Data Structures & Algorithms

Teacher: Dr. Adeel Muzaffar Syed

DATE: 23-09-2024

Assignment # 1

Name #	Enrollment #
Muhammad Sami Shahid	01-131232-066
Mudassir Raiz	01-131232-046

Department of Software Engineering

Muhammad Sami Shahid
01-131232-066

Mudassir Raiz
01-131232-046

Asgn 1
Deadline Tuesday Sep 24, 2024.
Group size: 2

Read a text file containing at least 10 different lines having expressions in it. Ask the user which line to check. Whichever line is selected by the user, the program should tell if the expression is valid or not. Use stacks to perform this check. All conventions are to be followed.

Git-hub link:

https://github.com/samishahid516/DSA-Assign-1

Code:

Stack.h:

```
#include <iostream>
#include <stdexcept>

template <class ItemType>
class Stack
{
private:
    ItemType* items;
    int capacity;
    int top;

public:
    Stack();
    ~Stack();
    void push(const ItemType& element);
    void pop();
    ItemType Top() const;
    bool IsEmpty() const;
    bool IsFull() const;
};
```

Stack.cpp:

```
#include "Stack.h"

template <class ItemType>
Stack<ItemType>::Stack()
{
    capacity = 50;
    top = -1;
    items = new ItemType[capacity];
}
```

```cpp
template <class ItemType>
Stack<ItemType>::~Stack()
{
    delete[] items;
}

template <class ItemType>
void Stack<ItemType>::push(const ItemType& element)
{
    if (IsFull())
    {
        throw std::overflow_error("Stack is full");
    }
    top++;
    items[top] = element;
}

template <class ItemType>
void Stack<ItemType>::pop()
{
    if (IsEmpty())
    {
        throw std::out_of_range("Stack is empty");
    }
    top--;
}

template <class ItemType>
ItemType Stack<ItemType>::Top() const
{
    if (IsEmpty())
    {
        throw std::out_of_range("Stack is empty");
    }
    return items[top];
}

template <class ItemType>
bool Stack<ItemType>::IsEmpty() const
{
    if (top == -1)
    {
        return true;
    }
    else
        return false;
}

template <class ItemType>
bool Stack<ItemType>::IsFull() const
{
    if (top == capacity - 1)
    {
        return true;
    }
    else
        return false;
}

template class Stack<char>;
```

Main.cpp:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include "Stack.cpp"

using namespace std;

bool isValidExpression(const std::string& expression)
{
    Stack<char> stack;  // Use char for the stack

    for (char ch : expression)
    {
        if (ch == '(' || ch == '{' || ch == '[') {
            stack.push(ch);
        }
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (stack.IsEmpty()) return false;

            char top = stack.Top();
            if ((ch == ')' && top != '(') ||
                (ch == '}' && top != '{') ||
                (ch == ']' && top != '[')) {
                return false;
            }
            stack.pop();
        }
    }

    return stack.IsEmpty();
}

void readExpressionsFromFile(const std::string& filename, std::vector<std::string>&
expressions)
{
    std::ifstream file(filename);
    std::string line;

    if (file.is_open())
    {
        while (getline(file, line))
        {
            expressions.push_back(line);
        }
        file.close();
    }
    else
    {
        cerr << "Unable to open file: " << filename << endl;
    }
}

int main()
{
start:
    system("cls");
    vector<string> expressions;
    readExpressionsFromFile("expressions.txt", expressions);
```

```
    if (expressions.empty())
    {
        cout << "No expressions found in the file." << endl;
        return 1;
    }

    cout << "Available expressions:" << endl;
    for (size_t i = 0; i < expressions.size(); ++i)
    {
        cout << i + 1 << ": " << expressions[i] << endl;
    }

    int choice;

    do {
        cout << "Enter the line number to check (1 to " << expressions.size() << "):";
        cin >> choice;

        if (choice < 1 || choice > expressions.size()) {
            cout << "Invalid choice. Please try again:" << endl;
            choice = -1;
        }

    } while (choice < 1 || choice > expressions.size());

    string selectedExpression = expressions[choice - 1];

    if (isValidExpression(selectedExpression))
    {
        cout << "The expression \"" << selectedExpression << "\" is valid." << endl;
    }
    else {
        cout << "The expression \"" << selectedExpression << "\" is invalid." << endl;
    }

    system("pause");
    goto start;

    return 0;
}
```

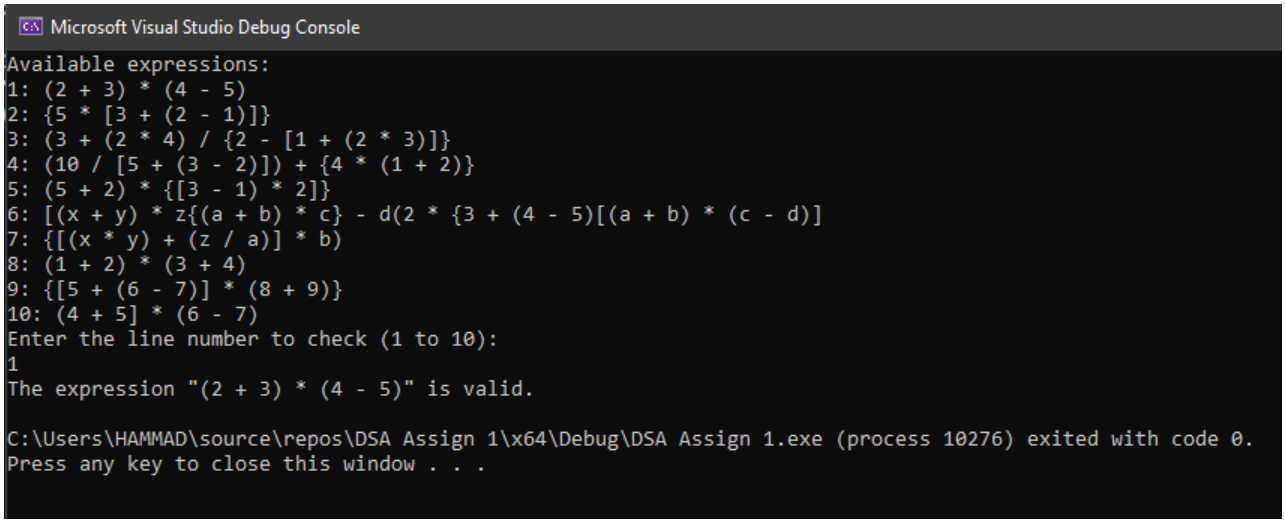
expressions.txt:

- (2 + 3) * (4 - 5)
- {5 * [3 + (2 - 1)]}
- (3 + (2 * 4) / {2 - [1 + (2 * 3)]}
- (10 / [5 + (3 - 2)]) + {4 * (1 + 2)}
- (5 + 2) * {[3 - 1) * 2]}
- [(x + y) * z{(a + b) * c} - d(2 * {3 + (4 - 5)[(a + b) * (c - d)]

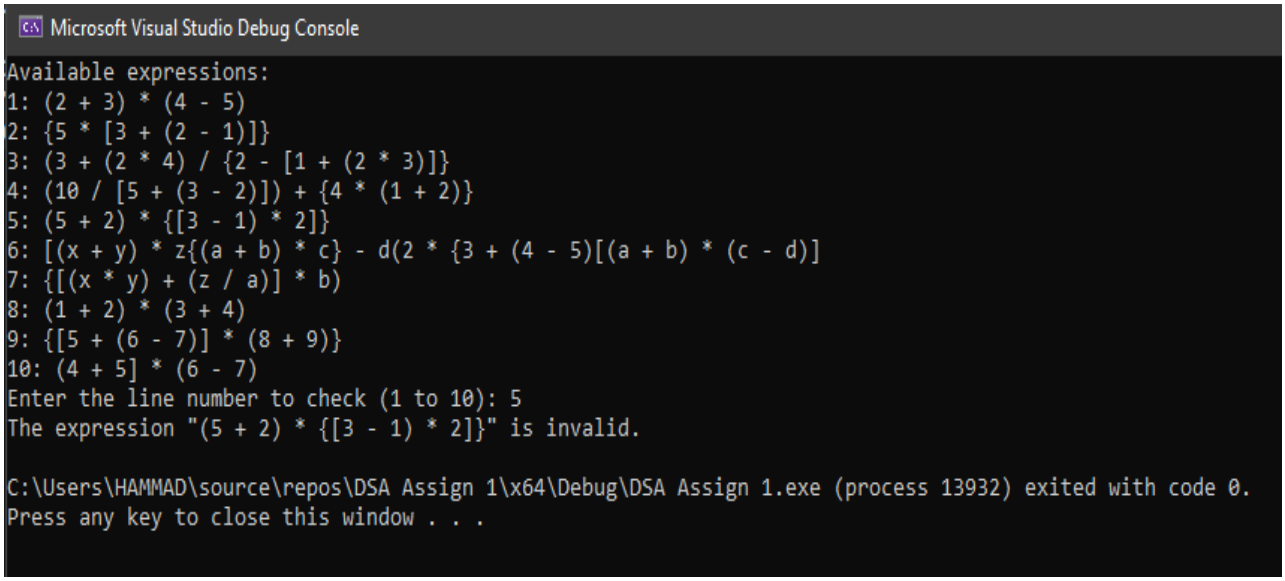
- {[(x * y) + (z / a)] * b)
- (1 + 2) * (3 + 4)
- {[5 + (6 - 7)] * (8 + 9)}
- (4 + 5] * (6 - 7)

Code Display:

For expression 1 which is valid:



For expression 5 which is invalid:



***********The End***********