Praktikum 4

Tumpukan (*Stack*)

POKOK BAHASAN:

- Konsep Tumpukan (*Stack*)
- Struktur data untuk Tumpukan
- Algoritma merubah Infix menjadi Postfix
- Implementasi Tumpukan dalam Bahasa C

TUJUAN BELAJAR:

Setelah melakukan praktikum dalam bab ini, mahasiswa diharapkan mampu:

- Memahami terminologi yang terkait dengan struktur data *stack*.
- Memahami operasi-operasi yang ada dalam *stack*.
- Mengidentifikasi permasalahan-permasalahan pemrograman yang harus diselesaikan dengan menggunakan *stack*, sekaligus menyelesaikannya

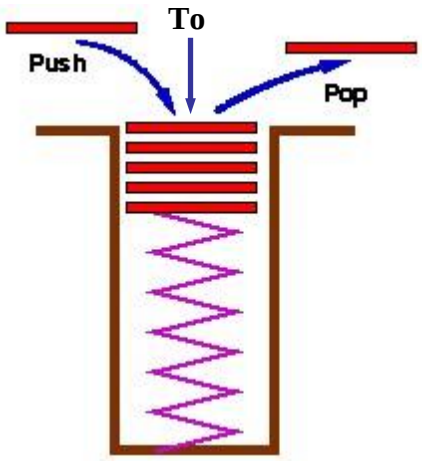
DASAR TEORI:

Salah satu konsep yang efektif untuk menyimpan dan mengambil data adalah "terakhir masuk sebagai yang pertama keluar" (*Last In First Out* / LIFO). Dengan konsep ini, pengambilan data akan berkebalikan urutannya dengan penyimpanan data.

Stack adalah sebuah kumpulan data dimana data yang diletakkan di atas data yang lain. Dengan demikian stack adalah struktur data yang menggunakan konsep LIFO. Dengan demikian, elemen terakhir yang disimpan dalam stack menjadi elemen pertama yang diambil. Dalam proses komputasi, untuk meletakkan sebuah elemen pada bagian atas dari stack, maka kita melakukan push. Dan untuk memindahkan dari tempat yang atas tersebut, kita melakukan pop.

Untuk menjelaskan pengertian diatas kita mengambil contoh sebagai berikut. Misalnya kita mempunyai dua buah kotak yang kita tumpuk, sehingga kotak kita letakkan di atas kotak yang lain. Jika kemudian stack dua buah kotak tersebut kita tambah dengan kotak ketiga dan seterusnya, maka akan kita peroleh sebuah stack kotak, yang terdiri dari N kotak.

Secara sederhana, sebuah stack bisa kita ilustrasikan seperti pada gambar 4.1 di bawah ini. Dari gambar di bawah ini, kita bisa mengatakan bahwa kotak B ada di atas kotak A dan ada di bawah kotak C. Dari gambar tersebut kita dapat melihat bahwa kita hanya bisa menambah atau mengambil sebuah kotak lewat satu ujung, yaitu ujung bagian atas. Nampak pula bahwa stack merupakan kumpulan data yang sifatnya dinamis, artinya kita bisa menambah atau mengambil data darinya.



Gambar 3.1 Ilustrasi Sebuah Stack

4.1 REPRESENTASI STACK dengan ARRAY

Ada beberapa cara untuk menyajikan sebuah stack tergantung pada permasalahan yang akan kita selesaikan. Dalam bab ini kita akan menggunakan cara yang paling sederhana, tipe data yang sudah kita kenal, yaitu array. Kita dapat menggunakan array untuk menyajikan sebuah stack, dengan anggapan bahwa banyaknya elemen maksimum dari stack tersebut tidak akan melebihi batas maksimum banyaknya elemen dalam array.

Pada saat ukuran stack, kalau kita teruskan menambah data lagi, akan terjadi overflow. Dengan demikian perlu data tambahan untuk mencatat posisi ujung stack.

Dengan kebutuhan seperti ini, kita dapat menyajikan stack dengan menggunakan tipe data struktur (struct) yang terdiri dari dua field. Field pertama bertipe array untuk menyimpan elemen stack, medan kedua bertipe integer untuk mencatat posisi ujung stack. Dengan demikian kita mendeklarasikan stack sebagai berikut:

```
#define MAXSTACK 100 typedef
int ItemType; /* Definisi
struktur stack */ typedef
struct
{
      int Item[MAXSTACK]; /* Array yang berisi data tumpukan
*/    int Count; /* menunjukkan indeks data paling atas dari
stack */
}Stack;
```

4.2 OPERASI PADA STACK

Operasi-operasi Dasar pada stack adalah sebagai berikut:

1. Operasi menciptakan T sebagai stack kosong `void InitializeStack(Stack *S)`

   ```
   {
        S->Count = 0;
   }
   ```

2. Fungsi yang melakukan pengecekan apakah stack dalam kondisi kosong `int Empty(Stack *S)`

   ```
   {
    return (S->Count == 0);
   }
   ```

3. Fungsi yang melakukan pengecekan apakah stack dalam kondisi penuh `int Full(Stack *S)`

   ```
   {  return (S->Count == MAXSTACK);
   }
   ```

4. Operasi menyisipkan elemen x ke stack T dan mengembalikan stack baru

   ```
   void Push(ItemType x, Stack *S)
   {         if (S-
>Count==MAXSTACK)
   ```

```
                printf("Stack penuh! Data tidak dapat masuk!");
else
            {
                S->Item[S->Count]=x;
                ++(S->Count);
          }
      }
```

5. Operasi mengambil elemen puncak stack T, (pop(T,x)) `int Pop(Stack *S, ItemType *x)`

```
      {           if (S->Count==0)//stack
kosong
            printf("Stack masih kosong!");
else
            {
                --(S->Count);
                *x = S->Item[S->Count];
            }
      }
```

ALGORITMA MERUBAH NOTASI INFIX MENJADI NOTASI POSTFIX

1. Baca ungkapan dalam notasi infix, misalnya S, tentukan panjang ungkapan tersebut, misalnya N karakter, siapkan sebuah stack kosong dan siapkan derajad masing-masing operator, misalnya: ^ berderajad 3, * dan / berderajad 2, + dan – berderajad 1 dan (berderajad 0.
2. Dimulai dari i = 1 sampai N kerjakan langkah-langkah sebagai berikut:
   a. R = S[I]
   b. Test nilai R. Jika R adalah:
      operand : langsung ditulis kurung buka
      : push ke dalam tumpukan
      kurung tutup : pop dan tulis semua isi tumpukan sampai ujung tumpukan = ‘(‘. Pop juga tanda ‘(‘ ini, tetapi tidak usah ditulis
      operator : jika tumpukan kosong atau derajad R lebih tinggi dibanding derajad ujung tumpukan, push operator ke dalam tumpukan. Jika tidak, pop ujung tumpukan dan tulis; kemudian ulangi pembandingan R dengan ujung tumpukan. Kenudian R di-push
   c. Jika akhir notasi infix telah tercapai, dan tumpukan masih belum kosong, pop semua isi tumpukan dan tulis hasilnya

Untuk memahami algoritma di atas, kita coba mengubah ungkapan berikut, yang ditulis menggunakan notasi infix, menjadi notasi postfix

(A + B) / ((C – D) * E ^ F)

Ilustrasi pengubahan notasi infix di atas menjadi notasi postfix secara lengkap tersaji dalam tabel sebagai berikut:

Karakter dibaca	Isi Tumpukan	Karakter tercetak	Hasil Notasi Postfix Yang Terbentuk
(	(		
A	(+	A	A
+	(+		
B		B	A B
)		+	A B +
/	/		
(	/ (		
(	/ ((		
C	/ ((	C	A B + C
-	/ ((-		
D	/ ((-	D	A B + C D
)	/ (	-	A B + C D -
*	/ (*		
E	/ (*	E	A B + C D – E
^	/ (* ^		
F	/ (* ^	F	A B + C D – E F
)	/ (*	^	A B + C D – E F ^
	/ (	*	A B + C D – E F ^ *
	/		
		/	A B + C D – E F ^ *

Dari ilustrasi di atas, bisa kita lihat bahwa notasi postfix dari ungkapan:

(A + B) / ((C – D) * E ^ F)

adalah

A B + C D – E F ^ */

TUGAS PENDAHULUAN:

1. Buatlah flowchart untuk masing-masing operasi stack seperti yang sudah dijelaskan di dasar teori
2. Buatlah flowchart untuk latihan no soal 1 & 2.

PERCOBAAN:

1. Buatlah workspace untuk praktikum Struktur Data dengan menggunakan Visual C++.
2. Buatlah project untuk praktikum KEEMPAT.
3. Cobalah untuk masing-masing percobaan di bawah ini.
4. Selesaikan soal-soal yang ada dengan mengimplementasikan flowchart yang anda buat pada Tugas Pendahuluan.

Percobaan 1 : Penerapan Fungsi-Fungsi Tumpukan, Memasukkan Isi Tumpukan & Menampilkan Isi Tumpukan

```
#include <stdio.h>
#define MAXSTACK 5

typedef int ItemType;
/* Definisi struktur stack */
typedef struct
{
      int Item[MAXSTACK];
      /* Array yang berisi data tumpukan */
      int Count;
      /* menunjukkan indeks data paling atas dari stack */
}
Stack;
```

```
void InitializeStack(Stack *S)
{
      S->Count = 0;
}

int Empty(Stack *S)
{
      return (S->Count == 0);
}

int Full(Stack *S)
{
      return (S->Count == MAXSTACK);
}

void Push(ItemType x, Stack *S)
{
      if (S->Count == MAXSTACK)
            printf("Stack penuh! Data tidak dapat masuk!");
      else
      {
            S->Item[S->Count] = x;
            ++(S->Count);
      }
}

int Pop(Stack *S)
{
      if (S->Count == 0)//stack kosong
            printf ("Stack masih kosong!");
      else
      {
            --(S->Count);
            return (S->Item[S->Count]);
      }
}
```

```
void main()
{
      int i, input;
      Stack tumpukan;



      InitializeStack(&tumpukan);

      for (i = 0; i < MAXSTACK; i++)
      {
            printf("Masukkan isi stack ke- %d: ", i + 1);
            scanf("%d",&input);
            Push(input, &tumpukan);
      }

      for (i = MAXSTACK; i > 0; i--)
      {
      printf("Isi stack ke- %d: ", i);
      printf(" %d \n", Pop(&tumpukan));
      }
}
```

Percobaan 2 : Program Merubah Notasi Infix menjadi Postfix

```
/* Implementasi algoritma utk mengubah infix menjadi postfix */

#include <stdio.h>
#include <string.h>

#define MAXSTACK 100

typedef char ItemType;
typedef struct
{
      char item[MAXSTACK]; /* Array yg berisi data tumpukan */
      int count; /* menunjukkan indeks data paling atas dari stack
*/ } Stack;

Stack tumpukan;
```

```
void InitializeStack(Stack *);
int Empty(Stack *);
int Full(Stack *);
void Push(ItemType, Stack *);
ItemType Pop(Stack *);
int drjt(char);
void konversi_cetak(char[]);

main()
{
      char tampung[MAXSTACK], jawab;

      puts("MENGUBAH NOTASI INFIX MENJADI PSOTFIX");
      puts("DENGAN MEMANFAATKAN STRUKTUR STACK");
      do
      {
      InitializeStack(&tumpukan);
      fflush(stdin);
      puts("");
      printf("Masukan ekspresi dalam notasi infix : ");
      fflush(stdin);
      fgets(tampung, sizeof(tampung), stdin);
      fflush(stdin);
      printf("\nUngkapan postfixnya = ");
      fflush(stdin);
      konversi_cetak(tampung);
      puts("");
      fflush(stdin);
      printf("\nMau mencoba lagi (y/t) ? ");
      scanf("%c", &jawab);
      } while ((jawab == 'y') || (jawab == 'Y'));
}

void InitializeStack(Stack *S)
{
      S->count = 0;
}

int Empty(Stack *S)
{
      return (S->count == 0);
}
int Full(Stack *S)
{
      return (S->count == MAXSTACK);
}

void Push(ItemType x, Stack *S)
{
```

```
      if (Full(S))  //stack penuh
            printf("Stack penuh! Data tidak dapat masuk!");
      else
      {
            ++(S->count);
            S->item[S->count]=x;
      }
}

            ItemType Pop(Stack *S)
            {
            ItemType x;

            if (Empty(S))
            {  //stack kosong
                  printf ("STACK KOSONG!");
                  return 0;
            }
            else
            {
                  x = (S->item[S->count]);
                  --(S->count);   return x;
            }
      }
int drjt(char x)  //menentukan derajat operator
{
      if(x == '(')
            return 0;
      else if((x == '+') || (x == '-'))
            return 1;
      else if((x == '*') || (x == '/'))
            return 2;
      else if(x == '^')
            return 3;
      else
            return -1;  //invalid operator
}

void konversi_cetak(char temp[])
{
                  int i, pjg, valid = 1;
                  char kar, smtr;
                  pjg = strlen(temp) - 1;
                  for (i = 0; i < pjg; i++)
                  {
                        kar = temp[i]; //membaca input per karakter
                        switch(kar)
                        {
                              //if kar = '(' -> push ke dalam tumpukan
                        case '(' :
```

```
                              Push(kar, &tumpukan);
                              break;

  //if kar = operand -> langsung ditulis
                        case '0': case '1': case '2': case '3': case
'4': case '5' : case '6': case '7': case '8': case'9':
                              printf("%c", kar);
                              break;
                        /* if kar = operator -> jika tumpukan kosong
atau derajad kar lbh tinggi dibanding derajad ujung tumpukan, push
operator ke  dalam tumpukan.  Jika tidak, pop ujung tumpukan dan
tulis;   Ulangi pembandingan kar dengan ujung tumpukan.   Kemudian
kar di-push    */
                        case '+': case '-': case'*': case '/': case
'^':
                              if ((Empty(&tumpukan)) || ((drjt(kar) >
drjt(tumpukan.item[tumpukan.count])))      Push(kar, &tumpukan);
                              else
                              {
                                    do
                                    {
                                          smtr = Pop(&tumpukan);
                                          printf("%c", smtr);
                                    }
                                    while         (drjt(kar)         <
drjt(tumpukan.item[tumpukan.count]));
                                    Push(kar, &tumpukan);
                              }
                              break;

                        /* if kar = ')' -> pop dan tulis semua isi
tumpukan sampai ujung tumpukan = '('.    Pop juga tanda '(' ini,
tetapi tidak usah ditulis    */
                        case ')':
                              while (tumpukan.item[tumpukan.count] !=
'('
                              {
                                    smtr = Pop(&tumpukan);
                                    printf("%c", smtr);
                              }
                              Pop(&tumpukan);
                              break;

                        default:  //selain dari kar yang diijinkan
                              valid = 0;
                              puts("INVALID STSTEMEN");
                              break;
                        } //end of switch-case
                  } //end of looping for
```

```
/*    Jika statemen valid, akhir notasi infix telah tercapai,
sedangkan tumpukan masih belum kosong, pop semua isi tumpukan dan
tulis hasilnya  */

                    if ((valid != 0) && (!Empty(&tumpukan)))
                    {
                          smtr = Pop(&tumpukan);
                          printf("%c", smtr);
                    }
}
```

LATIHAN

1. Buatlah sebuah program yang melakukan konversi dari bilangan desimal ke bilangan biner, octal, heksa dengan menggunakan stack
2. Buatlah sebuah program yang melakukan pembalikan terhadap kalimat dengan menggunakan stack Contoh:

   Kalimat : Struktur Data

   Hasil setelah dibalik : ataD rutkurtS
3. Tentukan apakah sebuah kalimat yang diinputkan dalam program (dengan menggunakan stack) adalah sebuah palindrom atau bukan. Palindrom adalah kalimat yang jika dibaca dari depan dan dari belakang, maka bunyinya sama.

   Contoh:

   Kalimat : sugus

   Kalimat tersebut adalah palindrom
4. Tambahkan implementasi Percobaan2 hingga mendapatkan hasil penghitungannya.

   Contoh:

   infix : 5 * (4 – 2)

   postfix : 542-* hasil :

   10