BỘ GIÁO DỤC ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**🙤🕮🙦**

****

**THỰC TẬP CƠ SỞ**

**VẬN DỤNG PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG XÂY DỰNG CẤU TRÚC NGĂN XẾP (STACK) ÁP DỤNG TÍNH GIÁ TRỊ BIỂU THỨC ĐẠI SỐ.**

**Giảng viên hướng dẫn:Nguyễn Đình Hưng**

**Sinh viên thực hiện:Đinh Văn Bảo**

**Mã số sinh viên: 61133384**

**LỜI NÓI ĐẦU**

**Lời cảm ơn!**

Lời đầu tiên, cho phép em gửi lời cảm ơn sâu sắc và chân thành nhất đến quý Thầy/Cô và các bạn học đã tạo điều kiện giúp em trong suốt quá trình học tập và thực hiện đề tài. Sự quan tâm và giúp đỡ của quý Thầy/Cô và các bạn học là nguồn động viên rất lớn giúp em hoàn thành tốt đề tài này.

Với lòng biết ơn sâu sắc nhất, em xin gửi đến quý Thầy/Cô ở Khoa Công Nghệ Thông Tin đã truyền đạt vốn kiến thức quý báu cho chúng em trong suốt thời gian học tập tại trường. Nhờ có những lời hướng dẫn, dạy bảo của các Thầy/Cô nên đề tài nghiên cứu của em mới có thể hoàn thiện tốt đẹp.

Một lần nữa, em xin chân thành cảm ơn những Thầy/Cô – người đã trực tiếp giúp đỡ, quan tâm, hướng dẫn chúng em hoàn thành tốt bài báo cáo này trong thời gian qua.

Bước đầu đi vào thực tế của em còn nhiều hạn chế và bỡ ngỡ nên không tránh khỏi những thiếu sót, em rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy/Cô để kiến thức của em trong bài báo cáo này được hoàn thiện hơn đồng thời có điều kiện bổ sung, nâng cao ý thức của mình.

Em xin chân thành cảm ơn!

**Mục tiêu cần đạt được**

Vận dụng phương pháp lập trình hướng đối tượng xây dựng cấu trúc ngăn xếp (Stack) áp dụng tính giá trị biểu thức đại số.

MỤC LỤC

[CHƯƠNG I. TỔNG QUAN 3](#_Toc91090708)

[1. Lý do chọn đề tài 3](#_Toc91090709)

[2. Phương pháp nghiên cứu 3](#_Toc91090710)

[CHƯƠNG II. PHÂN TÍCH BÀI TOÁN 4](#_Toc91090711)

[I. Mô tả thuật toán 4](#_Toc91090712)

[**1.** **Phân tích bài toán** 4](#_Toc91090713)

[**2.** **Tìm hiểu giải thuật.** 5](#_Toc91090714)

[**3.** **Tìm hiểu công cụ lập trình.** 7](#_Toc91090715)

[II. Cài đặt thuật toán 7](#_Toc91090716)

[1. Mô tả giải thuật 7](#_Toc91090717)

[2. Cài đặt bài toán 14](#_Toc91090718)

[CHƯƠNG III. KẾT QUẢ NGHIÊN CỨU 19](#_Toc91090719)

[1. Code index.html 19](#_Toc91090720)

[2. Code style.css 21](#_Toc91090721)

[3. Giao diện Web 23](#_Toc91090722)

[CHƯƠNG IV. KẾT LUẬN 24](#_Toc91090723)

# CHƯƠNG I. TỔNG QUAN

1. Lý do chọn đề tài

Như chúng ta đã biết Các biểu thức đại số được [biểu diễn dưới dạng trung tố](https://www.stdio.vn/article/ung-dung-stack-bieu-thuc-trung-to-infix-zjru1) (Infix). Tuy nhiên, để máy tính tính được giá trị của một biểu thức thì cần biểu diễn các biểu thức đại số từ trung tố sang một dạng khác là tiền tố hoặc hậu tố. Chuyển từ biểu thức trung tố (Infix) sang hậu tố (Postfix) và tính toán biểu thức hậu tố bằng cấu trúc Stack.

Với mong muốn ôn lại kiến thức lập trình hướng đối tượng và cấu trúc Stack cho công viếc sau này em đã quyết định chọn đề tài này. Đề tài vận dụng phương pháp lập trình hướng đối tượng xây dựng cấu trúc Stack áp dụng tính giá trị biểu thức đại số đã tạo điều kiện cho em hoàn thành mong muốn này. Như chúng ta đã biết cấu trúc Stack là một cấu trúc rất phổ biến và được ứng dụng vào nhiều bài toán lập trình đơn giản đến phức tạp. Và em cũng đang tìm hiểu về Call Stack trong ngôn ngữ Javascript nên em quyết định làm đề tài này bằng ngôn ngữ Javascript. Lập trình trên nền tảng ứng dụng Visual Studio Code.

1. Phương pháp nghiên cứu

Phướng pháp nghiên cứu lý thuyết em sử dụng phương pháp phân tích tổ hợp lý thuyết bằng việc tìm hiểu về cấu trúc và cách thức hoạt động của cấu trúc Stack. Sau đó nghiên cứu lý thuyết về biểu thức tiền tố và hậu tố, ký pháp nghich đảo Balan trong việc chuyển đổi từ biểu thức trung tố sang biểu thức hậu tố. Tiếp theo tìm hiểu cách thức tính toán biểu thức hậu tố. Sau đó là nghiên cứu lại lý thuyết lập trình hướng đối tượng, xây dựng các lớp trên ngôn ngữ Javascript.

Phương pháp nghiên cứu thực tiễn em sử dụng phương pháp điều tra, tìm hiểu code trên mạng sau đó nghiên cứu lại và viết lại nó trên ngỗn ngữ Javascript đồng thời thử xây dựng một ứng dụng, một giao diện nhỏ để có thể thực hiện việc chuyển đổi và tính toán biểu thức đại số, bằng biểu thức đại số do người dụng nhập vào. Sau đó chương trình sẽ xử lý chuyển đổi biểu thức tiền tố sang hậu tố và rồi tính toán biểu thức hậu tố đó. Biểu thức hậu tố và kết quả sẽ được in ra màn hình giao diện, hiện thị cho người dùng xem.

# CHƯƠNG II. PHÂN TÍCH BÀI TOÁN

## Mô tả thuật toán

1. **Phân tích bài toán**

Các biểu thức đại số được [biểu diễn dưới dạng trung tố](https://www.stdio.vn/article/ung-dung-stack-bieu-thuc-trung-to-infix-zjru1) (Infix). Tuy nhiên, để máy tính tính được giá trị của một biểu thức thì cần biểu diễn các biểu thức đại số từ trung tố sang một dạng khác là tiền tố hoặc hậu tố. Chuyển từ biểu thức trung tố (Infix) sang hậu tố (Postfix) và tính toán biểu thức hậu tố bằng cấu trúc Stack.

Biểu thức hậu tố (Postfix) là thuật toán được biểu diễn bằng cách đặt các toán tử ra sau các toán hạng.

VD)

|  |  |
| --- | --- |
| Infix | Postfix |
| A / B – C \* D | A B / C D \* + |
| A / ( B – C \* D) | A B C D \* - / |
| A / (B – C) \* D | A B C - / D \* |

*Cấu trúc Stack*

Trong khoa học máy tính, một ngăn xếp (còn gọi là bộ xếp chồng, tiếng Anh: Stack) là một cấu trúc dữ liệu trừu tượng hoạt động theo nguyên lý “vào sau ra trước” (Last In First Out (LIFO). Tức là, phần tử cuối cùng được chèn vào ngăn xếp sẽ là phần tử đầu tiên được lấy ra khỏi ngăn xếp.

Việc thao tác với ngăn xếp của chúng ta bao gồm các hành động sau:

**Push**: Thêm một phần tử vào đỉnh của ngăn xếp, số phần tử của ngăn xếp tăng lên 1.

**Pop**: Xóa bỏ phần tử đầu tiên ở đỉnh của ngăn xếp, số phần tử của ngăn xếp giảm đi 1.

**Top**: Lấy giá trị của phần tử đầu tiên ở đỉnh của ngăn xếp, số phần tử của ngăn xếp không thay đổi.

**IsEmpty**: Kiểm tra ngăn xếp trống hay không. Ngăn xếp trống là ngăn xếp không có phần tử nào.

**IsFull**: Kiểm tra ngăn xếp đã đầy hay chưa. Thao tác này không phải lúc nào cũng có.

**Size**: Lấy số lượng phần tử stack đang có.



1. **Tìm hiểu giải thuật.**

*Ký pháp ngịch đảo Ba-lan*

Khi lập trình, tính giá trị một biểu thức toán học là điều quá đỗi bình thường. Tuy nhiên, trong nhiều ứng dụng (như chương trình vẽ đồ thị hàm số chẳng hạn, trong đó chương trình cho phép người dùng nhập vào hàm số), ta cần phải tính giá trị của một biểu thức được nhập vào từ bàn phím dưới dạng một chuỗi. Với các biểu thức toán học đơn giản (như a+b) thì bạn có thể tự làm bằng các phương pháp tách chuỗi “thủ công”. Nhưng để “giải quyết” các biểu thức có dấu ngoặc, ví dụ như (a+b)\*c + (d+e)\*f ,  thì các phương pháp tách chuỗi đơn giản đều không khả thi.Trong tình huống này, ta phải dùng đến Ký Pháp Nghịch Đảo Ba Lan Để đơn giản cho việc minh họa, ta giả định rằng chuỗi biểu thức mà ta nhận được từ bàn phím chỉ bao gồm: các dấu mở ngoặc/đóng ngoặc; 4 toán tử cộng, trừ, nhân và chia (+, -, \*, /); các toán hạng đều chỉ là các con số nguyên từ 0 đến 9; không có bất kỳ khoảng trắng nào giữa các ký tự.Quá trình tính toán giá trị của biểu thức hậu tố là đọc biểu thức từ trái sang phải, nếu gặp một toán hạng (con số hoặc biến) thì push toán hạng này vào ngăn xếp; nếu gặp toán tử, lấy hai toán hạng ra khỏi ngăn xếp (stack), tính kết quả, đẩy kết quả trở lại ngăn xếp. Khi quá trình kết thúc thì con số cuối cùng còn lại trong ngăn xếp chính là giá trị của biểu thức đó.

Độ ưu tiên của các toán tử được xắp sếp theo thứ tự: Mức 1 là hai toán tử ‘(‘ và ‘)’, mức 2 là toán tử ‘+’ và ‘-’, mức 3 là toán tử ‘/’ và ‘\*’, mức 4 là toán tử ‘^’.Các toán tử ở cùng mức có độ ưu tiên là như nhau.

Nếu gặp một toán hạng (con số hoặc biến) thì ghi nó vào chuỗi kết quả (chuỗi kết quả là biểu thức trung tố). Nếu gặp dấu mở ngoặc, đưa nó vào ngăn xếp. Nếu gặp một toán tử (gọi là o1 ), thực hiện hai bước sau: Chừng nào còn có một toán tử o2 ở đỉnh ngăn xếp.Và độ ưu tiên của o1 nhỏ hơn hay bằng độ ưu tiên của o2 thì lấy o2 ra khỏi ngăn xếp và ghi vào kết quả, Push o1 vào ngăn xếp.Nếu gặp dấu đóng ngoặc thì cứ lấy các toán tử trong ngăn xếp ra và ghi vào kết quả cho đến khi lấy được dấu mở ngoặc ra khỏi ngăn xếp. Khi đã duyệt hết biểu thức trung tố, lần lượt lấy tất cả toán tử (nếu có) từ ngăn xếp ra và ghi vào chuỗi kết quả.

*Thuật toán chuyển từ trung tố sang hậu tố*

Bước 1: Khởi tạo Stack rỗng.

Bước 2: Khởi tạo 2 chuỗi x và token; i, j lần lượt là index của Infix và Postfix.

Bước 3: Duyệt vòng lặp for từ i = 1 cho đến cuối chuỗi Infix:

* + Nếu Infix[i] là toán hạng thì đưa vào Postfix.
  + 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 Postfix.

*Output: Postfix là biểu thức hậu tố.*

*Tính giá trị biểu thức hậu tố*

* Duyệt biểu thức dạng chuỗi từ trái sang phải.
* Dùng hàm isdigit để kiểm tra.
* Nếu là toán hạng thì dùng Push() đưa vào ngăn xếp S.
* Nếu là toán tử thì Pop() 2 toán hạng trong ngăn xếp S ra, sau đó tính toán giá trị của chúng dựa vào toán tử này, sau đó Push() lại vào S.
* Thực hiện cho đến khi gặp kí tự \0 kết thúc chuỗi.
* Kết quả của biểu thức chính là phần tử còn lại cuối cùng trong ngăn xếp S.

1. **Tìm hiểu công cụ lập trình.**

Công cụ lập trình ta sử dụng ở đây là Visual Studio Code

|  |  |
| --- | --- |
| C:\Users\ASUS\AppData\Local\Temp\ksohtml\wps9BB9.tmp.jpg | Visual Studio Code là một trình soạn thảo mã nguồn nhẹ nhưng mạnh mẽ, chạy trên máy tính để bàn của bạn và có sẵn cho Windows, macOS và Linux. Nó đi kèm với hỗ trợ tích hợp cho JavaScript, TypeScript và Node.js và có một hệ sinh thái mở rộng phong phú cho các ngôn ngữ khác (chẳng hạn như C ++, C #, Java, Python, PHP…) |

Để thực hiện đề tài thực tập này em chọn ngôn ngữ lập trình Javascript. Javascript là một ngôn ngữ lập trình kịch bản dựa vào đối tượng phát triển có sẵn hoặc tự định nghĩa. Javascript được sử dụng rộng rãi trong các ứng dụng Website. Javascript được hỗ trợ hầu như trên tất cả các trình duyệt như Firefox, Chrome, ... trên máy tính lẫn điện thoại.

## Cài đặt thuật toán

1. Mô tả giải thuật

*Chuyển biểu thức tiền tố sang hậu tố*

Để thự hiện bài toán đầu tiên ta tạo một stack rỗng với biến khởi tạo bằng -1

var stackarr = [];

var topp = -1;

Sau đó chúng ta sẽ tạo các hàm sự lý việc Push và Pop các phần tử vào stack để sử dụng cho hàm xử lý phía sau:

Hàm Push() đẩy phần tử vào Stack và tăng biên đếm lên 1 đơn vị

function push(e) {

topp++;

stackarr[topp] = e;

}

Hàm Pop() lấy 1 phần tử trên cùng của Stack ra ngoài trả về giá trị vừa lấy và giảm biến đếm của Stack đi 1 nếu Stack rỗng trả về 0.

function pop() {

if (topp == -1)

return 0;

else {

var popped\_ele = stackarr[topp];

topp--;

return popped\_ele;

}

}

Tiếp theo ta xây dựng hàm kiểm tra, để kiểm tra xem phần tử nhập vào có phải toán tử hay không.

function operator(op) {

if (op == '+' || op == '-' || op == '^' || op == '\*' || op == '/' || op == '(' || op == ')' ) {

return true;

}

Else return false; }

Xây dựng hàm kiểm tra độ ưu tiên của mộ toán tử, giá trị trả về là độ ưu tiên biều hiện bằng số, số càng lớn thì độ ưu tiên càng cao.

function precedency(pre) {

if (pre == '@' || pre == '(' || pre == ')') {

return 1;

}

else if (pre == '+' || pre == '-') {

return 2;

}

else if (pre == '/' || pre == '\*') {

return 3;

}

else if (pre == '^') {

return 4;

}

else

return 0;

}

Sau khi xây dựng các hàm hỗ trợ ta đi vào xây dựng hàm chính là hàm Tính biểu thức đại số.

Đầu tiên ta tạo 1 mảng postfix mảng này chứa biểu thức hậu tố, khới tạo biến bắt đầu temp = 0.

var postfix = [];

var temp = 0;

Sau đó ta Push(phần\_tử\_bất\_kỳ) và để tạo điểm dừng cho truy xuất Stack về sau, lên là ký từ thay vì 1 toán tử hay 1 toán hạng, ở đây em nhập @ và cho độ ưu tiên của nó là 1.

push('@');

Tiếp theo với ngôn ngữ Javascript ta dùng lệnh document.getElementById ("infixvalue").value; để lấy biểu thức đại số do người dùng nhập vào, với id của thẻ Input là id= “infixvalue” và gán nó cho biến infixvalue viết tắt là infixval.

infixval = document.getElementById("infixvalue").value;

Tiếp theo ta tạo vòng lặp chạy từ i đến < infixval.length , lặp qua các phần tử và gán lần lượt phần tử cho biến el.

for (var i = 0; i < infixval.length; i++) {

var el = infixval[i];

…. Các điều kiện

}

Sau đó ta bắt đầu kiểm tra nếu phần tử nhập vào là toán tử:

Ta sẽ tiếp tục kiểm tra nếu nó là dấu ‘)’ ngoặt đóng ta sẽ Pop() từng phần tử trong Stack ra lưu vào mảng postfix với biến temp tăng lên một với mỗi phần từ được Pop vào, ta sẽ Pop cho đến khi gặp dấu ‘(’ ngoặc đóng,sau khi Pop song ra tiếp tuc Pop để xóa ngoặc mở.

if (operator(el)) {

if (el == ')') {

while (stackarr[topp] != "(") {

postfix[temp++] = pop();

}

pop();

}

}

Còn nếu phần tử nhập vào là dấu ‘(’ ngoặc mở thì ta trực tiếp push nó vào Stack.

else if (el == '(') {

push(el);

}

Nếu nó là toán tử ‘+,-,\*,/,^’ ta tiến hành so sánh độ ưu tiên với @ và độ ưu tiên của @ là 1 nên mặc định toán tử đầu tiên nó sẽ được đẩy thằng vào Stack.

else if (precedency(el) > precedency(stackarr[topp])) {

push(el);

}

Và với các phần tử tiếp theo nếu độ ưu tiên của phần tử mới thêm vào vẫn còn lớn hơn độ ưu tiên của các phần tử trong Stack với vị trị so sánh >-1 tức là không so sánh nó với @.Thì ta sẽ Pop nó vào mảng posfix cho đên khi gặp phần tử có độ ưu tien >= phần tử mới thêm vào.Sau khi lấy ra hết ta Push phần tử mới thêm vào mảng.

else {

while (precedency(el) <=precedency(stackarr[topp]) && topp > -1) {

postfix[temp++] = pop();

}

push(el);

}

Sau khi kiểm tra toán tử song nếu phần tử truyền vào không phải toán tử ta trực tiếp gán nó vào mảng posfix.

else {

postfix[temp++] = el;

}

Cuối cùng là gàn nốt những phần tử còn lại trong Stack vào mảng postfix trừ biến @.

while (stackarr[topp] != '@') {

postfix[temp++] = pop();

}

*Tính toán biểu thức đại số*

Đầu tiên ta tạo Stack rỗng, sau đó ta dùng vòng lặp quét qua từng phần tử trong mảng fostfix.

let stack=[];

Dùng vòng lặp duyệt qua từng phần tử của mảng postfix gán các phần tử cho biến c để kiểm tra

for(let i=0;i<exp.length;i++){

let c=exp[i];

…..Các điều kiện và tính toán ở dưới.

}

Sau đó ta bắt đầu kiểm tra phần tử nhập vào đó nếu phần tử đó là số thì phương thức parseInt(<number>) sẽ trả về số đó làm cho hàm isNaN bằng fasle bằng False nhưng dấu ! đạo lại thành True phù hợp điều kiện if. Khi vào if ta tiến hành Push số hay toán hạng đó vào Stack bằng phương thức charCodeAt trả về số thứ tự ở bảng UTF-8 để lấy giá trị thực ta lấy number.charCodeAt(0) – “0”. charCodeAt(0).

if(! isNaN( parseInt(c) ))

stack.push(c.charCodeAt(0) - '0'.charCodeAt(0));

Ngược lại nếu phần tử đưa vào là toán tử, ta tiến hành gắn phần tử thứ n-1 và n-2 vào hai biếm val1 và val2 sau đó tùy thuộc vào toán tử truyền vào ta dùng Swich(toán tử) tiến hành +,-,\*,/ hai biến val1 và val2.

else

{

let val1 = stack.pop();

let val2 = stack.pop();

switch(c)

{

case '+':

stack.push(val2+val1);

break;

case '-':

stack.push(val2- val1);

break;

case '/':

stack.push(val2/val1);

break;

case '\*':

stack.push(val2\*val1);

break;

}

}

}

Cuối dùng ta trả về giá trị đã tính toán trong Stack ra ngoài đó chính là kết quả của biểu thức đại số mà ta cần tính.

return stack.pop();

}

1. Cài đặt bài toán

// Created an empty array - Tạo 1 mảng trống

var stackarr = [];

// Variable topp initialized with -1 ->Biến topp được khởi tạo bằng -1

var topp = -1;

// Push function for pushing -> hàm đẩy các phần tử vào stack

// elements inside stack phần tử bên trong stack

function push(*e*) {

    topp++;

    stackarr[topp] = *e*;

}

// Pop function for returning top element ->Hàm Pop để trả về phần tử trên đầu

function pop() {

    if (topp == -1)

        return 0;

    else {

        var popped\_ele = stackarr[topp];

        topp--;

        return popped\_ele;

    }

}

// Function to check whether the passed ->// Hàm kiểm tra xem có được thông qua hay không

// character is operator or not ->// ký tự có phải là toán tử hay không

function operator(*op*) {

    if (*op* == '+' || *op* == '-' ||

*op* == '^' || *op* == '\*' ||

*op* == '/' || *op* == '(' ||

*op* == ')') {

        return true;

    }

    else

        return false;

}

// Function to return the precedency of operator ->// Hàm trả về thứ tự ưu tiên của một toán tử

function precedency(*pre*) {

    if (*pre* == '@' || *pre* == '(' || *pre* == ')') {

        return 1;

    }

    else if (*pre* == '+' || *pre* == '-') {

        return 2;

    }

    else if (*pre* == '/' || *pre* == '\*') {

        return 3;

    }

    else if (*pre* == '^') {

        return 4;

    }

    else

        return 0;

}

// Function to convert Infix to Postfix ->// Hàm chuyển Infix thành Postfix

function InfixtoPostfix() {

    // Postfix array created ->tạo mảng postfix.

    var postfix = [];

    var temp = 0;

    push('@');

    infixval = document.getElementById("infixvalue").value;

    // Iterate on infix string -> lặp chuỗi Infix

    for (var i = 0; i < infixval.length; i++) {

        var el = infixval[i];

        // Checking whether operator or not ->Kiểm tra nó có phải toán tử hay không

        if (operator(el)) {

            if (el == ')') {

                while (stackarr[topp] != "(") {

                    postfix[temp++] = pop();

                }

                pop();

            }

            // Checking whether el is ( or not -> kiểm tra phần tử el[i] có phải là "(" hay không

            else if (el == '(') {

                push(el);

            }

            // Comparing precedency of el and -> so sanh độ ưu tiên củab el và stackarr

            // stackarr[topp]

            else if (precedency(el)>

precedency(stackarr[topp])) {

                push(el);

            }

            else {

                while (precedency(el) <=

                    precedency(stackarr[topp]) && topp > -1) {

                    postfix[temp++] = pop();

                }

                push(el);

            }

        }

        else {

            postfix[temp++] = el;

        }

    }

    // Adding character until stackarr[topp] is @ -> Thêm ký tự cho đến khi stackarr = @

    while (stackarr[topp] != '@') {

        postfix[temp++] = pop();

    }

    // String to store postfix expression -> Tạo chuỗi lưu chữ biểu thức hậu tố

    var st = "";

    for (var i = 0; i < postfix.length; i++)

        st += postfix[i];

    // To print postfix expression in HTML

    function evaluatePostfix(*exp*)

{

    //create a stack

        let stack=[];

        // Scan all characters one by one

        for(let i=0;i<*exp*.length;i++)

        {

            let c=*exp*[i];

            // If the scanned character is an operand (number here),

            // push it to the stack.

            if(! isNaN( parseInt(c)   ))

            stack.push(c.charCodeAt(0) - '0'.charCodeAt(0));

            //  If the scanned character is an operator, pop two

            // elements from stack apply the operator

            else

            {

                let val1 = stack.pop();

                let val2 = stack.pop();

                switch(c)

                {

                    case '+':

                    stack.push(val2+val1);

                    break;

                    case '-':

                    stack.push(val2- val1);

                    break;

                    case '/':

                    stack.push(val2/val1);

                    break;

                    case '\*':

                    stack.push(val2\*val1);

                    break;

              }

            }

        }

        return stack.pop();

}

// Driver program to test above functions

let exp = st;//"231\*+9-";

document.getElementById("text").innerHTML = st;

document.getElementById("text1").innerHTML = evaluatePostfix(exp);

}

// This code is contributed by avanitrachhadiya2155

class TinhBTDS {

    constructor(*name*){

*this*.name = *name*;

    }

    BTDS() {

        var infixval = document.getElementById("infixvalue").value;

        var stackarr = [];

        var topp = -1

// Push phần tử vào Stack và tăng biến đếm lên 1 đơn vị

function push(*e*) {}

// Pop lấy ra 1 phần tử đầu stack giảm biển đếm xuống 1 đơn vị

// và trả về giá trị biến đã lấy.

function pop() {}

// hàm kiểm tra phần tử truyền vào có phải toán tử không

function precedency(*pre*) {}

// Hàm chuyển đổi biểu thức tiền tố sang hậu tố

function InfixtoPostfix() {}

//Tính toán biểu thức đại số

function evaluatePostfix(*exp*){}

let exp = InfixtoPostfix();

        document.getElementById("text").innerHTML = InfixtoPostfix();

        document.getElementById("text1").innerHTML = evaluatePostfix(exp);

var ob = **new** TinhBTDS ('2\*3(-6+4)\*5')

ob.getName()

ob.BTDS()

# CHƯƠNG III. KẾT QUẢ NGHIÊN CỨU

Xây dựng thuật toán bằng Javascript và mô tả thuật toán trên web form.Yêu cầu người dùng nhập vào một biểu thức đại số bất kỳ. Sau đó thuật toán sẽ xử lý và in ra biểu thức hậu tố, sau đó từ biểu thức hậu tố đó tính toán và in giá kết quả của biểu thức đại số.

### Code index.html

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *http-equiv*="X-UA-Compatible" *content*="IE=edge">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <script *src*="../Code\_TTCS/design.js"></script>

    <link *rel*="stylesheet" *href*="../Code\_TTCS/style.css">

    <title>Convert infix to postfix</title>

</head>

<body>

    <nav *class*="navbar navbar-light bg-light">

        <div *class*="container-fluid"

*style*="justify-content: center;">

            <span *class*="navbar-brand mb-0 h1">

                Infix to Postfix Converter in JavaScript

            </span>

        </div>

    </nav>

    <div *class*="inputdata">

        <!-- Taking infix expression as input -->

        <span *class*="span1">

            Enter Valid Infix Expression

        </span>

        <br>

        <input *type*="text" *placeholder*="Infix Expression"

*id*="infixvalue" *class*="input1">

        <br>

        <!-- InfixtoPostfix() function call-->

        <button *onclick*="InfixtoPostfix()" *class*="btn1">

            Convert</button>

    </div>

    <br>

    <div *class*="output">

        <!-- Postfix expression printed -->

        <span *style*="font-size:1.4vw;  font-size: 2.2vw;">

            Postfix Expression:-

            <span *id*="text"

*style*="color: black; font-weight: 600;">

            </span>

        </span>

    </div>

    <div *class*="output1">

        <span *style*="font-size:1.4vw;  font-size: 2.2vw;">

            Evaluate Postfix =

            <span *id*="text1"

*style*="color: black; font-weight: 600;">

            </span>

        </span>

    </div>

</body>

</html>

### Code style.css

body {

    background-color: #fff0c3;

}

*.container-fluid* {

    background-color: #6f459e;

    justify-content: center;

}

*.navbar-brand* mb-0 h1 {

    color: black;

}

*.navbar-light* *.navbar-brand* {

    color: white;

}

*.navbar* navbar-light bg-light {

    width: 100%;

}

*.navbar* {

    padding: 0%;

}

*.navbar-brand* {

    font-size: 1.8rem;

}

*.navbar-light* *.navbar-brand:hover* {

    color: white;

}

*.inputdata* {

    text-align: center;

    margin-top: 21vh;

}

*.span1* {

    font-size: 2vw;

    font-weight: 500;

    color: black;

}

*.input1* {

    width: 58vw;

    text-align: center;

    height: 3vw;

    place-items: center;

    font-size: 20px;

    border: 2px solid white;

    border-radius: 18px;

    margin-top: 2vw;

}

*.btn1* {

    border: wheat;

    background-color: #6f459e;

    text-align: center;

    color: white;

    font-weight: 500;

    border-radius: 8px;

    margin-top: 1.5vw;

    width: 7vw;

    height: 6vh;

}

button*:hover* {

    background-color: #c694ff;

}

input*:focus* {

    outline: none;

}

*.output* {

    text-align: center;

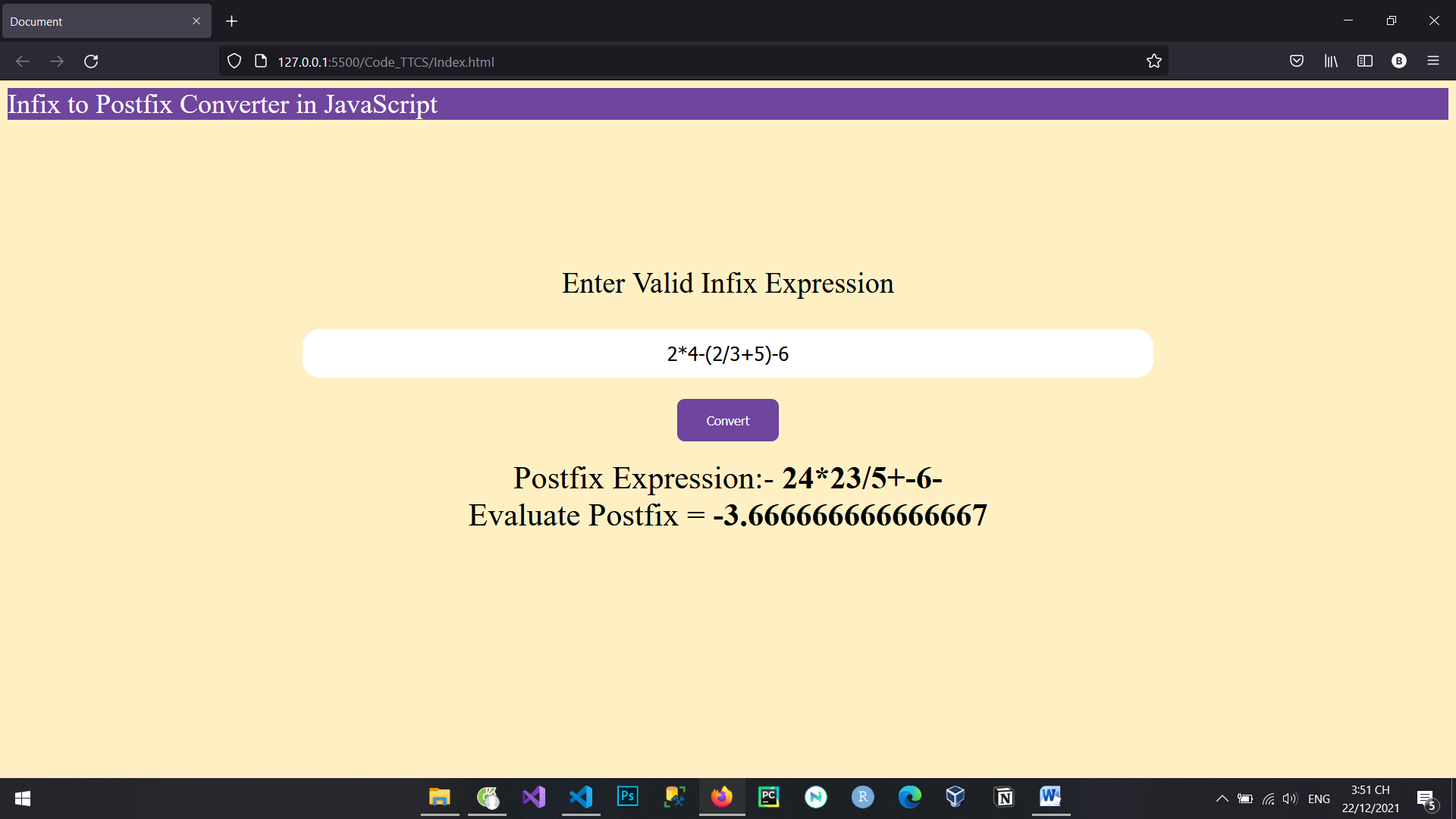
}

*.output1* {

    text-align: center;

}

### Giao diện Web



# CHƯƠNG IV. KẾT LUẬN

# link github

https://github.com/Jemmyn1206/TTCS.git