BLM267

Bölüm 7: Yığınlar

C Kullanarak Veri Yapıları, İkinci Baskı
Reema Thareja

- Yığınlara Giriş
- Yığınların Dizi Gösterimi
- Bir Yığın Üzerindeki İşlemler
- Yığınların Bağlantılı Gösterimi
- Bağlantılı Yığın Üzerindeki İşlemler
- Çoklu Yığınlar
- Yığınların Uygulaması

Yığınlara Giriş

- Yığın, elemanlarını düzenli bir şekilde saklayan önemli bir veri yapısıdır.
- Yığın kavramını bir benzetme kullanarak açıklayalım.
- Şekil 7.1'de görüldüğü gibi, bir tabağın diğerinin üzerine yerleştirildiği bir tabak yığınını görmüşsünüzdür.
- Şimdi bir tabağı çıkarmak istediğinizde önce en üstteki tabağı çıkarmalısınız.
- Bu nedenle, bir elemanı (örneğin bir plakayı) yalnızca en üst konumdan ekleyebilir veya kaldırabilirsiniz.

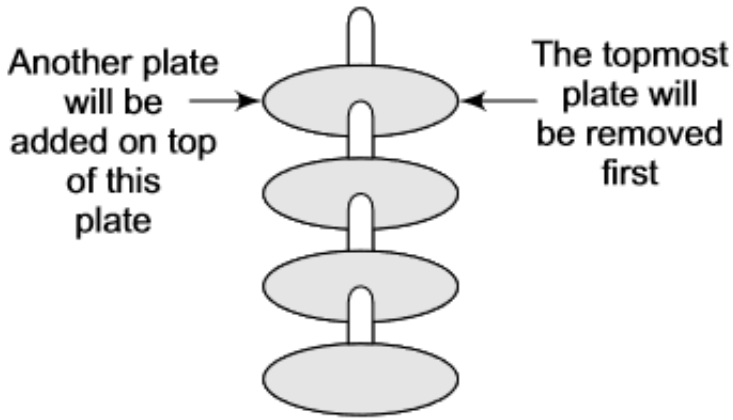

Figure 7.1 Stack of plates

Yığınlara Giriş

- Yığın, aynı prensibi kullanan doğrusal bir veri yapısıdır, yani yığındaki elemanlar yalnızca TOP adı verilen bir uçtan eklenir veya çıkarılır.
- Bu nedenle, bir yığına LIFO (Son Giren İlk Çıkar) veri yapısı denir, çünkü en son eklenen eleman ilk çıkarılacak elemandır.
- Peki şimdi soru şu: Bilgisayar biliminde yığınlara nerede ihtiyacımız var?
- Cevap fonksiyon çağrılarındadır.
- Bir örnek verelim; A fonksiyonunu çalıştırıyoruz.
- A fonksiyonu çalışması sırasında başka bir B fonksiyonunu çağırır.
- Fonksiyon B, sırayla başka bir fonksiyon olan C'yi çağırır ve C de fonksiyon D'yi çağırır.

Yığınlara Giriş

- Her aktif fonksiyonun geri dönüş noktasını takip edebilmek için sistem yığını veya çağrı yığını adı verilen özel bir yığın kullanılır.
- Bir fonksiyon başka bir fonksiyonu çağırdığında, çağıran fonksiyon yığının en üstüne itilir.
- Bunun sebebi çağrılan fonksiyon çalıştırıldıktan sonra kontrolün tekrar çağıran fonksiyona geçmesidir.
- Bu kavramı gösteren Şekil 7.2'ye bakın.

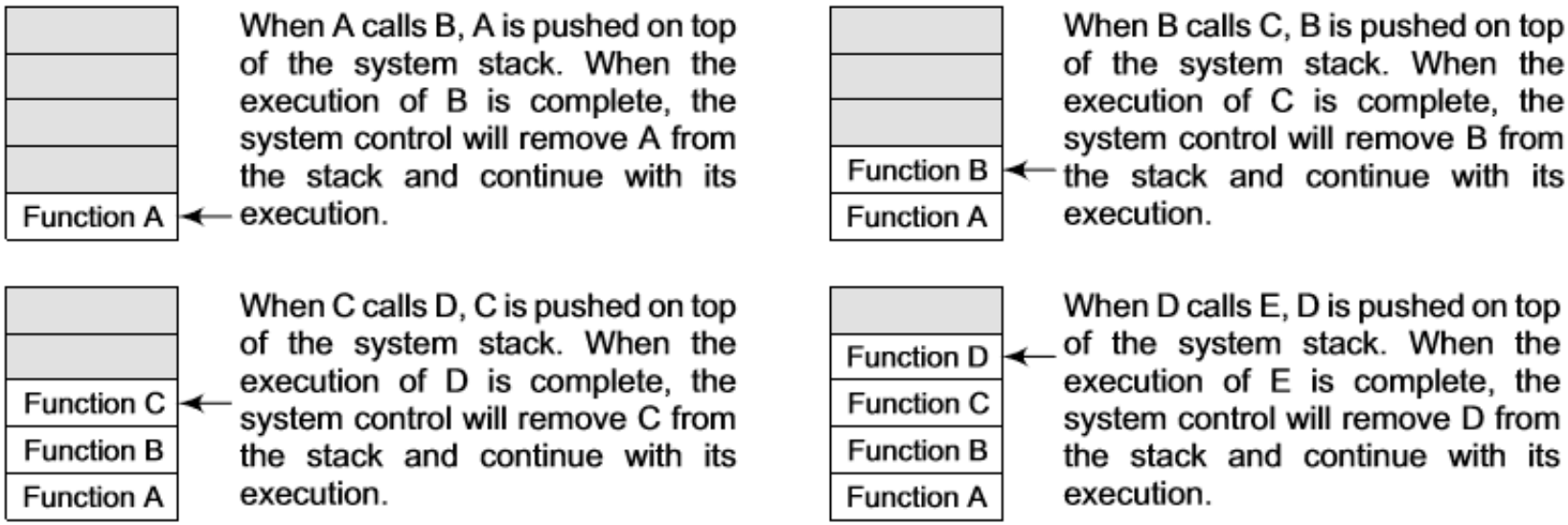

Figure 7.2 System stack in the case of function calls

Yığınlara Giriş

- Artık E fonksiyonu çalıştırıldığında D fonksiyonu yığının en üstünden kaldırılacak ve çalıştırılacak.
- D fonksiyonu tamamen yürütüldüğünde, C fonksiyonu yürütülmek üzere yığından kaldırılacaktır.
- Tüm fonksiyonlar yürütülene kadar tüm prosedür tekrarlanacaktır.
- Her fonksiyon çalıştırıldıktan sonra yığına bakalım.
- Bu durum Şekil 7.3'te gösterilmiştir. Sistem yığını, fonksiyonların düzgün bir şekilde yürütülme sırasını garanti eder.
- Bu nedenle, özellikle diğer koşullar sağlanana kadar işlemin ertelenmesi gerektiğinde, işlem sırasının çok önemli olduğu durumlarda yığınlar sıklıkla kullanılır.
- Yığınlar diziler veya bağlı listeler kullanılarak uygulanabilir. Aşağıdaki bölümlerde, yığınların hem dizi hem de bağlı liste uygulamasını ele alacağız.

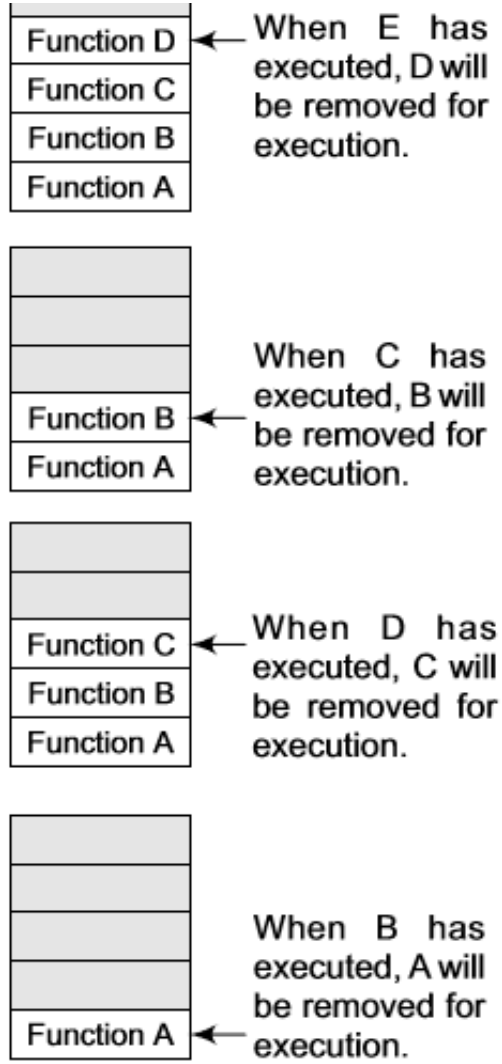

Figure 7.3 System stack when a called function returns

Yığınların Dizi Gösterimi

- Bilgisayarın belleğinde yığınlar doğrusal bir dizi olarak temsil edilebilir.
- Her yığının, yığının en üst elemanının adresini depolamak için kullanılan TOP adında bir değişkeni vardır.
- Elemanın ekleneceği veya silineceği konum burasıdır.
- Yığının tutabileceği maksimum eleman sayısını depolamak için kullanılan MAX adında bir değişken daha vardır.
- Eğer TOP = NULL ise yığının boş olduğunu, eğer TOP = MAX–1 ise yığının dolu olduğunu gösterir.
- Şekil 7.4'teki yığın TOP = 4'ü gösterir, bu nedenle eklemeler ve silmeler bu konumda yapılacaktır. Yukarıdaki yığında, beş eleman daha saklanabilir.

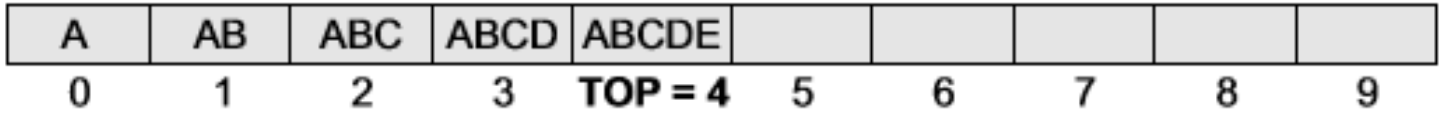

Figure 7.4 Stack

Bir Yığın Üzerindeki İşlemler

- Bir yığın üç temel işlemi destekler: itme, çıkarma ve göz atma.
- İtme işlemi bir elemanı yığının en üstüne ekler ve çıkarma işlemi elemanı yığının en üstünden kaldırır.
- Peek işlemi yığının en üst elemanının değerini döndürür.
- **İtme İşlemi**
- İtme işlemi, yığına bir eleman eklemek için kullanılır.
- Yeni eleman yığının en üst noktasına eklenir.
- Ancak değeri eklemeden önce TOP=MAX–1 olup olmadığını kontrol etmeliyiz, çünkü eğer durum buysa yığın doludur ve daha fazla ekleme yapılamaz.
- Zaten dolu olan bir yığına bir değer ekleme girişimi yapılırsa, bir OVERFLOW mesajı yazdırılır. Şekil 7.5'te verilen yığını ele alalım.

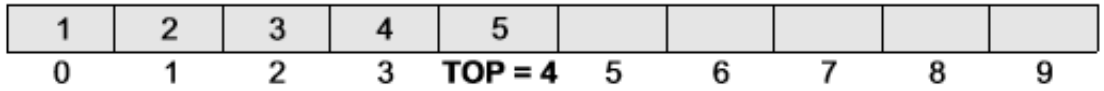

Figure 7.5 Stack

Bir Yığın Üzerindeki İşlemler

- Değeri 6 olan bir eleman eklemek için öncelikle TOP=MAX–1 olup olmadığını kontrol ederiz.
- Eğer koşul yanlışsa, TOP değerini artırırız ve yeni elemanı stack[TOP] tarafından verilen konuma depolarız.
- Böylece güncellenen yığın Şekil 7.6'daki gibi olur.
- Şekil 7.7 bir yığına eleman ekleme algoritmasını göstermektedir.
- 1. Adımda öncelikle OVERFLOW durumunu kontrol ediyoruz.
- 2. Adımda TOP, dizideki bir sonraki konuma işaret edecek şekilde artırılır.
- Adım 3'te değer, TOP'un işaret ettiği konumdaki yığında saklanır.

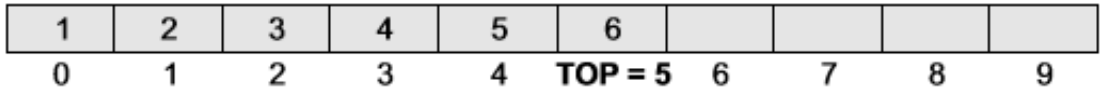

Figure 7.6 Stack after insertion

```
Step 1: IF TOP = MAX-1
            PRINT "OVERFLOW"
            Goto Step 4
        [END OF IF]
Step 2: SET TOP = TOP + 1
Step 3: SET STACK[TOP] = VALUE
Step 4: END
```

Figure 7.7 Algorithm to insert an element in a stack

Bir Yığın Üzerindeki İşlemler

- Pop Operasyonu
- Pop işlemi yığındaki en üst elemanı silmek için kullanılır.
- Ancak değeri silmeden önce TOP=NULL olup olmadığını kontrol etmeliyiz çünkü eğer durum böyleyse, yığın boş demektir ve daha fazla silme işlemi yapılamaz.
- Zaten boş olan bir yığından bir değeri silmeye çalışılırsa UNDERFLOW mesajı yazdırılır.
- Şekil 7.8'de verilen yığını ele alalım.

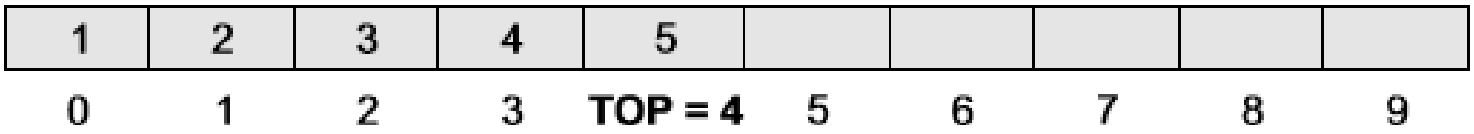

Figure 7.8 Stack

Bir Yığın Üzerindeki İşlemler

- Pop Operasyonu
- En üstteki öğeyi silmek için önce TOP=NULL olup olmadığını kontrol ederiz. Koşul yanlışsa, TOP tarafından işaret edilen değeri azaltırız.
- Böylece güncellenen yığın Şekil 7.9'daki gibi olur.
- Şekil 7.10 bir yığından bir elemanı silmek için kullanılan algoritmayı göstermektedir.
- 1. Adımda öncelikle UNDERFLOW koşulunu kontrol ediyoruz.
- Adım 2'de TOP'un işaret ettiği yığındaki konumun değeri VAL'de saklanır.
- 3. Adımda TOP azaltılır.

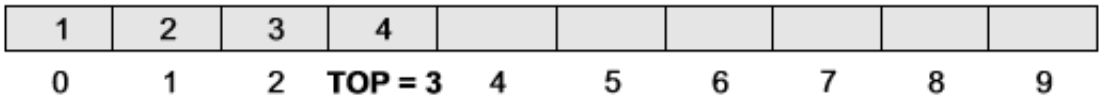

Figure 7.9 Stack after deletion

```
Step 1: IF TOP = NULL
            PRINT "UNDERFLOW"
            Goto Step 4
        [END OF IF]
Step 2: SET VAL = STACK[TOP]
Step 3: SET TOP = TOP - 1
Step 4: END
```

Figure 7.10 Algorithm to delete an element from a stack

Bir Yığın Üzerindeki İşlemler

- Gözetleme Operasyonu
- Peek, yığının en üstündeki elemanın değerini, onu yığından silmeden döndüren bir işlemdir.
- Peek işleminin algoritması Şekil 7.11'de verilmiştir.
- Ancak Peek işlemi öncelikle yığının boş olup olmadığını kontrol eder, yani eğer TOP = NULL ise uygun bir mesaj yazdırılır, aksi takdirde değer döndürülür.
- Şekil 7.12'de verilen yığını ele alalım.
- Burada Peek işlemi, yığının en üst elemanının değeri olduğu için ir.

```
Step 1: IF TOP = NULL
            PRINT "STACK IS EMPTY"
            Goto Step 3
Step 2: RETURN STACK[TOP]
Step 3: END
```

Figure 7.11 Algorithm for Peek operation

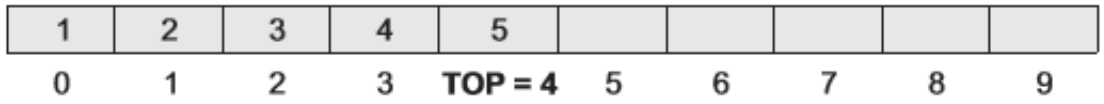

Figure 7.12 Stack

Bir Yığın Üzerindeki İşlemler

```
void push(int st[], int val)
{
        if(top == MAX-1)
        {
                printf("\n STACK OVERFLOW");
        }
        else
        {
                top++;
                st[top] = val;
        }
}
int pop(int st[])
{
        int val;
        if(top == -1)
        {
                printf("\n STACK UNDERFLOW");
                return -1;
        }
        else
        {
                val = st[top];
                top--;
                return val;
        }
}
```

Bir Yığın Üzerindeki İşlemler

```
void display(int st[])
{
        int i;
        if(top == -1)
        printf("\n STACK IS EMPTY");
        else
        {
                for(i=top;i>=0;i--)
                printf("\n %d",st[i]);
                printf("\n"); // Added for formatting purposes
        }
}
int peek(int st[])
{
        if(top == -1)
        {
                printf("\n STACK IS EMPTY");
                return -1;
        }
        else
        return (st[top]);
}
```

Yığınların Bağlantılı Gösterimi

- Bir dizinin kullanılarak yığının nasıl oluşturulduğunu gördük.
- Yığın oluşturma tekniği kolaydır, ancak dezavantajı dizinin sabit bir boyuta sahip olacak şekilde bildirilmesi gerekliliğidir.
- Yığın çok küçükse veya yığının maksimum boyutu önceden biliniyorsa, yığının dizi uygulaması verimli bir uygulama verir.
- Ancak dizinin boyutu önceden belirlenemiyorsa, diğer alternatif, yani bağlı gösterim kullanılır.
- n elemanlı yığının bağlantılı gösteriminin depolama gereksinimi O(n)'dir ve işlemler için tipik zaman gereksinimi O(1)'dir.

Yığınların Bağlantılı Gösterimi

- Bağlantılı bir yığında, her düğümün iki bölümü vardır: Biri verileri depolar, diğeri ise bir sonraki düğümün adresini depolar.
- Bağlı listenin BAŞLANGIÇ işaretçisi TOP olarak kullanılır.
- Tüm eklemeler ve silmeler TOP tarafından işaret edilen düğümde yapılır. TOP = NULL ise, yığının boş olduğunu gösterir.
- Bir yığının bağlantılı gösterimi Şekil 7.13'te gösterilmiştir.

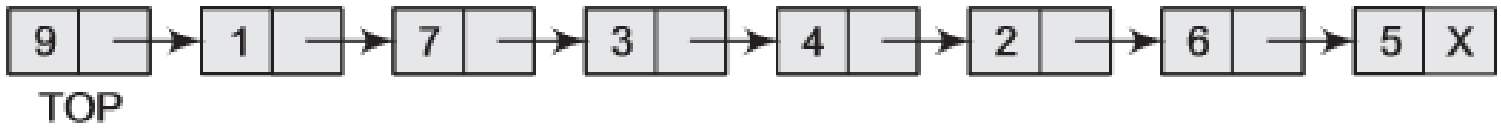

Figure 7.13 Linked stack

Bağlantılı Yığın Üzerindeki İşlemler

- İtme İşlemi
- İtme işlemi, yığına bir eleman eklemek için kullanılır.
- Yeni eleman yığının en üst pozisyonuna eklenir. Şekil 7.14'te gösterilen bağlantılı yığını ele alalım.

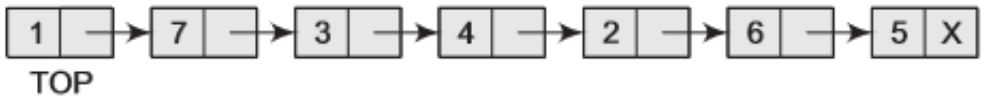

Figure 7.14 Linked stack

- Değeri 9 olan bir eleman eklemek için öncelikle TOP=NULL olup olmadığını kontrol ederiz.
- Eğer durum böyleyse, yeni bir node için bellek ayırıp, DATA kısmına değeri, NEXT kısmına ise NULL değerini depolarız.
- Yeni düğüme daha sonra TOP adı verilecek.
- Ancak eğer TOP!=NULL ise, yeni düğümü bağlı yığının başına ekleriz ve bu yeni düğüme TOP adını veririz.
- Böylece güncellenen yığın Şekil 7.15'te görüldüğü gibi olur.

Figure 7.15 Linked stack after inserting a new node

Bağlantılı Yığın Üzerindeki İşlemler

- Şekil 7.16, bir elemanı bağlantılı yığına itmek için kullanılan algoritmayı göstermektedir.
- Adım 1'de, yeni düğüm için bellek tahsis edilir. Adım 2'de, yeni düğümün DATA kısmı, düğümde depolanacak değerle başlatılır.
- 3. Adımda yeni düğümün bağlı listenin ilk düğümü olup olmadığını kontrol ediyoruz.
- Bu, TOP = NULL olup olmadığını kontrol ederek yapılır.
- Eğer IF ifadesi doğru olarak değerlendirilirse, düğümün NEXT kısmına NULL kaydedilir ve yeni düğüme TOP adı verilir.
- Ancak eğer yeni düğüm listedeki ilk düğüm değilse, listenin ilk düğümünden (yani TOP düğümü[illegible]landırılır.

```
Step 1: Allocate memory for the new
        node and name it as NEW_NODE
Step 2: SET NEW_NODE -> DATA = VAL
Step 3: IF TOP = NULL
            SET NEW_NODE -> NEXT = NULL
            SET TOP = NEW_NODE
        ELSE
            SET NEW_NODE -> NEXT = TOP
            SET TOP = NEW_NODE
        [END OF IF]
Step 4: END
```

Figure 7.16 Algorithm to insert an element in a linked stack

Bağlantılı Yığın Üzerindeki İşlemler

- Pop Operasyonu
- Pop işlemi bir yığından en üstteki elemanı silmek için kullanılır.
- Ancak değeri silmeden önce TOP=NULL olup olmadığını kontrol etmeliyiz, çünkü eğer durum böyleyse, yığın boş demektir ve daha fazla silme işlemi yapılamaz.
- Zaten boş olan bir yığından bir değeri silmeye çalışılırsa UNDERFLOW mesajı yazdırılır.
- Şekil 7.17'de gösterilen yığını ele alalım.

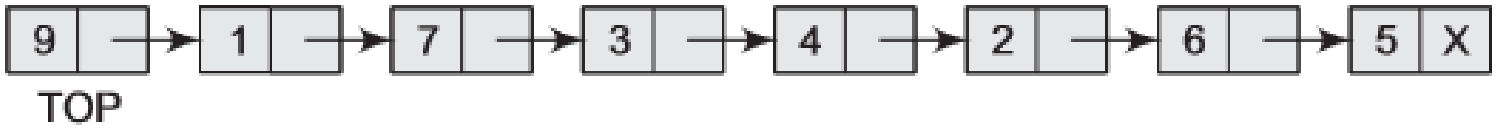

Figure 7.17 Linked stack

Bağlantılı Yığın Üzerindeki İşlemler

- Pop Operasyonu
- TOP!=NULL olması durumunda TOP'un işaret ettiği düğümü sileceğiz ve TOP'un bağlantılı yığının ikinci elemanına işaret etmesini sağlayacağız.
- Böylece güncellenen yığın Şekil 7.18'deki gibi olur.
- Şekil 7.19 bir yığından bir elemanı silmek için kullanılan algoritmayı göstermektedir.
- Adım 1'de, ilk olarak UNDERFLOW koşulunu kontrol ediyoruz. Adım 2'de, TOP'a işaret eden bir işaretçi PTR kullanıyoruz.
- 3. Adımda TOP'un sıradaki düğüme işaret etmesi sağlanır.
- 4. Adımda PTR'nin kapladığı bellek boş havuza geri verilir.

```
Step 1: IF TOP = NULL
            PRINT "UNDERFLOW"
            Goto Step 5
        [END OF IF]
Step 2: SET PTR = TOP
Step 3: SET TOP = TOP -> NEXT
Step 4: FREE PTR
Step 5: END
```

Figure 7.19 Algorithm to delete an element from a linked stack

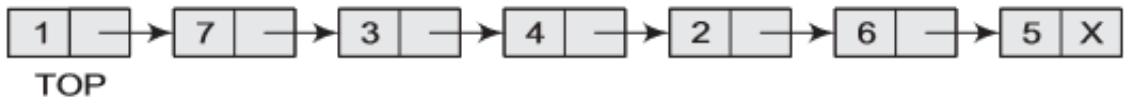

Figure 7.18 Linked stack after deletion of the topmost element

Bağlantılı Yığın Üzerindeki İşlemler

```
struct stack *push(struct stack *top, int val)
{
        struct stack *ptr;
        ptr = (struct stack*)malloc(sizeof(struct stack));
        ptr -> data = val;
        if(top == NULL)
        {
                ptr -> next = NULL;
                top = ptr;
        }
        else
        {
                ptr -> next = top;
                top = ptr;
        }
        return top;
}
```

Bağlantılı Yığın Üzerindeki İşlemler

```
struct stack *display(struct stack *top)
{
        struct stack *ptr;
        ptr = top;
        if(top == NULL)
        printf("\n STACK IS EMPTY");
        else
        {
                while(ptr != NULL)
                {
                        printf("\n %d", ptr -> data);
                        ptr = ptr -> next;
                }
        }
        return top;
}
```

Bağlantılı Yığın Üzerindeki İşlemler

```
struct stack *pop(struct stack *top)
{
        struct stack *ptr;
        ptr = top;
        if(top == NULL)
        printf("\n STACK UNDERFLOW");
        else
        {
                top = top -> next;
                printf("\n The value being deleted is: %d", ptr -> data);
                free(ptr);
        }
        return top;
}
int peek(struct stack *top)
{
        if(top==NULL)
        return -1;
        else
        return top ->data;
}
```

Çoklu Yığınlar

- Bir dizi kullanarak yığın uygularken, dizinin boyutunun önceden bilinmesi gerektiğini görmüştük.
- Yığına daha az alan tahsis edilirse, sık sık OVERFLOW durumuyla karşılaşılacaktır.
- Bu sorunla başa çıkabilmek için, diziye daha fazla alan tahsis etmek amacıyla kodun değiştirilmesi gerekecektir.
- Yığın için çok fazla alan ayırmamız durumunda, bu durum tamamen bellek israfına yol açabilir.
- Dolayısıyla taşma sıklığı ile ayrılan alan arasında bir denge söz konusudur.
- Dolayısıyla bu problemle başa çıkmak için daha iyi bir çözüm, birden fazla yığın kullanmak veya aynı dizide yeterli büyüklükte birden fazla yığın bulundurmaktır.
- Şekil 7.20 bu kavramı göstermektedir.

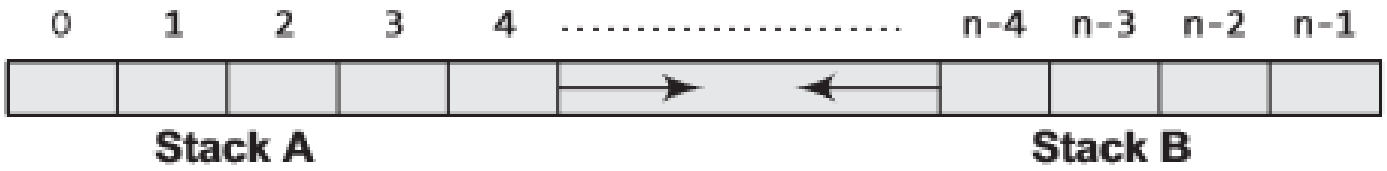

Figure 7.20 Multiple stacks

Çoklu Yığınlar

- Şekil 7.20'de, Yığın A ve Yığın B olmak üzere iki yığını temsil etmek için bir STACK[n] dizisi kullanılmıştır.
- n değeri, her iki yığının toplam boyutunun hiçbir zaman n'yi aşmayacağı şekildedir.
- Bu yığınlar üzerinde işlem yaparken bir noktaya dikkat etmek önemlidir: Yığın A soldan sağa doğru büyürken, Yığın B aynı anda sağdan sola doğru büyüyecektir.
- Bu kavramı birden fazla yığına genişletirsek, bir yığın aynı dizideki n sayıda yığını temsil etmek için de kullanılabilir.
- Yani, eğer bir STACK[n] varsa, o zaman her yığın I'e b[i] ve e[i] dizinleriyle sınırlanmış eşit miktarda alan tahsis edilecektir.
- Bu durum Şekil 7.21'de gösterilmiştir.

Figure 7.21 Multiple stacks

Yığınların Uygulamaları

- Bu bölümde, yığınların basit ve etkili bir çözüm için kolayca uygulanabileceği tipik problemleri tartışacağız.
- Bu bölümde ele alınacak konular şunlardır:
  - Bir listeyi tersine çevirme
  - Parantez denetleyicisi
  - Bir infix ifadesinin bir postfix ifadesine dönüştürülmesi
  - Bir postfix ifadesinin değerlendirilmesi
  - Bir infix ifadesinin bir prefix ifadesine dönüştürülmesi
  - Bir önek ifadesinin değerlendirilmesi

Yığınların Uygulamaları

- Bir Listeyi Tersine Çevirme
- Bir sayı listesini tersine çevirmek için, her sayıyı ilk indeksten başlayarak bir diziden okuyup bir yığına koymak gerekir.
- Tüm sayılar okunduktan sonra, sayılar tek tek seçilip ilk indeksten başlanarak diziye kaydedilebilir.

Yığınların Uygulamaları

4. Write a program to reverse a list of given numbers.

```
#include <stdio.h>
int main()
{
        int val, n, i,
        arr[10];
        clrscr();
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        printf("\n Enter the elements of the array : ");
        for(i=0;i<n;i++)
                scanf("%d", &arr[i]);
        for(i=0;i<n;i++)
                push(arr[i]);
        for(i=0;i<n;i++)
        {
                val = pop();
                arr[i] = val;
        }
        printf("\n The reversed array is : ");
        for(i=0;i<n;i++)
                printf("\n %d", arr[i]);
        getche");
        return 0;
}
void push(int val)
{
        stk[++top] = val;
```

Yığınların Uygulamaları

- **Parantez Denetleyicisini Uygulama**
- **Yığınlar, herhangi bir cebirsel ifadedeki parantezlerin geçerliliğini kontrol etmek için kullanılabilir.**
- **Örneğin, her açık parantezin bir kapanış parantezi varsa cebirsel bir ifade geçerlidir.**
- **Örneğin, (A+B} ifadesi geçersizdir ancak {A + (B – C)} ifadesi geçerlidir.**
- **Aşağıdaki programda, bir cebirsel ifadenin geçerliliğini kontrol etmek için ifadeyi tarayan bir program görebilirsiniz.**

5. Write a program to check nesting of parentheses using a stack.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 10
int top = -1;
int stk[MAX];
void push(char);

char pop();
```

Yığınların Uygulamaları

```
{
        char exp[MAX],temp;
        int i, flag=1;
        clrscr();
        printf("Enter an expression : ");
        gets(exp);
        for(i=0;i<strlen(exp);i++)
        {
                if(exp[i]=='(' || exp[i]=='{' || exp[i]=='[')
                        push(exp[i]);
                if(exp[i]==')' || exp[i]=='}' || exp[i]==']')
                        if(top == -1)
                                flag=0;
                        else
                        {
                                temp=pop();
                                if(exp[i]==')' && (temp=='{' || temp=='['))
                                        flag=0;
                                if(exp[i]=='}' && (temp=='(' || temp=='['))
                                        flag=0;
                                if(exp[i]==']' && (temp=='(' || temp=='{'))
                                        flag=0;
                        }
        }
        if(top>=0)
                flag=0;
        if(flag==1)
                printf("\n Valid expression");
        else
                printf("\n Invalid expression");
```

Yığınların Uygulamaları

```
void push(char c)
{
        if(top == (MAX-1))
                printf("Stack Overflow\n");
        else
        {
                top=top+1;
                stk[top] = c;
        }
}
char pop()
{
        if(top == -1)
                printf("\n Stack Underflow");
        else
                return(stk[top--]);
}
```

Output

```
Enter an expression : (A + (B - C))
Valid Expression
```

Yığınların Uygulamaları

- **Aritmetik İfadelerin Değerlendirilmesi**
- **Polonya Notasyonları**
- **Önek, sonek ve önek gösterimleri cebirsel ifadelerin yazımında kullanılan üç farklı ancak eşdeğer gösterimdir.**
- **Ancak önek ve sonek gösterimlerini öğrenmeden önce, ilk önce bir infix gösteriminin ne olduğuna bakalım.**
- **Cebirsel ifadelerin yazımında infix notasyonunun kullanıldığını hepimiz biliyoruz.**
- **İnfix gösterimi kullanılarak bir aritmetik ifade yazılırken operatör, işlenenlerin arasına yerleştirilir.**
- **Örneğin A+B; burada artı operatörü A ve B işlenenleri arasına yerleştirilmiştir.**
- **İnfiks gösterimini kullanarak ifadeleri yazmak bizim için kolay olsa da, bilgisayarın ifadeyi değerlendirebilmesi için çok fazla bilgiye ihtiyaç duyması nedeniyle, bilgisayarlar bunu ayrıştırmakta zorluk çekmektedir.**
- **Operatör önceliği ve ilişkisellik kuralları ve bu kuralları geçersiz kılan parantezler hakkında bilgiye ihtiyaç vardır.**
- **Bu nedenle bilgisayarlar, önek ve sonek gösterimleri kullanılarak yazılan ifadelerle daha verimli çalışır**

Yığınların Uygulamaları

- **Aritmetik İfadelerin Değerlendirilmesi**
- Postfix gösterimi Polonyalı mantıkçı, matematikçi ve filozof Jan Łukasiewicz tarafından geliştirilmiştir.
- Amacı, parantezsiz bir önek gösterimi (aynı zamanda Lehçe gösterimi olarak da bilinir) ve daha çok Ters Lehçe Gösterimi veya RPN olarak bilinen bir sonek gösterimi geliştirmekti.
- Adından da anlaşılacağı üzere postfix gösteriminde operatör, işlenenlerden sonra yerleştirilir.
- Örneğin, bir ifade ek gösterimde A+B şeklinde yazılıyorsa, aynı ifade ek gösterimde AB+ şeklinde de yazılabilir.
- Bir postfix ifadesinin değerlendirilme sırası her zaman soldan sağa doğrudur.
- Parantez bile değerlendirme sırasını değiştiremez.
- (A + B) * C ifadesi son ek gösteriminde AB+C* şeklinde yazılabilir.

Yığınların Uygulamaları

- Aritmetik İfadelerin Değerlendirilmesi
- Bir postfix işlemi operatör önceliği kurallarına bile uymuyor.
- İfadede ilk bulunan operatör, operandlar üzerinde ilk olarak işlem görür.
- Örneğin, AB+C* şeklinde bir sonek gösterimi verildiğinde.
- Değerlendirme yapılırken çarpma işleminden önce toplama işlemi yapılacaktır.
- Böylece postfix gösteriminde operatörlerin kendilerine hemen bırakılan işlenenlere uygulandığını görüyoruz.
- Örnekte AB+C*, A ve B'ye + uygulanır, ardından toplama işleminin sonucuna ve C'ye * uygulanır.

Yığınların Uygulamaları

- **Aritmetik İfadelerin Değerlendirilmesi**
- **Önek gösterimi de soldan sağa doğru değerlendirilse de, sonek gösterimi ile önek gösterimi arasındaki tek fark, önek gösteriminde işlecin işlenenlerden önce yerleştirilmesidir.**
- **Örneğin, A+B infix gösteriminde bir ifade ise, önek gösteriminde buna karşılık gelen ifade +AB ile verilir.**
- **Bir önek ifadesi değerlendirilirken operatörler, operatörün hemen sağında bulunan işlenenlere uygulanır.**
- **Postfix gibi, prefix ifadeleri de operatör önceliği ve ilişkisellik kurallarına uymaz ve hatta parantezler bile değerlendirme sırasını değiştiremez.**

Yığınların Uygulamaları

Example 7.1 Convert the following infix expressions into postfix expressions.

Solution

```
(a) (A-B) * (C+D)
    [AB-] * [CD+]
    AB-CD+*
(b) (A + B) / (C + D) - (D * E)
    [AB+] / [CD+] - [DE*]
    [AB+CD+/] - [DE*]
    AB+CD+/DE*-
```

Example 7.2 Convert the following infix expressions into prefix expressions.

Solution

```
(a) (A + B) * C
    (+AB)*C
    *+ABC
(b) (A-B) * (C+D)
    [-AB] * [+CD]
    *-AB+CD
(c) (A + B) / ( C + D) - ( D * E)
    [+AB] / [+CD] - [*DE]
    [/+AB+CD] - [*DE]
    -/+AB+CD*DE
```

Yığınların Uygulamaları

- **Bir İnfix İfadesinin Bir Postfix İfadesine Dönüştürülmesi**
- **I, infix gösteriminde yazılmış bir cebirsel ifade olsun.**
- **Parantez, işlenen ve operatör içerebilir.**
- **Algoritmanın basitliği için sadece +, –, *, /, % operatörlerini kullanacağız.**
- **Bu operatörlerin önceliği aşağıdaki gibi verilebilir:**
- **Daha yüksek öncelik *, /, %**
- **Daha düşük öncelik +, –**
- **Şüphesiz bu operatörlerin değerlendirme sırası parantezlerden yararlanılarak değiştirilebilir.**
- **Örneğin A + B * C şeklinde bir ifademiz varsa önce B * C işlemi yapılacak ve sonuç A'ya eklenecektir.**
- **Fakat aynı ifade (A + B) * C şeklinde yazıldığında önce A + B değerlendirilecek ve sonra sonuç C ile çarpılacaktır.**

Yığınların Uygulamaları

- **Bir İnfix İfadesinin Bir Postfix İfadesine Dönüştürülmesi**
- **Aşağıda verilen algoritma, Şekil 7.22'de gösterildiği gibi bir infix ifadesini postfix ifadesine dönüştürür.**
- **Algoritma, operatörler, işlenenler ve parantezleri içerebilen bir ön ek ifadesini kabul eder.**
- **Basitleştirmek adına, infix işleminin yalnızca modül (%), çarpma (*), bölme (/), toplama (+) ve çıkarma (—) operatörlerini içerdiğini ve aynı önceliğe sahip operatörlerin soldan sağa doğru gerçekleştirildiğini varsayalım.**
- **Algoritma, operatörleri geçici olarak tutmak için bir yığın kullanır.**
- **Postfix ifadesi, infix ifadesindeki işlenenler ve yığından çıkarılan operatörler kullanılarak soldan sağa doğru elde edilir.**
- **Bu algoritmadaki ilk adım, yığına bir sol parantez koymak ve infix ifadesinin sonuna karşılık gelen bir sağ parantez eklemektir.**
- **Yığın boşalana kadar algoritma tekrarlanır.**

Yığınların Uygulamaları

- **Bir İnfix İfadesinin Bir Postfix İfadesine Dönüştürülmesi**

```
Step 1: Add ")" to the end of the infix expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
       IF a "(" is encountered, push it on the stack
       IF an operand (whether a digit or a character) is encountered, add it to the
       postfix expression.
       IF a ")" is encountered, then
         a. Repeatedly pop from stack and add it to the postfix expression until a
            "(" is encountered.
         b. Discard the "(". That is, remove the "(" from stack and do not
            add it to the postfix expression
       IF an operator O is encountered, then
         a. Repeatedly pop from stack and add each operator (popped from the stack) to the
            postfix expression which has the same precedence or a higher precedence than O
         b. Push the operator O to the stack
       [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT
```

Figure 7.22 Algorithm to convert an infix notation to postfix notation

Yığınların Uygulamaları

- **Bir İnfix İfadesinin Bir Postfix İfadesine Dönüştürülmesi**

Solution

Infix Character Scanned	Stack	Postfix Expression
	(	
A	(	A
–	(–	A
(	(– (	A
B	(– (	A B
/	(– (/	A B
C	(– (/	A B C
+	(– (+	A B C /
(	(– (+ (	A B C /
D	(– (+ (	A B C / D
%	(– (+ (%	A B C / D
E	(– (+ (%	A B C / D E
*	(– (+ (% *	A B C / D E
F	(– (+ (% *	A B C / D E F
)	(– (+	A B C / D E F * %
/	(– (+ /	A B C / D E F * %
G	(– (+ /	A B C / D E F * % G
)	(–	A B C / D E F * % G / +
*	(– *	A B C / D E F * % G / +
H	(– *	A B C / D E F * % G / + H
)		A B C / D E F * % G / + H * –

Example 7.3 Convert the following infix expression into postfix expression using the algorithm given in Fig. 7.22.

(a) A – (B / C + (D % E * F) / G)* H

(b) A – (B / C + (D % E * F) / G)* H)

PROGRAMMING EXAMPLE

6. Write a program to convert an infix expression into its equivalent postfix notation.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
char st[MAX];
int top=-1;
void push(char st[], char);
char pop(char st[]);
```

Yığınların Uygulamaları

```
 InfixtoPostfix(char source[], char target[]);
getPriority(char);
main()

     char infix[100], postfix[100];
     clrscr();
     printf("\n Enter any infix expression : ");
     gets(infix);
     strcpy(postfix, "");
     InfixtoPostfix(infix, postfix);
     printf("\n The corresponding postfix expression is : '
     puts(postfix);
     getch();
     return 0;
```

Yığınların Uygulamaları

```
void InfixtoPostfix(char source[], char target[])
{
        int i=0, j=0;
        char temp;
        strcpy(target, "");
        while(source[i]!='\0')
        {
                if(source[i]=='(')
                {
                        push(st, source[i]);
                        i++;
                }
                else if(source[i] == ')')
                {
                        while((top!=-1) && (st[top]!='('))
                        {
                                target[j] = pop(st);
                                j++;
                        }
                        if(top==-1)
                        {
                                printf("\n INCORRECT EXPRESSION");
                                exit(1);
                        }
                        temp = pop(st);//remove left parenthesis
                        i++;
                }
                else if(isdigit(source[i]) || isalpha(source[i]))
                {
                        target[j] = source[i];
                        j++;
                        i++;
                }
                else if (source[i] == '+' || source[i] == '-' || source[i] == '*'
source[i] == '/' || source[i] == '%')
                {
                        while( (top!=-1) && (st[top]!= '(') && (getPriority(st[to
> getPriority(source[i])))
                        {
                                target[j] = pop(st);
                                j++;
                        }
                        push(st, source[i]);
                        i++;
                }
                else
```

Yığınların Uygulamaları

```
                        exit(1);
                }
        }
        while((top!=-1) && (st[top]!='('))
        {
                target[j] = pop(st);
                j++;
        }
        target[j]='\0';
}
int getPriority(char op)
{
        if(op=='/' || op == '*' || op=='%')
                return 1;
        else if(op=='+' || op=='-')
                return 0;
}
void push(char st[], char val)
{
        if(top==MAX-1)
                printf("\n STACK OVERFLOW");
        else
        {
                top++;
                st[top]=val;
        }
}
char pop(char st[])
{
        char val=' ';
        if(top==-1)
                printf("\n STACK UNDERFLOW");
        else
        {
                val=st[top];
                top--;
        }
        return val;
}
```

Output

Yığınların Uygulamaları

- **Bir Postfix İfadesinin Değerlendirilmesi**
- **Değerlendirmenin kolaylığı, bilgisayarların bir ön ek gösterimini bir son ek gösterimine dönüştürmesi için itici güç görevi görür.**
- **Yani, infix gösteriminde yazılmış bir cebirsel ifade verildiğinde, bilgisayar önce ifadeyi eşdeğer postfix gösterimine dönüştürür ve daha sonra postfix ifadesini değerlendirir.**
- **Bu iki görev de (infix gösterimini postfix gösterimine dönüştürme ve postfix ifadesini değerlendirme) birincil araç olarak yığınlardan kapsamlı bir şekilde yararlanır.**

Yığınların Uygulamaları

- **Bir Postfix İfadesinin Değerlendirilmesi**
- **Yığınlar kullanılarak herhangi bir postfix ifadesi çok kolay bir şekilde değerlendirilebilir.**
- **Postfix ifadesinin her karakteri soldan sağa doğru taranır.**
- **Karşılaşılan karakter bir operand ise yığına itilir.**
- **Ancak bir operatörle karşılaşıldığında yığından en üstteki iki değer çıkarılır ve operatör bu değerlere uygulanır.**
- **Sonuç daha sonra yığına itilir. Bir postfix ifadesini değerlendirmek için algoritmayı gösteren Şekil 7.23'e bakalım.**

Yığınların Uygulamaları

- Bir Postfix İfadesinin Değerlendirilmesi
- Şimdi bu algoritmayı kullanan bir örneği ele alalım.
- 9 – ((3 * 4) + 8) / 4 şeklinde verilen infix ifadesini ele alalım. İfadeyi değerlendirelim.
- 9 – ((3 * 4) + 8) / 4 ön ek ifadesi, son ek gösterimi kullanılarak 9 3 4 * 8 + 4 / – şeklinde yazılabilir.
- İşlemi gösteren Tablo 7.1'e bakın.

Yığınların Uygulamaları

- **Bir Postfix İfadesinin Değerlendirilmesi**

```
ep 1: Add a ")" at the end of the
      postfix expression
ep 2: Scan every character of the
      postfix expression and repeat
      Steps 3 and 4 until ")"is encountered
ep 3: IF an operand is encountered,
      push it on the stack
      IF an operator O is encountered, then
      a. Pop the top two elements from the
         stack as A and B as A and B
      b. Evaluate B O A, where A is the
         topmost element and B
         is the element below A.
      c. Push the result of evaluation
         on the stack
      [END OF IF]
ep 4: SET RESULT equal to the topmost element
      of the stack
ep 5: EXIT
```

ıre 7.23 Algorithm to evaluate a postfix expression

Table 7.1 Evaluation of a postfix expression

Character Scanned	Stack
9	9
3	9, 3
4	9, 3, 4
*	9, 12
8	9, 12, 8
+	9, 20
4	9, 20, 4
/	9, 5
-	4

Yığınların Uygulamaları

7. Write a program to evaluate a postfix expression.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#define MAX 100
float st[MAX];
int top=-1;
void push(float st[], float val);
float pop(float st[]);
float evaluatePostfixExp(char exp[]);
int main()
{
        float val;
        char exp[100];
        clrscr();
        printf("\n Enter any postfix expression : ");
        gets(exp);
        val = evaluatePostfixExp(exp);
        printf("\n Value of the postfix expression = %.2f", val);
        getch();
        return 0;
}
float evaluatePostfixExp(char exp[])
{
        int i=0;
        float op1, op2, value;
        while(exp[i] != '\0')
        {
                if(isdigit(exp[i]))
```

Yığınların Uygulamaları

```
                        push(st, (float)(exp[i]-'0'));
                else
                {
                        op2 = pop(st);
                        op1 = pop(st);
                        switch(exp[i])
                        {
                                case '+':
                                        value = op1 + op2;
                                        break;
                                case '-':
                                        value = op1 - op2;
                                        break;
                                case '/':
                                        value = op1 / op2;
                                        break;
                                case '*':
                                        value = op1 * op2;
                                        break;
                                case '%':
                                        value = (int)op1 % (int)op2;
                                        break;
                        }
                        push(st, value);
                }
                i++;
        }
        return(pop(st));
}
```

Yığınların Uygulamaları

```
void push(float st[], float val)
{
        if(top==MAX-1)
                printf("\n STACK OVERFLOW");
        else
        {
                top++;
                st[top]=val;
        }
}
float pop(float st[])
{
        float val=-1;
        if(top==-1)
                printf("\n STACK UNDERFLOW");
        else
        {
                val=st[top];
                top--;
        }
        return val;
}
```

Output

```
Enter any postfix expression : 9 3 4 * 8 + 4 / -
Value of the postfix expression = 4.00
```

Yığınların Uygulamaları

- **Bir İnfix İfadesinin Bir Prefix İfadesine Dönüştürülmesi**
- **Bir infix ifadesini eşdeğer prefix ifadesine dönüştürmek için iki algoritma vardır.**
- **İlk algoritma Şekil 7.24'te, ikinci algoritma ise Şekil 7.25'te gösterilmektedir.**

```
Step 1: Scan each character in the infix
        expression. For this, repeat Steps
        2-8 until the end of infix expression
Step 2: Push the operator into the operator stack,
        operand into the operand stack, and
        ignore all the left parentheses until
        a right parenthesis is encountered
Step 3: Pop operand 2 from operand stack
Step 4: Pop operand 1 from operand stack
Step 5: Pop operator from operator stack
Step 6: Concatenate operator and operand 1
Step 7: Concatenate result with operand 2
Step 8: Push result into the operand stack
Step 9: END
```

Figure 7.24 Algorithm to convert an infix expression into prefix expression

```
tep 1: Reverse the infix string. Note that
       while reversing the string you must
       interchange left and right parentheses.
tep 2: Obtain the postfix expression of the
       infix expression obtained in Step 1.
tep 3: Reverse the postfix expression to get
       the prefix expression
```

ure 7.25 Algorithm to convert an infix expression into prefix expression

Yığınların Uygulamaları

- **Bir İnfix İfadesinin Bir Prefix İfadesine Dönüştürülmesi**
- Karşılık gelen önek ifadesi operand yığınında elde edilir. Örneğin, (A – B / C) * (A / K – L) şeklinde bir önek ifadesi verildiğinde
- Adım 1: İnfix dizesini ters çevirin. Dizeyi ters çevirirken sol ve sağ parantezleri değiştirmeniz gerektiğini unutmayın. (L – K / A) * (C / B – A)
- Adım 2: Adım 1 sonucunda elde edilen infix ifadesinin karşılık gelen postfix ifadesini elde edin.
  İfade şu şekildedir: (L – K / A) * (C / B – A)
  Öyleyse,
  [L – (KA /)] * [(C B /) – A]
  = [LKA/–] * [CB/A–] = L K A / – C B/A – *
- Adım 3: Önek ifadesini elde etmek için sonek ifadesini tersine çevirin. Bu nedenle,
  önek ifadesi * – A / B C – /A K L'dir

Yığınların Uygulamaları

8. Write a program to convert an infix expression to a prefix expression.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
char st[MAX];
int top=-1;
void reverse(char str[]);
void push(char st[], char);
char pop(char st[]);
void InfixtoPostfix(char source[], char target[]);
int getPriority(char);
char infix[100], postfix[100], temp[100];
int main()
{
        clrscr();
        printf("\\n Enter any infix expression : ");
        gets(infix);
        reverse(infix);
        strcpy(postfix, "");
        InfixtoPostfix(temp, postfix);
        printf("\n The corresponding postfix expression is : ");
        puts(postfix);
        strcpy(temp,"");
        reverse(postfix);
```

Yığınların Uygulamaları

```
        printf("\n The prefix expression is : \n");
        puts(temp);
        getch();
        return 0;
}
void reverse(char str[])
{
        int len, i=0, j=0;
        len=strlen(str);
        j=len-1;
        while(j>= 0)
        {
                if (str[j] == '(')
                        temp[i] = ')';
                else    if ( str[j] == ')')
                        temp[i] = '(';
                else
                        temp[i] = str[j];
                i++,    j--;
        }
        temp[i] = '\0';
}
```

Yığınların Uygulamaları

```
char temp;
strcpy(target, "");
while(source[i]!= '\0')
{
        if(source[i]=='(')
        {
                push(st, source[i]);
                i++;
        }
        else if(source[i] == ')')
        {
                while((top!=-1) && (st[top]!='('))
                {
                        target[j] = pop(st);
                        j++;
                }
                if(top==-1)
                {
                        printf("\n INCORRECT EXPRESSION");
                        exit(1);
                }
                temp = pop(st); //remove left parentheses
                i++;
        }
        else if(isdigit(source[i]) || isalpha(source[i]))
        {
                target[j] = source[i];
                j++;
                i++;
        }
        else if( source[i] == '+' || source[i] == '-' || source[i] == '*' ||
source[i] == '/' || source[i] == '%')
        {
```

Yığınların Uygulamaları

```
    > getPriority(source[i])))
                                {
                                        target[j] = pop(st);
                                        j++;
                                }
                                push(st, source[i]);
                                i++;
                        }
                        else
                        {
                                printf("\n INCORRECT ELEMENT IN EXPRESSION");
                                exit(1);
                        }
                }
                while((top!=-1) && (st[top]!='('))
                {
                        target[j] = pop(st);
                        j++;
                }
                target[j]='\0';
    }
```

Yığınların Uygulamaları

```
{
        if(op=='/' || op == '*' || op=='%')
                return 1;
        else if(op=='+' || op=='-')
                return 0;
}
void push(char st[], char val)
{
        if(top==MAX-1)
                printf("\n STACK OVERFLOW");
        else
        {
                top++;
                st[top] = val;
        }
 }
char pop(char st[])
{
        char val=' ';
        if(top==-1)
                printf("\n STACK UNDERFLOW");
        else
        {
                val=st[top];
                top--;
        }
        return val;
}
```

Output

```
Enter any infix expression : A+B-C*D
The corresponding postfix expression is : AB+CD*-
```

Yığınların Uygulamaları

- **Bir Önek İfadesinin Değerlendirilmesi**
- **Bir önek ifadesini değerlendirmek için çeşitli teknikler vardır.**
- **Bir önek ifadesinin değerlendirilmesinin en basit yolu Şekil 7.26'da verilmiştir.**
- **Örneğin, + – 9 2 7 * 8 / 4 12 önek ifadesini ele alalım. Şimdi algoritmayı bu ifadeyi değerlendirmek için uygulayalım.**

```
Step 2: Repeat until all the characters
        in the prefix expression have
        been scanned
        (a) Scan the prefix expression
            from right, one character at a
            time.
        (b) If the scanned character is an
            operand, push it on the
            operand stack.
        (c) If the scanned character is an
            operator, then
              (i) Pop two values from the
                  operand stack
             (ii) Apply the operator on
                  the popped operands
            (iii) Push the result on the
                  operand stack
Step 3: END
```

Figure 7.26 Algorithm for evaluation of a prefix

Character scanned	Operand stack
12	12
4	12, 4
/	3
8	3, 8
*	24
7	24, 7
2	24, 7, 2
–	24, 5
+	29

Yığınların Uygulamaları

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int stk[10];
int top=-1;
int pop();
void push(int);
int main()
{
    char prefix[10];
    int len, val, i, opr1, opr2, res;
    clrscr();
    printf("\n Enter the prefix expression : ");
    gets(prefix);
    len = strlen(prefix);
      for(i=len-1;i>=0;i--)
      {
    switch(get_type(prefix[i]))
            {
                case 0:
                     val = prefix[i] - '0';
                     push(val);
                     break;
                case 1:
                     opr1 = pop();
                     opr2 = pop();
                     switch(prefix[i])
                     {
                          case '+':
                               res = opr1 + opr2;
                               break;
                          case '-':
                               res = opr1 - opr2;
                               break;
                          case '*':
                               res = opr1 * opr2;
                               break;
                          case '/':
                               res = opr1 / opr2;
                               break;
                     }
                     push(res);
            }
      }
      printf("\n RESULT = %d", stk[0]);
      getche();
      return 0;
}
void push(int val)
{
      stk[++top] = val;
```

Yığınların Uygulamaları

```
}
int pop()
{
        return(stk[top--]);
}
int get_type(char c)
{
        if(c == '+' || c == '-' || c == '*' || c == '/')
                return 1;
        else return 0;
}
```

Output

```
Enter the prefix expression : +-927
RESULT = 14
```