BÁO CÁO XÂY DỰNG CHƯƠNG TRÌNH ỨNG DỤNG STACK AND QUEU ĐỂ TÍNH TOÁN BIỂU THỨC

Họ tên: Trần Quảng

MSSV: 21880108

Mục lục:

1. Đọc dữ liệu file BIEUTHUC.txt từ thư mục data

```
// Khai báo đường dẫn và đọc dữ liệu từ file
var filePath = Directory.GetParent("CTDL_GT").Parent.Parent.Parent.Parent.FullName;
var filePath_read = filePath + @"\data\BIEUTHUC.txt";
List<string> lines = new List<string>();
List<string> box = new List<string>();
```

- Khai báo lines,box để chứa dữ liệu chuẩn bị thực hiện tính toán, kết quả sau khi thực hiện.

2. Xây dựng chương trình

2.1 Xử lý dữ liệu đầu vào trước khi tính toán

```
lines = File.ReadAllLines(filePath_read).ToList();
string s = "";
for (int i =1; i < lines.Count;i++)
{
    s = lines[i].Replace(" ","");
    s = change_Infix(s);
    Queue Posfix = new Queue();
    Posfix = infixToPosfix(s);
    box.Add(Calculator(Posfix).ToString());
}
```

- Thực hiện xét từng dòng trong file BIEUTHUC.txt để thực hiện tính toán qua các bước
- Xóa khoảng từng dòng dữ liệu, và thực hiện định nghĩa lại “-” và “!” chuyển thành toán tử hai ngôi để đồng nhất và dễ thực hiện tính toán.

```
// Hàm chuyển đổi toán hạng âm và mang giai thừa thành toán tử hai ngôi
1 reference
public static String change_Infix(string s)
{
   string a = "";
   if (s[0] == '-')
    {
        a += '0';
    }
   for (int i = 0; i <s.Length;i++)
    {
        if (i != 0 && s[i - 1] == '(' && s[i] == '-')
            a += 0;
        a += s[i];
        if (s[i] == '!')
        {
            a += '0';
        }
    }
    return a;
}
```

2. Xây dựng chương trình

2.2 Xử lý dữ liệu từ Infix sang Postfix

```
public static bool IsNumber(string pText)
{
    Regex regex = new Regex(@"^[-+]?[0-9]*.?[0-9]+$");
    return regex.IsMatch(pText);
}

1 reference
private static bool IsOperator(string str)
{
    return Regex.Match(str, @"\+|\-|\*|\/|\^|\!|\(|\)").Success;
}
```

- Viết hàm kiểm tra dữ liệu đầu vào có là hợp lệ hay không.

2. Xây dựng chương trình

2.2 Xử lý dữ liệu từ Infix sang Postfix

```
public static Queue infixToPosfix (String a)
{
    Queue posfix = new Queue();
    Stack stack = new Stack();
    string token,x;
    string newQueue = "";
    for (int i = 0; i<a.Length;i++)
    {
        token = a[i].ToString();
        if (token.All(Char.IsDigit) || token ==".")
        {
            newQueue += token;
        }
```

- Khởi tạo Stack rỗng và Queu rỗng
- Khởi tạo 2 chuỗi x và token.
- Duyệt vòng lặp for từ i = 1 cho đến cuối chuỗi Infix:
  Nếu Infix[i] là toán hạng thì đưa vào Queu.
  Nếu Infix[i] là toán tử thì Push vào ngăn xếp S.
  Nếu Infix[i] là “)” thì Pop vào ngăn xếp S (lấy giá trị trên đỉnh của S) sau đó đưa vào Queu.

Output: Queue Postfix là biểu thức hậu tố.

```
else
{
    while (!stack.isEmty() && (precedence(token) <= precedence(stack.Gethead())))
    {
        x = stack.Pop();
        posfix.Enqueue(x);
    }
    stack.Push(token);
}
```

- Trong các bước này thực hiện kiểm tra toán hạng và toán tử xem có hợp lệ.

2. Xây dựng chương trình

2.3 Xử lý tính toán và trả ra kết quả

```
public static double Calculator(Queue profix)
{
    Stack stack = new Stack();
    double result = 0;
    string x, opr1, opr2;
    while (!profix.isEmty())
    {
        x = profix.Dequeue();
        if (x.All(Char.IsDigit))
        {
            stack.Push(x);
        }
        else
        {
            opr1 = stack.Pop();
            opr2 = stack.Pop();
            result = cal(x, opr1, opr2);
            stack.Push(result.ToString());
        }
    }
    return result;
}
```

- Thực hiện Dequeu từ profix cho tới khi rỗng. Nếu gặp toán hạng thì push vào stack ,nếu gặp toán thử sẽ pop toán hạng từ stack ra hai lần để thực hiện tính toán, sau đó push ngược kết quả lại stack.

2. Xây dựng chương trình

2.3 Xử lý tính toán và trả ra kết quả

```
1 reference
public static double cal(string x, string opr1, string opr2)
{
    double result = 0;
    double a = double.Parse(opr1);
    double b = double.Parse(opr2);
    switch (x)
    {
        case "+":
            return b + a;
        case "-":
            return b - a;
        case "*":
            return b * a;
        case "/":
            return b / a;
        case "^":
            return Math.Pow(b, a);
        case "!":
            if (b % 1 != 0)
            {
                throw new Exception("Không có chức năng tính thập phân cho số có giai thừa");
            }
            return Giaithua(b,a);
    }
    return result;
}
```

- Hàm calutator tính toán dữ liệu từ các toán hạng lấy ra từ stack

3. Ghi dữ liệu tính toán xong bào KETQUA.txt ở thư mục data

```
// ghi kết quả xuống file

var filePath_write = filePath + @"\data\KETQUA.txt";
File.WriteAllLines(filePath_write, box);
```

- Sau khi thực hiện tính toán file BIEUTHUC. Dữ liệu được lưu trong box và thực hiện thao tác ghi file.
- Kết thúc chương trình.