Tekli Ve Çiftli Linked List ile Stack İşlemleri

```
static public int sp = -1;
static public int[] stack = new int[100];

static public void push(int data)
{
    sp++;
    stack[sp] = data;
}

static public int pop()
{
    int data = stack[sp];
    sp--;
    return data;
}

static public int read()
{
    int data = stack[sp];
    return data;
}

static Tekli sp_ = null;
static Tekli head_ = null;
static public void push_(int data)
{
    Tekli q = new Tekli();
    q.data = data;
    q.next = null;
    if(sp_ == null)
    {
      sp_ = q;
      head_ = q;

    }
    else
    {

      sp_.next = q;
      sp_ = q;

    }
}
```

```
static public int pop_()
{
    int data = sp_.data;
    Tekli temp = head_;
    while(temp.next != sp_)
    {
     temp = temp.next;
    }
    temp.next = null;
    sp_ = temp;


    temp = sp_;
    while(temp.next != sp_)
    {
     temp = sp_;
    }
    temp.next = sp_.next;

    return data;
}



static Ciftli sp__ = null;
static public void push__(int data)
{
    Ciftli q = new Ciftli();
    q.data = data;
    q.next = null;
    q.prev = sp__;
    if(sp__ == null)
    {
     sp__ = q;
    }
    else
    {
     sp__.next = q;
     sp__ = q;
    }
}

static public int pop__()
{
    int data = sp__.data;
    sp__ = sp__.prev;
    return data;
}
```

```
static int count()
{
    int adet = 0;
    Ciftli w = sp__;
    while (w != null)
    {
     w = w.prev;
     adet++;
    }
    return adet;
}

static int Count_(Ciftli q)
{
    if (q == null) return 0;
    return 1 + Count_(q.prev);
}

class Tekli
{
    public int data;
    public Tekli next;
}

class Ciftli
{
    public int data;
    public Ciftli next;
    public Ciftli prev;
}
```

Kuyruk işlemleri 1

```
//50 Elemanlı Bir sayısal dizi oluşturuyoruz
static int[] kuyruk = new int[50];
//Başlangıç front ve rear değerlerini belirliyoruz.
static int front = 0;
static int rear = -1;

//Kuyruk için yeniden sıralama yapıp verileri sıfırlıyoruz.
static void kuyrukyenile()
{
    for (int i = front; i <=rear ; i++)
    {
      kuyruk[i-front] = kuyruk[i];

    }
    rear = front;
    front = 0;
}

//Kuyruğa eleman ekliyoruz
static void ekle(int data)
{
    //sonraki eklenenin sona gitmesi için her eklemede rear değerini
arttırıyoruz
    rear++;
    kuyruk[rear] = data;
}
```

```
static int elemanal()

{
    //kuyruktaki ilk elemanı okuyoruz.
    int data = kuyruk[front];
    //ilk değeri arttırıyoruz sonraki okumada sıradaki veri.
    front++;

    //Eleman sayısını kontrol ediyoruz 0sa rear front sıfırlıyoruz.
    if (elemansayisi() == 0)
    {
     front = 0;
     rear = -1;
    }
    //Eleman sayısı 1se kuyruğun son elamanı ile ilk elemanı aynı.

    if (elemansayisi() == 1)
    {
     kuyruk[0] = kuyruk[front];
     front = 0;
     rear = 0;

    }
    return data;
}

//Eleman sayısını alıyoruz bunun için son kayıt-ilk kayıt+1 formülü
static int elemansayisi()
{
    return rear-front+1;
}


static void Main(string[] args)
{
    //listenin içine 20 eleman dolduruyoruz
    for (int i = 0; i < 20; i++)
    {
     ekle(i);
    }

    //Doldurulan bu 20 elemanı ekrana yazdırıyoruz.
    for (int i = 0; i < 20; i++)
    {
Console.WriteLine(elemanal());
    }

    Console.ReadLine();
}
```

Kuyruk işlemleri 2

```
//Tekli Linked list ilk elemanı için kullanılacak değişkene boş
değer atıyoruz.
static Tekli head = null;

//Tekli Liste içindeki ilk eleman alınıyor.
//Tekli liste ilk elemanı listenin sonraki elemanı olarak atanıyor.
static int elamanaltekli()
{
    int data = head.data;
    head = head.next;
    return data;
}

//Tekli Liste içine kayıt ekleniyor.
static void ekletekli(int data)
{
    Tekli q = new Tekli();
    q.data = data;
    q.next = null;
    if (head == null)
    {
     //Linked list içinde hiç kayıt yoksa head değeri yeni eklenen
     oluyor.
     head = q;
    }
    else
    {
     //Listede kayıt varsa listenin son elemanı ilk elemanı olarak
     gösteriliyor.
     //Liste içindeki next değeri boş oluncaya kadar yani son
     elemana kadar dönülüyor.
     Tekli last = head;
     while (last.next != null)
     {
         //Her Last değeri Listenin sonraki kaydı olarak
     güncelleniyor.
         last = last.next;
     }
     //döngü sonucu bulunan Tekli Listenin son kaydının nexti yeni
     eklenen q değeri olarak ekleniyor.
     last.next = q;
    }
}
```

```
//Çiftli Linked Listin ilk ve son elemanları için kullanılacak
değişkenlere boş değer atıyoruz
static Ciftli rearCiftli = null;
static Ciftli frontCiftli = null;

//Çiftli Linked List İçin Kuyruğa ekleme işlemi yapıyoruz.
static void ekleCiftli(int data)
{
    Ciftli q = new Ciftli();
    q.data = data;
    q.next = null;
    q.prev = null;
    if(frontCiftli == null) {
     //Çiftli Linked Listede hiç eleman olmadığı için Kuyruğun ilk
     ve son Kaydı yeni oluşturulan kayıt
     frontCiftli = q;
     rearCiftli = q;
    }
    Else {
     //son tanımlanan rearCiftli verisinin sonraki kaydı yeni
     eklenen değer olarak belirleniyor
     rearCiftli.next = q;
     //yeni eklenen kaydın önceki değeri Mevcut rearCiftli kaydı
     oluyor
     q.prev = rearCiftli;
     //Son olarak son kayıt olacak rearCiftli verisi yeni eklenen
     kayıt yapılıyor.
     rearCiftli = q;
    }
}
//Çiftli Linked List içinde Eleman Alınıyor
static int elamanalCiftli() {
    //Çiftli Linked list içindeki ilk kayıt datası alınıyor
    int data = frontCiftli.data;
    //Çiftli linked listin ilk kaydının sonraki değeri kontrol
ediliyor.
    if (frontCiftli.next != null) {
//Çiftli Linked listin ilk elemanı sonraki kaydının önceki verisi
boş
//burada cümle saçma gibi geliyor ama yapılan işlem bu. böylece bu
kayıt içinde veri olmayacak
frontCiftli.next.prev = null;
    }
    //Çiftli Linked Listin ilk kaydı ilk kayıttan sonra gelen kayıt
olarak atanıyor.
    frontCiftli = frontCiftli.next;
    return data;
}
```

```
//Çiftli Liste Sınıfı
public class Ciftli
{
    public Ciftli next;
    public Ciftli prev;
    public int data;
}

//Tekli Liste Sınıfı
public class Tekli
{
    public Tekli next;
    public int data;
}

static void Main(string[] args)
{
    //Tekli Linked listenin içine 20 eleman dolduruyoruz
    for (int i = 0; i < 20; i++)
    {
     ekletekli(i);
    }

    //Tekli Linked List içine Doldurulan bu 20 elemanı ekrana
yazdırıyoruz.
    for (int i = 0; i < 20; i++)
    {
     Console.WriteLine(elamanaltekli());
    }

    //Çiftli Linked listenin içine 20 eleman dolduruyoruz
    for (int i = 0; i < 20; i++)
    {
     ekleCiftli(i);
    }

    //Tekli Linked List içine Doldurulan bu 20 elemanı ekrana
yazdırıyoruz.
    for (int i = 0; i < 20; i++)
    {
     Console.WriteLine(elamanalCiftli());
    }

    Console.ReadLine();
}
```

Postfix, infix, prefix İşlemleri

```
static char[] stack = new char[100];
static int sp = -1;
static void push(char ch)
{
    sp++;
    stack[sp] = ch;
}
static char pop()
{
    char ch = stack[sp];
    sp--;
    return ch;
}

static char peek()
{
    return stack[sp];
}

static void Main(string[] args)
{
    string infix = "a*b+c*d";
    string postfix = "";
    push('$');
    string vars = "abcdefghıijklmonopqrstuvwxyz";
    string ops = "$(+-*/";
    int[] ort = new int[6];
    ort[0] = 0; ort[1] = 0;
    ort[2] = 1; ort[3] = 1;
    ort[4] = 2; ort[5] = 2;
```

```
    for (int i = 0; i < infix.Length; i++)
    {
     if (infix[i] == '(')
     {
         push(infix[i]);
         continue;

     }

     if (infix[i] == ')')
     {
         while (peek() != '(')
         {
          postfix += pop();
         }
         continue;

     }

     if (vars.IndexOf(infix[i]) != -1)
     {
         postfix += infix[i];
     }
     else {
         if (ort[ops.IndexOf(peek())] < ort[ops.IndexOf(infix[i])])
         {
          push(infix[i]);
         }
         else
         {
          postfix += pop(); push(infix[i]);
         }
     }


    }

    while (peek() != '$')
    {
     postfix += pop();
    }

    Console.WriteLine(postfix);
    Console.ReadLine();

}
```

Postfix, infix, prefix İşlemleri 2

```
static char[] stack = new char[100];
static int sp = -1;
static void push(char ch)
{
    sp++;
    stack[sp] = ch;
}
static char pop()
{
    char ch = stack[sp];
    sp--;
    return ch;
}

static char peek()
{
    return stack[sp];
}

static int topla(int a , int b)
{
    return a + b;
}
static int cikar(int a, int b)
{
    return a - b;
}
static int carp(int a, int b)
{
    return a * b;
}
static int bol(int a, int b)
{
    return a/b;
}
```

```
static void Main(string[] args)
{
    string postfix = "ab+cd*+e+";
    string ops = "*/+-";
    string harfler = "abcdefgh";
    int[] vars = new int[10];
    vars[0] = 2; //a harfinin işlem için değeri
    vars[1] = 3; //b harfinin işlem için değeri
    vars[2] = 4; //c harfinin işlem için değeri
    vars[3] = 5; //d harfinin işlem için değeri
    vars[4] = 6; //e harfinin işlem için değeri
    vars[5] = 7; //f harfinin işlem için değeri
    vars[6] = 8; //g harfinin işlem için değeri
    vars[7] = 9; //h harfinin işlem için değeri
    //işlem içinde her karakter için kontrol yapılacak

    for (int i = 0; i < postfix.Length; i++)
    {
     //indis değeri harf ise stack içine atılıyor
     if (harfler.IndexOf(postfix[i]) >= 0)
     {
         push(postfix[i]);
     }
     else
     {

         //indis değeri işlemse stack içindeki son 2 kayıt pop ile
     alınıp sayi1 ve sayi2 olarak atanıyor.
         char harf1 = pop();
         int sayi1 = vars[harfler.IndexOf(harf1)];
         char harf2 = pop();
         int sayi2 = vars[harfler.IndexOf(harf2)];
         int sonuc = 0;
         if (postfix[i] == '*')
         {
          sonuc = carp(sayi1, sayi2);
         }
         if (postfix[i] == '+')
         {
          sonuc = topla(sayi1, sayi2);
         }
         if (postfix[i] == '/')
         {
          sonuc = bol(sayi1, sayi2);
         }
         if (postfix[i] == '-')
         {
          sonuc = cikar(sayi1, sayi2);
         }
```

```
            push('a');
            vars[harfler.IndexOf('a')] = sonuc;

            Console.WriteLine(sonuc);

        }
    }

    Console.ReadLine();
```

Postfix, infix, prefix

Herhangi bir yere operatör koymamamızın önünde bir engel yoktur.

Operatör Önde (Prefix) : + a b

Biçim: işlem işlenen işlenen (operator operand operand) şeklindedir: + 2 7

İşlem sağdan sola doğru ilerler. Öncelik (parantez) yoktur.

Operatör Arada (Infix) : a + b

Biçim: işlenen işlem işlenen (operand operator operand) şeklindedir: 2 + 7

İşlem öncelik sırasına göre ve soldan sağa doğru ilerler.

Operatör Sonda (Postfix) : a b +

Biçim: işlenen işlenen işlem (operand operand operator) şeklindedir: 2 7 +

İşlem soldan sağa doğru ilerler. Öncelik (parantez) yoktur.

İşlem önceliği (büyükten küçüğe)

1. Parantez
2. Üs Alma
3. Çarpma /Bölme
4. Toplama/Çıkarma

- Parantezsiz ve aynı önceliğe sahip işlemcilerde soldan sağa doğu yapılır (üs hariç).
- Üs almada sağdan sola doğrudur. A-B+C'de öncelik (A-B)+C şeklindedir. A^B^C'de ise A^(B^C)

Parantez -Infix

2+3*5 işlemini gerçekleştiriniz.

+ önce ise:

(2+3)*5 = 5*5 = 25

***** önce ise:

2+(3*5) = 2+15 = 17

Infix gösterim paranteze ihtiyaç duyar.

Prefix Gösterim

Paranteze ihtiyaç yok!

+2*35=	
	=+2*35
	=+215=17
*+235=	
	=*+235
	=*55=25

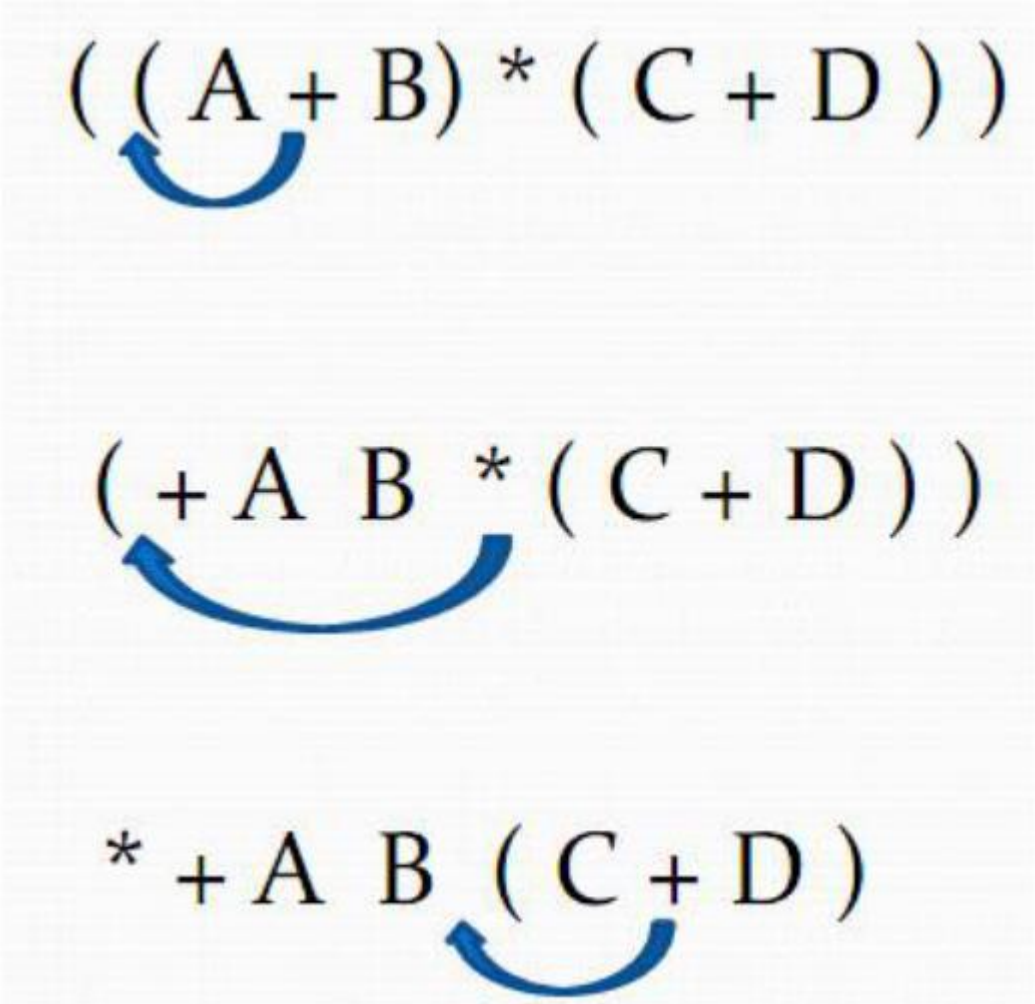
Postfix Gösterim

Paranteze ihtiyaç yok!

235*+=	
	=235*+
	=215+=17
23+5*=	
	=23+5*
	=55*=25

Infix'ten Prefix'e Dönüşüm

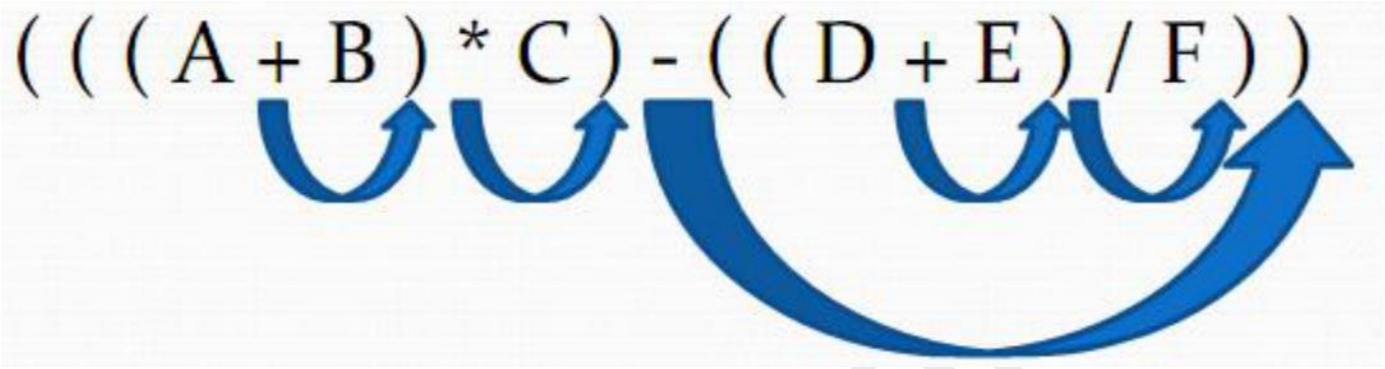
Her bir operatörü kendi işlenenlerinin soluna taşı veparantezleri kaldır. :



*+AB+C D

İşlenenlerin sırasında bir değişiklik olmadı!

Infix'ten Postfix'e Dönüşüm



((AB+*C)-((D+E)/F))
(AB+C*-((D+E)/F))
AB+C*((D+E)/F)-
AB+C* (DE+ / F)-
A B+C* D E+F/-

Infix, Prefix, Postfix

Infix	Postfix	Prefix
A+B-C	**AB+C-**	**-+ABC**
(A+B)*(C-D)	**AB+CD-***	***+AB-CD**
A^B*C-D+E/F/(G+H)	**AB^C*D-EF/GH+/+**	**+-*^ABCD//EF+GH**
((A+B)*C-(D-E))^(F+G)	**AB+C*DE—FG+^**	**^-*+ABC-DE+FG**
A-B/(C*D^E)	**ABCDE^*/-**	**-A/B*C^DE**

Infix, Prefix, Postfix İşlemleri

çevrilmesi :a + b * c – d		
Okunan	Yığıt	Çıktı /Operatör sonda ifade
a		a
+	+	a
b	+	a b
*	+ *	a b
c	+ *	a b c
-	+ *	a b c
	+	a b c *
	-	a b c * +
d	-	a b c * + d -

Operatör Sonda (Postfix) İfade işlenişi

Örnek: a b c * + d -	ifadesini a=2 b=3 c=5 d=10 -->	
Okunan	Yığıt	Hesaplanan
2	2	
3	2 3	
5	2 3 5	
*	2	islem=* pop1=5 pop2=3
	2 15	3 * 5=15
+	17	islem=+ pop1=15 pop2=2
		2+15=17
10	17 10	
-	7	islem=- pop1=10 pop2=17
		17–10=7
		a b c * + d -

Infix, Prefix, Postfix İşlemleri

çevrilmesi. Infix ifade: (2 + 8) / (4 – 3)

Okunan	Yığıt	Hesaplanan
(	(	
2	(	2
+	(+	2
8	(+	2 8
)		2 8 +
/	/	2 8 +
(	/ (	2 8 +
4	/ (	2 8 + 4
-	/ (-	2 8 + 4
3	/ (-	2 8 + 4 3
)	/	2 8 + 4 3 –
		28+43–
		2 8 + 43–/

Infix, Prefix, Postfix İşlemleri

Infix	Prefix	Postfix
2x(3+5)-7^2(2+1)	?	?
?	**++x23^-5721**	?
?	?	**235+7-^2x1+**
2x3+5-7^2+1	?	?