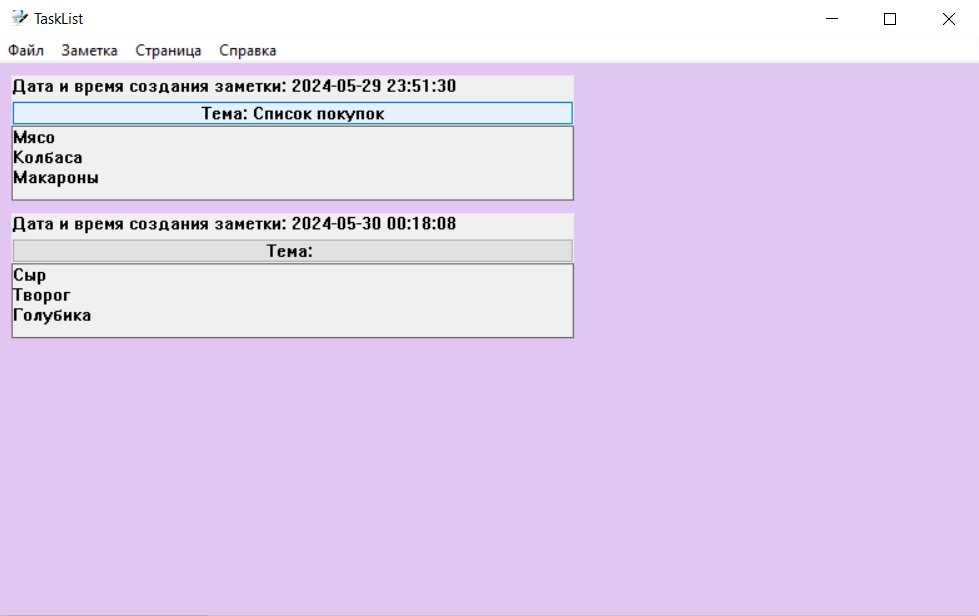


Рисунок 8.6. Уведомление с просьбой ввести текс заметки.

Чтобы отредактировать или удалить заметку, нажмите на кнопку, соответствующую теме нужной заметки (см. рис. 8.7). После этого откроется окно редактирования, в котором можно изменить как текст заметки, так и её тему (см. рис. 8.8). При нажатии кнопки «Сохранить» выводится диалоговое окно для подтверждения сохранения заметки (см. рис. 8.9). После нажатия кнопки "Да", изменения в тексте и дата изменений будут отображены на главном экране. После нажатия на кнопку «Нет», возвращаемся к редактированию заметки. При нажатии кнопки «Удалить» выводится диалоговое окно для подтверждения удаления заметки (см. рис. 8.10). После нажатия кнопки "Да", данная заметка будет удалена (см. рис. 8.11). После нажатия на кнопку «Нет», возвращаемся к редактированию заметки.

  
Рисунок 8.7. Открытие заметки для редактирования.

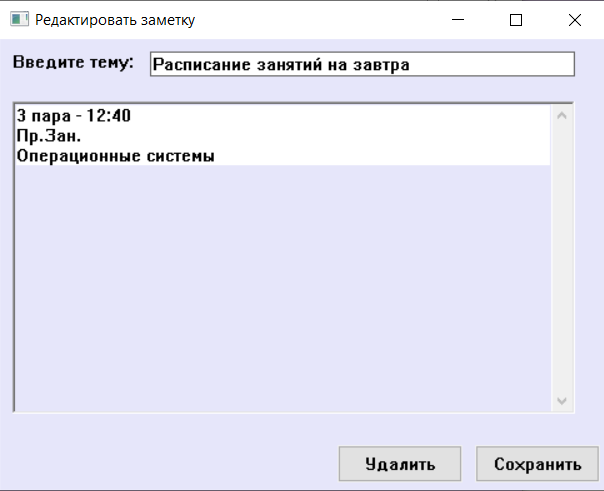


Рисунок 8.8. Редактирование заметки.

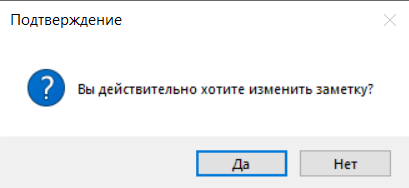


Рисунок 8.9. Подтверждение редактирования заметки.

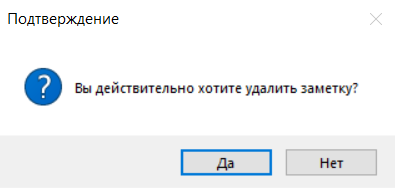


Рисунок 8.10. Подтверждение удаления заметок.

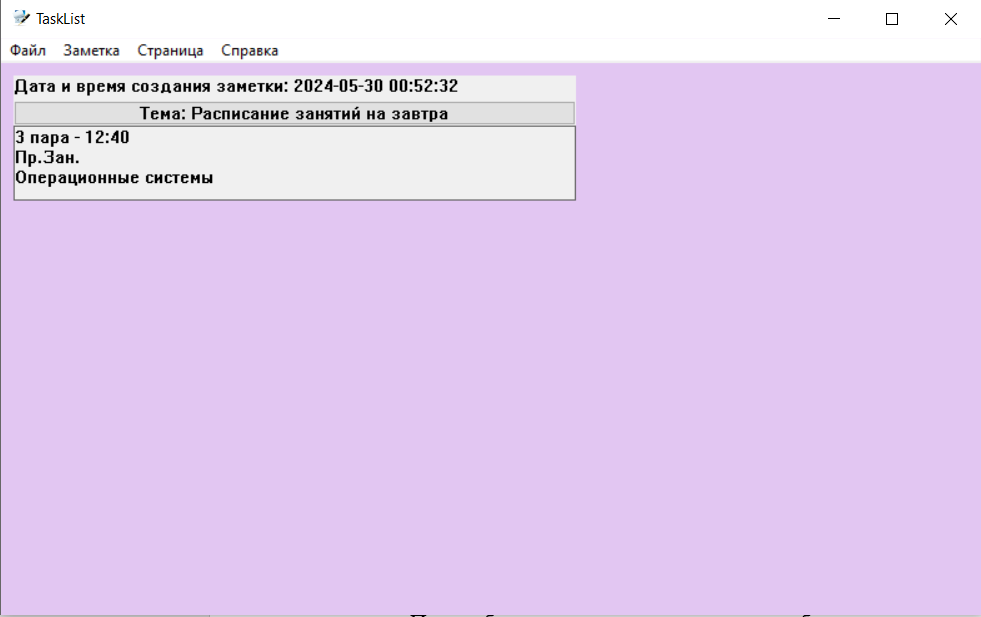


Рисунок 8.11. Главный экран после удаления заметки.

Если создано больше 4 заметок, то новые сохраняются на новой странице. Для перехода между страницами в пункте меню «Страница» необходимо нажать на кнопку «Следующая страница» для перехода на следующую страницу, либо на кнопку «Предыдущая страница» для перехода на предыдущую страницу (см. рис. 8.12).

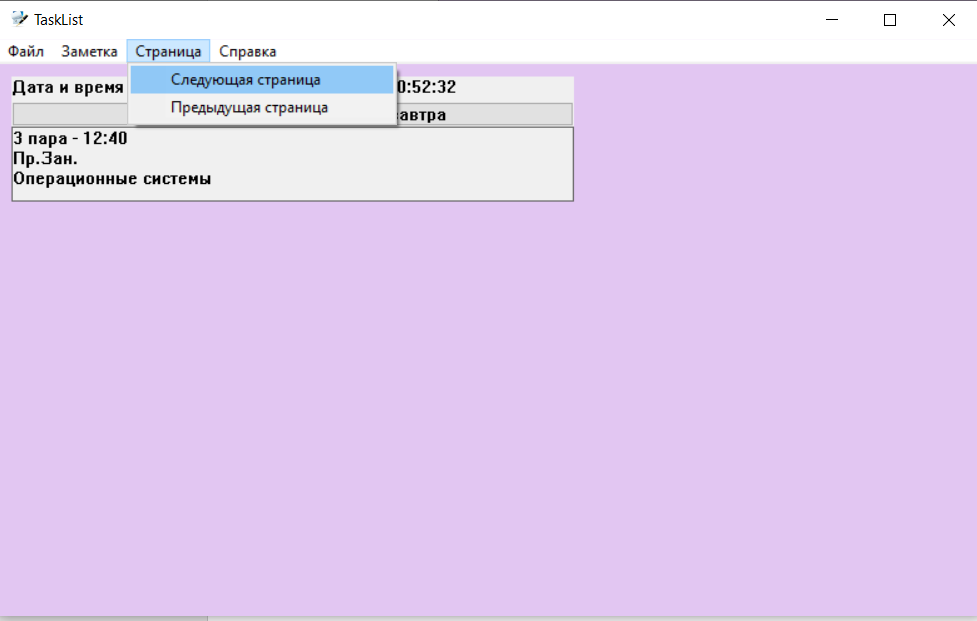
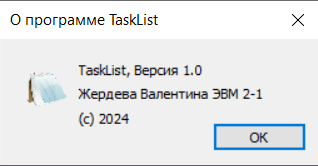


Рисунок 8.12. Кнопки для смены страниц.

При нажатии на кнопку «О программе» в пункте меню «Справка» выводится информация о программе (см. рис.8.13).

   
Рисунок 8.13. Вывод информации о программе

При нажатии на кнопку «Выход» в пункте меню «Файл» приложение закрывается.

# 9. Список литературы

1. Черкасова Н.И. «Пособие к выполнению курсовой работы по дисциплине «Операционные системы» для студентов направления 09.03.01 очной формы обучения», Москва ИД Академии Жуковского, 2020

2. Интернет ресурс: <https://learn.microsoft.com/ru-ru/windows/win32/api/>

3. Интернет ресурс: <http://www.vsokovikov.narod.ru/New_MSDN_API/Controls/fn_initcommoncontrolsex.htm>

# 10. Приложение

## 10.1. Листинги программ

*TaskList.cpp*

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#define WM\_USER\_UPDATE\_NOTE (WM\_USER + 1) // Определение пользовательского сообщения для обновления заметки

#pragma comment(linker,"\"/manifestdependency:type='win32' \

name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \

processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#pragma comment(lib,"Comctl32.lib")

#include "resource.h"

#include <windowsx.h>

#include <windows.h>

#include <commctrl.h>

#include <wchar.h>

#include <iostream>

#include <SDKDDKVer.h>

#include <stdlib.h>

#include <malloc.h>

#include <memory.h>

#include <tchar.h>

#include <vector>

#include <string>

#include <algorithm>

#include <ctime> // Для работы с датой и временем

HINSTANCE hInst;

HBRUSH hBrushEditTopic;

HWND hClild;

HWND hUpd;

HWND hParent;

HWND hEdit;

HWND hEditTopic;

HWND hEditUpd;

HWND hEditTopicUpd;

HWND hNoteText;

struct Note {

std::wstring time;

std::wstring topic;

std::wstring text;

};

std::vector<Note> notes;

std::vector<Note> noteslist;

int verticalPosition = 10; // Начальная вертикальная позиция для первой заметки

int editNoteIndex = -1; // Индекс редактируемой заметки (-1, если редактируется новая заметка)

int id\_upd;

int nowlist = 1;

void AddNoteToList(const Note& newNote, int number, int id\_menu)

{

std::wstring wTimeString = L"Дата и время создания заметки: ";

wTimeString += newNote.time;

verticalPosition = 10 + number \* 110;

HWND hNoteTime = CreateWindowEx(0, L"STATIC", wTimeString.c\_str(),

WS\_CHILD | WS\_VISIBLE | SS\_LEFT, 10, verticalPosition, 450, 20, hParent, NULL, hInst, NULL);

std::wstring noteTitle(L"Тема: ");

noteTitle += newNote.topic;

HWND hNoteTopic = CreateWindowEx(0, L"BUTTON", noteTitle.c\_str(),

WS\_CHILD | WS\_VISIBLE | SS\_LEFT, 10, verticalPosition + 20, 450, 20, hParent, (HMENU)(1000 + id\_menu), hInst, NULL);

HWND hNoteText = CreateWindowEx(0, L"STATIC", newNote.text.c\_str(),

WS\_CHILD | WS\_VISIBLE | SS\_LEFT | WS\_BORDER, 10, verticalPosition + 40, 450, 60, hParent, (HMENU)(2000 + id\_menu), hInst, NULL);

}

void ClearList()

{

HWND hwndChild = GetWindow(hParent, GW\_CHILD);

while (hwndChild)

{

HWND hwndNext = GetWindow(hwndChild, GW\_HWNDNEXT);

DestroyWindow(hwndChild);

hwndChild = hwndNext;

}

// Затем отправляем сообщение о том, что окно нужно перерисовать

RedrawWindow(hParent, NULL, NULL, RDW\_ERASE | RDW\_FRAME | RDW\_INVALIDATE | RDW\_ALLCHILDREN);

}

void UpdateList()

{

ClearList();

int number = 0;

for (int i = nowlist \* 4 - 4; i < notes.size() && i < nowlist \* 4; i++) {

AddNoteToList(notes[i], number, i);

number++;

}

}

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return TRUE;

}

break;

}

return FALSE;

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK WndProcD(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK WndProcUpd(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int CALLBACK wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR szCmdLine, int nCmdShow)

{

MSG msg{};

HWND hwnd{};

WNDCLASSEX wc{};

wc.hInstance = hInstance;

wc.lpszClassName = L"Родитель";

wc.lpfnWndProc = WndProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = (HBRUSH)CreateSolidBrush(RGB(226, 198, 242));

wc.hCursor = LoadCursor(hInstance, IDC\_ARROW);

wc.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_TASKLIST));

wc.hIconSm = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_SMALL));

wc.lpszMenuName = MAKEINTRESOURCE(IDC\_TASKLIST);

wc.style = CS\_VREDRAW | CS\_HREDRAW;

wc.cbSize = sizeof(WNDCLASSEX);

hInst = hInstance;

if (!RegisterClassEx(&wc)) return EXIT\_FAILURE;

hwnd = CreateWindow(wc.lpszClassName, L"TaskList", WS\_OVERLAPPEDWINDOW | WS\_SYSMENU , 200, 210, 800, 500, nullptr, nullptr, wc.hInstance, nullptr);

hParent = hwnd;

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

while (GetMessage(&msg, nullptr, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)(msg.wParam);

}

INT CALLBACK CreateWindowChild(HWND hwnd) {

WNDCLASSEX wcChild{};

wcChild.cbSize = sizeof(WNDCLASSEX);

wcChild.style = CS\_HREDRAW | CS\_VREDRAW;

wcChild.lpfnWndProc = WndProcD;

wcChild.cbClsExtra = 0;

wcChild.cbWndExtra = 0;

wcChild.hInstance = hInst;

wcChild.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcChild.hbrBackground = (HBRUSH)CreateSolidBrush(RGB(230, 230, 250));

wcChild.lpszClassName = L"NoteChildClass";

if (!RegisterClassEx(&wcChild))

return EXIT\_FAILURE;

HWND hChildWnd = CreateWindow(wcChild.lpszClassName, L"Создать заметку", WS\_OVERLAPPEDWINDOW | WS\_SYSMENU, 100, 100, 500, 400, hwnd, nullptr, nullptr, nullptr);

hClild = hChildWnd;

UpdateWindow(hChildWnd);

}

INT CALLBACK CreateWindowUpd(HWND hwnd) {

WNDCLASSEX wcChild{};

wcChild.cbSize = sizeof(WNDCLASSEX);

wcChild.style = CS\_HREDRAW | CS\_VREDRAW;

wcChild.lpfnWndProc = WndProcUpd;

wcChild.cbClsExtra = 0;

wcChild.cbWndExtra = 0;

wcChild.hInstance = hInst;

wcChild.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcChild.hbrBackground = (HBRUSH)CreateSolidBrush(RGB(230, 230, 250));

wcChild.lpszClassName = L"NoteUpdateClass";

if (!RegisterClassEx(&wcChild))

return EXIT\_FAILURE;

HWND hChildWnd = CreateWindow(wcChild.lpszClassName, L"Редактировать заметку", WS\_OVERLAPPEDWINDOW | WS\_SYSMENU, 100, 100, 500, 400, hwnd, nullptr, nullptr, nullptr);

hUpd = hChildWnd;

UpdateWindow(hChildWnd);

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) //обработка сообщений главного окна

{

switch (uMsg)

{

case WM\_CREATE:

{

CreateWindowChild(hwnd);

CreateWindowUpd(hwnd);

InvalidateRect(hwnd, NULL, TRUE);

return 0;

}

return 0;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDM\_EXIT:

{

DestroyWindow(hwnd);

}

break;

case IDM\_ABOUT:

{

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hwnd, About);

}

break;

case IDM\_CREATE:

{

ShowWindow(hClild, SW\_RESTORE);

}

break;

case IDM\_NEXT:

{

if (notes.size() > nowlist \* 4) {

nowlist++;

UpdateList();

}

else MessageBox(hwnd, L"Страница не существует!", L"Предупреждение", MB\_ICONWARNING | MB\_OK);

}

break;

case IDM\_LAST:

{

if (nowlist == 1) MessageBox(hwnd, L"Страница не существует!", L"Предупреждение", MB\_ICONWARNING | MB\_OK);

else {

nowlist--;

UpdateList();

}

}

break;

default:

// Обработка выбора заметки для редактирования

if (wmId >= 1000 && wmId < 2000)

{

editNoteIndex = wmId - 1000;

id\_upd = editNoteIndex;

const Note& noteToEdit = notes[editNoteIndex];

SetWindowText(hEditTopicUpd, noteToEdit.topic.c\_str());

SetWindowText(hEditUpd, noteToEdit.text.c\_str());

ShowWindow(hUpd, SW\_RESTORE);

std::cout << "123";

}

break;

}

}

return 0;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

EndPaint(hwnd, &ps);

}

return 0;

case WM\_DESTROY:

{

PostQuitMessage(EXIT\_SUCCESS);

}

return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

LRESULT CALLBACK WndProcD(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) //обработка сообщений главного окна

{

switch (uMsg)

{

case WM\_CREATE:

{

// Добавляем текст "Введите тему:"

CreateWindowEx(0, L"STATIC", L"Введите тему:",

WS\_CHILD | WS\_VISIBLE | SS\_LEFT, 10, 10, 100, 20, hwnd, NULL, hInst, NULL);

// Добавляем текстовое поле для ввода темы

hEditTopic = CreateWindowEx(0, L"EDIT", L"",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT, 120, 10, 340, 20, hwnd, (HMENU)IDC\_EDIT\_TOPIC, hInst, NULL);

// Устанавливаем обработчик сообщения WM\_CTLCOLOREDIT для изменения цвета фона текстового поля

// Добавляем текстовое поле для ввода заметки

hEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"",

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL,

10, 50, 450, 250, hwnd, (HMENU)IDC\_EDIT\_CHILD, hInst, NULL);

// Устанавливаем обработчик сообщения WM\_CTLCOLOREDIT для изменения цвета фона текстового поля

// Добавляем кнопку "Сохранить"

HWND hButton = CreateWindowEx(0, L"BUTTON", L"Сохранить",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 380, 325, 100, 30, hwnd, (HMENU)IDM\_SAVE, hInst, NULL);

ShowWindow(hwnd, SW\_HIDE);

}

return 0;

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case IDM\_EXIT:

{

ShowWindow(hwnd, SW\_HIDE);

ShowWindow(hParent, SW\_RESTORE);

}

break;

case IDM\_ABOUT:

{

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hwnd, About);

}

break;

case IDM\_SAVE:

{

// Добавляем сохраненную заметку в вектор

std::wstring titleText, noteText;

int lenTopic = GetWindowTextLength(hEditTopic);

int lenText = GetWindowTextLength(hEdit);

int len = GetWindowTextLength(hEdit);

if (len > 0) {

wchar\_t\* bufferTopic = new wchar\_t[lenTopic + 1];

wchar\_t\* bufferText = new wchar\_t[lenText + 1];

GetWindowText(hEditTopic, bufferTopic, lenTopic + 1);

GetWindowText(hEdit, bufferText, lenText + 1);

wchar\_t\* buffer = new wchar\_t[len + 1];

GetWindowText(hEdit, buffer, len + 1);

int result = MessageBox(hwnd, L"Вы действительно хотите сохранить заметку?", L"Подтверждение", MB\_YESNO | MB\_ICONQUESTION);

if (result == IDYES) {

Note newNote;

time\_t currentTime = time(nullptr);

tm\* localTime = localtime(&currentTime);

char timeStr[64];

strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", localTime);

std::string timeString(timeStr);

newNote.time = std::wstring(timeString.begin(), timeString.end());

newNote.topic = bufferTopic;

newNote.text = bufferText;

notes.push\_back(newNote);

UpdateList();

ShowWindow(hClild, SW\_HIDE); // Закрываем дочернее окно после сохранения

SetWindowPos(hParent, HWND\_TOP, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

SetWindowText(hEdit, L"");

SetWindowText(hEditTopic, L"");

}

else {

delete[] buffer;

}

}

else {

MessageBox(hwnd, L"Введите текст заметки", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

break;

}

}

return 0;

case WM\_CTLCOLORSTATIC:

{

HDC hdcStatic = (HDC)wParam;

SetBkColor(hdcStatic, RGB(230, 230, 250));

return (LRESULT)GetStockObject(NULL\_BRUSH);

}

case WM\_CTLCOLOREDIT:

{

HDC hdcEdit = (HDC)wParam;

SetBkColor(hdcEdit, RGB(255, 255, 255));

SetTextColor(hdcEdit, RGB(0, 0, 0));

return (INT\_PTR)hBrushEditTopic;

}

case WM\_CLOSE:

{

ShowWindow(hwnd, SW\_HIDE);

SetWindowPos(hParent, HWND\_TOP, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

}

return 0;

case WM\_DESTROY:

DeleteObject(hBrushEditTopic);

return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

LRESULT CALLBACK WndProcUpd(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) //обработка сообщений окна delete

{

switch (uMsg)

{

case WM\_CREATE:

{

// Добавляем текст "Введите тему:"

CreateWindowEx(0, L"STATIC", L"Введите тему:",

WS\_CHILD | WS\_VISIBLE | SS\_LEFT, 10, 10, 100, 20, hwnd, NULL, hInst, NULL);

// Добавляем текстовое поле для ввода темы

hEditTopicUpd = CreateWindowEx(0, L"EDIT", L"",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT, 120, 10, 340, 20, hwnd, (HMENU)IDC\_EDIT\_TOPIC\_UPD, hInst, NULL);

// Устанавливаем обработчик сообщения WM\_CTLCOLOREDIT для изменения цвета фона текстового поля

// Добавляем текстовое поле для ввода заметки

hEditUpd = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"",

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL,

10, 50, 450, 250, hwnd, (HMENU)IDC\_EDIT\_UPD, hInst, NULL);

// Устанавливаем обработчик сообщения WM\_CTLCOLOREDIT для изменения цвета фона текстового поля

// Добавляем кнопку "Сохранить"

HWND hButtonUpdate = CreateWindowEx(0, L"BUTTON", L"Сохранить",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 380, 325, 100, 30, hwnd, (HMENU)IDM\_UPDATE, hInst, NULL);

HWND hButtonDelete = CreateWindowEx(0, L"BUTTON", L"Удалить",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 270, 325, 100, 30, hwnd, (HMENU)IDM\_DELETE, hInst, NULL);

ShowWindow(hwnd, SW\_HIDE);

}

return 0;

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case IDM\_EXIT:

{

ShowWindow(hwnd, SW\_HIDE);

ShowWindow(hParent, SW\_RESTORE);

}

break;

case IDM\_ABOUT:

{

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hwnd, About);

}

break;

case IDM\_UPDATE:

{

std::wstring titleText, noteText;

int lenTopic = GetWindowTextLength(hEditTopicUpd);

int lenText = GetWindowTextLength(hEditUpd);

int len = GetWindowTextLength(hEditUpd);

if (len > 0) {

wchar\_t\* bufferTopic = new wchar\_t[lenTopic + 1];

wchar\_t\* bufferText = new wchar\_t[lenText + 1];

GetWindowText(hEditTopicUpd, bufferTopic, lenTopic + 1);

GetWindowText(hEditUpd, bufferText, lenText + 1);

wchar\_t\* buffer = new wchar\_t[len + 1];

GetWindowText(hEditUpd, buffer, len + 1);

int result = MessageBox(hwnd, L"Вы действительно хотите изменить заметку?", L"Подтверждение", MB\_YESNO | MB\_ICONQUESTION);

if (result == IDYES) {

Note newNote;

time\_t currentTime = time(nullptr);

tm\* localTime = localtime(&currentTime);

char timeStr[64];

strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", localTime);

std::string timeString(timeStr);

newNote.time = std::wstring(timeString.begin(), timeString.end());

newNote.topic = bufferTopic;

newNote.text = bufferText;

notes[id\_upd] = newNote;

UpdateList();

ShowWindow(hUpd, SW\_HIDE); // Закрываем дочернее окно после сохранения

SetWindowPos(hParent, HWND\_TOP, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

SetWindowText(hEditUpd, L"");

SetWindowText(hEditTopicUpd, L"");

}

else {

delete[] buffer;

}

}

else {

MessageBox(hwnd, L"Введите текст заметки", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

break;

case IDM\_DELETE:

{

int result = MessageBox(hwnd, L"Вы действительно хотите удалить заметку?", L"Подтверждение", MB\_YESNO | MB\_ICONQUESTION);

if (result == IDYES) {

notes.erase(notes.begin() + id\_upd);

if (notes.size() < nowlist \* 4 - 3) nowlist--;

UpdateList();

ShowWindow(hUpd, SW\_HIDE); // Закрываем дочернее окно

SetWindowPos(hParent, HWND\_TOP, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

SetWindowText(hEditUpd, L"");

SetWindowText(hEditTopicUpd, L"");

}

}

}

break;

}

return 0;

case WM\_CTLCOLORSTATIC:

{

HDC hdcStatic = (HDC)wParam;

SetBkColor(hdcStatic, RGB(230, 230, 250));

return (LRESULT)GetStockObject(NULL\_BRUSH);

}

case WM\_CTLCOLOREDIT:

{

HDC hdcEdit = (HDC)wParam;

SetBkColor(hdcEdit, RGB(255, 255, 255));

SetTextColor(hdcEdit, RGB(0, 0, 0));

return (INT\_PTR)hBrushEditTopic;

}

case WM\_CLOSE:

{

ShowWindow(hwnd, SW\_HIDE);

SetWindowPos(hParent, HWND\_TOP, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

}

return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

*TaskList.rc*

//Сценарий ресурсов, созданный в Microsoft Visual C++.

//

#include "resource.h"

#define APSTUDIO\_READONLY\_SYMBOLS

/////////////////////////////////////////////////////////////////////////////

//

// Создано из ресурса TEXTINCLUDE.

//

#ifndef APSTUDIO\_INVOKED

#include "targetver.h"

#endif

#define APSTUDIO\_HIDDEN\_SYMBOLS

#include "windows.h"

#undef APSTUDIO\_HIDDEN\_SYMBOLS

/////////////////////////////////////////////////////////////////////////////

#undef APSTUDIO\_READONLY\_SYMBOLS

#if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_RUS)

LANGUAGE 25, 1

/////////////////////////////////////////////////////////////////////////////

//

// Значок

//

// Значок с наименьшим значением идентификатора помещается первым, чтобы гарантировать использование

// идентичного значка приложения для всех систем.

IDI\_TASKLIST ICON "TaskList.ico"

IDI\_SMALL ICON "small.ico"

/////////////////////////////////////////////////////////////////////////////

//

// Меню

//

IDC\_TASKLIST MENU

BEGIN

POPUP "&Файл"

BEGIN

MENUITEM "Вы&ход", IDM\_EXIT

END

POPUP "&Заметка"

BEGIN

MENUITEM "&Создать заметку", IDM\_CREATE

END

POPUP "&Страница"

BEGIN

MENUITEM "&Следующая страница", IDM\_NEXT

MENUITEM "&Предыдущая страница", IDM\_LAST

END

POPUP "&Справка"

BEGIN

MENUITEM "&О программе...", IDM\_ABOUT

END

END

/////////////////////////////////////////////////////////////////////////////

//

// Ускоритель

//

IDC\_TASKLIST ACCELERATORS

BEGIN

"?", IDM\_ABOUT, ASCII, ALT

"/", IDM\_ABOUT, ASCII, ALT

END

/////////////////////////////////////////////////////////////////////////////

//

// Диалоговое окно

//

IDD\_ABOUTBOX DIALOGEX 0, 0, 170, 62

STYLE DS\_SETFONT | DS\_MODALFRAME | DS\_FIXEDSYS | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "О программе TaskList"

FONT 8, "MS Shell Dlg"

BEGIN

ICON IDI\_TASKLIST,IDC\_STATIC,14,14,21,20

LTEXT "TaskList, Версия 1.0",IDC\_STATIC,42,11,114,8,SS\_NOPREFIX

LTEXT "Жердева Валентина ЭВМ 2-1 ", IDC\_STATIC, 42, 23, 114, 8, SS\_NOPREFIX

LTEXT "(c) 2024",IDC\_STATIC,42,35,114,8

DEFPUSHBUTTON "ОК",IDOK,113,41,50,14,WS\_GROUP

END

/////////////////////////////////////////////////////////////////////////////

//

// DESIGNINFO

//

#ifdef APSTUDIO\_INVOKED

GUIDELINES DESIGNINFO

BEGIN

IDD\_ABOUTBOX, DIALOG

BEGIN

LEFTMARGIN, 7

RIGHTMARGIN, 163

TOPMARGIN, 7

BOTTOMMARGIN, 55

END

END

#endif // APSTUDIO\_INVOKED

#ifdef APSTUDIO\_INVOKED

/////////////////////////////////////////////////////////////////////////////

//

// TEXTINCLUDE

//

1 TEXTINCLUDE

BEGIN

"resource.h\0"

END

2 TEXTINCLUDE

BEGIN

"#ifndef APSTUDIO\_INVOKED\r\n"

"#include ""targetver.h""\r\n"

"#endif\r\n"

"#define APSTUDIO\_HIDDEN\_SYMBOLS\r\n"

"#include ""windows.h""\r\n"

"#undef APSTUDIO\_HIDDEN\_SYMBOLS\r\n"

"\0"

END

3 TEXTINCLUDE

BEGIN

"\r\n"

"\0"

END

#endif // APSTUDIO\_INVOKED

/////////////////////////////////////////////////////////////////////////////

//

// Таблица строк

//

STRINGTABLE

BEGIN

IDC\_TASKLIST "TASKLIST"

IDS\_APP\_TITLE "TaskList"

END

#endif

/////////////////////////////////////////////////////////////////////////////

#ifndef APSTUDIO\_INVOKED

/////////////////////////////////////////////////////////////////////////////

//

// Создано из ресурса TEXTINCLUDE.

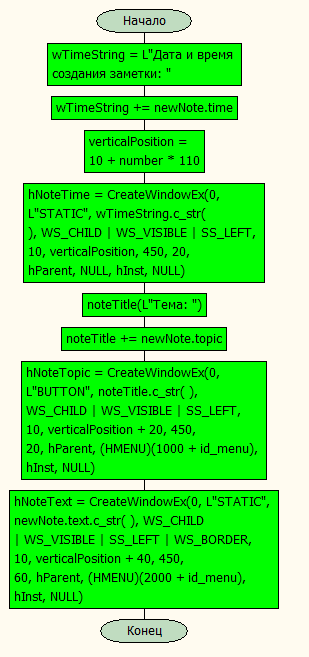
//

/////////////////////////////////////////////////////////////////////////////

#endif // not APSTUDIO\_INVOKED

## 10.2. Алгоритмы функций

AddNoteToList – функция, размещающая заметку на главном экране. Алгоритм функции представлен на рисунке 10.1.

  
Рисунок 10.1. Алгоритм функции AddNoteToList.

ClearList – функция, очищающая родительское окно. Алгоритм функции представлен на рисунке 10.2.

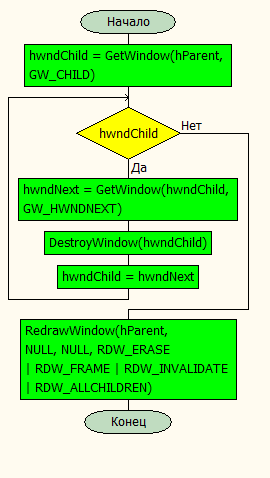
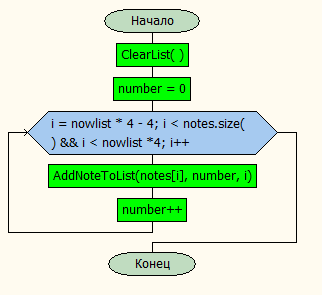


Рисунок 10.2. Алгоритм функции ClearList.

UpdateList – функция, отображающая выбранную страницу. Алгоритм функции представлен на рисунке 10.3.

  
Рисунок 10.3. Алгоритм функции UpdateList