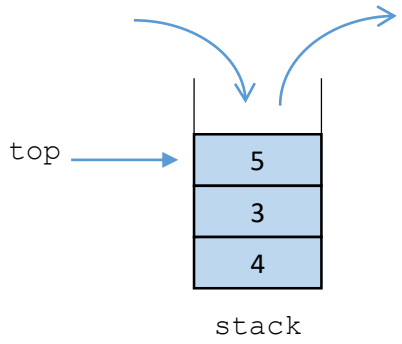
7 Yığınlar

7.1 Giriş

Veri yapılarının önemli konularından birisidir. Nesne ekleme ve çıkarmalarının en üstten (*top*) yapıldığı veri yapısına **stack** (**yığın**) adı verilir. Soyut bir veri tipidir. Bazı kaynaklarda yığıt, nadiren çıkın olarak da isimlendirilir. Bir nesne ekleneceği zaman yığının en üstüne konulur. Bir nesne çıkarılacağı zaman da yığının en üstündeki nesne çıkarılır. Bu çıkarılan nesne, yığının elemanlarının içindeki en son eklenen nesnedir. Yani bir yığındaki elemanlardan sadece en son eklenene erişim yapılır. Bu nedenle yığınlara **LIFO** (*Last In First Out: Son giren ilk çıkar*) listesi de denilir. Mantıksal yapısı bir konteyner gibi düşünülebilir.



Şekil. Stack (yığın) mantıksal yapısı

Stack yapılarına gerçek hayattan benzetmeler yapacak olursak;

- Şarjör,
- Yemekhanedeki tepsiler,
- El feneri pilleri,

gibi son girenin ilk çıktığı örnekler verilebilir. Bilgisayar'da nerelerde kullanıldığını ilerleyen sayfalarda göreceğiz.

Yığınlar statik veriler olabileceği gibi, dinamik veriler de olabilirler. Bir yığın statik olarak tanımlanırken dizi şeklinde, dinamik olarak tanımlanırken ise bağlı liste biçiminde tanımlanabilir. Şimdi stack'lerin C programlama dilini kullanarak bilgisayarlardaki implementasyonunun nasıl gerçekleştirildiğini görelim.

7.2 Stack'lerin Dizi (Array) İmplementasyonu

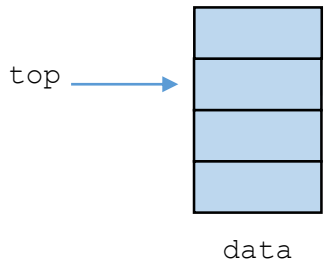
Stack'lerin bir boyutu (kapasitesi) olacağından önce bu boyutu tanımlamalıyız.

```
#define STACK_SIZE 4
```

C derleyicilerinin `#define` önişlemci komutunu biliyor olmalısınız. C programlama dilinde # ile başlayan bütün satırlar, önişlemci programa verilen komutlardır (directives). Üstteki satır bir anlamda derleyiciye `STACK_SIZE` gördüğün yere 4 yaz demektir. Program içerisinde dizi boyutunun belirtilmesi gereken alana bir sabit ifadesi olan 4 rakamını değil `STACK_SIZE` yazacağız. Böyle bir tanımlama ile uzun kodlardan oluşabilecek program içerisinde, dizi boyutunun değiştirilmesi ihtiyacı olduğunda 4 rakamını güncellemek yeterli olacaktır ve programın uzun kod satırları arasında dizi boyutunu tek tek değiştirme zahmetinden kurtaracaktır. Şimdi bir `stack`'in genel yapısını tanımlayalım;

```
#define STACK_SIZE 4
typedef struct {
     int data[STACK_SIZE]; // ihtiyaca göre veri türü değişebilir
     int top;
     /* sürekli eleman ekleme ve silme işlemi yapılacağı için en üstteki
     * elemanın indisini tutan top adında bir değişken tanımladık */
} stack;
```

Bu tanımlama ile meydana gelen durum aşağıdaki şekilde görülmektedir:



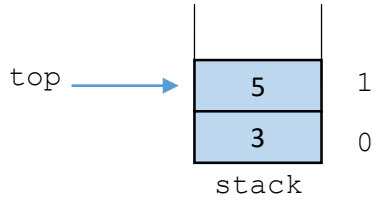
Dikkat ederseniz top değişkeni şu an için herhangi bir `data` 'nın indisini tutmuyor. C derleyicisi tarafından bir çöp değer atanmış durumdadır. `stack`'e ekleme işlemi yapan `push` isimli fonksiyonumuzu yazalım.

7.2.1 Stack'lere Eleman Ekleme İşlemi (push)

Bir `stack`'e eleman ekleme fonksiyonu `push` olarak bilinir ve bu fonksiyon, `stack`'i ve eklenecek elemanın değerini parametre olarak alır. Geri dönüş değeri olmayacağı için türü `void` olmalıdır. Ayrıca eleman ekleyebilmek için yığının dolu olmaması gerekir.

```
void push(stack *stk, int c) {
     if(stk -> top == STACK_SIZE - 1) // top son indisi tutuyorsa doludur
          printf("\nStack dolu\n\n");
     else {
          stk -> top ++;
          stk -> data[stk -> top] = c;
     }
}
```

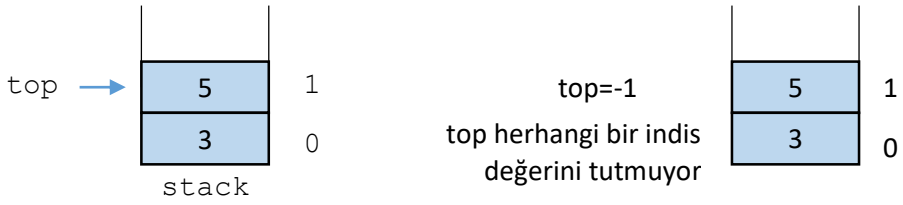
Eleman eklendikten sonra top değişkeni en son eklenen elemanın indis değerini tutmaktadır (aşağıdaki şekilde 1 değerini tutuyor):



7.2.2 Bir Stack'in Tüm Elemanlarını Silme İşlemi (reset)

`stack`'i resetlemek için `top` değişkeninin değerini -1 yapmamız yeterlidir. Bu atamadan sonra top herhangi bir indis değeri tutmuyor demektir. Aslında veriler silinmedi değil mi? Bu veriler hala hafızada bir yerlerde tutuluyor. Lakin `stack`'e hiç ekleme yapılmamış bile olsa `stack`'lerin dizi ile implementasyonu bellekte yine de yer işgal edecekti. Yeni eklenecek veriler ise öncekilerinin üzerine yazılacaktır. Şimdi reset ismindeki fonksiyonumuzu yazalım.

```
void reset(stack *stk) {
     stk -> top = -1;
}
```



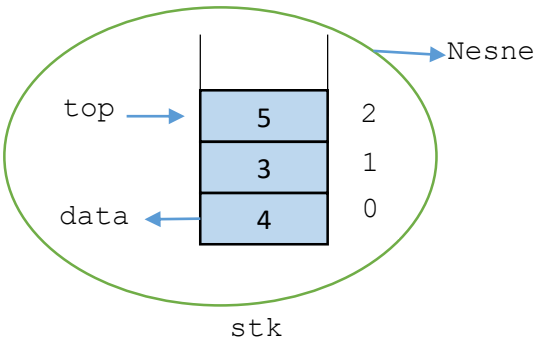
7.2.3 Stack'lerden Eleman Çıkarma İşlemi (pop)

`pop` fonksiyonu olarak bilinir. Son giren elemanı çıkarmak demektir. Çıkarılan elemanın indis değeriyle geri döneceği için fonksiyonun tipi `int` olmalıdır.

```
int pop(stack *stk) {
     if(stk -> top == -1) // stack boş mu diye kontrol ediliyor
          printf("Stack bos");
     else {
          int x = stk -> top--; // -- operatörünün işlem sırasına dikkat
          return x; // çıkarılan elemanın indis değeriyle geri dönüyor
     }
}
```

else kısmı şu tek satır kodla da yazılabilirdi. Fakat bu kez geri dönüş değerinin `stack`'den çıkarılan verinin kendisi olduğuna dikkat ediniz.

```
else
     return(stk -> data[stk -> top--]);
```



Şekil. "stk" nesne modeli

Aşağıda `push()` ve `pop()` fonksiyonlarının `main()` içerisinde ki kullanımını görüyorsunuz. `stack` yapısının ve fonksiyon tanımlamalarının `main()` bloğu içinde yapıldığını varsayıyoruz.

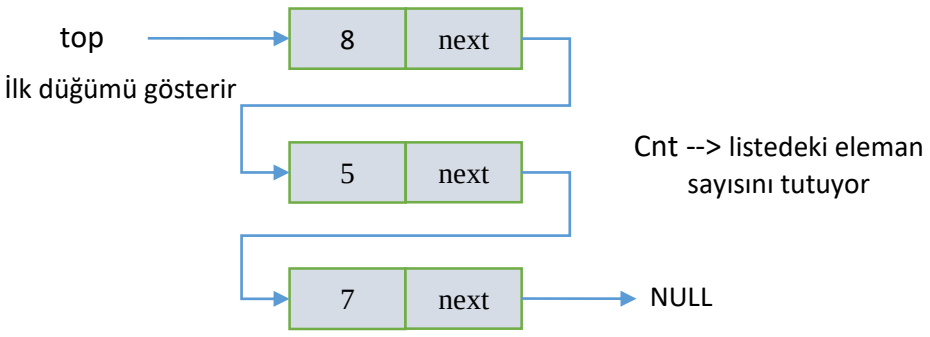
```
int main() {
```

```
    int x;
    stack n;
    reset(&n);
    push(&n, 4);
    push(&n, 2);
    push(&n, 5);
    push(&n, 7);
    push(&n, 11);
    x = pop(&n);
    printf("%d\n", x);
    x = pop(&n);
    printf("%d\n", x);
    x = pop(&n);
    printf("%d\n", x);
    x = pop(&n);
    printf("%d\n", x);
    x = pop(&n);
    printf("%d\n", x);
    getch();
    return 0;
}
```

Alıştırma-2: Yukarıdaki kod satırlarında her `pop` işleminden sonra `return` edilen x değerinin son `pop` işleminden sonraki değeri ne olur?

7.3 Stack'lerin Bağlı Liste (Linked List) İmplementasyonu

`Stack`'in bağlı liste uygulamasında, elemanlar bir yapının içinde yapı işaretçileriyle beraber bulunur. Mantıksal yapısını daha önce gördüğümüz bağlı listelerin mantıksal yapısının üst üste sıralanmış şekli gibi düşünebilirsiniz.



Şekil. Bağlı liste kullanılarak gerçekleştirilen `stack` yapısı

Dizi uygulamasının verdiği maksimum genişlik sınırı kontrolü bu uygulamada yoktur. Tüm fonksiyonlar sabit zamanda gerçekleşir. Çünkü fonksiyonlarda `stack`'in genişliği referans alınır. Hemen hemen bütün fonksiyonların görevleri, dizi uygulamasındaki görevleriyle aynıdır. Fakat iki uygulama arasında algoritma farkı olduğu için fonksiyon tanımlamaları da farklı olur. Tek bağlı doğrusal liste kullanarak veri yapısını yazalım.

```
typedef struct {
     struct node *top;
     int cnt;
}stack;
```

Görüldüğü üzere `stack`, liste uzunluğunu tutacak `int` türden bir `cnt` değişkeni ve `struct node` türden göstericisi olan bir yapıdır. `stack`'in tanımı `typedef` bildirimiyle yapılmıştır. Bu yapıdan nesneler oluşturulacağı zaman `struct stack` tür belirtimi yerine yalnızca `stack` yazılması geçerli olacaktır.

7.3.1 Stack'in Boş Olup Olmadığının Kontrolü (isEmpty)

C'de `enumaration` (*numaralandırma*) konusunu hatırlıyor olmalısınız. Biz de ilk olarak enumaration yoluyla `boolean` türünü tanımlıyoruz. C++'ta ise `boolean` türü öntanımlıdır. Ayrıca tanımlamaya gerek yoktur.

```
typedef enum {false, true}boolean;
```

`typedef` bildiriminden sonra `boolean`, yalnızca `false` ve `true` değerlerini alabilen bir türün ismidir. `false` 0 değerini, `true` ise 1 değerini alacaktır. Artık `enum` anahtar sözcüğü kullanılmadan direk `boolean` türden tanımlamalar yapılabilir. Aşağıda `isEmpty` isimli fonksiyonu görüyorsunuz.

```
boolean isEmpty(stack *stk) {
     return(stk -> cnt == 0);
     // parantez içerisindeki ifadeye dikkat
}
```

Fonksiyonun geri dönüş değeri türü `boolean` türüdür. Geri döndürülen değer sadece yanlış veya doğru olabilir. `return` parantezi içerisindeki ifade ile `stk -> cnt` değeri 0'a eşitse `true`, eşit değil ise `false` değeri geri döndürülecektir.

7.3.2 Stack'in Dolu Olup Olmadığının Kontrolü (isFull)

Yine geri dönüş değeri türü `boolean` türden olan `isFull` isimli fonksiyonu tanımlıyoruz. `stack` yapısının `cnt` değişkeninin değeri, `stack`'in boyutunu sınırlandıran `STACK_SIZE` değişkenine eşitse dolu demektir ve fonksiyondan geriye `true` döndürülür. Eşit değilse geri dönüş değeri `false` olacaktır.

```
boolean isFull(stack *stk) {
     return(stk -> cnt == STACK_SIZE);
}
```

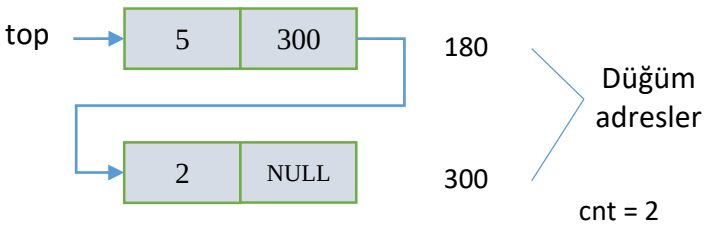
7.3.3 Stack'lere Yeni Bir Düğüm Eklemek (push)

Tek bağlı liste uygulamasında da `stack`'lerde ekleme işlemini yapan fonksiyon `push` fonksiyonu olarak bilinir. Dizi uygulamasındaki push fonksiyonuyla bazı farkları vardır. `stack`'lerin dizi implementasyonunda dizi boyutu önceden belirlenir. Dizi boyutu aşıldığında ise taşma hatası meydana gelir. Oysa tek bağlı liste uygulaması ile oluşturulan `stack`'lerde taşma hatası meydana gelmez. Aslında bellekte boş yer olduğu sürece ekleme yapılabilir. Fakat `stack`'in soyut veri yapısından dolayı sanki bir büyüklüğü varmış gibi ekleme yapılır. Yeni bir düğüm ekleneceğinden dolayı fonksiyon içerisinde struct node türden bir yapı oluşturmak gerekecektir.

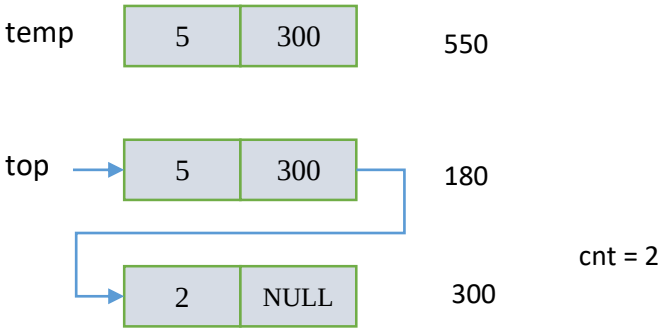
```
// stack'e ekleme yapan fonksiyon
void push(stack *stk, int c) {
     if(!isfull(stk)) {
          struct node *temp = (struct node *)malloc(sizeof(struct node));
          /* struct node *temp = new node(); // C++'ta bu şekilde */
          temp -> data = c;
          temp -> next = stk -> top;
          stk -> top = temp;
          stk -> cnt++;
     }
     else
          printf("Stack dolu\n");
}
```

Görüldüğü gibi daha önce anlatılan tek bağlı listelerde eleman ekleme işleminden tek farkı `stk -> cnt++` ifadesidir. Yapılan işlemler aşağıdaki şekilde adım adım belirtilmiştir.

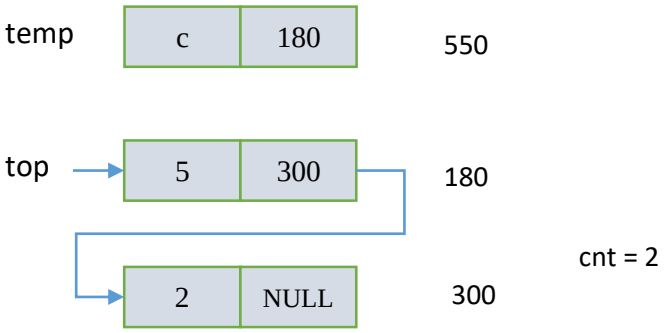
Adım 1: Başlangıç durumu. `top` şu anda 180 adresini gösteriyor:



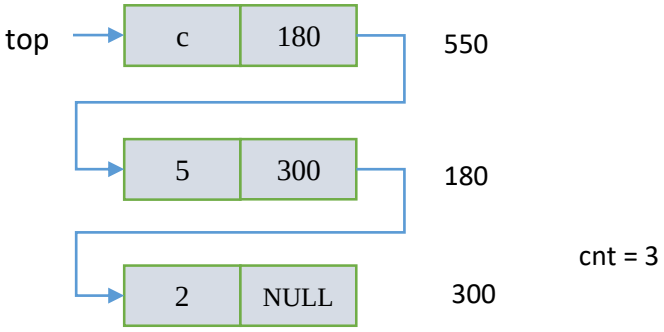
Adım 2: `temp` düğümü için bellekte yer ayrılıyor:



Adım 3: Yeni düğüme veri atanıyor:



Adım 4: `top` listenin başını gösteriyor:



7.3.4 Stack'lerden Bir Düğümü Silmek (pop)

Yine dizi uygulamasında olduğu gibi `pop` fonksiyonu olarak bilinir. Silinecek düğümü belleğe geri kazandırmak için `struct node` türden `temp` adında bir işaretçi tanımlanarak silinecek düğümün adresi bu işaretçiye atanır. `top` göstericisine ise bir sonraki düğümün adresi atanır ve `temp` düğümü `free()` fonksiyonuyla silinerek belleğe geri kazandırılır. Eleman sayısını tutan `cnt` değişkeninin değeri de 1 azaltılır. Fonksiyon silinen düğümün verisiyle geri döneceği için türü `int` olmalıdır.

```
int pop(stack *stk) {
     if(!isEmpty(stk)) { // stack boş değilse
          struct node *temp = stk -> top;
          stk -> top = stk -> top -> next;
          int x = temp -> data;
          free(temp);
          stk -> cnt--;
          return x;
     }
}
```

Fonksiyon, eğer `stack` boş değilse düğüm silme işlemini gerçekleştiriyor ve silinen düğümün verisini geri döndürüyor. `stack`'in boş olması halinde fonksiyonun `else` bölümüne ekrana uyarı mesajı veren bir kod satırı da eklenebilirdi.

7.3.5 Stack'in En Üstteki Verisini Bulmak (top)

`stack` boş değilse en üstteki düğümün verisini geri döndüren bir fonksiyondur. Geri dönüş değeri türü, verinin türüyle aynı olmalıdır.

```
int top(stack *stk) {
     if(!isEmpty(stk))
          return(stk -> top -> data);
          // En üstteki elemanın verisiyle geri döner
}
```

7.3.6 Bir Stack'e Başlangıç Değerlerini Vermek (initialize)

Önemli bir fonksiyondur. Değişkenlere bir değer ataması yapılmadığı zaman çoğu C derleyicisi bu değişkenlere, çöp değer diye de tabir edilen rastgele değerler atayacaktır. Yığın işlemlerinden önce mutlaka bu fonksiyonla başlangıç değerleri verilmelidir.

```
void initialize(stack *stk) {
     stk -> top = NULL;
     stk -> cnt = 0;
}
```

7.3.7 Stack'ler Bilgisayar Dünyasında Nerelerde Kullanılır?

Yığın mantığı bilgisayar donanım ve yazılım uygulamalarında yaygın olarak kullanılmaktadır. Bunlardan bazıları;

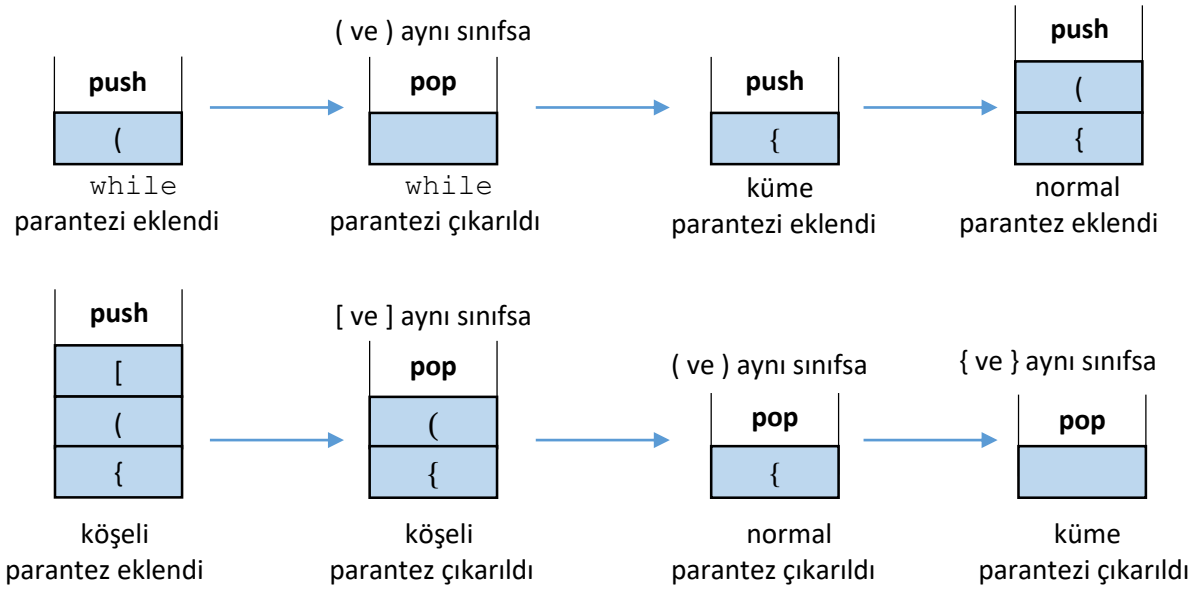
- Rekürsif olarak tanımlanan bir fonksiyon çalışırken hafıza kullanımı bu yöntem ile ele alınır,
- (, {, [,], },) ayıraçlarının C/C++ derleyicisinin kontrollerinde,
- postfix → infix dönüştürmelerinde,
- Yazılım uygulamalarındaki Parsing ve Undo işlemlerinde,
- Web browser'lardaki Back butonu (önceki sayfaya) uygulamasında,
- Ayrıca, mikroişlemcinin içyapısında **stack** adı verilen özel hafıza alanı ile mikroişlemci arasında, bazı program komutları ile (`push` ve `pop` gibi), bir takım işlemlerde (alt program çağırma ve kesmeler gibi), veri transferi gerçekleşir. Mikroişlemcinin içinde, hafızadaki bu özel alanı gösteren bir yığın işaretçisi (*Stack Pointer* –SP) bulunur. `Stack Pointer` o anda bellekte çalışılan bölgenin adresini tutar. `push` fonksiyonu, `stack`'e veri göndermede (yazmada), `pop` fonksiyonu ise bu bölgeden veri almada (okumada) kullanılır.

Özyinelemeli Fonksiyonlar konusunda gösterilen rekürsif faktöryel uygulamasında, fonksiyonun kendisini her çağırmasında `stack`'e ekleme yapılır. `stack`'in en üstünde `fact(0)` bulunmaktadır. Fonksiyon çağırıldığı yere her geri döndüğünde ise `stack`'ten çıkarma işlemi yapılmaktadır.

Örnek 16: Anlamsız bir kod parçası olsun;

```
while(x == 7) {
     printf("%d", a[2]);
}
```

Üstteki kodlarda bulunan parantezlerin kullanımlarının doğruluğu `stack`'lerle kontrol edilir. Bunun nasıl yapıldığı alttaki şekilde adımlar halinde görülüyor. Bu uygulamada yukarıdaki kod parçasında bulunan bir parantez açıldığında `push()` fonksiyonuyla `stack`'e ekleme, kapatıldığında ise `pop()` fonksiyonuyla çıkarma yapacağız.



`pop()` işleminde çıkarılacak parantez, kapanan parantezle uyuşmuyorsa hata durumu ortaya çıkacaktır. Son işlemden sonra `stack`'te eleman kalmadığından, parantezlerin doğru bir şekilde kullanıldığı görülmektedir.

7.4 Infix, Prefix ve Postfix Notasyonları

Bilgisayarlarda infix yazım türünün çözümlenmesi zordur. Acaba `x=a/b-c+d*e-a*c` şeklindeki bir ifadeyi çözümlerken, `((4/2)-2)+(3*3)-(4*2)` gibi bir ifadenin değerini hesaplarken ya da `a/(b-c)+d*(e-a)*c` gibi parantezli bir ifadeyi işlerken derleyiciler sorunun üstesinden nasıl geliyor? `32*(55-32-(11-4)+(533-(533-(533+(533-(533+212))*21-2))` gibi birçok operatör ve operand içeren bir işlemde nasıl operatör önceliklerine göre işlem sıralarını doğru belirleyip sonuç üretebiliyorlar?

Bir ifadede farklı önceliklere sahip operatörler yazılma sırasıyla işlenirse ifade yanlış sonuçlandırılabilir. Örneğin `3+4*2` ifadesi `7*2=14` ile sonuçlandırılabileceği gibi `3+8=11` ile de sonuçlandırılabilir. Bilgisayarlarda infix yazım türünün çözümlenmesi zordur. Bu yüzden ifadelerin operatör önceliklerine göre ayrıştırılması, ayrılan parçaların sıralanması ve bu sıralamaya uyularak işlem yapılması gerekir. Bu işlemler için prefix ya da postfix notasyonu kullanılır. Çoğu derleyici, kaynak kod içerisinde infix notasyonunu kullanmaktadır ve daha sonra `stack` veri yapısını kullanarak prefix veya postfix notasyonuna çevirir.

Bu bölümde bilgisayar alanındaki önemli konulardan biri olan infix, prefix ve postfix kavramları üzerinde duracağız ve bu kavramlarda `stack` kullanımını göreceğiz.

- Operatörler : +, -, /, *, ^

- Operandlar : A, B, C… gibi isimler ya da sayılar.

Infix notasyonu:

Alışa geldiğimiz ifadeler infix şeklindedir. Operatörlerin işlenecek operandlar arasına yerleştirildiği gösterim biçimidir. Bu gösterimde operatör önceliklerinin değiştirilebilmesi için parantez kullanılması şarttır. Örneğin infix notasyonundaki `2+4*6` ifadesi `2+24=26` ile sonuçlanır. Aynı ifadede `+` operatörüne öncelik verilmesi istenirse parantezler kullanılır; `(2+4)*6`. Böylece ifade `36` ile sonuçlandırılır.

Prefix notasyonu:

Prefix notasyonunda (PN, polish notation) operatörler, operandlarından önce yazılır. Örneğin `2+4*6` ifadesi infix notasyonundadır ve prefix notasyonunda `+2*46` şeklinde gösterilir. Benzer biçimde `(2+4)*6` ifadesi `*+246` şeklinde gösterilir. Görüldüğü gibi prefix notasyonunda işlem önceliklerinin sağlanması için parantezlere ihtiyaç duyulmamaktadır.

Postfix notasyonu:

Postfix notasyonunda (RPN, reverse polish notation) ise önce operandlar ve ardından operatör yerleştirilir. Aynı örnek üzerinden devam edersek; infix notasyonundaki `2+4*6` ifadesi postfix notasyonunda `2 4 6 * +` şeklinde, benzer biçimde `(2+4)*6` ifadesi de `2 4 + 6 *` şeklinde gösterilir. Yine prefix'te olduğu gibi bu gösterimde de parantezlere ihtiyaç duyulmamaktadır. Bazı bilgisayarlar matematiksel ifadeleri postfix olarak daha iyi saklayabilmektedir.

Tüm aritmetik ifadeler bu gösterimlerden birini kullanarak yazılabilir. Ancak, bir yazmaç (register) yığını ile birleştirilmiş postfix gösterimi, aritmetik ifadelerin hesaplanmasında en verimli yoldur. Aslında bazı elektronik hesap makineleri (*Hewlett-Packard* gibi) kullanıcının ifadeleri postfix gösteriminde girmesini ister. Bu hesap makinelerinde biraz alıştırma yapıldığında, iç içe birçok parantez

içeren uzun ifadeleri, terimlerin nasıl gruplandığını bile düşünmeden, hızlı bir şekilde hesaplamak mümkündür.

İşlem önceliği;

1- Parantez içi
2- Üs alma
3- Çarpma/Bölme
4- Toplama Çıkarma

Aynı önceliğe sahip işlemlerde sıra soldan sağa (→) doğrudur. Yalnız üs almada sağdan sola doğru işlem yapılır.

Tablo. Matematiksel ifadelerde infix, prefix ve postfix notasyonunun gösterimi.

Infix	Prefix	Postfix
A+B-C	-+ABC	AB+C-
(A+B)*(C-D)	*+AB-CD	AB+CD-*
A/B^C+D*E–A*C	–+/A^BC*DE*AC	ABC^/DE*+AC*–
(B^2–4*A*C)^(1/2)	(^–^B2**4AC/12)	(B2^4A*C*–12/^)
A^B*C-D+E/F/(G+H)	+-*^ABCD//EF+GH	AB^C*D-EF/GH+/+
((A+B)*C-(D-E))^(F+G)	^-*+ABC-DE+FG	AB+C*DE—FG+^
A-B/(C*D^E)	-A/B*C^DE	ABCDE^*/-

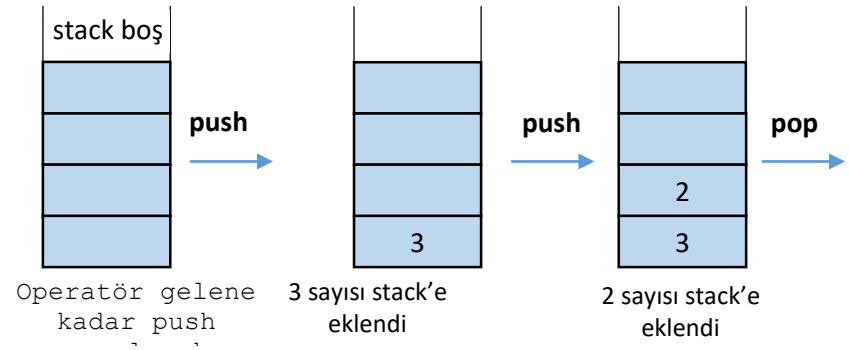
Dikkat edilecek olunursa, postfix ile prefix ifadeler birbirinin ayna görüntüsü değillerdir. Şimdi bir örnekle notasyon işlemlerinin stack kullanılarak nasıl gerçekleştirildiklerini görelim.

Örnek 17: 3 2 * 5 6 * + postfix ifadesini `stack` kullanarak hesaplayınız.

Çözüm algoritması şöyle olmalıdır;

1- Bir operandla karşılaşıldığında o operand stack'e eklenir,
2- Bir operatör ile karşılaşıldığında ise o operatörün gerek duyduğu sayıda operand stack'ten çıkarılır,
3- Daha sonra pop edilen iki operanda operatör uygulanır,
4- Bulunan sonuç tekrar stack'e eklenir.

Bu adımları kullanarak Postfix ifadesinin stack kullanılarak hesaplanması:

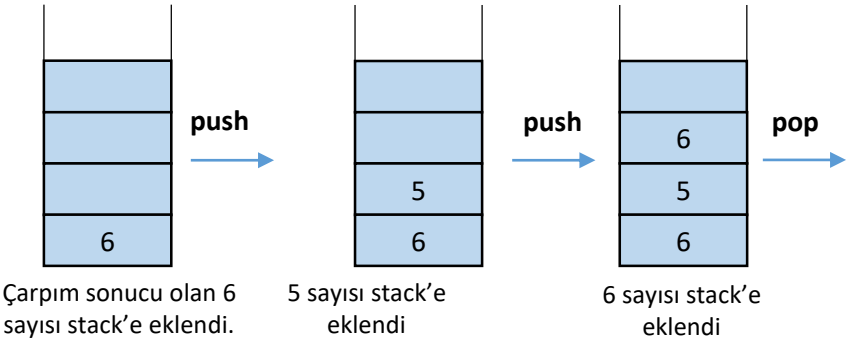


Operatör gelene kadar push

3 sayısı stack'e eklendi

2 sayısı stack'e eklendi

Şimdi bir operatörle karşılaşıldığı için stack'teki elemanlara 2 defa pop işlemi uygulandıktan sonra operatörle işleme sokulup sonuç push yapılacak.

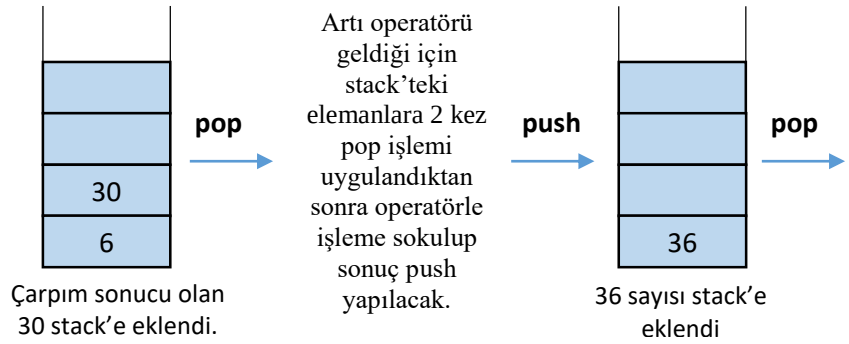


Çarpım sonucu olan 6 sayısı stack'e eklendi.

5 sayısı stack'e eklendi

6 sayısı stack'e eklendi

Çarpı operatörüyle karşılaşıldığı için stack'teki elemanlara 2 defa pop işlemi uygulandıktan sonra operatörle işleme sokulup sonuç push yapılacak.



Çarpım sonucu olan 30 stack'e eklendi.

Artı operatörü geldiği için stack'teki elemanlara 2 kez pop işlemi uygulandıktan sonra operatörle işleme sokulup sonuç push yapılacak.

36 sayısı stack'e eklendi

Artık hesaplanacak bir işlem kalmadığı için sonuç ekrana yazdırılabilir.