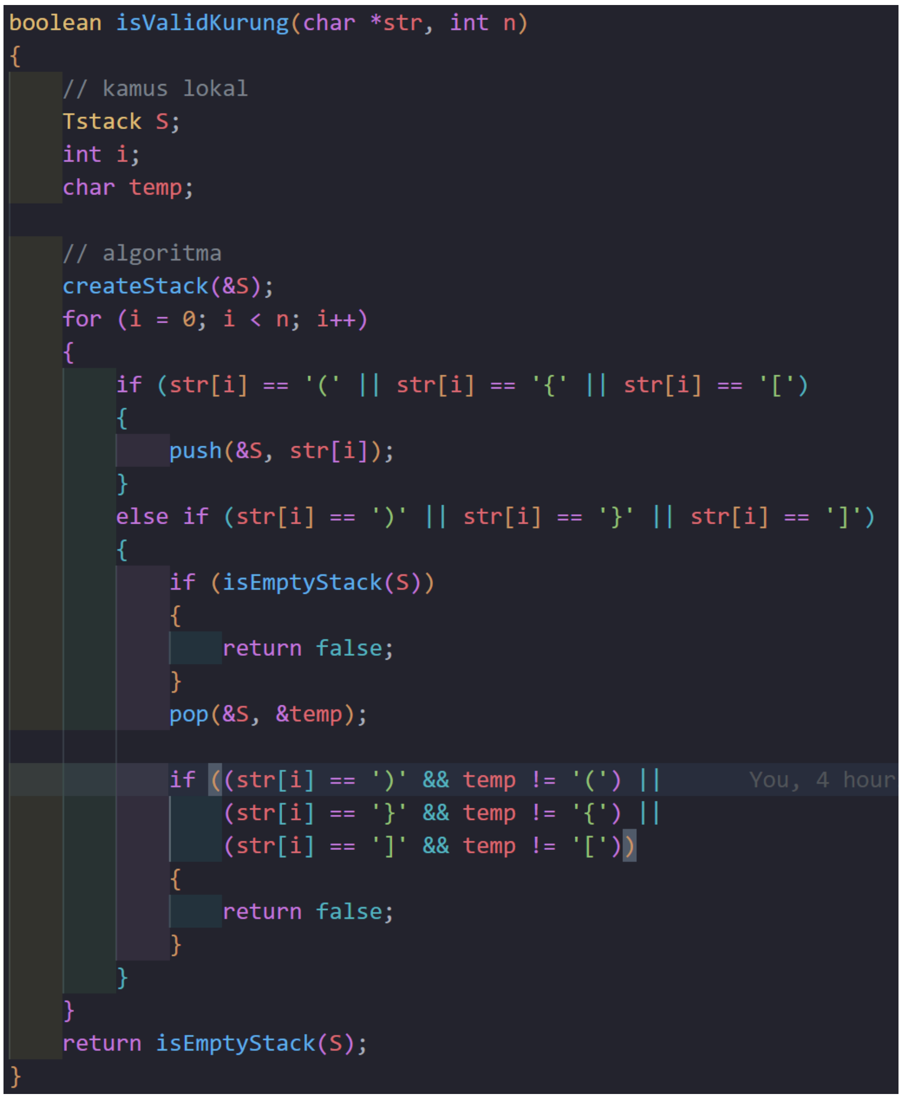
NOTASI ALGORITMIK
APLIKASI STACK

Nama Anggota Kelompok 7:

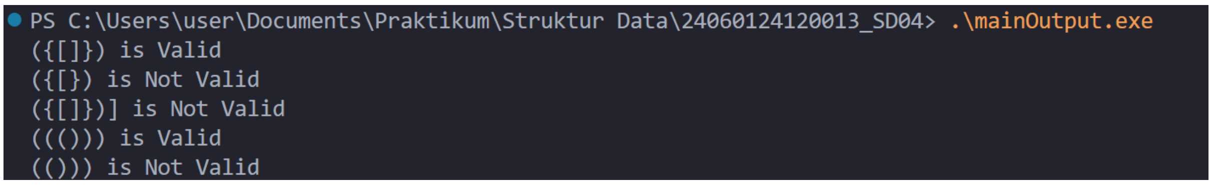
1. Muchammad Yuda Tri Ananda (24060124110142)
2. Muhammad Kemal Faza (24060124120013)
3. Nadia Azura Nurhaniya (24060124120019)
4. Muhammad Zaidaan Ardiyansyah (24060124140200)
5. Muhammad Farhan Abdul Azis (24060124

1. Parantheses Chekcker

```
                              isValidKurung

        DEKLARASI/DEFINISI & SPESIFIKASI TIPE & PROTOTIPE
function isValidKurung(kata:string) → boolean
{I.S.: kata terdefinisi}
{F.S.: Mengembalikan true jika seluruh tanda kurung di kata
sesuai pasangan}
{Proses:
  - Jika kurung buka → push ke stack
  - Jika kurung tutup → pop dari stalalu cekck  pasangannya
  - Jika tidak sesuai → return false
  - Setelah semua karakter diproses, jika stack kosong → return
true}
{Contoh:
  kata = "{[(A+B)]}" → true
  kata = "{[(A+B)]"  → false}

                     BODY/REALISASI PROTOTIPE
function isValidKurung(kata: string, n: integer) → boolean
Kamus Lokal
      i: integer
      S: Tstack
      temp : character

Algoritma
      createStack(S)
      i traversal [1..n]
         if str[i] = '(' or str[i] = '{' or str[i] = '[' then
             push(S, str[i])
         else if str[i] = ')' or str[i] = '}' or str[i] = ']'
      then
             if isEmptyStack(S) then
                → false
             {endif}

             pop(S, temp)
             if (str[i] = ')' and temp ≠ '(') or
                (str[i] = '}' and temp ≠ '{') or
                (str[i] = ']' and temp ≠ '[') then
                 → false
                {endif}
            {endif}
        {endtraversal}

        → isEmptyStack(S)
```



```c
boolean isValidKurung(char *str, int n)
{
    // kamus lokal
    Tstack S;
    int i;
    char temp;

    // algoritma
    createStack(&S);
    for (i = 0; i < n; i++)
    {
        if (str[i] == '(' || str[i] == '{' || str[i] == '[')
        {
            push(&S, str[i]);
        }
        else if (str[i] == ')' || str[i] == '}' || str[i] == ']')
        {
            if (isEmptyStack(S))
            {
                return false;
            }
            pop(&S, &temp);

            if ((str[i] == ')' && temp != '(') ||       You, 4 hour
                (str[i] == '}' && temp != '{') ||
                (str[i] == ']' && temp != '['))
            {
                return false;
            }
        }
    }
    return isEmptyStack(S);
}
```

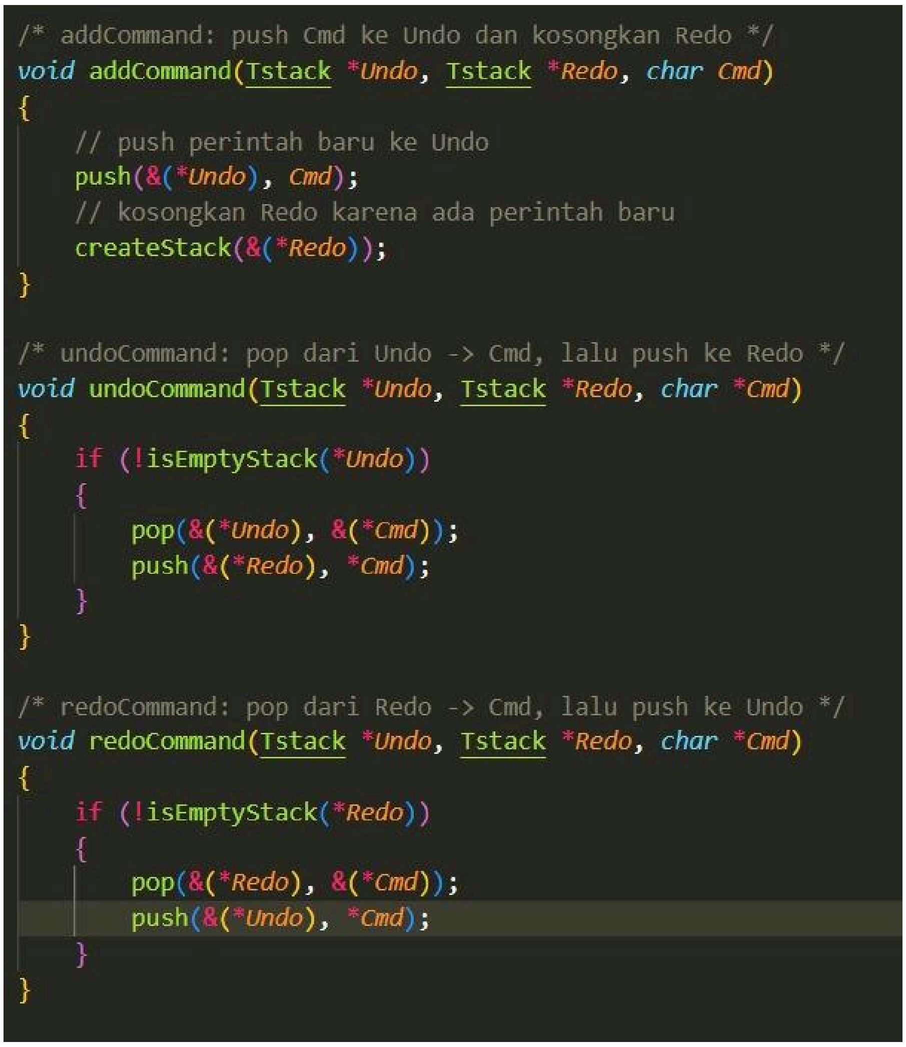


```
PS C:\Users\user\Documents\Praktikum\Struktur Data\24060124120013_SD04> .\mainOutput.exe
({[]}) is Valid
({[}) is Not Valid
({[]})] is Not Valid
((())) is Valid
(())) is Not Valid
```

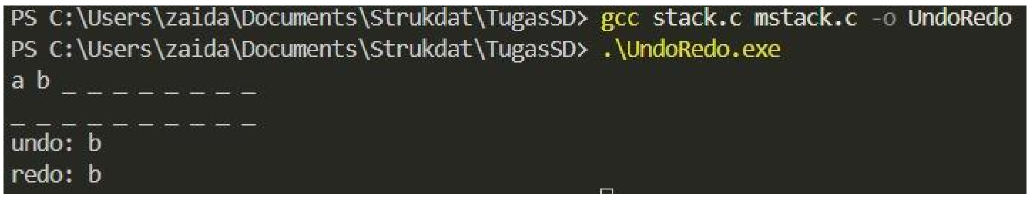
2. Undo-Redo pada aplikasi Text Editor

```
                              Undo-Redo

        DEKLARASI/DEFINISI & SPESIFIKASI TIPE & PROTOTIPE

procedure addCommand (input/output Undo: Tstack, Redo: Tstack,
input Cmd: character)
{I.S. : Undo & Redo terdefinisi, Cmd terdefinisi}
{F.S. : Cmd masuk ke Undo (push), Redo dikosongkan (createStack)}
{Proses : push Cmd ke Undo, createStack(Redo)}
{Contoh : Undo=['A'], Cmd='B' → Undo=['A','B'], Redo=[]}

procedure undoCommand (input/output Undo: Tstack, Redo: Tstack,
output Cmd: character)
{I.S. : Undo tidak kosong, Redo terdefinisi}
{F.S. : elemen top Undo dipindah ke Redo, Cmd berisi elemen
tersebut}
{Proses : pop dari Undo → Cmd, lalu push ke Redo}
{Contoh : Undo=['A','B','C'], Redo=[] → Undo=['A','B'],
Redo=['C']}

procedure redoCommand (input/output Undo: Tstack, Redo: Tstack,
output Cmd: character)
{I.S. : Redo tidak kosong, Undo terdefinisi}
{F.S. : elemen top pada Redo dipindah ke Undo, Cmd berisi elemen
tersebut}
{Proses : pop dari Redo → Cmd, lalu push ke Undo}
{Contoh : Undo=['A','B'], Redo=['C'] → Undo=['A','B','C'],
Redo=[]}

                     BODY/REALISASI PROTOTIPE

procedure addCommand (input/output Undo: Tstack, Redo: Tstack,
input Cmd: character)
Kamus
    -
Algoritma
    push(Undo, Cmd)
    createStack(Redo)

procedure UndoCommand (input/output Undo: Tstack, Redo: Tstack,
output Cmd: character)
Kamus
    -
Algoritma
    if not (isEmptyStack(Undo)) then
        pop(Undo, Cmd)
        push(Redo, Cmd)


procedure redoCommand(input/output Undo: Tstack, Redo: Tstack,
output Cmd: character)
Kamus
```

```
    -
Algoritma
    if not(isEmptyStack(Redo)) then
        pop(Redo, Cmd)
        push(Undo, Cmd)
```



```c
/* addCommand: push Cmd ke Undo dan kosongkan Redo */
void addCommand(Tstack *Undo, Tstack *Redo, char Cmd)
{
    // push perintah baru ke Undo
    push(&(*Undo), Cmd);
    // kosongkan Redo karena ada perintah baru
    createStack(&(*Redo));
}

/* undoCommand: pop dari Undo -> Cmd, lalu push ke Redo */
void undoCommand(Tstack *Undo, Tstack *Redo, char *Cmd)
{
    if (!isEmptyStack(*Undo))
    {
        pop(&(*Undo), &(*Cmd));
        push(&(*Redo), *Cmd);
    }
}

/* redoCommand: pop dari Redo -> Cmd, lalu push ke Undo */
void redoCommand(Tstack *Undo, Tstack *Redo, char *Cmd)
{
    if (!isEmptyStack(*Redo))
    {
        pop(&(*Redo), &(*Cmd));
        push(&(*Undo), *Cmd);
    }
}
```

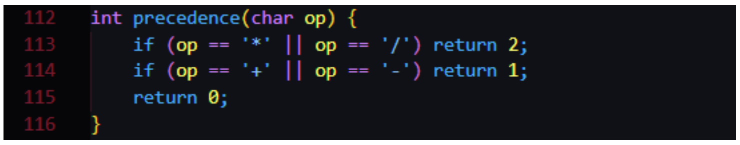


```
PS C:\Users\zaida\Documents\Strukdat\TugasSD> gcc stack.c mstack.c -o UndoRedo
PS C:\Users\zaida\Documents\Strukdat\TugasSD> .\UndoRedo.exe
a b _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
undo: b
redo: b
```

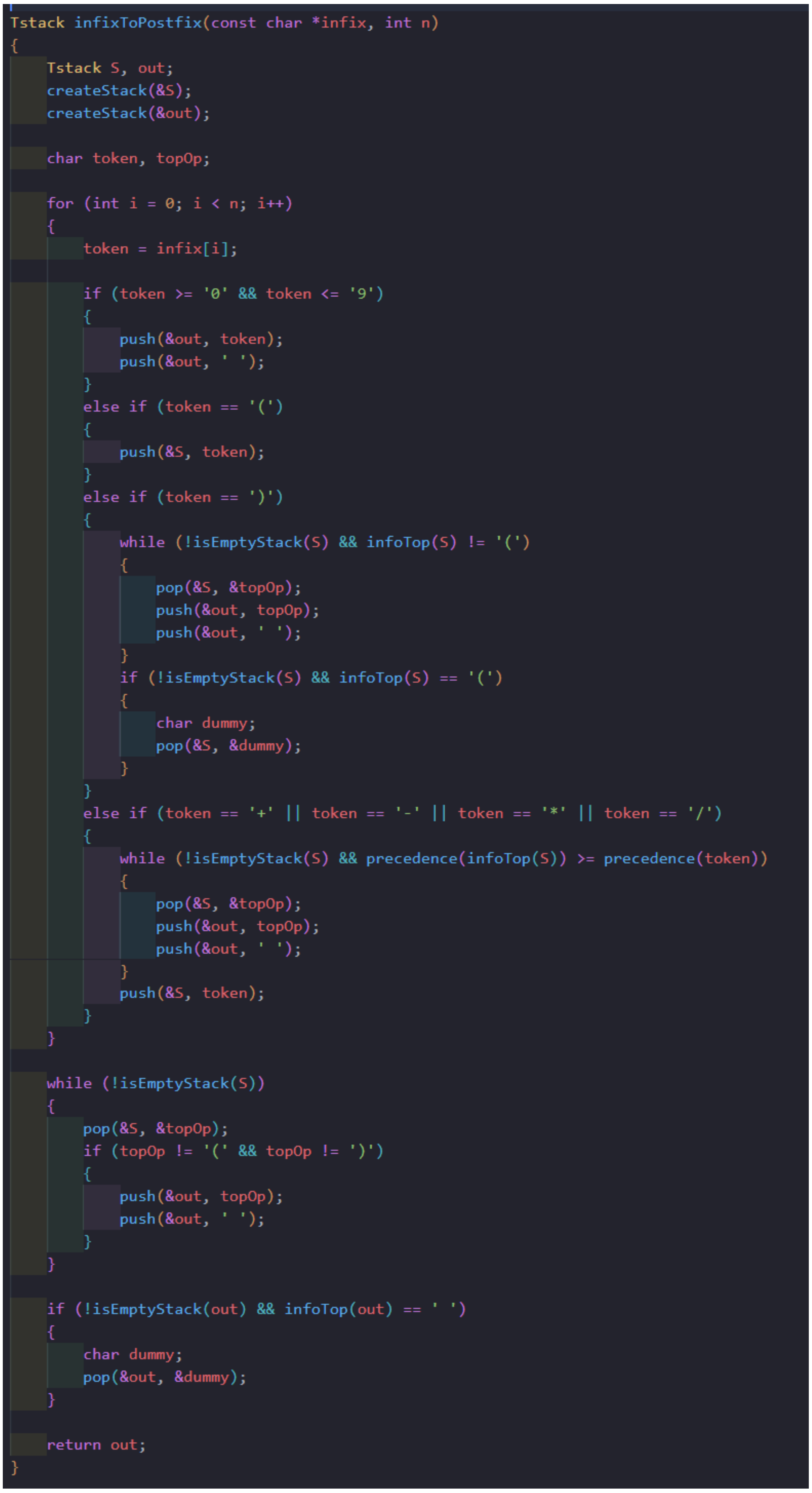
3. Konversi infix expression (operator di tengah) ke dalam postfix expression (opertor di akhir)

infixToPostfix
DEKLARASI/DEFINISI & SPESIFIKASI TIPE & PROTOTIPE procedure infixToPostfix (infix:array [1..10] of character, n: integer, out: Tstack) → character {I.S.:infix terdefinisi, berisi operand(0-9) dan operan(+,-,*,/)} {F.S.:Menghasilkan pointer ke array karakter berisi postfix expression sesuai aturan prioritas operator} {Proses: - Jika token operand → tambahkan langsung ke postfix - Jika token '(' → push ke stack - Jika token ')' → pop semua operator hingga '(' - Jika token operator: • Pop operator di stack selama precedence(top) ≥ precedence(current) • Push operator sekarang ke stack - Setelah semua token diproses → pop semua sisa operator ke postfix} {Contoh: infix = "3+(4*3)/4" → postfix = "3 4 3 * 4 / +" infix = "3*(4+5)/2" → postfix = "3 4 5 + * 2 /" infix = "3+(4*5)-2" → postfix = "3 4 5 * + 2 -"} function precedence(op: character)→ integer {Mengembalikan tingkat prioritas operator}

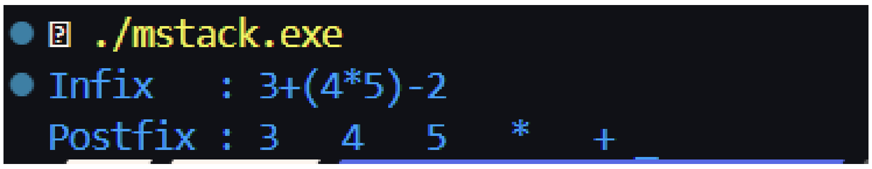
```
                      BODY/REALISASI PROTOTIPE

function precedence(op: character)→ integer
{Mengembalikan tingkat prioritas operator}
Kamus Lokal
   op : character (operator aritmatika)

Algoritma
   if op = '*' or op = '/' then
       → 2
   if op = '+' or op = '-' then
       → 1
   else
       → 0

function infixToPostfix (infix: array [1..10] of character, n:
integer)
Kamus Lokal
   i, n    : integer
   infix   : array [1..10] of character
   postfix : array [1..10] of character
   S, out       : Tstack
   token, topOp, dummy : character

Algoritma
    createStack(S)
    createStack(out)

    i traversal [1..n]
        token ← infix[i]

        if token ≥ '0' and token ≤ '9' then
            push(out, token)
            push(out, ' ')
        else if token = '(' then
            push(S, token)

        else if token = ')' then
            while not (isEmptyStack(S)) and infoTop(S) ≠ '(') do
                pop(S, topOp)
                push(out, topOp)
                push(out, ' ')
            {end while}
            if not (isEmptyStack(S)) and infoTop(S) = '(') then
                pop(S, dummy)
            {endif}

        else if token = '+' or token = '-' or token = '*' or token
= '/'  then
            while not (isEmptyStack(S)) and (precedence(top(S)) ≥
precedence(token)) do
                pop(S, topOp)
                push(out, topOp)
                push(out, ' ')
            {endwhile}
            push(S, token)

        {endif}
    {end traversal}

    while not (isEmptyStack(S)) do
        pop(S, topOp)
        if topOp ≠ '(' and topOp ≠ ')' then
            push(out, topOp)
            push(out, ' ')
        {endif}
    {end while}

    if not (isEmptyStack(out)) and(infoTop(out) = ' ') then
        pop(out, dummy)
    {endif}
    -> out
{endfunction}
```



```
int precedence(char op) {
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0;
}
```



```c
Tstack infixToPostfix(const char *infix, int n)
{
    Tstack S, out;
    createStack(&S);
    createStack(&out);

    char token, topOp;

    for (int i = 0; i < n; i++)
    {
        token = infix[i];

        if (token >= '0' && token <= '9')
        {
            push(&out, token);
            push(&out, ' ');
        }
        else if (token == '(')
        {
            push(&S, token);
        }
        else if (token == ')')
        {
            while (!isEmptyStack(S) && infoTop(S) != '(')
            {
                pop(&S, &topOp);
                push(&out, topOp);
                push(&out, ' ');
            }
            if (!isEmptyStack(S) && infoTop(S) == '(')
            {
                char dummy;
                pop(&S, &dummy);
            }
        }
        else if (token == '+' || token == '-' || token == '*' || token == '/')
        {
            while (!isEmptyStack(S) && precedence(infoTop(S)) >= precedence(token))
            {
                pop(&S, &topOp);
                push(&out, topOp);
                push(&out, ' ');
            }
            push(&S, token);
        }
    }

    while (!isEmptyStack(S))
    {
        pop(&S, &topOp);
        if (topOp != '(' && topOp != ')')
        {
            push(&out, topOp);
            push(&out, ' ');
        }
    }

    if (!isEmptyStack(out) && infoTop(out) == ' ')
    {
        char dummy;
        pop(&out, &dummy);
    }

    return out;
}
```

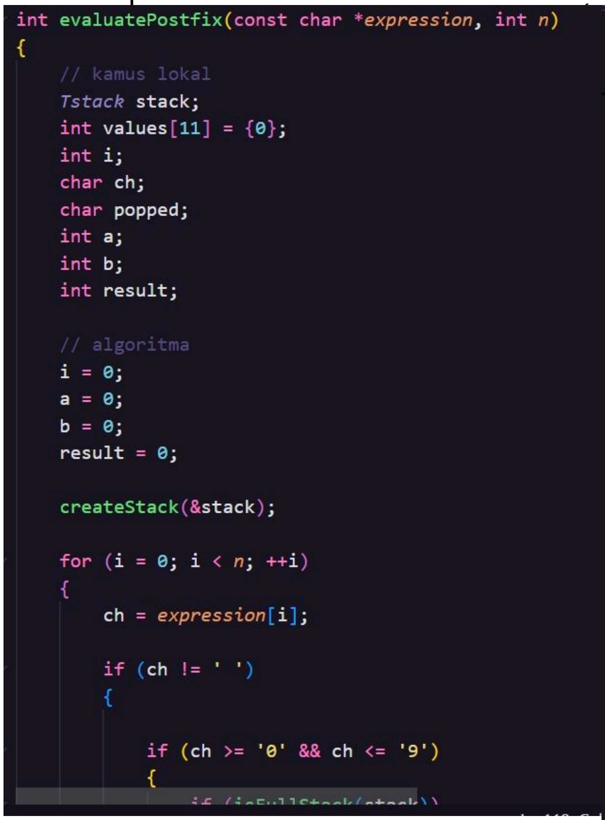


```
./mstack.exe
Infix   : 3+(4*5)-2
Postfix : 3  4  5  *  +
```

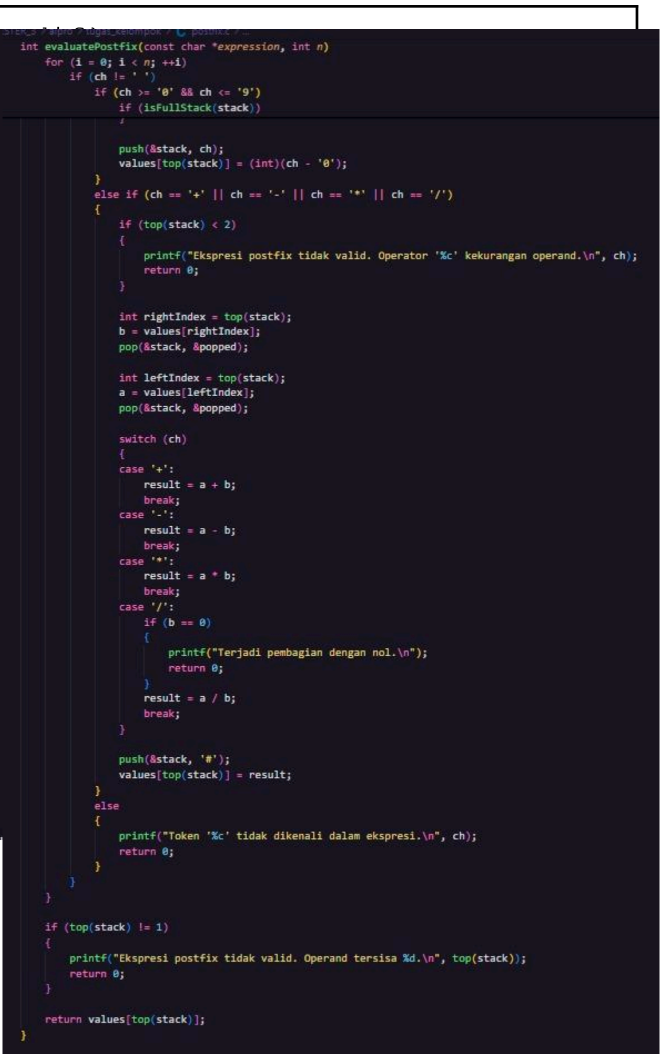
4. Evaluasi postfix expression

```
evaluatePostfix

      DEKLARASI/DEFINISI & SPESIFIKASI TIPE & PROTOTIPE

function evaluatePostfix (postfix:string, n:integer) → real
{I.S.: postfix terdefinisi, berisi operand (0-9) dan
operator (+,-,*,/)}
{F.S.: Menghasilkan nilai hasil evaluasi postfix}
{Proses:
   -  Baca token postfix satu per satu
   -  Jika token operand (angka) → push ke stack
   -  Jika token operator → pop 2 operand (b, a), hitung a
      op b, lalu push hasilnya kembali ke stack
   -  Setelah semua token selesai, elemen terakhir di stack
      adalah hasil akhir}
{Contoh:
  Postfix: "9 3 4 * 8 + 4 / -"
  langkah:
  Push 9
  Push 3
  Push 4
  '*' → pop 4,3 → 3*4=12 → push 12
  Push 8
  '+' → pop 8,12 → 12+8=20 → push 20
  Push 4
  '/' → pop 4,20 → 20/4=5 → push 5
  '-' → pop 5,9 → 9-5=4 → push 4
  Hasil akhir = 4
```

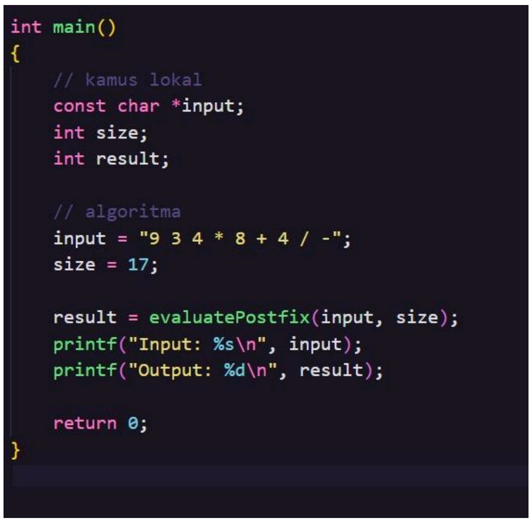
```
                    BODY/REALISASI PROTOTIPE

function evaluatePostfix (postfix: string, n: integer) → real
Kamus Lokal
    stack : Tstack
    values : array[0..10] of integer
    i : integer
    ch, popped : character
    a, b, result : integer

Algoritma
    createStack(stack)
    a ← 0
    b ← 0
    result ← 0

    i traversal [1..n]
        ch ← expression[i]

        if ch ≠ ' ' then
            if ch ≥ '0' and ch ≤ '9' then
                if isFullStack(stack) then
                  output("Stack penuh saat memproses angka")
                    → 0
                {endif}

                push(stack, ch)
                values[top(stack)] ← (ch - '0')

            else if(ch = '+' or ch = '-' or ch = '*' or ch = '/') then
                if top(stack) < 2 then
                    output("Operator kekurangan operand")
                    → 0
                {endif}

                b ← values[top(stack)]
                pop(stack, popped)

                a ← values[top(stack)]
                pop(stack, popped)

                switch(ch)
                    case '+': result ← a + b
                    case '-': result ← a - b
                    case '*': result ← a * b
                    case '/':
                        if b = 0 then
                           output("Terjadi pembagian dengan nol")
                            → 0
                        else
                            result ← a / b
```



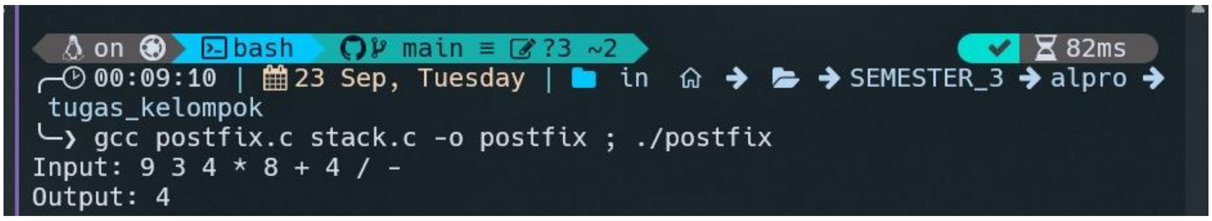
```c
int evaluatePostfix(const char *expression, int n)
{
    // kamus lokal
    Tstack stack;
    int values[11] = {0};
    int i;
    char ch;
    char popped;
    int a;
    int b;
    int result;

    // algoritma
    i = 0;
    a = 0;
    b = 0;
    result = 0;

    createStack(&stack);

    for (i = 0; i < n; ++i)
    {
        ch = expression[i];

        if (ch != ' ')
        {

            if (ch >= '0' && ch <= '9')
            {
```



```c
int evaluatePostfix(const char *expression, int n)
    for (i = 0; i < n; ++i)
        if (ch != ' ')
            if (ch >= '0' && ch <= '9')
                if (isFullStack(stack))

                push(&stack, ch);
                values[top(stack)] = (int)(ch - '0');
            }
            else if (ch == '+' || ch == '-' || ch == '*' || ch == '/')
            {
                if (top(stack) < 2)
                {
                    printf("Ekspresi postfix tidak valid. Operator '%c' kekurangan operand.\n", ch);
                    return 0;
                }

                int rightIndex = top(stack);
                b = values[rightIndex];
                pop(&stack, &popped);

                int leftIndex = top(stack);
                a = values[leftIndex];
                pop(&stack, &popped);

                switch (ch)
                {
                case '+':
                    result = a + b;
                    break;
                case '-':
                    result = a - b;
                    break;
                case '*':
                    result = a * b;
                    break;
                case '/':
                    if (b == 0)
                    {
                        printf("Terjadi pembagian dengan nol.\n");
                        return 0;
                    }
                    result = a / b;
                    break;
                }

                push(&stack, '#');
                values[top(stack)] = result;
            }
            else
            {
                printf("Token '%c' tidak dikenali dalam ekspresi.\n", ch);
                return 0;
            }
        }
    }

    if (top(stack) != 1)
    {
        printf("Ekspresi postfix tidak valid. Operand tersisa %d.\n", top(stack));
        return 0;
    }

    return values[top(stack)];
}
```



```c
int main()
{
    // kamus lokal
    const char *input;
    int size;
    int result;

    // algoritma
    input = "9 3 4 * 8 + 4 / -";
    size = 17;

    result = evaluatePostfix(input, size);
    printf("Input: %s\n", input);
    printf("Output: %d\n", result);

    return 0;
}
```



```
 on   bash   main ≡  ?3 ~2                                      ✓  82ms
╭─ 00:09:10 | 23 Sep, Tuesday |  in  → → SEMESTER_3 → alpro →
 tugas_kelompok
╰─› gcc postfix.c stack.c -o postfix ; ./postfix
Input: 9 3 4 * 8 + 4 / -
Output: 4
```