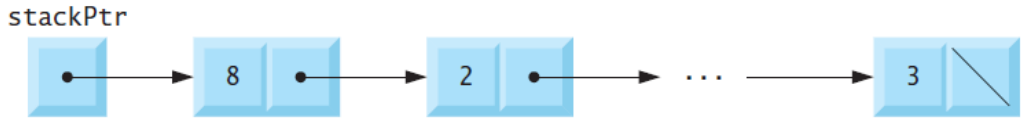
Yığınlar (Stacks)

NOT-1: Bu bölüm “Paul Deitel, Harvey Deitel - 9nth_editionC 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.

Yığın, bağlantılı bir listenin kısıtlanmış bir versiyonu olarak uygulanabilir. Yalnızca en üstte yeni düğümler ekler ve en üstten başlayarak mevcut olanları kaldırırsınız. Bu nedenle, bir yığına son giren ilk çıkar (Last In Firts Out - LIFO) veri yapısı denir. Bir yığına, üst elemanına bir işaretçi aracılığıyla erişirsiniz. Yığının son düğümündeki bağlantı üyesi, yığının dibini belirtmek için NULL olarak ayarlanır. Yığının NULL ile sonlandırılmaması çalışma zamanı hatalarına neden olabilir.

Aşağıdaki diyagram, birkaç düğüme sahip bir yığını göstermektedir; stackPtr, yığının en üst öğesini gösterir. Bu şekillerde yığınları ve bağlantılı listeleri aynı şekilde temsil ediliyor. Aralarındaki fark, ekleme ve silme işlemlerinin bağlantılı bir listede herhangi bir yerde, ancak bir yığının yalnızca en üstünde gerçekleşebilmesidir.



Bir yığının birincil fonksiyonları, itme (push) ve çekme (pop) fonksiyonlarıdır. push fonksiyonu, yeni bir düğüm oluşturur ve onu yığının üstüne yerleştirir. pop fonksiyonu, yığının tepesinden bir düğümü kaldırır, bu düğümün belleğini boşaltır ve değeri döndürür.

Yığın Uygulama

Şekil 12.2’deki program, basit bir tamsayı yığını uygulamaktadır. Program, yığına bir değer itmenize (push function), yığından bir değer çıkarmanıza (pop function) ve programı sonlandırmanıza izin verir. 7-10 arasındaki satırlar, yığının düğümlerini temsil etmek için kullanacağımız stackNode isimli struct yapısı tanımlar. Şekil 12.1'deki programda olduğu gibi, kodu daha okunaklı hale getirmek için typedef'leri (satır 12–13) kullanıyoruz.

```
1  // fig12_02.c
2  // A simple stack program
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  // self-referential structure
7  struct stackNode {
8     int data; // define data as an int
9     struct stackNode *nextPtr; // stackNode pointer
10 };
11
12 typedef struct stackNode StackNode; // synonym for struct stackNode
13 typedef StackNode *StackNodePtr; // synonym for StackNode*
14
15 // prototypes
16 void push(StackNodePtr *topPtr, int info);
17 int pop(StackNodePtr *topPtr);
18 int isEmpty(StackNodePtr topPtr);
19 void printStack(StackNodePtr currentPtr);
20 void instructions(void);
21
```

Başlangıçta, stackPtr (satır 23), boş bir yığını belirtmek için NULL olarak ayarlanır. Bu uygulamanın mantığının çoğu Şekil 12.1'deki programa benzer, bu yüzden burada sadece farklılıklara bakalım.

```
22 int main(void) {
23    StackNodePtr stackPtr = NULL; // points to stack top
24    int value = 0; // int input by user
25
26    instructions(); // display the menu
27    printf("%s", "? ");
28    int choice = 0; // user's menu choice
29    scanf("%d", &choice);
30
31    // while user does not enter 3
32    while (choice != 3) {
33       switch (choice) {
34          case 1: // push value onto stack
35             printf("%s", "Enter an integer: ");
36             scanf("%d", &value);
37             push(&stackPtr, value);
38             printStack(stackPtr);
39             break;
40          case 2: // pop value off stack
41             // if stack is not empty
42             if (!isEmpty(stackPtr)) {
43                printf("The popped value is %d.\n", pop(&stackPtr));
44             }
45
46             printStack(stackPtr);
47             break;
48          default:
49             puts("Invalid choice.\n");
50             instructions();
51             break;
52       }
53
54       printf("%s", "? ");
55       scanf("%d", &choice);
56    }
57
58    puts("End of run.");
59 }
60
61 // display program instructions to user
62 void instructions(void) {
63    puts("Enter choice:\n"
64       "1 to push a value on the stack\n"
65       "2 to pop a value off the stack\n"
66       "3 to end program");
67 }
68
69 // insert a node at the stack top
70 void push(StackNodePtr *topPtr, int info) {
71    StackNodePtr newPtr = malloc(sizeof(StackNode));
72
```

```
73      // insert the node at stack top
74      if (newPtr != NULL) {
75         newPtr->data = info;
76         newPtr->nextPtr = *topPtr;
77         *topPtr = newPtr;
78      }
79      else { // no space available
80         printf("%d not inserted. No memory available.\n", info);
81      }
82   }
83
84   // remove a node from the stack top
85   int pop(StackNodePtr *topPtr) {
86      StackNodePtr tempPtr = *topPtr;
87      int popValue = (*topPtr)->data;
88      *topPtr = (*topPtr)->nextPtr;
89      free(tempPtr);
90      return popValue;
91   }
92
93   // print the stack
94   void printStack(StackNodePtr currentPtr) {
95      if (currentPtr == NULL) { // if stack is empty
96         puts("The stack is empty.\n");
97      }
98      else {
99         puts("The stack is:");
100
101        while (currentPtr != NULL) { // while not the end of the stack
102           printf("%d --> ", currentPtr->data);
103           currentPtr = currentPtr->nextPtr;
104        }
105
106        puts("NULL\n");
107     }
108  }
109
110  // return 1 if the stack is empty, 0 otherwise
111  int isEmpty(StackNodePtr topPtr) {
112     return topPtr == NULL;
113  }
```

```
Enter choice:
1 to push a value on the stack
2 to pop a value off the stack
3 to end program
? 1

Enter an integer: 5
The stack is:
5 --> NULL
```

```
? 1
Enter an integer: 6
The stack is:
6 --> 5 --> NULL

? 1
Enter an integer: 4
The stack is:
4 --> 6 --> 5 --> NULL

? 2
The popped value is 4.
The stack is:
6 --> 5 --> NULL

? 2
The popped value is 6.
The stack is:
5 --> NULL

? 2
The popped value is 5.
The stack is empty.

? 2
The stack is empty.

? 4
Invalid choice.

Enter choice:
1 to push a value on the stack
2 to pop a value off the stack
3 to end program
? 3
End of run.
```

Function push (itme fonksiyonu)

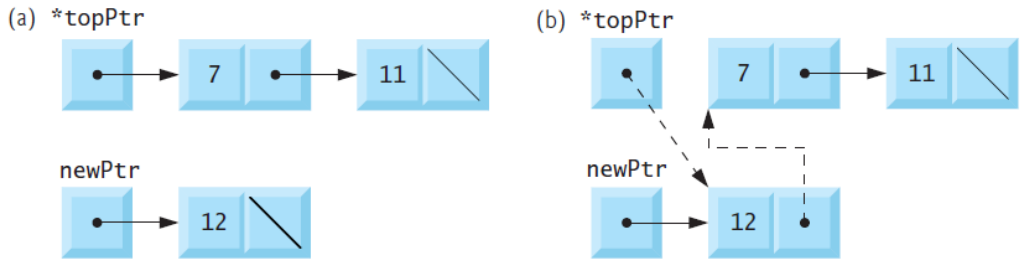
Fonksiyon push (satır 70–82), aşağıdaki adımları kullanarak yığına yeni bir düğüm yerleştirir:

1. Yeni bir düğüm oluşturmak için malloc'u çağırır, ardından tahsis edilen belleğin adresini newPtr'ye atar (satır 71).

2. newPtr->data'ya yığına itilecek değer atanır (satır 75) ve *topPtr (yığının tepesine işaretçi) newPtr->nextPtr'ye (satır 76) atanır. Yeni üst düğümün bağlantı üyesi artık bir önceki üst düğüme işaret ediyor.

3. newPtr ise *topPtr'ye atanır (satır 77)—bu, stackPtr'yi main'de yeni yığının tepesine işaret edecek şekilde değiştirir.

Aşağıdaki şemada bir push (itme) işlemi gösterilmektedir. Kısım (a), itme işlemi yeni düğümü yığının en üstüne eklemeden önceki yığını ve yeni düğümü gösterir—*topPtr, main'de stackPtr'yi temsil eder. Kısım (b)'deki noktalı oklar, 12'yi içeren düğümü en üste yerleştiren önceki tartışmanın 2. ve 3. adımlarını göstermektedir.



Function pop (çekme Fonksiyonu)

pop fonksiyonu (satır 85–91), yığının üst düğümünü kaldırır. main işlevi, pop çağrılmadan önce yığının boş olup olmadığını belirler. Pop işlemi beş adımdan oluşur:

1. Düğümün belleğini boşaltmak için kullanılacak *topPtr işaretçisi tempPtr'ye (satır 86) atanır.

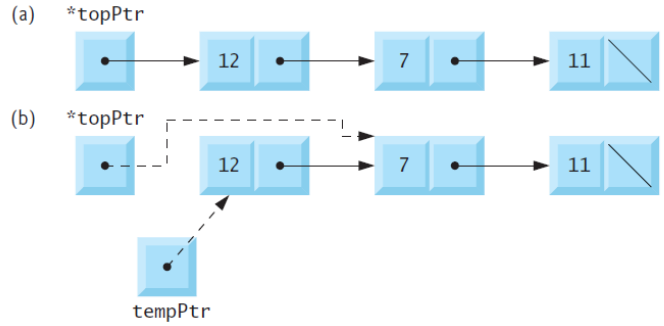
2. Döndürebilmemiz için üst düğümün değerini kaydetmek üzere (*topPtr)->data popValue'ye (satır 87) atanır.

3. (*topPtr)->nextPtr, *topPtr'ye (satır 88) atanır, böylece main'deki stackPtr daha önce yığının ikinci öğesini (veya başka öğe yoksa NULL) gösterir.

4. tempPtr (satır 89) tarafından işaret edilen bellek boşaltılır.

5. PopValue değeri döndürülür (satır 90).

Aşağıdaki şemada bir pop işlemi gösterilmektedir. Kısım (a), 12'yi içeren düğümü çıkarmadan önceki yığını gösterir—*topPtr, main'de stackPtr'yi temsil eder. Kısım (b), açılan düğüme işaret eden tempPtr'yi ve yeni üst düğüme işaret eden *topPtr'yi gösterir. Ardından tempPtr'nin işaret ettiği bellek serbest bırakılabilir.



Yığın Uygulamaları

Yığınların birçok ilginç uygulaması vardır. Örneğin, bir fonksiyon çağrısı yapıldığında, çağrılan fonksiyon çağırana nasıl geri döneceğini bilmelidir, böylece dönüş adresi bir yığına itilir. Bir dizi fonksiyon çağrısında, birbirini izleyen dönüş adresleri, her fonksiyonun çağırana geri dönebilmesi için son giren ilk çıkar sırasına göre yığına itilir.

Yığınlar, bir fonksiyonun her çağrılmasında otomatik yerel değişkenler için oluşturulan alanı içerir. Fonksiyon, çağırana geri döndüğünde, o fonksiyonun otomatik değişkenleri için alan yığından atılır ve bu değişkenler artık program tarafından bilinmez. Yığınlar bazen derleyiciler tarafından ifadeleri değerlendirme ve makine dili kodu oluşturma sürecinde de kullanılır.