



รายงาน

เรื่อง โปรแกรม Convert infix to postfix by assigned in variable

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

รายงาน

เรื่อง โปรแกรม Convert infix to postfix by assigned in variable

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา ENGCE124 โครงสร้างข้อมูลและขั้นตอนวิธี (Data Structures and Algorithms) หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์ สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่ ในระดับปริญญาตรีปีที่ 2 โดยมีจุดประสงค์ในการอธิบายโค้ดของโปรแกรม Convert infix to postfix by assigned in variable รวมถึงอธิบายหลักการทำงานของโปรแกรม Convert infix to postfix by assigned in variable และอธิบายผลลัพธ์การใช้งานโปรแกรม Convert infix to postfix by assigned in variable

ผู้จัดทำรายงานหวังว่า รายงานฉบับนี้จะเป็นประโยชน์กับผู้สนใจ หรือนักศึกษาทุกท่านที่กำลังหาศึกษาในหัวข้อของโปรแกรม Convert infix to postfix by assigned in variable หากมีข้อเสนอแนะหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้ และขอภัยมา ณ ที่นี้

ผู้จัดทำ

นายพงษ์พันธุ์ เลาวพงศ์

วันที่ 31/07/2567

สารบัญ

	หน้า
คำนำ	ก
สารบัญ	๗
โค้ดของโปรแกรม Convert infix to postfix by assigned in variable พร้อมคำอธิบาย	1
หลักการทำงานของโปรแกรม Convert infix to postfix by assigned in variable	8
ผลลัพธ์การใช้งานโปรแกรม Convert infix to postfix by assigned in variable	13
บรรณานุกรม	14

โค้ดของโปรแกรม Convert infix to postfix by assigned in variable พร้อมคำอธิบาย

```

/*
Program convert infix to postfix by assigned in variable.
=====
*/
#include <stdio.h> // ใช้ฟังก์ชัน printf()
#include <conio.h> // ใช้ฟังก์ชัน getch()
#include <string.h> // ใช้ฟังก์ชันเกี่ยวกับสตริง
#define MaxStack 40 // กำหนดขนาดสูงสุดของสแตก
char infix1[80] = {"A+B*(C^D*E/F)-G"}; // กำหนดนิพจน์อินฟิกซ์
char OpSt[MaxStack]; // ขนาดสแตกสำหรับเก็บโอเปอเรเตอร์
int SP = 0; // กำหนดค่าเริ่มต้นของตัวชี้สแตก (Stack Pointer, SP) เป็น 0
void push(char oper) // ฟังก์ชัน PUSH
{
    if (SP == MaxStack) // ตรวจสอบว่าสแตกเต็มหรือไม่
    {
        printf("ERROR STACK OVER FLOW!!!...\n"); // แสดงข้อผิดพลาดเมื่อสแตกเต็ม
    }
    else
    {
        SP = SP + 1; // เพิ่มค่า SP
        OpSt[SP] = oper; // ใส่ข้อมูลลงในสแตก
    }
}

```

โค้ดของโปรแกรม Convert infix to postfix by assigned in variable พร้อมคำอธิบาย (ต่อ)

```
int pop() // ฟังก์ชัน POP
{
    char oper;
    if (SP != 0) // ตรวจสอบว่าสแต็กไม่ว่างเปล่า
    {
        oper = OpSt[SP]; // ดึงข้อมูลจากสแต็ก
        SP--; // ลดค่า SP
        return (oper); // คืนค่าข้อมูล
    }
    else
        printf("\nERROR STACK UNDER FLOW!!!...\n"); // แสดงข้อผิดพลาด
        เมื่อสแต็กว่างเปล่า
}

int precedenceIP(char oper) // ฟังก์ชันสำหรับตรวจสอบลำดับความสำคัญของโอเปอเรเตอร์ที่
รับเข้ามา
{
    switch (oper)
    {
        case '+': return (1);
        case '-': return (1);
        case '*': return (2);
        case '/': return (2);
        case '^': return (4);
        case '(': return (4);
    }
}
```

โค้ดของโปรแกรม Convert infix to postfix by assigned in variable พร้อมคำอธิบาย (ต่อ)

```
int precedenceST(char oper) // ฟังก์ชันสำหรับตรวจสอบลำดับความสำคัญของโอเปอเรเตอร์ในสแตก
{
    switch (oper)
    {
        case '+': return (1);
        case '-': return (1);
        case '*': return (2);
        case '/': return (2);
        case '^': return (3);
        case '(': return (0);
    }
}

void infixTPostfix(char infix2[80])
{
    int i, j, len;
    char ch, temp;

    printf("INFIX : %s\n ", infix2); // แสดงนิพจน์อินฟิกซ์

    len = strlen(infix2); // หาความยาวของนิพจน์อินฟิกซ์

    printf("Infix Length = %d \n", len); // แสดงความยาวของนิพจน์อินฟิกซ์

    printf("POSTFIX IS : ");

    for (i = 0; i <= len - 1; i++) // แยกนิพจน์อินฟิกซ์
    {
        ch = infix2[i]; // ถ่ายโอนตัวอักษรไปยังตัวแปร ch

        if (strchr("+-* /^()", ch) == 0) // ตรวจสอบว่าเป็นโอเปอเรนด์หรือไม่

            printf("%c", ch); // แสดงผลเป็นโพสต์ฟิกซ์

        else // ถ้าเป็นโอเปอเรเตอร์ ทำงานด้านล่าง
    }
}
```

โค้ดของโปรแกรม Convert infix to postfix by assigned in variable พร้อมคำอธิบาย (ต่อ)

```
{
    if (SP == 0) // ถ้าสแตกว่าง
        push(ch); // ดันลงสแตก
    else
        if (ch != '(') // ถ้าไม่ใช่ '(' ทำงานด้านล่าง
        {
            if (precedenceIP(ch) >
precedenceST(OpSt[SP])) // ถ้าลำดับความสำคัญของอินพุตสูงกว่าที่อยู่ในสแตก
                push(ch); // ดันอินพุตโอเปอเรเตอร์ลงในสแตก
            else
            {
                printf("%c", pop()); // แสดงผลเป็นโพสต์ฟิกซ์
                while (precedenceIP(ch) <=
precedenceST(OpSt[SP]) && (SP != 0)) // ทำงานกว่าลำดับความสำคัญของอินพุตจะสูงกว่าที่อยู่ในสแตก
                    printf("%c", pop()); // แสดงผลเป็นโพสต์ฟิกซ์
                push(ch); // ดันโอเปอเรเตอร์อินพุตลงในสแตก
            }
        }
    else
    {
        temp = pop(); // ดึงโอเปอเรเตอร์จากสแตก
        while ((temp != '(')) // ทำงานถ้ายังไม่เจอ '('
        {
            printf("%c", temp); // แสดงผลเป็นโพสต์ฟิกซ์
            temp = pop(); // ดึงอีกครั้งและวนลูป
        }
    }
}
```


โค้ดของโปรแกรม Convert infix to postfix by assigned in variable พร้อมคำอธิบาย (ต่อ)

```

    }

    j = SP; // ใช้ j นับโอเปอเรเตอร์ที่เหลือในสแตก

    for (i = 1; i <= j; i++) // ดึงโอเปอเรเตอร์ที่เหลือออกจากสแตก

        printf("%c", pop()); // แสดงผลเป็นโพสต์ฟิกซ์

}

int main()
{
    printf("INFIX to POSTFIX CONVERSION PROGRAM\n");
    printf("=====\n");
    infixTPostfix(infix1);
    getch();
    return (0);
} // สิ้นสุด MAIN

```

หลักการทำงานของโปรแกรม Convert infix to postfix by assigned in variable

1. การนำเข้าไลบรารีและการกำหนดค่าตัวแปร

```
#include <stdio.h> // ใช้ฟังก์ชัน printf()

#include <conio.h> // ใช้ฟังก์ชัน getch()

#include <string.h> // ใช้ฟังก์ชันเกี่ยวกับสตริง

#define MaxStack 40 // กำหนดขนาดสูงสุดของสแตก

char infix1[80] = {"A+B*(C^D*E/F)-G"}; // กำหนดนิพจน์อินฟิกซ์

char OpSt[MaxStack]; // ขนาดสแตกสำหรับเก็บโอเปอเรเตอร์

int SP = 0; // กำหนดค่าเริ่มต้นของตัวชี้สแตก (Stack Pointer, SP) เป็น 0
```

ในส่วนของการนำเข้าไลบรารีและการกำหนดค่าตัวแปร ได้ทำการนำเข้าไลบรารีที่จำเป็น ได้แก่ stdio.h สำหรับการพิมพ์ข้อความ, conio.h สำหรับการรอรับการกดปุ่ม, และ string.h สำหรับการจัดการสตริง จากนั้นกำหนดขนาดสูงสุดของสแตกเป็น 40 และนิพจน์อินฟิกซ์ที่ต้องการแปลง พร้อมตัวแปร OpSt สำหรับเก็บโอเปอเรเตอร์ในสแตก และตัวชี้สแตก SP ที่เริ่มต้นเป็น 0 จากการอธิบายข้างต้น สามารถสรุปรายละเอียดได้ ดังนี้

- ไลบรารี stdio.h ใช้สำหรับฟังก์ชัน printf ที่ใช้ในการพิมพ์ข้อความออกทางหน้าจอ
- ไลบรารี conio.h ใช้สำหรับฟังก์ชัน getch ที่ใช้ในการรอรับการกดปุ่ม
- ไลบรารี string.h ใช้สำหรับการจัดการสตริง เช่น การหาความยาวของสตริง
- กำหนดค่าสูงสุดของสแตก (MaxStack) เป็น 40
- กำหนดนิพจน์อินฟิกซ์ (infix1) เป็น "A+B*(C^D*E/F)-G"
- ประกาศสแตก (OpSt) สำหรับเก็บโอเปอเรเตอร์
- ตัวชี้สแตก (SP) เริ่มต้นที่ 0

หลักการการทำงานของโปรแกรม Convert infix to postfix by assigned in variable (ต่อ)

2. ฟังก์ชัน push

```
void push(char oper) // ฟังก์ชัน PUSH
{
    if (SP == MaxStack) // ตรวจสอบว่าสแตกเต็มหรือไม่
    {
        printf("ERROR STACK OVER FLOW!!!...\n"); // แสดงข้อผิดพลาดเมื่อสแตกเต็ม
    }
    else
    {
        SP = SP + 1; // เพิ่มค่า SP
        OpSt[SP] = oper; // ใส่ข้อมูลลงในสแตก
    }
}
```

ฟังก์ชันนี้ใช้ในการดัน (push) โอเปอเรเตอร์ลงในสแตก โดยตรวจสอบว่าสแตกเต็มหรือไม่ หากเต็มจะแสดงข้อผิดพลาด หากไม่เต็ม จะเพิ่มค่า SP และใส่โอเปอเรเตอร์ลงในสแตก จากการอธิบายข้างต้น สามารถสรุปรายละเอียดได้ ดังนี้

- ตรวจสอบว่าสแตกเต็มหรือไม่ ถ้าเต็มจะแสดงข้อความ "ERROR STACK OVER FLOW!!!"
- ถ้าไม่เต็ม จะเพิ่มค่า SP และใส่โอเปอเรเตอร์ลงในสแตก

หลักการการทำงานของโปรแกรม Convert infix to postfix by assigned in variable (ต่อ)

3. ฟังก์ชัน pop

```
int pop() // ฟังก์ชัน POP
{
    char oper;
    if (SP != 0) // ตรวจสอบว่าสแต็กไม่ว่างเปล่า
    {
        oper = OpSt[SP]; // ดึงข้อมูลจากสแต็ก
        SP--; // ลดค่า SP
        return (oper); // คืนค่าข้อมูล
    }
    else
        printf("\nERROR STACK UNDER FLOW!!!...\n"); // แสดงข้อผิดพลาด
        เมื่อสแต็กว่างเปล่า
}
```

ฟังก์ชันนี้ใช้ในการดึง (pop) โอเปอเรเตอร์ออกจากสแต็ก โดยตรวจสอบว่าสแต็กไม่ว่างเปล่า หากไม่ว่างเปล่า จะดึงโอเปอเรเตอร์ออกและลดค่า SP จากนั้นคืนค่าโอเปอเรเตอร์ที่ดึงออกมา หากสแต็กว่างเปล่า จะแสดงข้อผิดพลาด จากการอธิบายข้างต้น สามารถสรุปรายละเอียดได้ ดังนี้

- ตรวจสอบว่าสแต็กไม่ว่างเปล่า ถ้าไม่ว่างเปล่า จะดึงโอเปอเรเตอร์ออกจากสแต็ก ลดค่า SP และคืนค่าโอเปอเรเตอร์ที่ดึงออกมา
- ถ้าสแต็กว่างเปล่า จะแสดงข้อความ "ERROR STACK UNDER FLOW!!!"

หลักการการทำงานของโปรแกรม Convert infix to postfix by assigned in variable (ต่อ)

4. ฟังก์ชัน precedenceIP

```
int precedenceIP(char oper) // ฟังก์ชันสำหรับตรวจสอบลำดับความสำคัญของโอเปอเรเตอร์ที่
รับเข้ามา
{
    switch (oper)
    {
        case '+': return (1);
        case '-': return (1);
        case '*': return (2);
        case '/': return (2);
        case '^': return (4);
        case '(': return (4);
    }
}
```

ฟังก์ชันนี้ใช้ในการตรวจสอบลำดับความสำคัญของโอเปอเรเตอร์ที่รับเข้ามา โดยการใช้การเปรียบเทียบใน switch ว่าโอเปอเรเตอร์ที่เข้ามามีลำดับความสำคัญเท่าใด และคืนค่าลำดับความสำคัญนั้นออกมา จากการอธิบายข้างต้น สามารถสรุปรายละเอียดได้ ดังนี้

- ตรวจสอบลำดับความสำคัญของโอเปอเรเตอร์ที่รับเข้ามา โดยใช้ switch เพื่อคืนค่าลำดับความสำคัญตามโอเปอเรเตอร์

หลักการการทำงานของโปรแกรม Convert infix to postfix by assigned in variable (ต่อ)

5. ฟังก์ชัน precedenceST

```
int precedenceST(char oper) // ฟังก์ชันสำหรับตรวจสอบลำดับความสำคัญของโอเปอเรเตอร์ในสแตก
{
    switch (oper)
    {
        case '+': return (1);
        case '-': return (1);
        case '*': return (2);
        case '/': return (2);
        case '^': return (3);
        case '(': return (0);
    }
}
```

ฟังก์ชันนี้คล้ายกับ precedenceIP แต่ใช้ในการตรวจสอบลำดับความสำคัญของโอเปอเรเตอร์ที่อยู่ในสแตก โดยการใช้การเปรียบเทียบใน switch ว่าโอเปอเรเตอร์ในสแตกมีลำดับความสำคัญเท่าใด และคืนค่าลำดับความสำคัญนั้นออกมา

- ตรวจสอบลำดับความสำคัญของโอเปอเรเตอร์ในสแตก โดยใช้ switch เพื่อคืนค่าลำดับความสำคัญตามโอเปอเรเตอร์

หลักการการทำงานของโปรแกรม Convert infix to postfix by assigned in variable (ต่อ)

6. ฟังก์ชัน infixTOpostfix

```
void infixTOpostfix(char infix2[80])
{
    int i, j, len;
    char ch, temp;

    printf("INFIX : %s\n ", infix2); // แสดงนิพจน์อินฟิกซ์

    len = strlen(infix2); // หาความยาวของนิพจน์อินฟิกซ์

    printf("Infix Length = %d \n", len); // แสดงความยาวของนิพจน์อินฟิกซ์

    printf("POSTFIX IS : ");

    for (i = 0; i <= len - 1; i++) // แยกนิพจน์อินฟิกซ์
    {
        ch = infix2[i]; // ถ่ายโอนตัวอักษรไปยังตัวแปร ch

        if (strchr("+-* / ^()", ch) == 0) // ตรวจสอบว่าเป็นโอเปอเรนด์หรือไม่

            printf("%c", ch); // แสดงผลเป็นโพสต์ฟิกซ์

        else // ถ้าเป็นโอเปอเรเตอร์ ทำงานด้านล่าง
        {
            if (SP == 0) // ถ้าสแต็กว่าง

                push(ch); // ดันลงสแต็ก

            else

                if (ch != ')') // ถ้าไม่ใช่ ')' ทำงานด้านล่าง
                {
                    if (precedenceIP(ch) >
precedenceST(OpSt[SP])) // ถ้าลำดับความสำคัญของอินพุตสูงกว่าที่อยู่ในสแต็ก

                        push(ch); // ดันอินพุตโอเปอเรเตอร์ลงในสแต็ก

                    else

                        {
```

หลักการการทำงานของโปรแกรม Convert infix to postfix by assigned in variable (ต่อ)

6. ฟังก์ชัน infixTOpostfix (ต่อ)

```

        printf("%c", pop()); // แสดงผลเป็นโพสต์ฟิกซ์

        while (precedenceIP(ch) <=
precedenceST(OpSt[SP]) && (SP != 0)) // ทำจนกว่าลำดับความสำคัญของอินพุตจะสูงกว่าที่อยู่ในสแตก

        printf("%c", pop()); // แสดงผลเป็นโพสต์ฟิกซ์

        push(ch); // ดันโอเปอเรเตอร์อินพุตลงในสแตก

    }

}

else
{

    temp = pop(); // ดึงโอเปอเรเตอร์จากสแตก

    while ((temp != '(')) // ทำงานถ้ายังไม่เจอ '('

    {

        printf("%c", temp); // แสดงผลเป็นโพสต์ฟิกซ์

        temp = pop(); // ดึงอีกครั้งและวนลูป

    }

}

}

j = SP; // ใช้ j นับโอเปอเรเตอร์ที่เหลือในสแตก

for (i = 1; i <= j; i++) // ดึงโอเปอเรเตอร์ที่เหลือออกจากสแตก

    printf("%c", pop()); // แสดงผลเป็นโพสต์ฟิกซ์

}

```

ฟังก์ชันนี้เป็นฟังก์ชันหลักที่ทำการแปลงนิพจน์อินฟิกซ์เป็นโพสต์ฟิกซ์ โดยเริ่มจากการหาความยาวของนิพจน์อินฟิกซ์ จากนั้นทำการวนลูปแต่ละตัวอักษรในนิพจน์อินฟิกซ์ หากตัวอักษรเป็นโอเปอเรนด์ จะแสดงผลทันที หากเป็นโอเปอเรเตอร์ จะทำการดันลงสแตกหรือดึงออกจากสแตกตามลำดับความสำคัญ จนกระทั่งนิพจน์อินฟิกซ์ถูกแปลงทั้งหมด จากนั้นจะแสดงผลโอเปอเรเตอร์ที่เหลือในสแตก

หลักการทำงานของโปรแกรม Convert infix to postfix by assigned in variable (ต่อ)

7. ฟังก์ชัน main

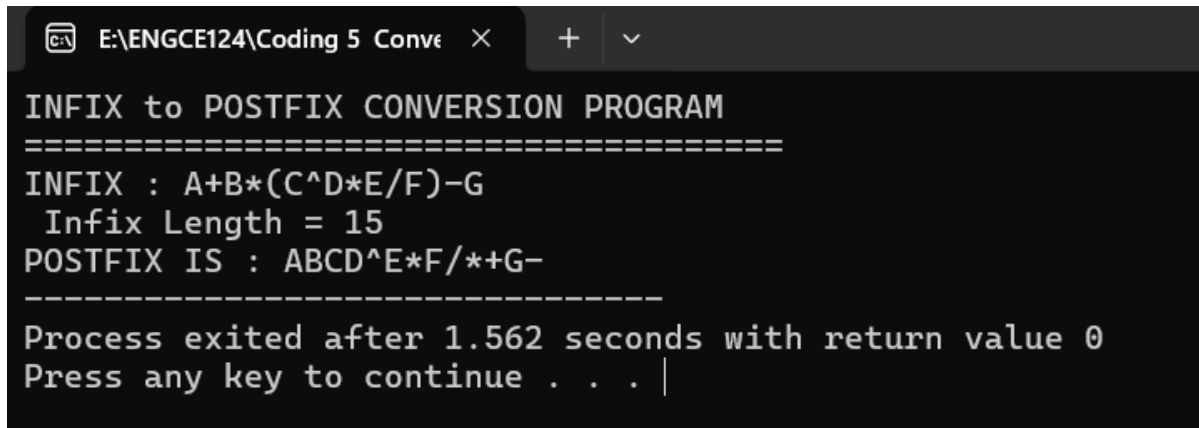
```
int main()
{
    printf("INFIX to POSTFIX CONVERSION PROGRAM\n");
    printf("=====\n");
    infixTPostfix(infix1);
    getch();
    return (0);
} // สิ้นสุด MAIN
```

ฟังก์ชัน main เป็นฟังก์ชันหลักของโปรแกรมที่ทำการเรียกใช้ฟังก์ชัน infixTPostfix เพื่อแปลงนิพจน์อินฟิกซ์ที่กำหนดไว้ จากนั้นรอรับการกดปุ่มเพื่อสิ้นสุด จากการอธิบายข้างต้น สามารถสรุปรายละเอียดได้ ดังนี้

- เรียกใช้ฟังก์ชัน infixTPostfix เพื่อแปลงนิพจน์อินฟิกซ์เป็นโพสต์ฟิกซ์
- รอรับการกดปุ่มเพื่อสิ้นสุดโปรแกรม

ผลลัพธ์การใช้งานโปรแกรม Convert infix to postfix by assigned in variable

ผลลัพธ์การใช้งานโปรแกรม Convert infix to postfix by assigned in variable พร้อมคำอธิบายการทำงานของโปรแกรม



```

E:\ENGCE124\Coding 5 Conve X + v
INFIX to POSTFIX CONVERSION PROGRAM
=====
INFIX : A+B*(C^D*E/F)-G
Infix Length = 15
POSTFIX IS : ABCD^E*F/*+G-
-----
Process exited after 1.562 seconds with return value 0
Press any key to continue . . . |
  
```

ผลลัพธ์ของโปรแกรมคือการแปลงนิพจน์ในรูปแบบอินฟิกซ์เป็นโพสต์ฟิกซ์ (Reverse Polish Notation) และแสดงผลลัพธ์ดังกล่าวออกมา โดยสำหรับนิพจน์อินฟิกซ์ที่กำหนดคือ "A+B*(C^D*E/F)-G" โปรแกรมจะทำการแปลงเป็นรูปแบบโพสต์ฟิกซ์ตามขั้นตอนต่อไปนี้

1. อ่านตัวอักษร 'A' ซึ่งเป็นโอเปอเรนด์ จะแสดงผล 'A'
2. อ่านเครื่องหมาย '+' ซึ่งเป็นโอเปอเรเตอร์ ดันลงสแตก
3. อ่านตัวอักษร 'B' ซึ่งเป็นโอเปอเรนด์ จะแสดงผล 'B'
4. อ่านเครื่องหมาย '*' ซึ่งเป็นโอเปอเรเตอร์ เปรียบเทียบลำดับความสำคัญกับเครื่องหมาย '+' ในสแตก พบว่ามีลำดับความสำคัญสูงกว่า จึงดันลงสแตก
5. อ่านตัวอักษร 'C' ซึ่งเป็นโอเปอเรนด์ จะแสดงผล 'C'
6. อ่านเครื่องหมาย '^' ซึ่งเป็นโอเปอเรเตอร์ เปรียบเทียบลำดับความสำคัญกับเครื่องหมาย '*' ในสแตก พบว่ามีลำดับความสำคัญสูงกว่า จึงดันลงสแตก
7. อ่านตัวอักษร 'D' ซึ่งเป็นโอเปอเรนด์ จะแสดงผล 'D'
8. อ่านเครื่องหมาย '/' ซึ่งเป็นโอเปอเรเตอร์ เปรียบเทียบลำดับความสำคัญกับเครื่องหมาย '^' ในสแตก พบว่ามีลำดับความสำคัญต่ำกว่า จึงดึง '^' ออกมาจากสแตกและแสดงผล จากนั้นเปรียบเทียบกับเครื่องหมาย '/' ที่เหลือในสแตก และดัน '*' ลงสแตก
9. อ่านตัวอักษร 'E' ซึ่งเป็นโอเปอเรนด์ จะแสดงผล 'E'
10. อ่านเครื่องหมาย '/' ซึ่งเป็นโอเปอเรเตอร์ เปรียบเทียบลำดับความสำคัญกับเครื่องหมาย '/' ในสแตก พบว่ามีลำดับความสำคัญเท่ากัน จึงดึง '/' ออกมาจากสแตกและแสดงผล จากนั้นดัน '/' ลงสแตก

11. อ่านตัวอักษร 'F' ซึ่งเป็นโอเปอเรนด์ จะแสดงผล 'F'
12. อ่านเครื่องหมาย ')' ซึ่งเป็นวงเล็บปิด จะดึงโอเปอเรเตอร์จากสแตกและแสดงผล จนกว่าจะเจอวงเล็บเปิด '('
13. อ่านเครื่องหมาย '-' ซึ่งเป็นโอเปอเรเตอร์ เปรียบเทียบลำดับความสำคัญกับเครื่องหมาย '+' ในสแตก พบว่ามีลำดับความสำคัญเท่ากัน จึงดึง '+' ออกมาจากสแตกและแสดงผล จากนั้นดัน '-' ลงสแตก
14. อ่านตัวอักษร 'G' ซึ่งเป็นโอเปอเรนด์ จะแสดงผล 'G'

หลังจากอ่านครบทุกตัวอักษรในนิพจน์อินฟิกซ์ จะดึงโอเปอเรเตอร์ที่เหลือในสแตกและแสดงผล

บรรณานุกรม

ChatGPT. (-). "Implementing an Infix to Postfix Expression Conversion Algorithm Using C Programming. สืบค้น 31 กรกฎาคม 2567, จาก <https://chatgpt.com/>