**3. ПРОЕКТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ**

1. **Разработка структуры приложения**

Для разработки приложения использовалась среда Microsoft Visual Studio 2019, включающая в себя широкий набор функциональных инструментов.

Для представления принципа работы программы ниже приведена ее

UML диаграмма вариантов использования (рисунок 1).

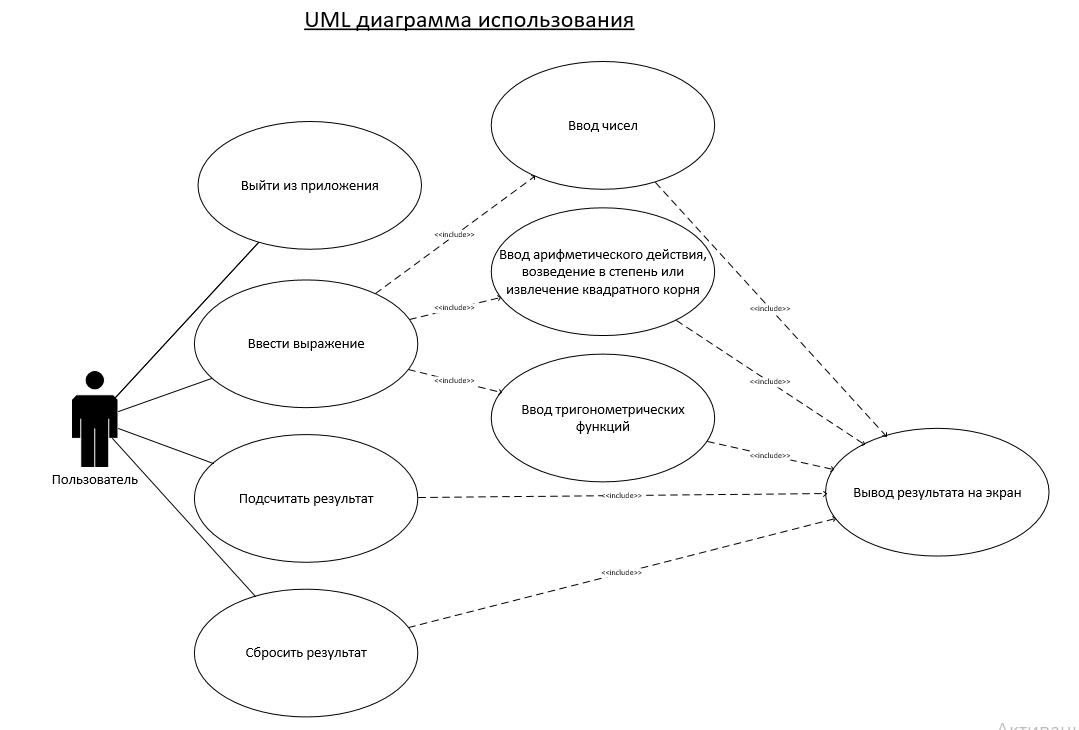


Рисунок 1 - Диаграмма использования программы «Калькулятор»

При запуске приложения у пользователь может выбрать, какое действие ему совершить.

В любой момент работы с приложением, пользователь может выйти из него.

Также, пользователь может ввести необходимо для вычисления выражение. Ввод состоит из чисел, арифметических действий, возведения в степень, извлечения квадратного корня, а также тригонометрических функций. Если возникла ошибка при вводе, то пользователь может сбросить результат.

После ввода, пользователь может подсчитать результат выражения и продолжить работу с полученным результатом или сбросить его.

В процессе работы с приложением, действия (ввод, вычисление, сброс), которые совершает пользователь, будут выводится на экран.

Общее назначение программного средства - выполнение арифметических и тригонометрических операций для использования в производственном, учебном процессах и повседневной жизни.

Реализуемая задача состоит в том, чтобы производилась корректная работа с тригонометрическими функциями, также чтобы при выборе действия выполнялась определенная операция, и имелась возможность сбросить полученный результат или же продолжить расчеты с данным результатом.

1. **Используемые функции Win32API**

Для создания программы «Калькулятор с расширенными возможностями», необходимо реализовать алгоритмы, позволяющие работать с арифметическими действиями, а также с тригонометрическими функциями. Необходимо организовать возможность сброса полученных результатов.

Блок-схемы алгоритмов представлены в Приложении 1.

Основные функции, реализуемые в программе из библиотеки User32.lib:

* LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) - производит обработку сообщений для окна (получает сообщения, которые Windows будет отсылать окну) :
  + hWnd – главное окно приложения,
  + message – сообщение, посылаемое операционной системе,
  + wParam, lParam – параметры сообщений, для последующего обращения.
* HCURSOR LoadCursor(NULL, IDC\_ARROW) - загружает заданный ресурс курсора из исполняемого файла:
  + NULL - дескриптор экземпляра модуля, исполняемый файл которого содержит загружаемый курсор.
  + IDC\_ARROW - cтандартная стрелка.
* HICON LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(MAIN\_ICON)) - загружает указанный ресурс значка из исполняемого файла:
  + GetModuleHandle - извлекает дескриптор указанного модуля;
  + MAKEINTRESOURCE - преобразует целочисленное значение в тип ресурса
  + MAIN\_ICON – предопределенная константа, служит для обозначения уникальности созданной иконки (.ico)
* ATOM RegisterClass(&wc) - регистрирует класс окна для последующего использования при вызове функции CreateWindow:
  + wc – структура, содержащая атрибуты окна.
* HWND CreateWindow(WMAIN\_CLASSNAME, WMAIN\_CAPTION, WS\_OVERLAPPED | WS\_CAPTION | WS\_SYSMENU | WS\_MINIMIZEBOX | WS\_MAXIMIZEBOX | WS\_VISIBLE, CW\_USEDEFAULT, CW\_USEDEFAULT, 320, 419, HWND\_DESKTOP, NULL, hInstance, 0) – создает основное окно приложения:
  + WMAIN\_CLASSNAME – зарегистрированное имя класса (Calculator),
  + WMAIN\_CAPTION – имя окна (Калькулятор),
  + WS\_OVERLAPPED – определяет базовый стиль окна - перекрывающееся окно,
  + WS\_CAPTION – заголовок,
  + WS\_SYSMENU – системное меню,
  + WS\_MINIMIZEBOX – кнопка минимизации размера окна,
  + WS\_MAXIMIZEBOX – кнопка максимального увеличения размера окна,
  + WS\_VISIBLE – первоначально, окно не отображается,
  + CW\_USEDEFAULT – горизонтальная позиция окна,
  + CW\_USEDEFAULT – вертикальная позиция окна,
  + 320 – ширина окна,
  + 419 – высота окна,
  + HWND\_DESKTOP – дескриптор родительского окна,
  + NULL - дескриптор меню или ID дочернего окна,
  + hInstance - дескриптор экземпляра приложения (это некий код оконной процедуры, идентификатор, по которой ОС будет отличать её от остальных окон)
  + 0 - указатель на данные создания окна (указывает на значение, переданное в параметре lParam сообщения WM\_CREATE).
* BOOL ShowWindow (hWnd, nCmdShow) - устанавливает состояние показа определяемого окна:
  + hWnd - главное окно приложения,
  + nCmdShow - состояние показа окна.
* BOOL UpdateWindow(hWnd) - обновляет рабочую область заданного окна, отправляя сообщение WM\_PAINT окну, если регион обновления окна не пуст:
  + hWnd - главное окно приложения.
* BOOL GetMessage(&msg, NULL, 0, 0) - извлекает сообщение из очереди сообщений вызывающего потока и помещает его в заданную структуру:
  + msg - указатель на структуру MSG, которая принимает информацию из очереди сообщений потока,
  + NULL – дескриптор окна,
  + 0 - определяет целочисленную величину самого маленького значения сообщения, которое будет извлечено,
  + 0 - определяет целочисленную величину самого большого значения сообщения, которое будет извлечено

Обе величины нулевые, следовательно, функция GetMessage возвращает все доступные сообщения (то есть никакой фильтрации в диапазоне значений не выполняется).

* BOOL TranslateMessage(&msg) - переводит сообщения виртуальных клавиш в символьные сообщения:
  + &msg - указатель на структуру MSG, которая содержит информацию о сообщении извлеченную из очереди сообщений вызывающего потока при помощи использования функции GetMessage.
* LRESULT DispatchMessage(&msg) - распределяет сообщение оконной процедуре, используется, чтобы доставить сообщение, извлеченное функцией GetMessage:
  + &msg - указатель на структуру MSG, которая содержит сообщение.
* LRESULT SendMessage(button[21], BM\_SETCHECK, BST\_CHECKED, 0) - отправляет заданное сообщение окну:
  + button[21] - дескриптор окна, оконная процедура которого примет сообщение (в данном случае радио - кнопка «RAD»);
  + BM\_SETCHECK - определяет сообщение, которое будет отправлено (возвращает состояние флажка для переключателя);
  + BST\_CHECKED – определяет дополнительную конкретизирующую сообщение информацию (устанавливает состояние флажка как отмеченное);
  + 0 - определяет дополнительную конкретизирующую сообщение информацию.
* WORD LOWORD(wParam) - выделяет из 32-битового целочисленного значения младшее слово:
  + wParam – преобразуемое значение;
* WORD HIWORD(wParam) - выделяет из 32-битового целочисленного значения старшее слово:
  + wParam – преобразуемое значение;
* int GetWindowText(hEdit, buf, sizeof(buf)) - копирует текст окна в буфер:
  + hEdit - дескриптор окна управления с текстом (окно ввода-вывода);
  + buf - адрес буфера для текста;
  + sizeof(buf) - максимальное число символов для копирования;
* BOOL SetWindowText(hEdit, buf) - изменяет текст окна:
  + hEdit - дескриптор окна, текст которого должен быть изменен (окно ввода-вывода);
  + buf – адрес буфера для текста.
* void PostQuitMessage(0) - указывает системе, что поток сделал запрос на то, чтобы завершить свою работу (выйти). Это используется в ответ на сообщение WM\_DESTROY:
  + 0 - определяет код завершения прикладной программы.
* LRESULT DefWindowProc(hWnd, message, wParam, lParam) - вызывается оконной процедурой по умолчанию, чтобы обеспечить обработку по умолчанию любого сообщения окна, которые приложение не обрабатывает. Эта функция гарантирует то, что обрабатывается каждое сообщение:
  + hWnd – главное окно приложения, которое получает сообщение;
  + message – передаваемое сообщение;
  + wParam, lParam - определяют дополнительную информацию о сообщении.

1. **Разработка архитектуры приложения**

Приложение разрабатывалась по архитектуре многоуровневого шаблона.

Данное приложение использует многоуровневый архитектурный шаблон. Данный шаблон используется для структурирования программ, которые можно разложить на группы неких подзадач, находящихся на определенных уровнях абстракции. Каждый слой представляет службы для следующего, более высокого слоя. На рисунке 2 представлена схема приложения.

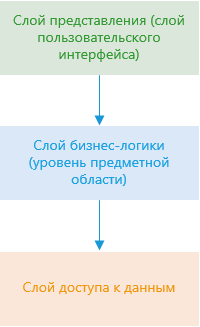


Рисунок 2 - Схема построения приложения по многоуровневому шаблону

Слой пользовательского интерфейса в разрабатываемом приложении представлен в файле Calculator.cpp, в котором указаны следующие параметры:

* главное окно. Расположение кнопок, поля ввода-вывода и параметрами главного окна;
* иконка приложения при сборке проекта.

Слой бизнес-логики описан основным исполняемым файлом Calculator.cpp и файлом заголовков resource.h. Исполняемый файл Calculator.cpp состоит из функции обработки нажатия клавиш. Файл заголовков состоит из логики вычислений и вывода результата на экран.

Слой данных представляется в виде символьных массивов, в которых хранятся введенные и вычисленные выражения.

1. **Разработка алгоритмов приложения**

Основной алгоритм приложения – непосредственное вычисление арифметических и тригонометрических выражений и последующий вывод на экран результата вычислений.

Для этого используется функция TCHAR\* Calculate(TCHAR\* buf) , где buf – это глобальный строчный массив символов, который был заполнен пользователем числами и знаками с помощью предыдущей функции.

Сперва, определяются приоритеты знаков, с помощью встроенной функции “map” из библиотеки “map”.

После определения приоритетов, необходимо перенести данные из символьного массива buf[256] в строку basic\_string, а после этого в строку потока basic\_stringstream.

Далее, используя basic\_stringstream, символы поочередно переводятся в числа типа double. После получения чисел, данная функция определяет какой знак действия ввел пользователь, и в зависимости от этого выполняются необходимые вычисления.

Последний шаг в функции – это вывод на экран вычисленного выражения.

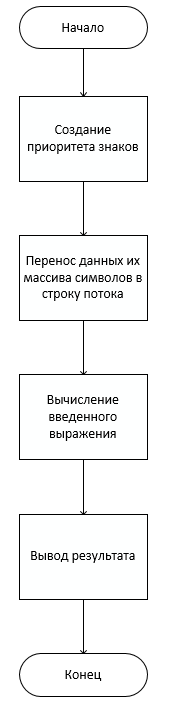


Рисунок 3 - Блок-схема функции Calculate

1. **Разработка интерфейса взаимодействия пользователя с системой**

Основная и самая главная часть приложения – создание окна и дальнейшее взаимодействие с ним.

Для этого используется функция

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) , где

hWnd – дескриптор окна,

message – сообщение, посылаемое операционной системе,

wParam, lParam – параметры сообщений, для последующего обращения.

В данной функции взаимодействие с окном происходит с помощью множественного условного оператора Switch.

В оператор передается переменная message, в зависимости от значения которой будут выполняться определенные действия.

При message = WM\_CREATE, создается интерфейс взаимодействия пользователя с приложением (окно, кнопки) с помощью встроенной функции CreateWindow.

Message = WM\_COMMAND становится при нажатии пользователем на любую кнопку интерфейса, далее, в зависимости от того, какая именно кнопка была нажата, в программе выполняются определенные действия, например, ввод чисел, знаков, подсчитывание результата.

Если пользователь нажмет на крестик в правом верхнем углу программы, то переменной message присвоится значение WM\_DESTROY и произойдет удаление окна с экрана (программа закроется).

Message = default – это обработчик программы по умолчанию, для того чтобы обеспечить обработку любого сообщения окна, которое приложение не обрабатывает.

Таким образом, для того чтобы производилась корректная работа с тригонометрическими функциями, при выборе действия выполнялась определенная операция, и имелась возможность сбросить полученный результат или же продолжить расчеты с данным результатом необходимо реализовать определенные алгоритмы: алгоритм непосредственного вычисления выражений и алгоритм взаимодействия пользователя с интерфейсом приложения. Для правильной работы приложения, необходимо использовать многоуровневый архитектурный шаблон.