

UJIAN TENGAH SEMESTER

Untuk memenuhi mata kuliah
Praktikum Struktur Data & Algoritma

OLEH :

NOVIA ANGREINI (2208107010068)

SYIFA SALSABILA (2408107010018)



FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

PROGRAM STUDI INFORMATIKA

UNIVERSITAS SYIAH KUALA

BANDA ACEH

2025

1. PENDAHULUAN

Dalam pemrograman, penggunaan struktur data yang tepat sangat penting untuk meningkatkan efisiensi dan optimasi program. Struktur data seperti **stack**, **queue**, dan **linked list** sering digunakan dalam berbagai aplikasi, termasuk sistem manajemen data, algoritma pencarian, dan pengolahan antrian.

Laporan ini bertujuan untuk menganalisis kode program yang telah dibuat, khususnya dalam aspek fungsi yang digunakan, struktur data yang diterapkan, dan jumlah fungsi yang terdapat dalam program. Dengan melakukan analisis ini, diharapkan dapat memberikan pemahaman yang lebih dalam mengenai implementasi struktur data dalam pemrograman serta bagaimana fungsi-fungsi tersebut bekerja dalam mengelola data secara optimal.

Melalui laporan ini, kita akan mengidentifikasi fungsi yang digunakan dalam kode program, menentukan jenis struktur data yang diimplementasikan, serta menghitung jumlah fungsi yang terdapat dalam kode program.

2. LANDASAN TEORI & ANALISIS PROGRAM

2.1 Ekspresi Aritmatika : Infix, Prefix, Postfix

Ekspresi aritmatika adalah kombinasi dari **angka (operand)**, **operator**, dan **kurung** yang membentuk perhitungan matematika. Ekspresi ini dapat ditulis dalam berbagai bentuk, tergantung pada urutan operator dan operandnya.

Ekspresi aritmatika memiliki tiga bentuk utama:

a. Infix

Infix adalah bentuk ekspresi aritmatika yang umum digunakan dalam matematika, dimana operator berada di antara dua operan

Contoh:

- $A + B$
- $(A + B) * C$
- $A * B + C / D$

b. Prefix

Prefix adalah bentuk ekspresi di mana operator ditempatkan sebelum kedua operandnya.

Contoh:

- $+ A B$ (untuk $A + B$)
- $+ A * B C$ (Untuk $A + (B * C)$)
- $- + A * B C D$ (Untuk $A + (B * C) - D$)

c. Postfix

Postfix adalah bentuk ekspresi di mana operator ditempatkan setelah kedua operandnya.

Contoh:

- $A B +$ (Untuk $A + B$)
- $A B C * +$ (Untuk $A + (B * C)$)
- $A B C * + D -$ (Untuk $A + (B * C) - D$)

Konversi antar ekspresi aritmatika:

Dari	Ke	Metode
Infix	Postfix	Gunakan Stack , baca operand langsung, pindahkan operator sesuai prioritas.

Infix	Prefix	Balik string, konversi ke Postfix , lalu balik kembali.
Postfix	Infix	Gunakan Stack , susun ulang operand dan operator.
Prefix	Infix	Gunakan Stack , susun ulang operand dan operator.
Prefix	Postfix	Gunakan Stack , konversi satu per satu.
Postfix	Prefix	Gunakan Stack , konversi satu per satu.

2.2 STRUKTUR DATA STACK

Stack adalah struktur data **LIFO (Last In, First Out)**, di mana elemen terakhir yang dimasukkan akan menjadi elemen pertama yang keluar. Dalam program ini, **Stack digunakan untuk menyimpan operand dan operator** selama proses konversi ekspresi aritmatika.

Struktur Stack didefinisikan sebagai berikut:

```
// Struktur stack
typedef struct {
    char data[MAX][MAX];
    int top;
} Stack;
```

Operasi utama stack:

- **Push:** Menambahkan elemen ke dalam stack.

```
// Push ke stack
void push(Stack *s, char *str) {
    if (s->top < MAX - 1) {
        strcpy(s->data[++(s->top)], str);
    }
}
```

- **Pop:** Menghapus elemen dari stack

```
// Pop dari stack
void pop(Stack *s, char *str) {
    if (!isEmpty(s)) {
        strcpy(str, s->data[(s->top)--]);
    }
}
```

- **isEmpty:** Mengecek apakah stack kosong.

```
// Cek apakah stack kosong
int isEmpty(Stack *s) {
    return s->top == -1;
}
```

- **initStack:** Menginisiasi stack agar tidak kosong

```
// Inisialisasi stack
void initStack(Stack *s) {
    s->top = -1;
}
```

2.3 ANALISIS FUNGSI DALAM PROGRAM

a. FUNGSI KONVERSI EKSPRESI

- **infixToPostfix():**

Digunakan untuk mengonversi ekspresi infix menjadi postfix dengan cara menyimpan operator dalam stack saat membaca ekspresi infix dan menghasilkan ekspresi postfix.

Operand langsung ditulis, operator disimpan dan dikeluarkan sesuai prioritas

berikut codenya:

```
// Fungsi untuk konversi Infix ke Postfix
void infixToPostfix(char *infix, char *postfix) {
    Stack s;
    initStack(&s);
    int j = 0;
    for (int i = 0; infix[i] != '\0'; i++) {
        char c = infix[i];
```

```

        if (isalnum(c)) {
            postfix[j++] = c;
        } else if (c == '(') {
            char temp[2] = {c, '\0'};
            push(&s, temp);
        } else if (c == ')') {
            char temp[MAX];
            while (!isEmpty(&s)) {
                pop(&s, temp);
                if (temp[0] == '(') break;
                postfix[j++] = temp[0];
            }
        } else {
            while (!isEmpty(&s) &&
precedence(s.data[s.top][0]) >= precedence(c)) {
                postfix[j++] = s.data[s.top][0];
                s.top--;
            }
            char temp[2] = {c, '\0'};
            push(&s, temp);
        }
    }
    while (!isEmpty(&s)) {
        postfix[j++] = s.data[s.top][0];
        s.top--;
    }
    postfix[j] = '\0';
}

```

output yang dihasilkan:

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
Konversi Ekspresi Aritmatika				
1. Infix ke Postfix				
2. Postfix ke Infix				
3. Infix ke Prefix				
4. Prefix ke Infix				
5. Prefix ke Postfix				
6. Postfix ke Prefix				
7. Keluar				
Pilih menu: 1				
Masukkan ekspresi Infix: A+B*C				
Ekspresi Postfix: ABC*+				

- postfixToInfix():

Digunakan untuk mengonversi ekspresi postfix menjadi infix dengan cara menyusun ulang operand dan operator dalam bentuk infix menggunakan stack

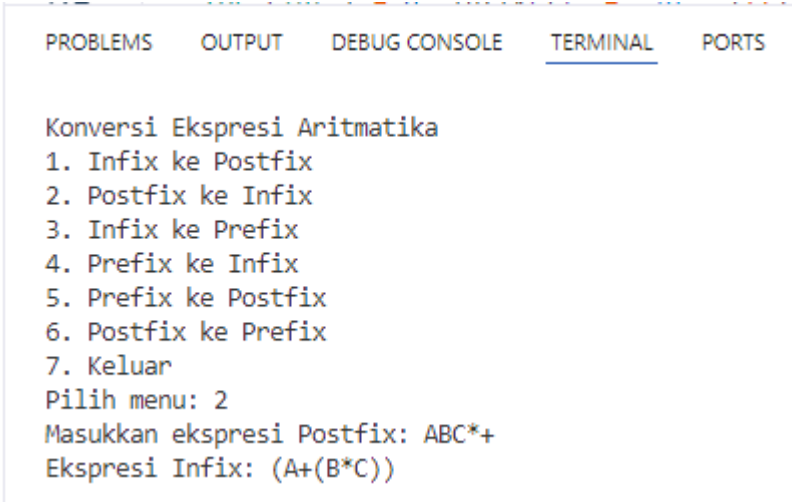
Operator digunakan untuk menyusun ulang operand.

code:

```
// Fungsi untuk konversi Postfix ke Infix
void postfixToInfix(char *postfix, char *infix) {
    Stack s;
    initStack(&s);
    char op1[MAX], op2[MAX], temp[MAX];

    for (int i = 0; postfix[i] != '\0'; i++) {
        if (isalnum(postfix[i])) {
            char operand[2] = {postfix[i], '\0'};
            push(&s, operand);
        } else {
            pop(&s, op2);
            pop(&s, op1);
            sprintf(temp, MAX, "(%s%c%s)", op1,
postfix[i], op2);
            push(&s, temp);
        }
    }
    pop(&s, infix);
}
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Konversi Ekspresi Aritmatika
1. Infix ke Postfix
2. Postfix ke Infix
3. Infix ke Prefix
4. Prefix ke Infix
5. Prefix ke Postfix
6. Postfix ke Prefix
7. Keluar
Pilih menu: 2
Masukkan ekspresi Postfix: ABC*+
Ekspresi Infix: (A+(B*C))
```

- infixToPrefix():

Digunakan untuk mengonversi ekspresi infix menjadi prefix dengan cara membalik ekspresi, mengonversinya ke postfix, lalu membalik hasilnya kembali.

Menggunakan stack dan prioritas operator

Code:

```
// Fungsi untuk konversi Infix ke Prefix
void infixToPrefix(char *infix, char *prefix) {
    reverseString(infix);
    for (int i = 0; infix[i] != '\0'; i++) {
        if (infix[i] == '(') infix[i] = ')';
        else if (infix[i] == ')') infix[i] = '(';
    }
    infixToPostfix(infix, prefix);
    reverseString(prefix);
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Konversi Ekspresi Aritmatika
1. Infix ke Postfix
2. Postfix ke Infix
3. Infix ke Prefix
4. Prefix ke Infix
5. Prefix ke Postfix
6. Postfix ke Prefix
7. Keluar
Pilih menu: 3
Masukkan ekspresi Infix: A+B*C
Ekspresi Prefix: +A*BC
```


- prefixToInfix():

Digunakan untuk mengonversi ekspresi prefix menjadi infix dengan cara menyusun ulang operand dan operator menggunakan stack.

Operator digunakan untuk menyusun ulang operand.

Code:

```
// Fungsi untuk konversi Prefix ke Infix
void prefixToInfix(char *prefix, char *infix) {
    Stack s;
    initStack(&s);
    char op1[MAX], op2[MAX], temp[MAX];
    reverseString(prefix);

    for (int i = 0; prefix[i] != '\0'; i++) {
        if (isalnum(prefix[i])) {
            char operand[2] = {prefix[i], '\0'};
            push(&s, operand);
        } else {
            pop(&s, op1);
            pop(&s, op2);
            sprintf(temp, "(%s%c%s)", op1,
prefix[i], op2);
            push(&s, temp);
        }
    }
    pop(&s, infix);
}
```

Outputnya:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Konversi Ekspresi Aritmatika
1. Infix ke Postfix
2. Postfix ke Infix
3. Infix ke Prefix
4. Prefix ke Infix
5. Prefix ke Postfix
6. Postfix ke Prefix
7. Keluar
Pilih menu: 4
Masukkan ekspresi Prefix: +A*BC
Ekspresi Infix: (A+(B*C))
```

- `prefixToPostfix()`:

Digunakan untuk mengonversi ekspresi prefix menjadi postfix dengan cara menggunakan stack untuk mengatur urutan operand dan operator.

Code:

```
// Fungsi untuk konversi Prefix ke Postfix
void prefixToPostfix(char *prefix, char *postfix) {
    Stack s;
    initStack(&s);
    char op1[MAX], op2[MAX], temp[MAX];
    reverseString(prefix);

    for (int i = 0; prefix[i] != '\0'; i++) {
        if (isalnum(prefix[i])) {
            char operand[2] = {prefix[i], '\0'};
            push(&s, operand);
        } else {
            pop(&s, op1);
            pop(&s, op2);
            sprintf(temp, MAX, "%s%s%c", op1, op2,
prefix[i]);
            push(&s, temp);
        }
    }
    pop(&s, postfix);
}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Konversi Ekspresi Aritmatika
1. Infix ke Postfix
2. Postfix ke Infix
3. Infix ke Prefix
4. Prefix ke Infix
5. Prefix ke Postfix
6. Postfix ke Prefix
7. Keluar
Pilih menu: 5
Masukkan ekspresi Prefix: +A*BC
Ekspresi Postfix: ABC*+
```

- postfixToPrefix():

Digunakan untuk mengonversi ekspresi postfix menjadi prefix dengan cara membalik urutan operand dan operator menggunakan stack.

Code:

```
// Fungsi untuk konversi Postfix ke Prefix
void postfixToPrefix(char *postfix, char *prefix) {
    Stack s;
    initStack(&s);
    char op1[MAX], op2[MAX], temp[MAX];

    for (int i = 0; postfix[i] != '\0'; i++) {
        if (isalnum(postfix[i])) {
            char operand[2] = {postfix[i], '\0'};
            push(&s, operand);
        } else {
            pop(&s, op2);
            pop(&s, op1);
            snprintf(temp, MAX, "%c%s%s", postfix[i], op1,
op2);
            push(&s, temp);
        }
    }
    pop(&s, prefix);
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Konversi Ekspresi Aritmatika
1. Infix ke Postfix
2. Postfix ke Infix
3. Infix ke Prefix
4. Prefix ke Infix
5. Prefix ke Postfix
6. Postfix ke Prefix
7. Keluar
Pilih menu: 6
Masukkan ekspresi Postfix: ABC*+
Ekspresi Prefix: +A*BC
```

b. FUNGSI PENDUKUNG STACK

- precedence()

Fungsi ini digunakan dalam konversi ekspresi aritmatika untuk menentukan tingkat prioritas operator. Operator dengan prioritas lebih tinggi akan dieksekusi lebih dulu dibandingkan operator dengan prioritas lebih rendah.

```
// Mendapatkan prioritas operator
int precedence(char op) {
    switch (op) {
        case '+': case '-': return 1;
        case '*': case '/': return 2;
        case '^': return 3;
        default: return 0;
    }
}
```

- reverseString()

Fungsi ini digunakan dalam proses konversi Infix ke Prefix. String akan dibalik agar dapat diproses dengan lebih mudah sesuai aturan konversi.

```
// Membalikkan string
void reverseString(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - i - 1];
    }
}
```

```
        str[len - i - 1] = temp;
    }
}
```

Dengan adanya fungsi-fungsi ini, program dapat melakukan operasi stack dengan efisien untuk menangani berbagai konversi ekspresi aritmatika. Total terdapat 12 fungsi dalam program, tidak termasuk fungsi main().

Link repository GitHub : <https://github.com/NoviaAngreini/UTS-SDA-2025>