Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Basic Data Structures

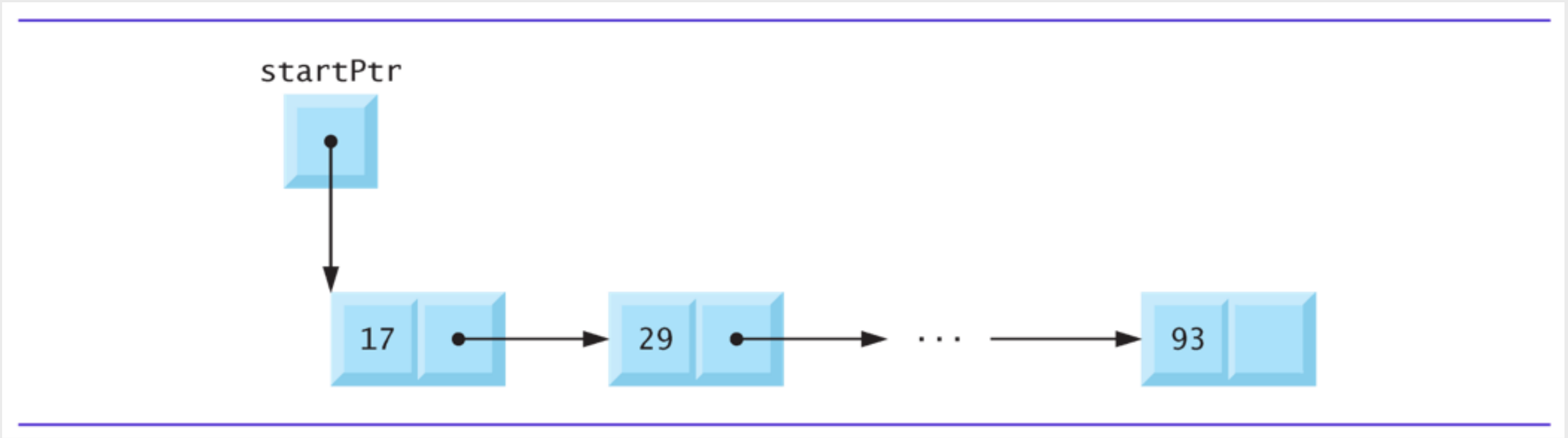
Linked Lists

- A linked list is a linear collection of self-referential structures, called nodes, connected by pointer links—hence, the term “linked” list.
- A linked list is accessed via a pointer to the first node of the list.
- By convention, the link pointer in the last node of a list is set to NULL to mark the end of the list.
- Linked lists are dynamic, so the length of a list can increase or decrease as necessary.

Linked Lists

- The size of an array, however cannot be altered once memory is allocated.
  - Arrays can become full.
- Linked lists become full only when the system has insufficient memory to satisfy dynamic storage allocation requests.

Linked Lists



Linked Lists

- Linked list nodes are normally not stored contiguously in memory.
- Logically, however, the nodes of a linked list appear to be contiguous.

Linked Lists

- Example: fig12_03.c

```c
/* Fig. 12.3: fig12_03.c
   Operating and maintaining 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*/
}; /* end structure listNode */

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 );
```

Linked Lists

- Example: fig12_03.c

```c
/* Fig. 12.3: fig12_03.c
   Operating and maintaining 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*/
}; /* end structure listNode */

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 );
```

Linked Lists

- Example: fig12_03.c

```c
/* Fig. 12.3: fig12_03.c
   Operating and maintaining 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*/
}; /* end structure listNode */

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 );
```

Linked Lists

- Example: fig12_03.c

```
22 int main( void ) {
23     ListNodePtr startPtr = NULL; /* initially there are no nodes */
24     int choice; /* user's choice */
25     char item; /* char entered by user */
26
27     instructions(); /* display the menu */
28     printf( "? " );
29     scanf( "%d", &choice );
30
31     /* loop while user does not choose 3 */
32     while ( choice != 3 ) {
33
34         switch ( choice ) {
35             case 1:
36                 printf( "Enter a character: " );
37                 scanf( "\n%c", &item );
38                 insert( &startPtr, item ); /* insert item in list */
39                 printList( startPtr );
40                 break;
```

Linked Lists

- Example: fig12_03.c

```
22 int main( void ) {
23     ListNodePtr startPtr = NULL; /* initially there are no nodes */
24     int choice; /* user's choice */
25     char item; /* char entered by user */
26
27     instructions(); /* display the menu */
28     printf( "? " );
29     scanf( "%d", &choice );
30
31     /* loop while user does not choose 3 */
32     while ( choice != 3 ) {
33
34         switch ( choice ) {
35             case 1:
36                 printf( "Enter a character: " );
37                 scanf( "\n%c", &item );
38                 insert( &startPtr, item ); /* insert item in list */
39                 printList( startPtr );
40                 break;
```

Linked Lists

- Example: fig12_03.c

```
22 int main( void ) {
23     ListNodePtr startPtr = NULL; /* initially there are no nodes */
24     int choice; /* user's choice */
25     char item; /* char entered by user */
26
27     instructions(); /* display the menu */
28     printf( "? " );
29     scanf( "%d", &choice );
30
31     /* loop while user does not choose 3 */
32     while ( choice != 3 ) {
33
34         switch ( choice ) {
35             case 1:
36                 printf( "Enter a character: " );
37                 scanf( "\n%c", &item );
38                 insert( &startPtr, item ); /* insert item in list */
39                 printList( startPtr );
40                 break;
```

Linked Lists

- Example: fig12_03.c

```
41              case 2:
42                  /* if list is not empty */
43                  if ( !isEmpty( startPtr ) ) {
44                      printf( "Enter character to be deleted: " );
45                      scanf( "\n%c", &item );
46
47                      /* if character is found, remove it */
48                      if ( delete( &startPtr, item ) ) { /* remove item */
49                          printf( "%c deleted.\n", item );
50                          printList( startPtr );
51                      } /* end if */
52                      else {
53                          printf( "%c not found.\n\n", item );
54                      } /* end else */
55                  } /* end if */
56                  else {
57                      printf( "List is empty.\n\n" );
58                  } /* end else */
59                  break;
60              default:
61                  printf( "Invalid choice.\n\n" );
62                  instructions();
63                  break;
64          } /* end switch */
```

Linked Lists

- Example: fig12_03.c

```
41              case 2:
42                  /* if list is not empty */
43                  if ( !isEmpty( startPtr ) ) {
44                      printf( "Enter character to be deleted: " );
45                      scanf( "\n%c", &item );
46
47                      /* if character is found, remove it */
48                      if ( delete( &startPtr, item ) ) { /* remove item */
49                          printf( "%c deleted.\n", item );
50                          printList( startPtr );
51                      } /* end if */
52                      else {
53                          printf( "%c not found.\n\n", item );
54                      } /* end else */
55                  } /* end if */
56                  else {
57                      printf( "List is empty.\n\n" );
58                  } /* end else */
59                  break;
60              default:
61                  printf( "Invalid choice.\n\n" );
62                  instructions();
63                  break;
64          } /* end switch */
```

Linked Lists

- Example: fig12_03.c

```
41              case 2:
42                  /* if list is not empty */
43                  if ( !isEmpty( startPtr ) ) {
44                      printf( "Enter character to be deleted: " );
45                      scanf( "\n%c", &item );
46
47                      /* if character is found, remove it */
48                      if ( delete( &startPtr, item ) ) { /* remove item */
49                          printf( "%c deleted.\n", item );
50                          printList( startPtr );
51                      } /* end if */
52                      else {
53                          printf( "%c not found.\n\n", item );
54                      } /* end else */
55                  } /* end if */
56                  else {
57                      printf( "List is empty.\n\n" );
58                  } /* end else */
59                  break;
60              default:
61                  printf( "Invalid choice.\n\n" );
62                  instructions();
63                  break;
64          } /* end switch */
```

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

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

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

```c
char delete( ListNodePtr *sPtr, char value )
{
    ListNodePtr previousPtr; /* pointer to previous node in list */
    ListNodePtr currentPtr; /* pointer to current node in list */
    ListNodePtr tempPtr; /* temporary node pointer */

    /* delete first node */
    if ( value == ( *sPtr )->data ) {
        tempPtr = *sPtr; /* hold onto node being removed */
        *sPtr = ( *sPtr )->nextPtr; /* de-thread the node */
        free( tempPtr ); /* free the de-threaded node */
        return value;
    } /* end if */
    else {
        previousPtr = *sPtr;
        currentPtr = ( *sPtr )->nextPtr;

        /* loop to find the correct location in the list */
        while ( currentPtr != NULL && currentPtr->data != value ) {
            previousPtr = currentPtr; /* walk to ...   */
            currentPtr = currentPtr->nextPtr; /* ... next node */
        } /* end while */

        /* delete node at currentPtr */
        if ( currentPtr != NULL ) {
            tempPtr = currentPtr;
            previousPtr->nextPtr = currentPtr->nextPtr;
            free( tempPtr );
            return value;
        } /* end if */
    } /* end else */

    return '\0';
} /* end function delete */
```

```c
char delete( ListNodePtr *sPtr, char value )
{
    ListNodePtr previousPtr; /* pointer to previous node in list */
    ListNodePtr currentPtr; /* pointer to current node in list */
    ListNodePtr tempPtr; /* temporary node pointer */

    /* delete first node */
    if ( value == ( *sPtr )->data ) {
        tempPtr = *sPtr; /* hold onto node being removed */
        *sPtr = ( *sPtr )->nextPtr; /* de-thread the node */
        free( tempPtr ); /* free the de-threaded node */
        return value;
    } /* end if */
    else {
        previousPtr = *sPtr;
        currentPtr = ( *sPtr )->nextPtr;

        /* loop to find the correct location in the list */
        while ( currentPtr != NULL && currentPtr->data != value ) {
            previousPtr = currentPtr; /* walk to ...    */
            currentPtr = currentPtr->nextPtr; /* ... next node */
        } /* end while */

        /* delete node at currentPtr */
        if ( currentPtr != NULL ) {
            tempPtr = currentPtr;
            previousPtr->nextPtr = currentPtr->nextPtr;
            free( tempPtr );
            return value;
        } /* end if */
    } /* end else */

    return '\0';
} /* end function delete */
```

Linked Lists

- Example: fig12_03.c

Linked Lists

- Example: fig12_03.c

```
157 int isEmpty( ListNodePtr sPtr )
158 {
159     return sPtr == NULL;
160 } /* end function isEmpty */
```

Linked Lists

- Example: fig12_03.c

```
157 int isEmpty( ListNodePtr sPtr )
158 {
159     return sPtr == NULL;
160 } /* end function isEmpty */
```

```
163 void printList( ListNodePtr currentPtr )
164 {
165     /* if list is empty */
166     if ( currentPtr == NULL ) {
167         printf( "List is empty.\n\n" );
168     } /* end if */
169     else {
170         printf( "The list is:\n" );
171 
172         /* while not the end of the list */
173         while ( currentPtr != NULL ) {
174             printf( "%c --> ", currentPtr->data );
175             currentPtr = currentPtr->nextPtr;
176         } /* end while */
177 
178         printf( "NULL\n\n" );
179     } /* end else */
180 } /* end function printList */
```

Linked Lists

- Example: fig12_03.c

```
157 int isEmpty( ListNodePtr sPtr )
158 {
159     return sPtr == NULL;
160 } /* end function isEmpty */
```

```
163 void printList( ListNodePtr currentPtr )
164 {
165     /* if list is empty */
166     if ( currentPtr == NULL ) {
167         printf( "List is empty.\n\n" );
168     } /* end if */
169     else {
170         printf( "The list is:\n" );
171 
172         /* while not the end of the list */
173         while ( currentPtr != NULL ) {
174             printf( "%c --> ", currentPtr->data );
175             currentPtr = currentPtr->nextPtr;
176         } /* end while */
177 
178         printf( "NULL\n\n" );
179     } /* end else */
180 } /* end function printList */
```

Linked Lists

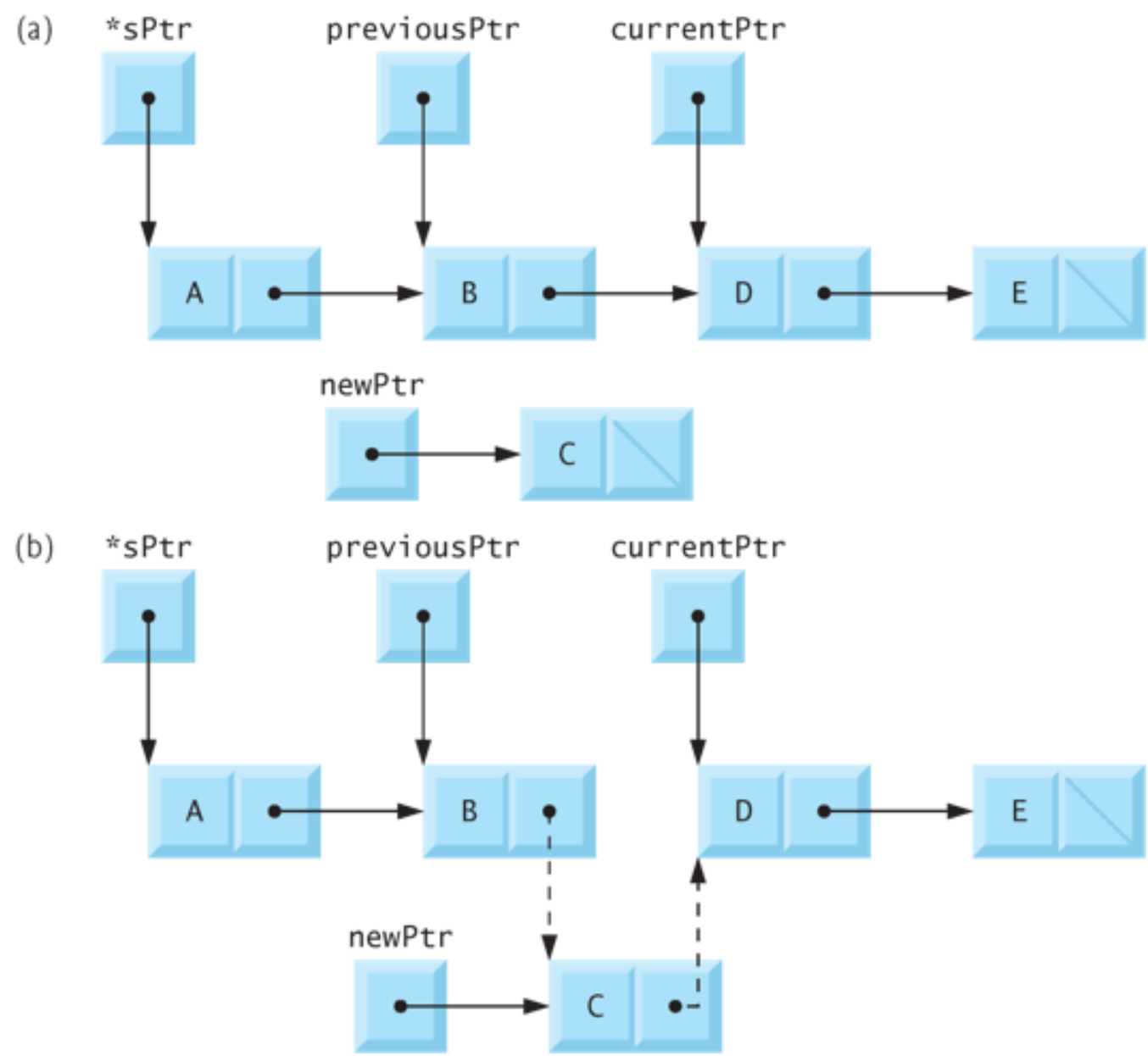


Fig. 12.5 | Inserting a node in order in a list.

Linked Lists

Error-Prevention Tip 12.2

Assign NULL *to the link member of a new node. Pointers should be initialized before they are used.*

Linked Lists

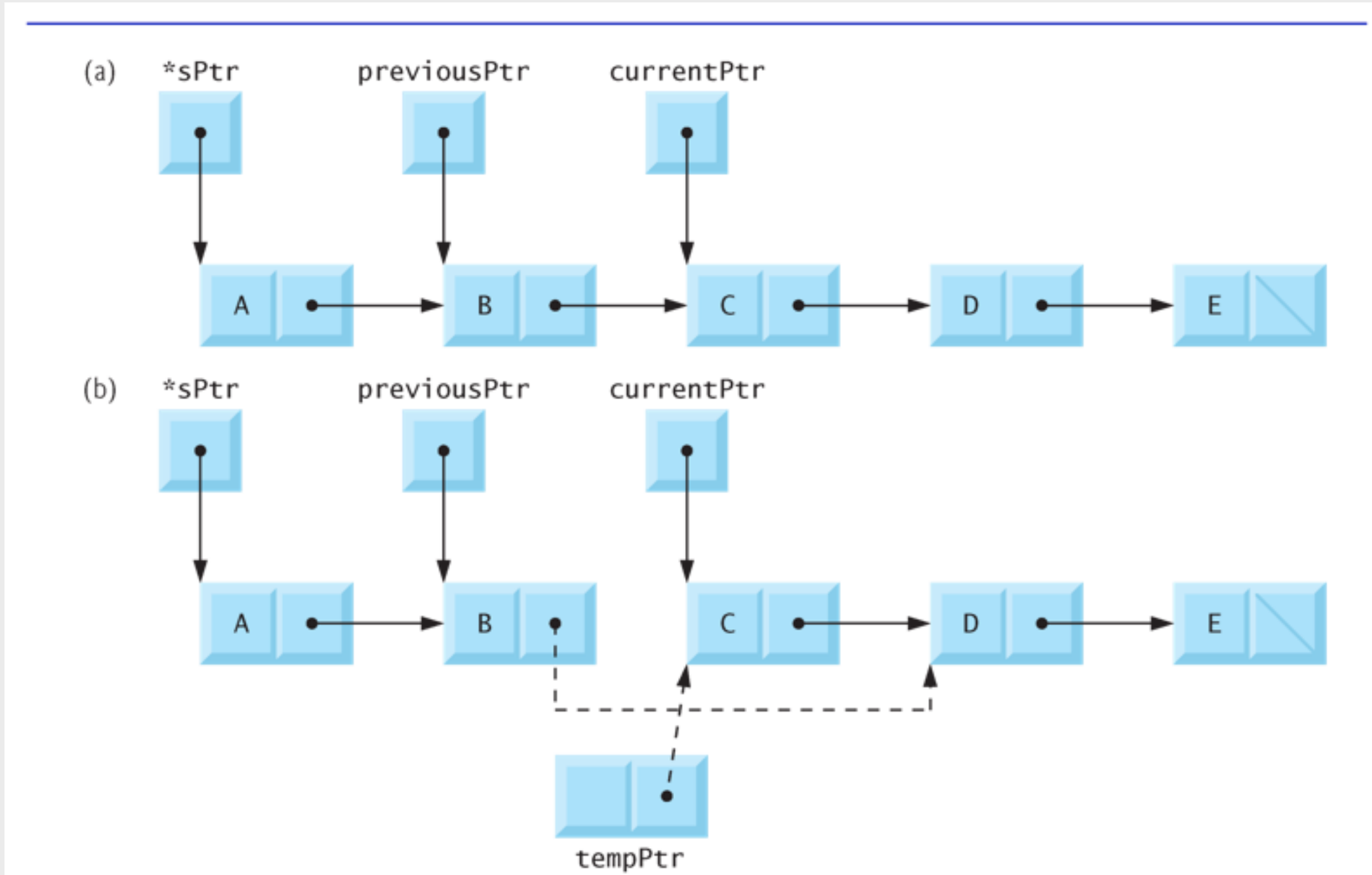


Fig. 12.6 | Deleting a node from a list.

Stacks

- A stack is a constrained version of a linked list.
- New nodes can be added to a stack and removed from a stack only at the top.
- For this reason, a stack is referred to as a last-in, first-out (LIFO) data structure.
- A stack is referenced via a pointer to the top element of the stack.
- The link member in the last node of the stack is set to NULL to indicate the bottom of the stack.

Stacks

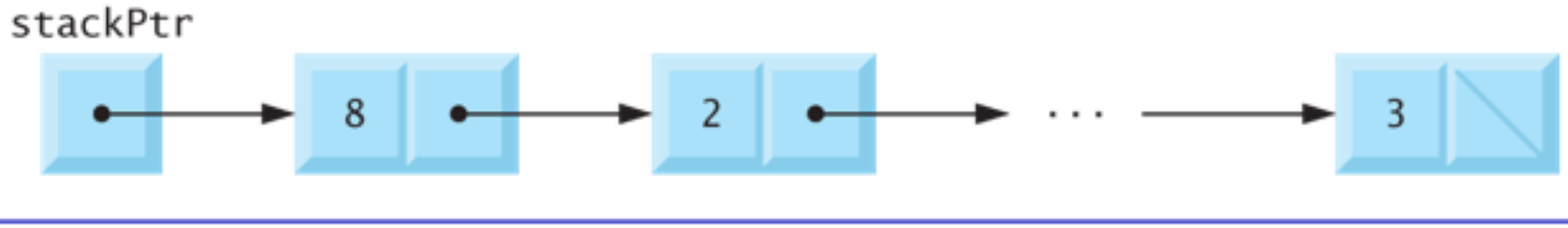


Fig. 12.7 | Stack graphical representation.

Stacks

- The primary functions used to manipulate a stack are push and pop.
- Function push creates a new node and places it on top of the stack.
- Function pop removes a node from the top of the stack, frees the memory that was allocated to the popped node and returns the popped value.

Stacks

- Example: fig12_08.c

```
 7 struct stackNode {···
 8     int data; /* define data as an int */
 9     struct stackNode *nextPtr; /* stackNode pointer */
10 }; /* end structure stackNode */
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 );
```

Stacks

- Example: fig12_08.c

```c
struct stackNode {
    int data; /* define data as an int */
    struct stackNode *nextPtr; /* stackNode pointer */
}; /* end structure stackNode */

typedef struct stackNode StackNode; /* synonym for struct stackNode */
typedef StackNode *StackNodePtr; /* synonym for StackNode* */

/* prototypes */
void push( StackNodePtr *topPtr, int info );
int pop( StackNodePtr *topPtr );
int isEmpty( StackNodePtr topPtr );
void printStack( StackNodePtr currentPtr );
void instructions( void );
```

Stacks

- Example: fig12_08.c

```
 7 struct stackNode {···
 8     int data; /* define data as an int */
 9     struct stackNode *nextPtr; /* stackNode pointer */
10 }; /* end structure stackNode */
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 );
```

Stacks

- Example: fig12_08.c

```
23 int main( void )
24 {
25     StackNodePtr stackPtr = NULL; /* points to stack top */
26     int choice; /* user's menu choice */
27     int value; /* int input by user */
28
29     instructions(); /* display the menu */
30     printf( "? " );
31     scanf( "%d", &choice );
32
33     /* while user does not enter 3 */
34     while ( choice != 3 ) {
35
36         switch ( choice ) {
37             /* push value onto stack */
38             case 1:
39                 printf( "Enter an integer: " );
40                 scanf( "%d", &value );
41                 push( &stackPtr, value );
42                 printStack( stackPtr );
43                 break;
```

Stacks

- Example: fig12_08.c

```
23 int main( void )
24 {
25     StackNodePtr stackPtr = NULL; /* points to stack top */
26     int choice; /* user's menu choice */
27     int value; /* int input by user */
28
29     instructions(); /* display the menu */
30     printf( "? " );
31     scanf( "%d", &choice );
32
33     /* while user does not enter 3 */
34     while ( choice != 3 ) {
35
36         switch ( choice ) {
37             /* push value onto stack */
38             case 1:
39                 printf( "Enter an integer: " );
40                 scanf( "%d", &value );
41                 push( &stackPtr, value );
42                 printStack( stackPtr );
43                 break;
```

Stacks

- Example: fig12_08.c

```
23 int main( void )
24 {
25     StackNodePtr stackPtr = NULL; /* points to stack top */
26     int choice; /* user's menu choice */
27     int value; /* int input by user */
28
29     instructions(); /* display the menu */
30     printf( "? " );
31     scanf( "%d", &choice );
32
33     /* while user does not enter 3 */
34     while ( choice != 3 ) {
35
36         switch ( choice ) {
37             /* push value onto stack */
38             case 1:
39                 printf( "Enter an integer: " );
40                 scanf( "%d", &value );
41                 push( &stackPtr, value );
42                 printStack( stackPtr );
43                 break;
```

Stacks

- Example: fig12_08.c

```
45              case 2:......
46                  /* if stack is not empty */
47                  if ( !isEmpty( stackPtr ) ) {
48                      printf( "The popped value is %d.\n", pop( &stackPtr ) );
49                  } /* end if */
50
51                  printStack( stackPtr );
52                  break;
53              default:
54                  printf( "Invalid choice.\n\n" );
55                  instructions();
56                  break;
57          } /* end switch */
58
59          printf( "? " );
60          scanf( "%d", &choice );
61      } /* end while */
62
63      printf( "End of run.\n" );
64      return 0; /* indicates successful termination */
65  } /* end main */
```

Stacks

- Example: fig12_08.c

```
45              case 2:......
46                  /* if stack is not empty */
47                  if ( !isEmpty( stackPtr ) ) {
48                      printf( "The popped value is %d.\n", pop( &stackPtr ) );
49                  } /* end if */
50
51                  printStack( stackPtr );
52                  break;
53              default:
54                  printf( "Invalid choice.\n\n" );
55                  instructions();
56                  break;
57          } /* end switch */
58
59          printf( "? " );
60          scanf( "%d", &choice );
61      } /* end while */
62
63      printf( "End of run.\n" );
64      return 0; /* indicates successful termination */
65  } /* end main */
```

Stacks

- Example: fig12_08.c

```
75 void push( StackNodePtr *topPtr, int info ) {
76     StackNodePtr newPtr; /* pointer to new node */
77
78     newPtr = malloc( sizeof( StackNode ) );
79
80     /* insert the node at stack top */
81     if ( newPtr != NULL ) {
82         newPtr->data = info;
83         newPtr->nextPtr = *topPtr;
84         *topPtr = newPtr;
85     } /* end if */
86     else { /* no space available */
87         printf( "%d not inserted. No memory available.\n", info );
88     } /* end else */
89 } /* end function push */
```

Stacks

- Example: fig12_08.c

```
75 void push( StackNodePtr *topPtr, int info ) {
76     StackNodePtr newPtr; /* pointer to new node */
77
78     newPtr = malloc( sizeof( StackNode ) );
79
80     /* insert the node at stack top */
81     if ( newPtr != NULL ) {···
82         newPtr->data = info;
83         newPtr->nextPtr = *topPtr;
84         *topPtr = newPtr;
85     } /* end if */
86     else { /* no space available */
87         printf( "%d not inserted. No memory available.\n", info );
88     } /* end else */
89 } /* end function push */
```

Stacks

- Example: fig12_08.c

```
75 void push( StackNodePtr *topPtr, int info ) {
76     StackNodePtr newPtr; /* pointer to new node */
77
78     newPtr = malloc( sizeof( StackNode ) );
79
80     /* insert the node at stack top */
81     if ( newPtr != NULL ) {
82         newPtr->data = info;
83         newPtr->nextPtr = *topPtr;
84         *topPtr = newPtr;
85     } /* end if */
86     else { /* no space available */
87         printf( "%d not inserted. No memory available.\n", info );
88     } /* end else */
89 } /* end function push */
```

Stacks

- Example: fig12_08.c

```
92  int pop( StackNodePtr *topPtr ) {
93      StackNodePtr tempPtr; /* temporary node pointer */
94      int popValue; /* node value */
95
96      tempPtr = *topPtr;
97      popValue = ( *topPtr )->data;
98      *topPtr = ( *topPtr )->nextPtr;
99      free( tempPtr );
100     return popValue;
101 } /* end function pop */
```

Stacks

- Example: fig12_08.c

```
92  int pop( StackNodePtr *topPtr ) {
93      StackNodePtr tempPtr; /* temporary node pointer */
94      int popValue; /* node value */
95
96      tempPtr = *topPtr;
97      popValue = ( *topPtr )->data;
98      *topPtr = ( *topPtr )->nextPtr;
99      free( tempPtr );
100     return popValue;
101 } /* end function pop */
```

Stacks

- Example: fig12_08.c

```
92  int pop( StackNodePtr *topPtr ) {
93     StackNodePtr tempPtr; /* temporary node pointer */
94     int popValue; /* node value */
95
96     tempPtr = *topPtr;
97     popValue = ( *topPtr )->data;
98     *topPtr = ( *topPtr )->nextPtr;
99     free( tempPtr );
100    return popValue;
101 } /* end function pop */
```

Stacks

- Example: fig12_08.c

```
104 void printStack( StackNodePtr currentPtr ) {
105     /* if stack is empty */
106     if ( currentPtr == NULL ) {
107         printf( "The stack is empty.\n\n" );
108     } /* end if */
109     else {
110         printf( "The stack is:\n" );
111 
112         /* while not the end of the stack */
113         while ( currentPtr != NULL ) {
114             printf( "%d --> ", currentPtr->data );
115             currentPtr = currentPtr->nextPtr;
116         } /* end while */
117 
118         printf( "NULL\n\n" );
119     } /* end else */
120 } /* end function printList */
```

```
123 int isEmpty( StackNodePtr topPtr ) {
124     return topPtr == NULL;
125 } /* end function isEmpty */
```

Stacks

- Example: fig12_08.c

```
104 void printStack( StackNodePtr currentPtr ) {
105     /* if stack is empty */
106     if ( currentPtr == NULL ) {
107         printf( "The stack is empty.\n\n" );
108     } /* end if */
109     else {
110         printf( "The stack is:\n" );
111 
112         /* while not the end of the stack */
113         while ( currentPtr != NULL ) {
114             printf( "%d --> ", currentPtr->data );
115             currentPtr = currentPtr->nextPtr;
116         } /* end while */
117 
118         printf( "NULL\n\n" );
119     } /* end else */
120 } /* end function printList */
```

```
123 int isEmpty( StackNodePtr topPtr ) {
124     return topPtr == NULL;
125 } /* end function isEmpty */
```

Stacks

- Example: fig12_08.c

```
104 void printStack( StackNodePtr currentPtr ) {
105     /* if stack is empty */
106     if ( currentPtr == NULL ) {
107         printf( "The stack is empty.\n\n" );
108     } /* end if */
109     else {
110         printf( "The stack is:\n" );
111 
112         /* while not the end of the stack */
113         while ( currentPtr != NULL ) {
114             printf( "%d --> ", currentPtr->data );
115             currentPtr = currentPtr->nextPtr;
116         } /* end while */
117 
118         printf( "NULL\n\n" );
119     } /* end else */
120 } /* end function printList */
```

```
123 int isEmpty( StackNodePtr topPtr ) {
124     return topPtr == NULL;
125 } /* end function isEmpty */
```

Stacks

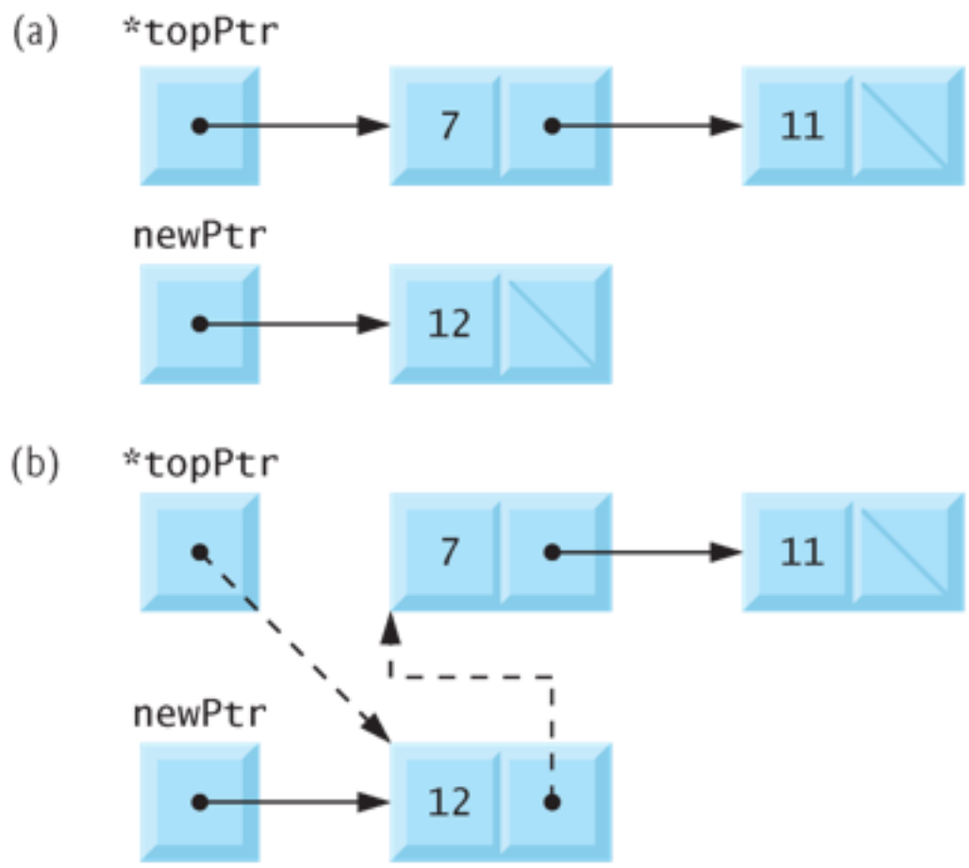


Fig. 12.10 | `push` operation.

Stacks

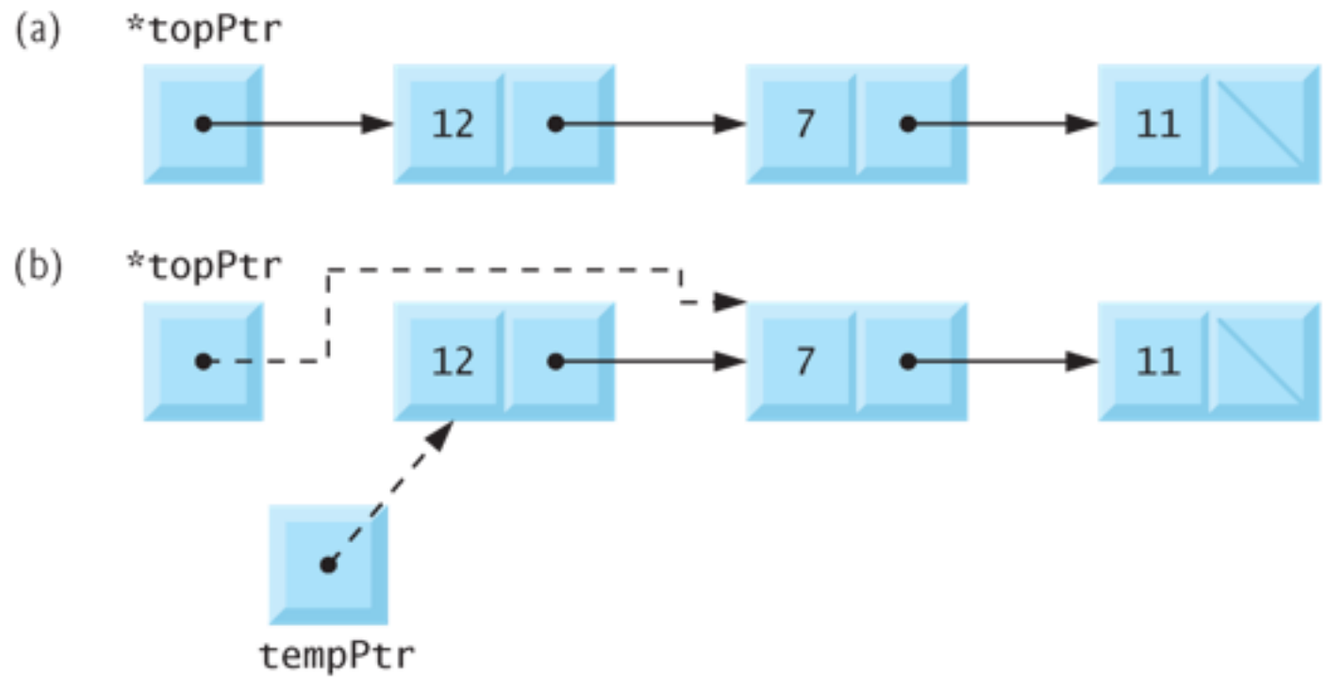


Fig. 12.11 | pop operation.

Stacks

- Stacks have many interesting applications.
- For example, whenever a function call is made, the called function must know how to return to its caller, so the return address is pushed onto a stack.
- If a series of function calls occurs, the successive return values are pushed onto the stack in last-in, first-out order so that each function can return to its caller.