Austin Johnson
0649
Teammates: Subrajit Surendran, Bailey Nguyen

Assignment 2 Part 2

```
#include <iostream>

int storedValue = 0;

void store(int value)
{
    storedValue = value;
}

int retrieve()
{
    return storedValue;
}

void clear()
{
    storedValue = 0;
    std::cout << "Stored Value Cleared." << std::endl;
}

int main()
{
    int num1, num2, result;
    char operation;
    while (true)
    {
        std::cout << "Enter first number: ";
        std::cin >> num1;
        std::cout << "Enter an operator (+, -, *, /, %, q=quit, s=store,
r=retrieve, c=clear): ";
        std::cin >> operation;

        if (operation == 'q')
        {
            break;
```

Commented [1]: Verify user input does not exceed max value that an "int" can be ... INT_MAX

Commented [2]: User input vulnerability: If input can only be int, create a check to ensure no other type is being input, or else program may crash

Commented [3]: Unvalidated user input vulnerability. Validate user input follows expected convention

```
        }
        else if (operation == 'r')
        {
            std::cout << "Retrieved Value: " << retrieve() << std::endl;
            continue;
        }
        else if (operation == 's')
        {
            store(num1);
            std::cout << "Stored: " << num1 << std::endl;
            continue;
        }
        else if (operation == 'c')
        {
            clear();
            continue;
        }

        std::cout << "Enter second number: ";
        std::cin >> num2;

        switch (operation)
        {
        case '+':
            result = num1 + num2;
            break;
        case '-':
            result = num1 - num2;
            break;
        case '*':
            result = num1 * num2;
            break;
        case '/':
            result = num1 / num2;
            break;
        case '%':
            result = num1 % num2;
            break;
        default:
            std::cout << "Invalid operator";
```

Commented [4]: Overflow vulnerability, should check if overflow has happened and let user know

Commented [5]: Divide by zero vulnerability. Check if divider is zero and handle gracefully

Commented [6]: Modulo by zero vulnerability. Check if if divider is zero and handle gracefully

```
            continue;
        }

        std::cout << "Result: " << result << std::endl;
    }
    return 0;
}
```

Modified Code Based on Teamates Comments

```
#include <iostream>
#include <limits>

int storedValue = 0;

void store(int value)
{
    storedValue = value;
}

int retrieve()
{
    return storedValue;
}

void clear()
{
    storedValue = 0;
    std::cout << "Stored Value Cleared." << std::endl;
}

// Implemented to catch if the input is not an integer
// This also catches if the input is greater than the max number allowed
or less than the lowest number allowed.
int getValidInt()
{
    int value;
    while (true)
    {
        std::cin >> value;

        if (std::cin.fail())
        {
            std::cin.clear();
            std::cin.ignore(1000, '\n');
            std::cout << "Invalid input. Please enter an integer: ";
        }
        else
        {
            return value;
        }
    }
}

// Implemented to catch if the operation input is valid operator or not
char getValidOperator()
{
    char operation;
    while (true)
    {
        std::cin >> operation;
        if (operation == '+' || operation == '-' || operation == '*' ||
            operation == '/' || operation == '%' ||
            operation == 'q' || operation == 's' ||
            operation == 'r' || operation == 'c')
        {
            return operation;
        }
        else
        {
            std::cout << "Invalid operator. Please enter a valid operator
(+, -, *, /, %, q=quit, s=store, r=retrieve, c=clear): ";
        }
    }
}

// Implemented to check for addition overflow
bool checkAddOverflow(int a, int b)
{
    if ((b > 0 && a > std::numeric_limits<int>::max() - b) ||
        (b < 0 && a < std::numeric_limits<int>::min() - b))
    {
        return true;
    }
    return false;
}

// Implemented to check for subtraction overflow
```

```
bool checkSubOverflow(int a, int b)
{
    if ((b < 0 && a > std::numeric_limits<int>::max() + b) ||
        (b > 0 && a < std::numeric_limits<int>::min() + b))
    {
        return true;
    }
    return false;
}

// Implemented to check for multiplication overflow
bool checkMulOverflow(int a, int b)
{
    if (a > 0 && b > 0 && a > std::numeric_limits<int>::max() / b)
        return true;
    if (a < 0 && b < 0 && a < std::numeric_limits<int>::max() / b)
        return true;
    if (a > 0 && b < 0 && b < std::numeric_limits<int>::min() / a)
        return true;
    if (a < 0 && b > 0 && a < std::numeric_limits<int>::min() / b)
        return true;

    return false;
}

// Implemented to check for division by zero
bool checkDivideByZero(int b)
{
    return b == 0;
}

int main()
{
    int num1, num2, result;
    char operation;
    while (true)
    {
        std::cout << "Enter first number: ";
        num1 = getValidInt();
```

```cpp
        std::cout << "Enter an operator (+, -, *, /, %, q=quit, s=store,
r=retrieve, c=clear): ";
        operation = getValidOperator();

        if (operation == 'q')
        {
            break;
        }
        else if (operation == 'r')
        {
            std::cout << "Retrieved Value: " << retrieve() << std::endl;
            continue;
        }
        else if (operation == 's')
        {
            store(num1);
            std::cout << "Stored: " << num1 << std::endl;
            continue;
        }
        else if (operation == 'c')
        {
            clear();
            continue;
        }

        std::cout << "Enter second number: ";
        num2 = getValidInt();

        try
        {
            switch (operation)
            {
            case '+':
                if (checkAddOverflow(num1, num2))
                    throw std::overflow_error("Addition overflow!");
                result = num1 + num2;
                break;
            case '-':
                if (checkSubOverflow(num1, num2))
                    throw std::overflow_error("Subtraction overflow!");
                result = num1 - num2;
                break;
            case '*':
                if (checkMulOverflow(num1, num2))
                    throw std::overflow_error("Multiplication overflow!");
                result = num1 * num2;
                break;
            case '/':
                if (checkDivideByZero(num2))
                    throw std::invalid_argument("Division by zero is not
allowed.");
                result = num1 / num2;
                break;
            case '%':
                if (checkDivideByZero(num2))
                    throw std::invalid_argument("Modulo by zero is not
allowed.");
                result = num1 % num2;
                break;
            default:
                std::cout << "Invalid operator";
                continue;
            }

            std::cout << "Result: " << result << std::endl;
        }
        catch (std::exception &e)
        {
            std::cout << "Error: " << e.what() << std::endl;
        }
    }
    return 0;
}
```