

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.

Kuyruklar

Kuyruklar, bir mağazadaki ödeme kuyruğuna benzer: Sıradaki ilk kişi hizmeti önce alır. Diğer müşteriler hatta sadece sonundan girerler ve servis beklerler. Kuyruk düğümlerini yalnızca başından (önden) kaldırır ve düğümleri yalnızca kuyruğuna (arkadan) eklersiniz. Bu nedenle kuyruk, ilk giren ilk çıkar (First In First Out-FIFO) veri yapısı olarak adlandırılır. Ekleme ve çıkarma işlemleri, sırasıyla enqueue ve dequeue olarak bilinir.

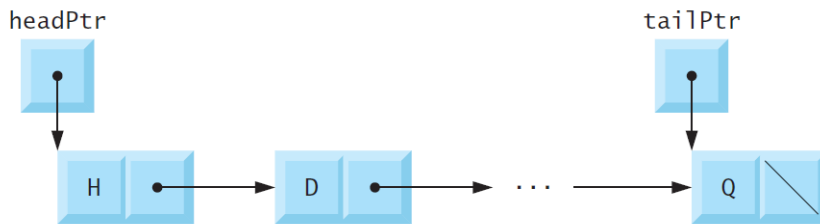
Kuyruk Uygulamaları

Kuyrukların bilgisayar sistemlerinde birçok uygulaması vardır:

- Yalnızca tek işlemciye sahip bilgisayarlarda aynı anda yalnızca bir kullanıcıya hizmet verilebilir. Diğer kullanıcılar için girişler bir kuyruğa yerleştirilir. Kullanıcılar hizmet aldıkça her giriş kademeli olarak sıranın önüne ilerler. Kuyruğun önündeki giriş, hizmeti alacak sonraki giriştir.
- Benzer şekilde, günümüzün çok çekirdekli sistemleri için, çalışan işlemci sayısından daha fazla program olabilir. Şu anda çalışmayan programlar, o anda meşgul olan bir işlemci kullanılabilir hale gelene kadar bir kuyruğa yerleştirilir.
- Kuyruklar ayrıca yazdırma biriktirmeyi de destekler. Bir ofiste yalnızca bir yazıcı olabilir. Birçok kullanıcı belirli bir zamanda yazdırılacak belgeleri gönderebilir. Yazıcı meşgul olduğunda ek belgeler bellekte kuyruğa alınır. Belgeler, yazıcı kullanılabilir hale gelene kadar kuyrukta bekler.
- İnternet gibi bilgisayar ağları üzerinden dolaşan bilgi paketleri de kuyruklarda bekler. Bir paket bir ağ düğümüne her ulaştığında, nihai hedefine giden yol boyunca ağdaki bir sonraki düğüme yönlendirilmelidir. Yönlendirme düğümü her seferinde bir paket yönlendirir, böylece yönlendirici onları yönlendirene kadar ek paketler kuyruğa alınır.

Bir Kuyruğu Gösterme

Aşağıdaki diyagram, birkaç düğüm içeren bir kuyruğu göstermektedir. Kuyruğun başı ve sonu için ayrı işaretçilere dikkat edin. Bağlantılı listelerde ve yığınlarda olduğu gibi, kuyruğun son düğümündeki bağlantının NULL olarak ayarlanmaması mantık hatalarına yol açabilir.



Kuyruk Uygulaması

Şekil 12.3'te verilen program, kuyruk işlemlerini gerçekleştirir. Program, kuyruğa bir düğüm eklemek (enqueue fonksiyonu), bir düğümü sıradan çıkarmak (dequeue fonksiyonu) ve programı sonlandırmak için seçenekler sunar. 7-10 arasındaki satırlar, kuyruğun düğümlerini temsil etmek için kullanacağımız struct queueNode'u tanımlar. Yine, kodu daha okunaklı hale getirmek için typedef'leri (satır 12-13) kullanıyoruz. Bu uygulamadaki mantığın çoğu, bağlantılı liste ve yığın örneklerimize (fig_12_1 ve fig_12_2 deki programlar) benzer, bu nedenle burada sadece farklılıklara bakalım. Başlangıçta, headPtr ve tailPtr (23-24. satırlar) boş bir kuyruğu belirtmek için NULL'dur.

```
1 // fig12_03.c
2 // Operating and maintaining a queue
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 // self-referential structure
7 struct queueNode {
8     char data; // define data as a char
9     struct queueNode *nextPtr; // queueNode pointer
10 };
11
12 typedef struct queueNode QueueNode;
13 typedef QueueNode *QueueNodePtr;
14
15 // function prototypes
16 void printQueue(QueueNodePtr currentPtr);
17 int isEmpty(QueueNodePtr headPtr);
18 void enqueue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value);
19 char dequeue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr);
20 void instructions(void);
21
22 int main(void) {
23     QueueNodePtr headPtr = NULL; // initialize headPtr
24     QueueNodePtr tailPtr = NULL; // initialize tailPtr
25     char item = '\0'; // char input by user
26
27     instructions(); // display the menu
28     printf("%s", "? ");
29     int choice = 0; // user's menu choice
30     scanf("%d", &choice);
31
32     // while user does not enter 3
33     while (choice != 3) {
34         switch(choice) {
35             case 1: // enqueue value
36                 printf("%s", "Enter a character: ");
37                 scanf("%c", &item);
38                 enqueue(&headPtr, &tailPtr, item);
39                 printQueue(headPtr);
40                 break;
41             case 2: // dequeue value
42                 // if queue is not empty
43                 if (!isEmpty(headPtr)) {
44                     item = dequeue(&headPtr, &tailPtr);
45                     printf("%c has been dequeued.\n", item);
46                 }
47
48                 printQueue(headPtr);
49                 break;
50             default:
51                 puts("Invalid choice.\n");
52                 instructions();
53                 break;
54         }
55
56         printf("%s", "? ");
57         scanf("%d", &choice);
58     }
```

```

59
60     puts("End of run.");
61 }
62
63 // display program instructions to user
64 void instructions(void) {
65     printf("Enter your choice:\n"
66           "    1 to add an item to the queue\n"
67           "    2 to remove an item from the queue\n"
68           "    3 to end\n");
69 }
70
71 // insert a node at queue tail
72 void enqueue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value) {
73     QueueNodePtr newPtr = malloc(sizeof(QueueNode));
74
75     if (newPtr != NULL) { // is space available?
76         newPtr->data = value;
77         newPtr->nextPtr = NULL;
78
79         // if empty, insert node at head
80         if (isEmpty(*headPtr)) {
81             *headPtr = newPtr;
82         }
83         else {
84             (*tailPtr)->nextPtr = newPtr;
85         }
86
87         *tailPtr = newPtr;
88     }
89     else {
90         printf("%c not inserted. No memory available.\n", value);
91     }
92 }
93
94 // remove node from queue head
95 char dequeue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr) {
96     char value = (*headPtr)->data;
97     QueueNodePtr tempPtr = *headPtr;
98     *headPtr = (*headPtr)->nextPtr;
99
100     // if queue is empty
101     if (*headPtr == NULL) {
102         *tailPtr = NULL;
103     }
104
105     free(tempPtr);
106     return value;
107 }
108
109 // return 1 if the queue is empty, 0 otherwise
110 int isEmpty(QueueNodePtr headPtr) {
111     return headPtr == NULL;
112 }
113
114 // print the queue
115 void printQueue(QueueNodePtr currentPtr) {
116     if (currentPtr == NULL) { // if queue is empty
117         puts("Queue is empty.\n");
118     }
119     else {
120         puts("The queue is:");
121
122         while (currentPtr != NULL) { // while not end of queue
123             printf("%c --> ", currentPtr->data);
124             currentPtr = currentPtr->nextPtr;
125         }
126
127         puts("NULL\n");
128     }
129 }

```

```
Enter your choice:
 1 to add an item to the queue
 2 to remove an item from the queue
 3 to end
```

```
? 1
Enter a character: A
The queue is:
A --> NULL
```

```
? 1
Enter a character: B
The queue is:
A --> B --> NULL
```

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

```
? 2
A has been dequeued.
The queue is:
B --> C --> NULL
```

```
? 2
B has been dequeued.
The queue is:
C --> NULL
```

```
? 2
C has been dequeued.
Queue is empty.
```

```
? 2
Queue is empty.
```

```
? 4
Invalid choice.
```

```
Enter your choice:
 1 to add an item to the queue
 2 to remove an item from the queue
 3 to end
```

```
? 3
End of run.
```

Function enqueue (kuyruğa ekleme fonksiyonu)

Sıraya alma (Kuyruğa ekleme) fonksiyonu (72–92 satırları) üç bağımsız değişken alır:

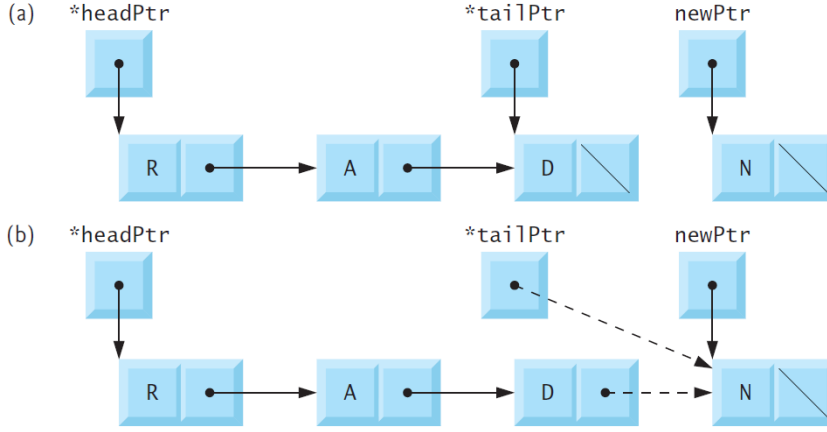
- headPtr'nin adresi—kuyruğun başına işaretçi,
- tailPtr'nin adresi—kuyruğun sonuna işaretçi ve
- value- girilecek değer.

Fonksiyon aşağıdaki adımları gerçekleştirir:

1. Satır 73, yeni bir düğüm oluşturmak için malloc'u çağırır ve ayrılan hafıza konumunu newPtr'ye atar.
2. Bellek doğru bir şekilde tahsis edildiyse, 76–77 satırları newPtr->data eklenecek değeri atar ve newPtr->nextPtr'ye NULL atar.
3. Kuyruk boşsa (satır 80), 81. satır newPtr'yi *headPtr'ye atar çünkü yeni düğüm sıranın hem başı hem de kuyruğudur; aksi halde, 84. satır newPtr'yi (*tailPtr)->nextPtr'ye atar, çünkü yeni düğüm yeni kuyruk düğümüdür.
4. Satır 87, kuyruk işaretçisini yeni kuyruk düğümüne güncellemek için newPtr'yi *tailPtr'ye atar.

Bir kuyruğa alma işlemini Gösterme

Aşağıdaki diyagram bir kuyruğa alma işlemini göstermektedir. Kısım (a), işlemten önce main'in headPtr ve tailPtr'sini ve enqueue'nin yeni düğümünü gösterir. Kısım (b)'deki noktalı oklar, önceki durumda boş olmayan bir sıranın kuyruğuna yeni düğüm ekleyen Adım 3 ve 4'ü göstermektedir.



Kuyruktan çıkarma Fonksiyonu (Function dequeue)

Kuyruktan çıkarma fonksiyonu (satır 95–107), sıranın baş ve kuyruk işaretçilerinin adreslerini argüman olarak alır ve sıranın ilk düğümünü kaldırır. Sıradan çıkarma işlemi aşağıdaki adımları gerçekleştirir:

1. Satır 96, kuyruktan çıkarılmakta olan verileri kaydetmek için (*headPtr)->data 'yı value' değişkenine atar.
2. Satır 97, hafızayı boşaltmak için kullanılacak *headPtr'yi tempPtr'ye atar.
3. Satır 98, (*headPtr)->nextPtr'yi *headPtr'ye atar, böylece kuyruğun ana işaretçisi şimdi yeni ana düğümü gösterir.
4. Satır 101, *headPtr'nin NULL olup olmadığını kontrol eder. Eğer öyleyse, sıra artık boş olduğu için 102. satır *tailPtr'ye NULL atar.
5. Satır 105, tempPtr tarafından işaret edilen belleği boşaltır.
6. Satır 106, değeri döndürür.

Kuyruktan çıkarma İşlemini Gösterme

Aşağıdaki diyagram, kuyruktan çıkarma işlemi göstermektedir. Kısım (a), kuyruktan çıkarma işleminden önceki kuyruğu gösterir—*headPtr ve *tailPtr, kuyruktan çıkarmada main'de headPtr ve tailPtr'yi değiştirmek için kullanılır. Kısım (b), tempPtr'nin kuyruktan çıkarılan düğümü işaret ettiğini ve main'in headPtr'sinin kuyruğun yeni ilk düğümünü işaret edecek şekilde güncellendiğini gösterir.

