# CSE 115 Section 2

Group Project (Number 9)


| Group Members | ID Number | Email Address |
|---|---|---|
| Kazi Abu Baqr Khalil | 2511608042 | kazi.khalil.251@northsouth.edu |
| Sidratul Muntaha Rahman Khan | 2514009642 | sidratul.khan.251@northsouth.edu |
| Prothom Khondker | 2512647042 | prothom.khondker.251@northsouth.edu |
| Shakibul Alam Ashik | 2514440043 | shakibul.ashik.251@northsouth.edu |


## Project Summary


This project is a graphical calculator application developed using the C programming language and the Windows API (WinAPI). It serves as a functional calculator allowing users to input and evaluate mathematical expressions using a graphical interface. It replicates the behavior and layout of a typical pocket calculator, providing buttons for digits, arithmetic operators, decimal points, and parentheses.

The calculator was built from scratch without using any third-party GUI frameworks or external libraries. Its design relies solely on WinAPI functions and C standard libraries. The application is developed and tested in Codeblocks and works on any standard Windows environment. It includes support for multi-digit numbers, decimal values, complex expressions with parentheses, and correct operator precedence during evaluation.

This project not only fulfills its functional goal but also reinforces fundamental concepts of low-level UI programming, event-driven application logic, manual expression evaluation, and memory-safe C development.


## Objective of the Project

- To develop a fully functioning graphical calculator using only C and native Windows libraries.
- To gain hands-on experience with the Windows API, creating buttons, handling user events, and updating UI elements dynamically.

- To construct a custom infix expression parser and evaluator for computing user-entered math expressions.
- To build an educational tool that reflects real-world GUI app structure, but in a student-friendly and lightweight way.
- To ensure runtime stability, usability, and user-friendly design without overcomplication or excessive abstraction.

## Background and Significance

Many C programming courses stop at console-based applications. Hence people often miss out on learning how GUI apps are created without the use of high-level frameworks. This project addresses that gap by introducing WinAPI, which is a powerful interface for building native Windows applications.

Using WinAPI for a calculator:

- Encourages deeper understanding of how GUI apps work at the OS level.
- Reinforces the role of event-driven design, where applications respond to user inputs like button clicks or window changes.
- Provides students with confidence in creating real applications that go beyond text-based interaction.

Significance:

- WinAPI is lightweight and requires no external setup.
- Real-world calculators follow expression evaluation logic, not just step-by-step arithmetic.
- This project brings together UI, math, parsing, and event management into a cohesive student-level application.

## Program Execution Flow (Textual Flowchart)

The following steps describe the internal workflow of the calculator program from the moment it is started until the user exits the application:

1. Program Launch (WinMain)
    - Initializes application window and registers the window class.
    - Sets up message loop to listen for user actions.
2. Window Creation (CreateWindowEx)
    - Creates main application window.
    - Creates all buttons (digits, operators, functions) and a textbox.
3. Event Handling (WndProc)

- Listens for WM_COMMAND messages when buttons are clicked.
- Identifies which button was clicked using its ID.

4. Expression Building
   - Appends clicked button's character to the expression array.
   - Updates the display textbox using SetWindowText().
5. Evaluation Trigger (Equals '=' button)
   - Parses the built expression using custom evaluation logic.
   - Handles operator precedence and parentheses using stacks.
6. Result Display
   - Sets the result back into the textbox for the user to see.
7. Clear Button
   - Clears the current expression and resets the UI.
8. Program Exit
   - Waits for the user to close the application (click the X).
   - Properly deallocates resources and exits the message loop.

# Application Usage Guidelines

## How to Compile and Run

- Open Code::Blocks on a Windows system.
- Create a new Windows application project (not console).
- Paste the full code into the main.c or default source file.
- Make sure to link the application with the correct WinAPI settings (-mwindows in linker options, if needed).
- Build and run the project using the "Build & Run" command.

## Input and Output Format

- Input: All inputs are given via mouse clicks on calculator buttons. Each button click appends a number, operator, or parenthesis to the current expression.
- Output: The final result of the expression is displayed in the top display box (edit control) once = is clicked. If the expression is malformed, a 0 or blank output is shown.

## Instructions for Use

- Digits (0–9): Click to input numbers.
- Operators (+, –, ×, ÷): Click to add math operations.

- Parentheses ( and ): Use to group parts of expressions.
- Decimal ( . ): Allows entry of decimal values.
- C (Clear): Clears the current expression.
- = (Equals): Evaluates the full expression and shows the result.

# 5. Features and Implementation

## Graphical Interface (UI Design)

- Built using CreateWindowEx() and CreateWindow() for all GUI elements.
- Buttons dynamically created in a grid layout.
- Edit control (textbox) displays current expression and result.
- Responsive to clicks using WM_COMMAND messages.

## Expression Building Logic

- All characters (numbers/operators) are stored in a character array expression[].
- Button click events add the character corresponding to the clicked button into the array.
- The expression is only evaluated when = is clicked.
- Allows expressions like: 6*(9+2)/3.5

## Expression Evaluation

- A manual parsing algorithm is used to:
    1. Separate numbers and operators.
    2. Apply operator precedence.
    3. Handle parentheses correctly.
- Two stacks are implemented:
    1. Operator Stack: Handles +, –, *, /
    2. Operand Stack: Handles numbers (integers and floats)
- After parsing, each operator is applied based on precedence rules using apply_op().

## Decimal and Floating-Point Support

- strtod() is used to convert substrings to floating point values.
- Prevents multiple decimals in a single number.
- All results are shown with floating-point precision (e.g., 12.5 ÷ 2 = 6.25).

### Error Handling and Validation

- Invalid operations (e.g., division by zero) are caught and default to zero.
- Overflow and buffer issues prevented using length checks (strlen()).
- Only calculator buttons can alter the input, ensuring valid expression construction.

## 6. Testing and Results

### Test Cases

| Test Case | Input Expression | Expected Output | Actual Output | Status |
|---|---|---|---|---|
| 1 | 2+3*4 | 14 | 14 | Valid |
| 2 | 10/(2+3) | 2 | 2 | Valid |
| 3 | (1+2)*(3+4) | 21 | 21 | Valid |
| 4 | 5.5 + 2.3 | 7.8 | 7.8 | Valid |
| 5 | 9/0 | Error or handled | 0 | Valid(Handled) |
| 6 | ((2+3) | Invalid format | 0 | Valid |
| 7 | 2.5*4.2 | 10.5 | 10.5 | Valid |
| 8 | 100 - 50 / 5 | 90 | 90 | Valid |
| 9 | Pressing = without input | 0 or no action | 0 | Valid |
| 10 | Using multiple decimal points like 1.2.3 | Invalid | 0 | Valid |

### Summary of Results and Performance

- The calculator handles basic arithmetic, parentheses, and floating point numbers accurately.
- It maintains stability for all valid and invalid expressions, including division by zero and malformed inputs.
- Performance is instant even for longer expressions due to lightweight logic and absence of external libraries.
- The UI remains responsive, and expression input is real-time without lag.
- The button layout provides an intuitive calculator-like experience.

- Evaluation algorithm accuracy was verified against standard calculators and manual computations.

# 6. Challenges and Solutions

### Buttons Causing Crash

Problem: Clicking any button would crash the application.
Cause: Button ID mapping outside valid index range.
Solution: Added range check before accessing button label array.

### Only One Character Appearing

Problem: Only one character shown in display at a time.
Cause: SetWindowText() was replaced instead of appending.
Solution: We built appendToExpression() to manage expression state and properly append text.

### 6.3 Decimal Parsing Issues

Problem: Incorrect or duplicate decimal inputs.
Solution: Allowed one decimal per number using simple character checks before appending.

### 6.4 Manual Evaluation Complexity

Problem: No standard library function to evaluate full expressions in C.
Solution: Implemented a custom stack-based evaluator, mimicking infix logic with correct precedence and parentheses handling.

### 6.5 UI Scaling and Layout

Issue: Hardcoded button positions were difficult to manage.
Solution: Used grid-based layout calculation using x = col * button_width + margin formulas.

# 7. Code Structure Overview

- WinMain() – Entry point for the Windows app.
- WndProc() – Handles all window messages.
- CreateWindow() calls for each button and textbox.
- appendToExpression() – Safely appends input to expression string.
- evaluate() – Parses and evaluates the math expression.
- precedence() – Returns priority of an operator.
- apply_op() – Executes math operations on operands.

## 8. Learning Outcomes

From this project, the following concepts and skills were reinforced:

- Event-Driven Programming: Understanding how events like button clicks are handled through messages.
- Low-Level GUI Programming: Building a UI without the help of higher-level GUI libraries.
- String and Buffer Management: Using arrays and memory-safe operations (strcat, strncpy, etc.).
- Math Expression Evaluation: Learning how to parse and evaluate expressions using stacks.
- Debugging Windows Apps: Tracing and fixing crashes caused by UI or logic bugs.
- Practical C Programming: Bridging the gap between console applications and user-facing software.

## 9. Future Improvements

Some planned or possible upgrades include:

- Keyboard Input: Allow input via keyboard for better usability.
- Backspace Button: Let users delete the last character.
- Memory Functions: Add M+, M-, MR (memory recall) like a real calculator.
- Scientific Mode: Introduce advanced functions: sin, cos, sqrt, log.
- Enhanced UI: Improve layout with better colors, fonts, hover effects.
- Better Error Messages: Notify users of invalid inputs explicitly.
- Calculation History: Show a list of previous expressions and results.

# 10. Conclusion

This project represents a comprehensive exploration of how to build a fully functional Windows GUI application in C using only WinAPI. It involved hands-on practice with:

- GUI component creation
- Event handling
- Manual expression parsing and math logic
- String and memory safety

Despite several technical challenges, the project now runs smoothly and reliably, offering a solid demonstration of programming competence at the system and application level.

The project's success shows how foundational knowledge in C and a bit of creativity can be used to build impressive and useful applications, even without modern frameworks. It also sets the stage for more advanced projects and concepts in native Windows development.