NOT-1: Bu bölüm “Paul Deitel, Harvey Deitel - 9nth_edition C How to Program-Pearson (2022)” kitabı 9ncu baskıdan anlatılmıştır. Bu bölüm 8nci baskıya göre daha açıklamalı verildiğinden 9ncu baskı tercih edilmiştir.

NOT-2: Bu bölündeki programları kaydederken C++ compiler kullanmayın, hata verebilir. Bunun yerine yazdığınız programı C compiler olarak kaydedin ve çalıştırın.

Dinamik Bellek Yönetimi

Şimdiye kadar, tek boyutlu diziler, iki boyutlu diziler ve struct yapıları gibi sabit boyutlu veri yapılarını inceledik. Bu bölümde ise, programın çalışması esnasında büyüyebilen ve küçülebilen dinamik veri yapılarını işleyeceğiz.

• Bağlantılı listeler, "arka arkaya dizilmiş" veri öğeleri bir araya gelmesidir.; eklemeler ve silmeler, bağlantılı bir listede herhangi bir yerde yapılır.

• Yığınlar (stack), derleyiciler ve işletim sistemlerinde önemlidir—eklemeler ve silmeler yığının yalnızca bir ucunda, yani tepesinde yapılır.

• Kuyruklar bekleme sıralarını temsil eder; eklemeler kuyruğun yalnızca arkasından (sonunda) silmeler ise kuyruğun yalnızca önünden (başında) yapılır.

• İkili ağaçlar (binary tree), tekrarlı veri öğelerini kaldırarak ve ifadeleri makine dilinde derleyerek, verilerin yüksek hızda aranmasını ve sıralanmasını kolaylaştırır.

Kendine Referanslı Yapılar

Kendine referanslı bir yapı, aynı struct yapısı tipindeki bir yapıya işaret eden bir işaretçi üye içerir. Örneğin;

```
struct node {
        int data;
        struct node *nextPtr;
};
```

Node yapısının iki üyesi vardır: tamsayı “data” üyesi ve işaretçi “nextPtr” üyesi. nextPtr üyesi, struct node türünde bir başka yapıya işaret eder - aynı türde bir yapı, dolayısıyla kendine referanslı yapı terimidir. nextPtr üyesine bağlantı denir - yani, struct node türündeki bir yapıyı aynı türdeki başka bir yapıya "bağlamak" için kullanılabilir.

Kendini referans alan yapılar; sıralar (kuyruklar), yığınlar ve ağaçlar gibi yararlı veri yapıları oluşturmak için birbirine bağlanabilir. Aşağıdaki Şekil, bir liste oluşturmak için birbirine bağlanan iki kendine referanslı yapı nesnesini göstermektedir. Bağlantının başka bir yapıya işaret etmediğini belirtmek için ikinci kendine referanslı yapının bağlantı üyesine bir NULL işaretçisini temsil eden bir eğik çizgi yerleştirilmiştir. Eğer başka bir bağlantı yapılmayacaksa son bağlantıya NULL eklenir. Bir NULL işaretçisi, tıpkı boş karakterin bir dizenin sonunu göstermesi gibi, normalde bir veri yapısının sonunu gösterir. [Not: Eğik çizgi yalnızca açıklama amaçlıdır; C'deki ters eğik çizgi karakterine karşılık gelmez.]

Dinamik Bellek Ayırma

Program çalışırken büyüyebilen ve küçülebilen dinamik veri yapılarının oluşturulması ve sürdürülmesi, dinamik bellek tahsisi gerektirir; bu, bir programın yürütme sırasında yeni düğümleri tutmak ve artık ihtiyaç duyulmayan alanı serbest bırakmak için daha fazla bellek alanı elde edebilme yeteneğidir.

malloc ve free fonksiyonları ile sizeof operatörü, dinamik bellek tahsisi için gereklidir.

malloc fonksiyonu; ayrılacak bayt sayısını bağımsız değişken olarak alır ve ayrılan belleğe bir işaretçi döndürür. malloc fonksiyonu normalde sizeof operatörüyle birlikte kullanılır. Örneğin;

```
newPtr = malloc(sizeof(struct node));
```

ifadesi, bir struct node nesnesinin bayt cinsinden boyutunu belirlemek için sizeof(struct node) öğesini kullanır, bu bayt sayısı kadar bellekte yeni bir alan tahsis eder ve newPtr ayrılan belleğe bir işaretçi verir. Kullanılabilir bellek yoksa malloc, NULL değerini döndürür.

free fonksiyonu; İşlevsiz belleği serbest bırakır; yani bellek, gelecekte yeniden tahsis edilebilmesi için sisteme geri gönderilir. Önceki malloc çağrısı tarafından dinamik olarak ayrılan belleği boşaltmak için aşağıdaki deyim kullanılır;

```
free(newPtr);
```

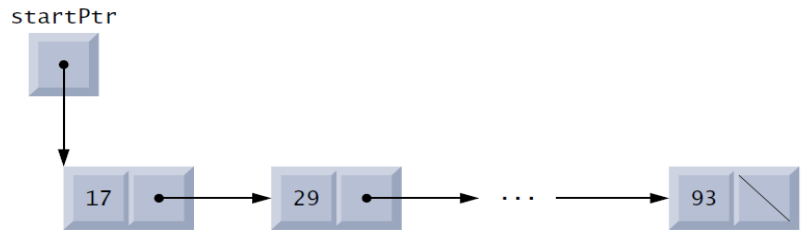
C dilinde ayrıca dinamik diziler oluşturmak ve değiştirmek için calloc ve realloc fonksiyonları da kullanılabilir.

Bağlantılı Listeler

Bağlantılı bir liste, düğüm adı verilen ve işaretçi bağlantılarıyla birbirine bağlanan, kendine referanslı yapıların doğrusal bir koleksiyonudur - bu nedenle "bağlı" liste terimini kullanır. Bağlantılı bir listeye, listenin ilk düğümüne atanan bir işaretçi aracılığıyla erişilir. Sonraki düğümlere, her düğümde bulunan bağlantı işaretçisi aracılığıyla erişilir. Geleneksel olarak, bir listenin son düğümündeki bağlantı işaretçisi, listenin sonunu işaretlemek için NULL olarak ayarlanır. Veriler dinamik olarak bağlantılı bir listede saklanır;

Bir düğüm, diğer struct yapıları da dahil olmak üzere herhangi bir türde veri içerebilir. Bağlantılı listeler, yığınlar ve kuyruklar doğrusal veri yapılarıdır. Ağaçlar (tree) ise doğrusal olmayan veri yapılarıdır.

Veri listeleri dizilerde de saklanabilir, ancak bağlantılı listeler çeşitli avantajlar sağlar. Bağlantılı bir liste, veri yapısında temsil edilecek veri öğelerinin sayısı tahmin edilemez olduğunda uygundur. Bağlantılı listeler dinamiktir, bu nedenle bir listenin uzunluğu yürütme sırasında gerektiği gibi artabilir veya azalabilir. Ancak derleme zamanında oluşturulan bir dizinin boyutu değiştirilemez. Bağlantılı liste düğümleri normalde bellekte bitişik olarak depolanmaz. Bununla birlikte, mantıksal olarak, bağlantılı bir listenin düğümleri bitişik gibi görünür. Aşağıdaki Şekil, birkaç düğüm içeren bağlantılı bir listeyi göstermektedir.



Şekil 12.1 olarak verilen aşağıdaki program bir karakter listesini yeniden düzenlemektedir. Bu karakter listesine alfabetik sırayla "function insert" ile bir karakter eklenebilir, yada "function delete" ile bir karakter silinebilir. Bu programı daha iyi anlamak için parça parça inceleyelim. Program çalıştığında oluşabilecek çıktılar aşağıdaki tabloda verilmiştir.

```
Enter your choice:
   1 to insert an element into the list.
   2 to delete an element from the list.
   3 to end.
? 1
Enter a character: B
The list is:
B --> NULL

? 1
Enter a character: A

The list is:
A --> B --> NULL

? 1
Enter a character: C
The list is:
A --> B --> C --> NULL

? 2
Enter character to be deleted: D
D not found.

? 2
Enter character to be deleted: B
B deleted.
The list is:
A --> C --> NULL
```

```
? 2
Enter character to be deleted: C
C deleted.
The list is:
A --> NULL

? 2
Enter character to be deleted: A
A deleted.
List is empty.

? 4
Invalid choice.

Enter your choice:
   1 to insert an element into the list.
   2 to delete an element from the list.
   3 to end.
? 3
End of run.
```

İlk bölümde oluşturulan fonksiyon prototipleri ve fonksiyonlar yer almaktadır. Programda 7-10 satırları bağlantılı liste oluşturmak için kendine referanslı bir yapı, (struct listNode) tanımlamaktadır. 12 ve 13 ncü satırlar daha iyi okunabilir bit kod yazabilmek için "typedefs"leri tanımlar. Buradaki "ListNode" ismi "struct listNode" nesnesini tanımlar. "ListNodePtr" ise "struct listNode" nesnesine bir işaretçi tanımlar.

Ana program (main function), karakter ekleme (34-39 satırlar), listeden bir karakter silme (40-58 satırlar) veya programı sonlandırmak için düzenlenmiştir. Program çalıştığında karakter ekleme, silme veya programı sonlandırma tercihleri 73-77 satırlarda yapılmaktadır. Başlangıçta "startPtr" (satır 23) listenin boş olduğunu göstermek için NULL yapılmıştır.

```c
// fig12_01.c
// Inserting and deleting nodes in a list
#include <stdio.h>
#include <stdlib.h>

// self-referential structure
struct listNode {
   char data; // each listNode contains a character
   struct listNode *nextPtr; // pointer to next node
};

typedef struct listNode ListNode; // synonym for struct listNode
typedef ListNode *ListNodePtr; // synonym for ListNode*

// prototypes
void insert(ListNodePtr *sPtr, char value);
char delete(ListNodePtr *sPtr, char value);
int isEmpty(ListNodePtr sPtr);
void printList(ListNodePtr currentPtr);
void instructions(void);

int main(void) {
   ListNodePtr startPtr = NULL; // initially there are no nodes
   char item = '\0'; // char entered by user

   instructions(); // display the menu
   printf("%s", "? ");
   int choice = 0; // user's choice
   scanf("%d", &choice);

   // loop while user does not choose 3
   while (choice != 3) {
      switch (choice) {
         case 1: // insert an element
            printf("%s", "Enter a character: ");
            scanf("\n%c", &item);
            insert(&startPtr, item); // insert item in list
            printList(startPtr);
            break;
         case 2: // delete an element
            if (!isEmpty(startPtr)) { // if list is not empty
               printf("%s", "Enter character to be deleted: ");
               scanf("\n%c", &item);

               // if character is found, remove it
               if (delete(&startPtr, item)) { // remove item
                  printf("%c deleted.\n", item);
                  printList(startPtr);
               }
               else {
                  printf("%c not found.\n\n", item);
               }
            }
```

```
54              else {
55                 puts("List is empty.\n");
56              }
57
58              break;
59           default:
60              puts("Invalid choice.\n");
61              instructions();
62              break;
63        }
64
65        printf("%s", "? ");
66        scanf("%d", &choice);
67     } // end while
68
69     puts("End of run.");
70  }
71
72  // display program instructions to user
73  void instructions(void) {
74     puts("Enter your choice:\n"
75        "   1 to insert an element into the list.\n"
76        "   2 to delete an element from the list.\n"
77        "   3 to end.");
78  }
79
```

insert fonksiyonu

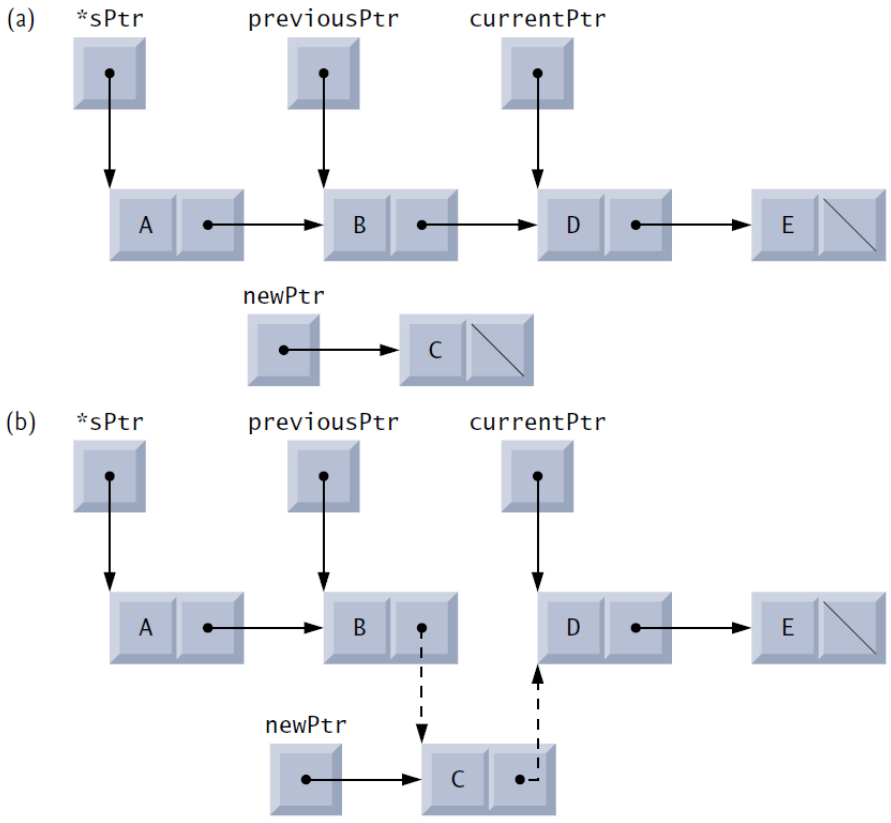
Bu örnekte, karakterler listeye alfabetik sırayla eklenir. insert fonksiyonu (satır 81–110), listenin adresini ve eklenecek bir karakteri alır. Listenin başına bir değer eklenecekse listenin adresi gereklidir. Adresin sağlanması, listenin referansa göre çağrı yoluyla değiştirilmesini sağlar. Böylece referansla işaretçiye geçilir. İşaretçinin adresine geçmek işaretçiden işaretçiye geçmeyi (pointer to a pointer) sağlar. Listenin kendisi (ilk öğesine) bir işaretçi olduğundan, adresinin iletilmesi bir işaretçiye bir işaretçi oluşturur (yani, çift dolaylı yönlendirme).

```
80  // insert a new value into the list in sorted order
81  void insert(ListNodePtr *sPtr, char value) {
82     ListNodePtr newPtr = malloc(sizeof(ListNode)); // create node
83
84     if (newPtr != NULL) { // is space available?
85        newPtr->data = value; // place value in node
86        newPtr->nextPtr = NULL; // node does not link to another node
87
88        ListNodePtr previousPtr = NULL;
89        ListNodePtr currentPtr = *sPtr;
90
91        // loop to find the correct location in the list
92        while (currentPtr != NULL && value > currentPtr->data) {
93           previousPtr = currentPtr; // walk to ...
94           currentPtr = currentPtr->nextPtr; // ... next node
95        }
96
97        // insert new node at beginning of list
98        if (previousPtr == NULL) {
99           newPtr->nextPtr = *sPtr;
100          *sPtr = newPtr;
101       }
102       else { // insert new node between previousPtr and currentPtr
103          previousPtr->nextPtr = newPtr;
104          newPtr->nextPtr = currentPtr;
105       }
106    }
107    else {
108       printf("%c not inserted. No memory available.\n", value);
109    }
110 }
111
```

Listeye karakter ekleme adımları aşağıdaki gibidir;

1. malloc'u çağırarak bir node yaratılır ve ayrılan bellek adresine “newPtr” atanır (satır 82).
2. Bellek tahsis edilirse, eklenecek karakter newPtr->data'ya (satır 85) ve NULL ise (satır 86) newPtr->nextPtr'ye atanır. Yeni node’un bağlantı üyesine her zaman NULL atanır. Önce işaretçiler başlatılmalıdır.
3. previousPtr, ve currentPtr işaretçileri ekleme noktasının sırasıyla, öncesi ve sonrasındaki düğümün (node’un) yerini belirlemek için kullanılır. previousPtr işaretçisi NULL olarak (satır 88) ve currentPtr işaretçisi ise *sPtr (satır 89) olarak başlatılır—listenin başlangıcına işaretçi olarak ilk nodun adresidir.
4. Yeni düğümün ekleme noktası bulunur. Bunun için, currentPtr, NULL değilken ve eklenecek değer currentPtr->data'dan (satır 92) büyükken, currentPtr, previousPtr'ye (satır 93) atanır ve currentPtr listedeki bir sonraki düğüme yükseltilir (satır 94).
5. Yeni değer listeye eklenir. Bunun için, eğer previousPtr NULL ise (satır 98), yeni düğüm listede ilk olarak eklenir (satır 99–100). *sPtr, newPtr->nextPtr'ye atanır (yeni düğüm bağlantısı önceki ilk düğümü gösterir) ve newPtr, *sPtr'ye atanır (*sPtr yeni düğümü gösterir), böylece “startPtr” yeni düğümün ilk temel noktası olur. Eğer, previousPtr NULL değilse, yeni düğüm yerine yerleştirilir (satır 103–104). newPtr işaretçisi, previousPtr->nextPtr'ye atanır (önceki düğüm yeni düğümü gösterir) ve currentPtr ise newPtr->nextPtr'ye atanır (yeni düğüm bağlantısı mevcut düğümü gösterir).

Aşağıdaki Şekil, 'C' karakterini içeren bir düğümün sıralı bir listeye eklenmesini göstermektedir. Şeklin (a) kısmı listeyi ve eklemeden hemen önceki yeni düğümü gösterir. Şeklin (b) kısmı, yeni düğümün eklenmesinin sonucunu gösterir. Yeniden atanan işaretçiler noktalı oklardır.



Silme Fonksiyonu

Sil fonksiyonu (satır 113–141), listenin başına işaretçinin adresini ve silinecek bir karakteri alır.

```
112 // delete a list element
113 char delete(ListNodePtr *sPtr, char value) {
114    // delete first node if a match is found
115    if (value == (*sPtr)->data) {
116       ListNodePtr tempPtr = *sPtr; // hold onto node being removed
117       *sPtr = (*sPtr)->nextPtr; // de-thread the node
118       free(tempPtr); // free the de-threaded node
119       return value;
120    }

121    else {
122       ListNodePtr previousPtr = *sPtr;
123       ListNodePtr currentPtr = (*sPtr)->nextPtr;
124
125       // loop to find the correct location in the list
126       while (currentPtr != NULL && currentPtr->data != value) {
127          previousPtr = currentPtr; // walk to ...
128          currentPtr = currentPtr->nextPtr; // ... next node
129       }
130
131       // delete node at currentPtr
132       if (currentPtr != NULL) {
133          ListNodePtr tempPtr = currentPtr;
134          previousPtr->nextPtr = currentPtr->nextPtr;
135          free(tempPtr);
136          return value;
137       }
138    }
139
140    return '\0';
141 }
142
```

Silme Fonksiyonunun adımları şöyle sıralanabilir;

1. Silinecek karakter listenin ilk düğümündeki (satır 115) karakterle eşleşiyorsa ilk düğüm silinir. Düğümlerin bulunduğu belleği boşaltmak için tempPtr'ye *sPtr atanır (tempPtr gereksiz belleği boşaltmak için kullanılacaktır). (*sPtr)->nextPtr ise *sPtr 'ye atanır böylece startPtr artık önceden ikinci düğüm olan düğümü göstermektedir (*sPtr artık listedeki ikinci düğümü işaret ediyor). Free fonksiyonu ile tempPtr tarafından işaret edilen bellek boşaltılır ve silinen karakter geri getirilir.

2. Diğer durumda, ikinci düğüme ilerlemek için previousPtr, *sPtr ile başlatılır ve currentPtr ise (*sPtr)->nextPtr (satır 122–123) ile başlatılır.

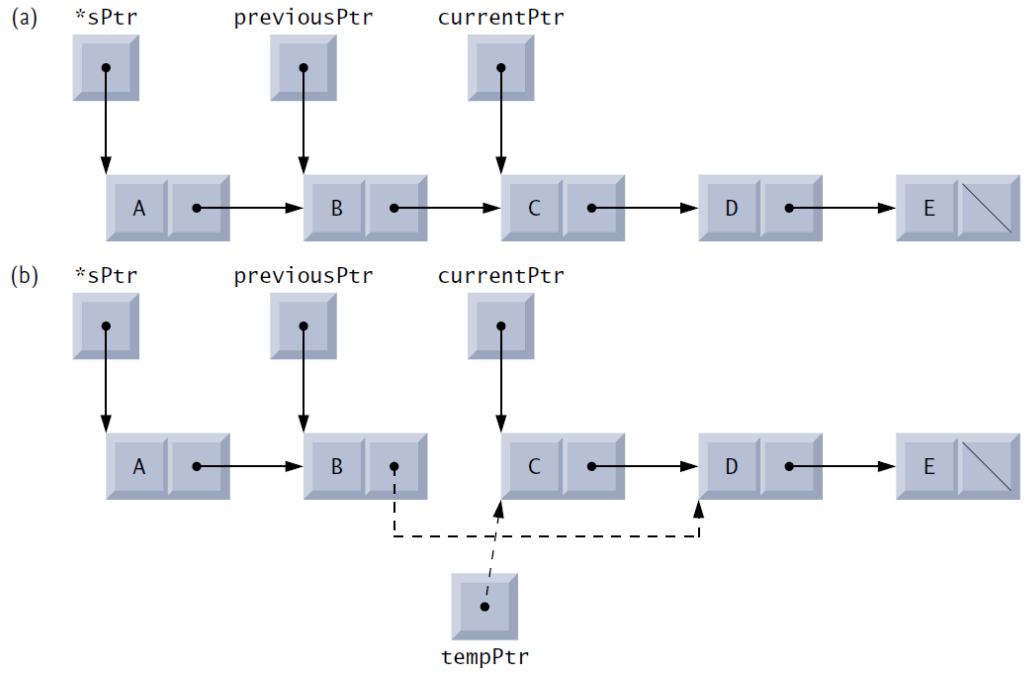
3. Listede varsa silinecek karakter bulunur. Bunun için, currentPtr NULL değilken ve silinecek değer currentPtr->data'ya (satır 126) eşit değilken, currentPtr previousPtr'ye (satır 127) atanır ve currentPtr->nextPtr ise currentPtr'ye (satır 128) atanır.

4. currentPtr, NULL değilse (satır 132) karakter listededir. Bu durumda, currentPtr, düğümü serbest bırakmak için kullanacağımız tempPtr'ye atanır (satır 133), currentPtr->nextPtr ise kaldırılandan önceki düğümü ve sonraki düğümü bağlamak için previousPtr->nextPtr'ye (satır 134) atanır. tempPtr tarafından işaret edilen düğüm serbest bırakılır (satır 135) ve listeden silinen karakter döndürülür (satır

136). currentPtr NULL ise, silinecek karakterin listede bulunmadığını belirtmek için boş karakteri ('\0') döndürülür (satır 140).

Aşağıdaki şekil, bağlantılı bir listeden 'C' karakterini içeren düğümün silinmesini göstermektedir. Şeklin (a) kısmı, önceki ekleme işleminden sonraki bağlantılı listeyi gösterir. Kısım (b), previousPtr'nin bağlantı öğesinin yeniden atanmasını ve currentPtr'nin tempPtr'ye atanmasını gösterir.

İşaretçi tempPtr, 'C' karakterini depolayan düğüme ayrılan belleği boşaltmak için kullanılır. 118 ve 135. satırlarda tempPtr'nin serbest bırakıldığına dikkat edin.



isEmpty ve printList Fonksiyonları

isEmpty fonksiyonu (satır 144–146) bir doğrulama fonksiyonudur.—listeyi hiçbir şekilde değiştirmez. Bunun yerine isEmpty, listenin boş olup olmadığını belirler; bu durumda, ilk düğümün işaretçisi NULL'dur. Liste boşsa, isEmpty 1 döndürür; aksi takdirde, 0 döndürür.

printList fonksiyonu (satır 149–165) bir liste yazdırır. Fonksiyonun currentPtr parametresi, listenin ilk düğümüne bir işaretçi alır. Fonksiyon önce listenin boş olup olmadığını belirler (satır 151–153) ve boşsa "Liste boş" yazdırır ve sonlandırır. Aksi takdirde, 155–163 satırları listenin verilerini yazdırır. currentPtr NULL değilken, 159. satır, currentPtr->data'daki değeri yazdırır ve 160. satır, bir sonraki düğüme ilerlemek için currentPtr->nextPtr'yi currentPtr'ye atar. Listenin son düğümündeki bağlantı NULL değilse, yazdırma algoritması bir mantık hatası olan listenin sonunu yazdırmaya çalışır. Bu yazdırma algoritması bağlantılı listeler, yığınlar ve kuyruklar için de aynıdır.

```c
// return 1 if the list is empty, 0 otherwise
int isEmpty(ListNodePtr sPtr) {
   return sPtr == NULL;
}

// print the list
void printList(ListNodePtr currentPtr) {
   // if list is empty
   if (isEmpty(currentPtr)) {
      puts("List is empty.\n");
   }
   else {
      puts("The list is:");

      // while not the end of the list
      while (currentPtr != NULL) {
         printf("%c --> ", currentPtr->data);
         currentPtr = currentPtr->nextPtr;
      }

      puts("NULL\n");
   }
}
```