LAPORAN PRAKTIKUM UJIAN TENGAH SEMESTER

STRUKTUR DATA DAN ALGORITMA

Kelas B

Disusun Oleh:

Teuku Fikram Al Syahbanna (2408107010044)

Muhammad Rayan Zishan (2408107010063)

Asisten Lab Pengampu:

Athar Rayyan Muhammad

Rafli Afriza Nugraha

Muhammad Syahidal Akbar Zas

PROGRAM STUDI INFORMATIKA

UNIVERSITAS SYIAH KUALA

2025

A. Apa Itu Ekspresi Infix, Postfix, dan Prefix?

Sebelum kita membahas kode programnya, mari kita pahami dulu konsep dasar dari ekspresi Infix, Postfix, dan Prefix.

1. Infix Notation (Notasi Infix)

Ini adalah cara kita biasa menulis ekspresi matematika.

Contoh:

A + B

(A + B) * C

Operator berada di antara operand. Namun, komputer tidak selalu bisa langsung memahami notasi ini karena perlu memperhitungkan prioritas operator dan urutan eksekusi dengan tanda kurung.

2. Postfix Notation (Reverse Polish Notation, RPN)

Dalam notasi ini, operator berada setelah operand.

Contoh:

AB+

AB+C*

Keunggulan utama: tidak perlu tanda kurung, Karena prioritas operator sudah ditentukan dengan urutan.

3. Prefix Notation (Polish Notation)

Ini adalah kebalikan dari Postfix. Operator ditempatkan sebelum operand.

Contoh:

+AB

*+ABC

Sama seperti Postfix, tidak membutuhkan tanda kurung karena urutan eksekusinya sudah jelas.

B. Apa Tujuan Program ini?

Program ini adalah konverter ekspresi matematika yang bisa mengubah ekspresi dari satu notasi ke notasi lainnya. Dengan program ini, kita bisa melakukan konversi berikut:

Infix ke Postfix

Infix ke Prefix

Postfix ke Infix

Postfix ke Prefix

Prefix ke Infix

Prefix ke Postfix

Dan tentu saja, ada opsi keluar dari program jika kita sudah selesai.

C. Struktur Data Apa yang Digunakan?

Untuk menangani konversi ini, kami menggunakan **Stack (tumpukan)** sebagai struktur data utama. Stack adalah struktur data yang mengikuti prinsip LIFO (Last In, First Out), yaitu elemen yang dimasukkan terakhir akan keluar lebih dulu. Yang kami anggap cara ini lebih efektif dan sesuai dengan algoritma konversi.

Perlu diketahui bahwa pengguna mungkin saja akan memasukkan karakter yang amat Panjang, karena di sini kami menggunakan stack berukuran 100(MAX = 100), untuk mempermudah berjalannya algoritma, maka stack di sini akan menerima string, bukan char. Dapat dilihat pada char *item[MAX], struct hanya akan menyimpan alamatnya saja, yang berarti per indeks dapat menyimpan sebuah string, cara ini lebih ribet, tapi setidaknya memberikan hasil yang memuaskan

Di dalam kode, Stack ini dibuat sebagai berikut:

```
typedef struct {
    int top;
    char *item[MAX]; // Stack menerima string
} Stack;
```

D. Fungsi-fungsi yang Digunakan Dalam Kode Program

1. inisialisasi(Stack *s) - Menginisialisasi stack agar kosong.

```
void inisialisasi(Stack *s) {
    s->top = -1; // buat stack kosong
}
```

2. cek_kosong(Stack *s) - Mengecek apakah stack kosong.

```
bool cek_kosong(Stack *s) {
    return (s->top == -1);
}
```

3. cek_penuh(Stack *s) - Mengecek apakah stack penuh.

```
bool cek_penuh(Stack *s) {
    return (s->top == MAX - 1);
}
```

4. push(Stack *s, char *value) - Menambahkan elemen ke dalam stack.

```
void push(Stack *s, char *value) {
    if (cek_penuh(s)) {
        printf("Stacknya Udah Penuh hai!\n");
        return;
    }
    s->item[++(s->top)] = value;
}
```

5. pop(Stack *s) - Menghapus elemen teratas dari stack dan mengembalikannya.

```
char* pop(Stack *s) {
    if (cek_kosong(s)) {
        printf("Stacknya Masih Kosong lah cik!\n");
        exit(EXIT_FAILURE);
    }
    return s->item[(s->top)--];
}
```

6. peek(Stack *s) - Mengembalikan elemen teratas dari stack tanpa menghapusnya.

```
char* peek(Stack *s) {
    if (cek_kosong(s)) {
        printf("Stack Kosong! mau tengok apa coba\n");
        exit(EXIT_FAILURE);
    }
    return s->item[s->top];
}
```

7. prioritas(char op) - Menentukan prioritas operator dalam ekspresi.

```c
int prioritas(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        default:
            return 0;
    }
}
```

8. balik_string(char str[]) - Membalik urutan karakter dalam string.

```c
void balik_string(char str[]) {
    int length = strlen(str);
    for (int i = 0, j = length - 1; i < j; i++, j--) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
}
```

9. apakah_operator(char ch) - Mengecek apakah karakter adalah operator.

```c
int apakah_operator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}
```

10. infixToPostfix(char infix[], char postfix[]) - Mengubah ekspresi infix menjadi postfix.

```c
void infixToPostfix(char infix[], char postfix[]) {
    Stack s;
    inisialisasi(&s);
    int i = 0, j = 0;

    while (infix[i] != '\0') {
        char ch = infix[i];

        if (OPERAND(ch)) {
            // tambah ke postfix kalau operand
            postfix[j++] = ch;
        } else if(ch ==' '){
            i++;
            continue;
        }
        else if (ch == '(') {
            // push ke stack
            char *parenthesis = (char *)malloc(2 * sizeof(char));
            sprintf(parenthesis, "%c", ch);
            push(&s, parenthesis);
```

```c
    } else if (ch == ')') {
        // kalau ), pop operator sampai jumpa (
        while (!cek_kosong(&s) && strcmp(peek(&s), "(") != 0) {
            char *op = pop(&s);
            postfix[j++] = op[0]; // masukkan operator ke postfix
            free(op);
        }
        char *leftParenthesis = pop(&s); // pop ( dari stack
        free(leftParenthesis);
    } else {
        //kalau operator, pop operator dgn prioritas lebih tinggi dari stack
        while (!cek_kosong(&s) && prioritas(peek(&s)[0]) >= prioritas(ch)) {
            char *op = pop(&s);
            postfix[j++] = op[0]; //masukkan operator ke postfix
            free(op);
        }
        //push operator
        char *operator = (char *)malloc(2 * sizeof(char));
        sprintf(operator, "%c", ch);
        push(&s, operator);
    }
    i++;
}
```

11. infixToPrefix(char infix[], char prefix[]) - Mengubah ekspresi infix menjadi prefix.

```
void infixToPrefix(char infix[], char prefix[]) {
    Stack s;
    inisialisasi(&s);
    char temp[MAX];

    // Balik infix
    balik_string(infix);

    // Konversi infix yang sudah dibalik ke postfix
    infixToPostfix(infix, temp);

    // Balik hasil postfix jadi prefix
    balik_string(temp);
    strcpy(prefix, temp);
}
```

12. PostfixToInfix(char postfix[], char infix[]) - Mengubah ekspresi postfix menjadi infix.

```
void PostfixToInfix(char postfix[], char infix[]) {
    Stack s;
    inisialisasi(&s);

    for (int i = 0; postfix[i] != '\0'; i++) {
        char ch = postfix[i];

        // push kalau jumpa operand
        if (OPERAND(ch)) {
            char *operand = (char *)malloc(2 * sizeof(char));
            sprintf(operand, "%c", ch); // ubah operand ke string
            push(&s, operand); // push operand
        } else if(ch == ' '){
            continue;//ini untuk skip spasi
        }
        // kalau operator
        else if (apakah_operator(ch)) {
            // pop 2 operand
            char *op2 = pop(&s); // operand 2
            char *op1 = pop(&s); // operand 1

            char *newExpr = (char *)malloc((strlen(op1) + strlen(op2) + 4) * sizeof(char));
            sprintf(newExpr, "(%s%c%s)", op1, ch, op2); // menggabungkan operand dan operator
```

```
            // Push infix baru ke stack
            push(&s, newExpr);

            // free memory tuk operand
            free(op1);
            free(op2);
        }
    }
    // masukkan hasil ke infix(sekaligus)
    strcpy(infix, pop(&s));
}
```

13. PostfixToPrefix(char postfix[], char prefix[]) - Mengubah ekspresi postfix menjadi prefix.

```
void PostfixToPrefix(char postfix[], char prefix[]) {
    Stack s;
    inisialisasi(&s);

    for (int i = 0; postfix[i] != '\0'; i++) {
        char ch = postfix[i];

        // push jika operand
        if (OPERAND(ch)) {
            char *operand = (char *)malloc(2 * sizeof(char));
            sprintf(operand, "%c", ch); // ubah operand ke string
            push(&s, operand);
        }else if(ch == ' '){
            continue;
        }
        // jika operator, maka
        else if (apakah_operator(ch)) {
            // Pop 2 operand
            char *op2 = pop(&s); // Operand 2
            char *op1 = pop(&s); // Operand 1
            // alokasi memori tuk prefix baru
            char *newExpr = (char *)malloc((strlen(op1) + strlen(op2) + 2) * sizeof(char));
            sprintf(newExpr, "%c%s%s", ch, op1, op2); // gabungkan operator dan operand
            // push prefix baru
```

```
            push(&s, newExpr);
            free(op1);
            free(op2);
        }
    }
    // masukkan stack ke prefix
    strcpy(prefix, pop(&s));
}
```

14. PrefixToInfix(char prefix[], char infix[]) - Mengubah ekspresi prefix menjadi infix.

```
void PrefixToInfix(char prefix[], char infix[]) {
    Stack s;
    inisialisasi(&s);

    int length = strlen(prefix);

    // perulangan dari kanan ke kiri
    for (int i = length - 1; i >= 0; i--) {
        char ch = prefix[i];

        // push kalau operand
        if (OPERAND(ch)) {
            char *operand = (char *)malloc(2 * sizeof(char));
            sprintf(operand, "%c", ch);
            push(&s, operand);
        }else if(ch ==' '){
            continue;
        }
        // kalau operator
        else if (apakah_operator(ch)) {
            // pop dua operand secara urut
            char *op1 = pop(&s); // Operand 1
            char *op2 = pop(&s); // Operand 2
```

```
            char *newExpr = (char *)malloc((strlen(op1) + strlen(op2) + 4) * sizeof(char));
            sprintf(newExpr, "(%s%c%s)", op1, ch, op2); // gabungkan

            // push semua ke stack
            push(&s, newExpr);

            free(op1);
            free(op2);
        }
    }

    // pop semua stack ke infix
    strcpy(infix, pop(&s));
}
```

15. PrefixToPostfix(char prefix[], char postfix[]) - Mengubah ekspresi prefix menjadi postfix.

```
void PrefixToPostfix(char prefix[], char postfix[]) {
    Stack s;
    inisialisasi(&s);

    int length = strlen(prefix);

    // perulangan dari kanan ke kiri
    for (int i = length - 1; i >= 0; i--) {
        char ch = prefix[i];

        if (OPERAND(ch)) {
            char *operand = (char *)malloc(2 * sizeof(char));
            sprintf(operand, "%c", ch);
            push(&s, operand);
        }else if(ch ==' '){
            continue;
        }
        else if (apakah_operator(ch)) {
            char *op1 = pop(&s);
            char *op2 = pop(&s);
            char *newExpr = (char *)malloc((strlen(op1) + strlen(op2) + 2) * sizeof(char));
            sprintf(newExpr, "%s%s%c", op1, op2, ch);
            push(&s, newExpr);
```

```
            free(op1);
            free(op2);
        }
    }
```

Masing-masing fungsi bekerja dengan konsep yang mirip, namun dengan susunan operator dan operand yang berbeda, intinya program akan menjalankan push jika ditemukan operand, melangkahi iterasi jika ditemukan spasi, dan menjalankan pop jika ditemukan operator. Prosedurnya bisa berbeda-beda di tiap fungsi, baik dari susunan karakter, jumlah push dan pop, dan perulangannya. Namun cara kerjanya tetap sama, hal ini pula dirancang agar trouble shooting dan bug fixing menjadi lebih bersahabat bagi engineer.

Keismpulannya, program ini punya total 15 fungsi yang masing-masing punya perannya sendiri dalam mengolah ekspresi matematika. Ada fungsi buat ngatur stack kayak push, pop, peek, terus ada juga yang khusus buat ngecek kondisi stack biar nggak error pas dipakai. Selain itu, ada fungsi utama yang

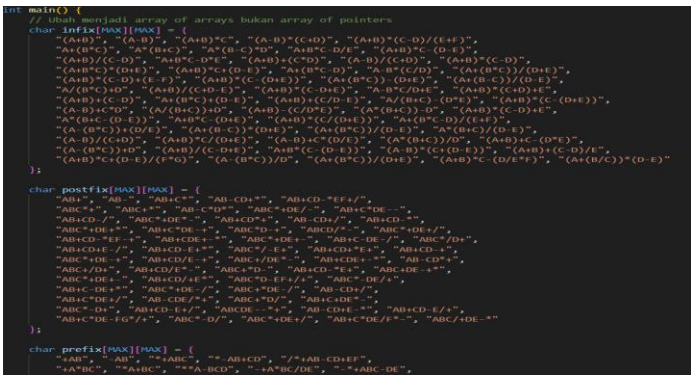
ngurusin konversi antara Infix, Postfix, dan Prefix, jadi program ini bisa ngubah ekspresi matematika sesuai kebutuhan pengguna. Semua proses ini dirancang biar bisa berjalan sistematis dan nggak bikin ribet.

Yang bikin program ini unik adalah pemanfaatan Struktur Data Stack, yang bikin konversi ekspresi jadi lebih efisien. Stack ini dipakai buat menyusun ulang elemen ekspresi tanpa harus bingung sama tanda kurung, karena urutan operatornya otomatis diatur sesuai prioritasnya. Dengan pendekatan ini, program bisa menangani berbagai jenis ekspresi dengan cepat dan akurat. Bisa dibilang, program ini cukup fleksibel, jadi kalau mau dikembangkan lebih lanjut, tinggal tambahin fitur tanpa harus ubah banyak bagian.

Jika dibaca saja mungkin kode ini terasa simple, namun sebenarnya yang menjadi kekurangan program ini adalah saat pembuatan kodenya, pengalokasian memori pada variable “newExpr” dan “operand” masih sedikit rumit, ditambah lagi stack dan variable di tiap fungsi sudah saling terintegrasi dengan pointer, yang juga sedikit menyulitkan bagi pemula.

Banyak fungsi yang masih dapat dipersingkat dan diefisiensikan, namun tidak kami lakulkan karena kami merasa ini adalah hasil yang terbaik. Program ini sendiri dapat berjalan dengan mulus dan **memiliki akurasi sebanyak 100%,** kami sudah buktikan dengan membuat 150 test case yang kemudian akan dikoreksi oleh ChatGPT, Adapun prosedurnya ialah sebagai berikut :

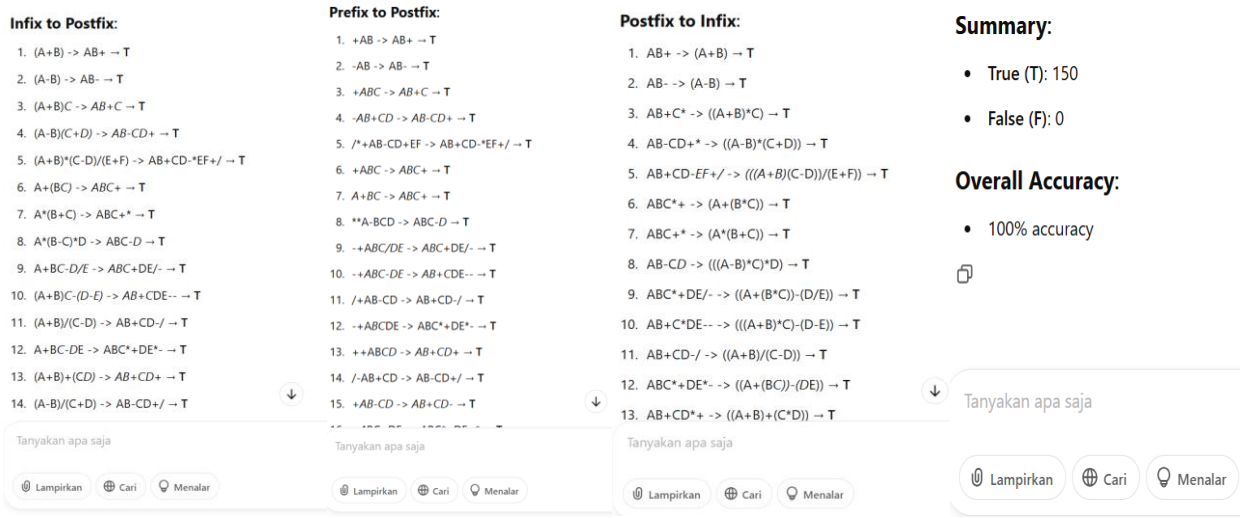
1. Membuat array berisi infix, postfix, dan prefix, dengan total 150



2. Membuat perulangan sederhana

```
char hasil[MAX];

printf("\nPostfix to Infix\n");
for (int i = 0; i < 10; i++) { //nyan testing beuh, bek dron pike kamoe menyelewengkan hasee
    bersihkan_jawa(hasil);
    PostfixToInfix(postfix[i], hasil);
    printf("%d. %s -> %s\n", i + 1, postfix[i], hasil);
}

printf("\nPostfix to Prefix\n");
for (int i = 0; i < 10; i++) {
    bersihkan_jawa(hasil);
    PostfixToPrefix(postfix[i], hasil);
    printf("%d. %s -> %s\n", i + 1, postfix[i], hasil);
}

printf("\nPrefix to Infix\n");
for (int i = 0; i < 10; i++) {
    bersihkan_jawa(hasil);
    PrefixToInfix(prefix[i], hasil);
    printf("%d. %s -> %s\n", i + 1, prefix[i], hasil);
}

printf("\nPrefix to Postfix\n");
for (int i = 0; i < 10; i++) {
    bersihkan_jawa(hasil);
    PrefixToPostfix(prefix[i], hasil);
    printf("%d. %s -> %s\n", i + 1, prefix[i], hasil);
}

return 0;
```

3. Yang terakhir, setelah program dijalankan, maka akan disalin ke ChatGPT dan akan dikoreksi secara otomatis



NB : Adapun berikut adalah command yang kami berikan kepada chatGPT

“Hi! i have these infix to postfix, infix to prefix, postfix to prefix, and vice versa, can you check are they correct? and please give me the summary of the true and false with the overall accuracy”

Memang 150 test case belum bisa dijadikan acuan seberapa handal kode tersebut, namun setidaknya ini sudah memberi sedikit gambaran bahwa kode kami memang teruji kebenarannya