Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Basic Data Structures

Queue

- Another common data structure is the queue.
- A queue is similar to a checkout line in a grocery store—the first person in line is serviced first, and other customers enter the line only at the end and wait to be serviced.
- Queue nodes are removed only from the head of the queue and are inserted only at the tail of the queue.
- For this reason, a queue is referred to as a first-in, first-out (FIFO) data structure.
- The insert and remove operations are known as enqueue and dequeue.

Queue

Common Programming Error 12.7

Not setting the link in the last node of a queue to NULL can lead to runtime errors.

Queue

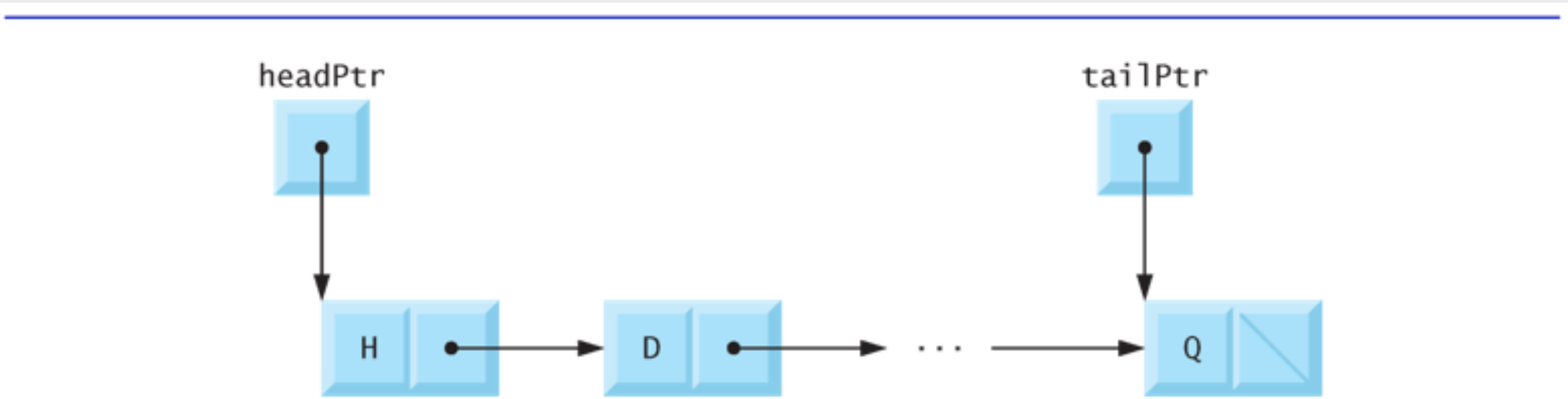

Fig. 12.12 | Queue graphical representation.

Queue

- Example: fig12_13.c

```
 7 struct queueNode {···
 8     char data; /* define data as a char */
 9     struct queueNode *nextPtr; /* queueNode pointer */
10 }; /* end structure queueNode */
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 char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr );
19 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value );
20 void instructions( void );
```

Queue

- Example: fig12_13.c

```c
struct queueNode {
    char data; /* define data as a char */
    struct queueNode *nextPtr; /* queueNode pointer */
}; /* end structure queueNode */

typedef struct queueNode QueueNode;
typedef QueueNode *QueueNodePtr;

/* function prototypes */
void printQueue( QueueNodePtr currentPtr );
int isEmpty( QueueNodePtr headPtr );
char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr );
void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value );
void instructions( void );
```

Queue

- Example: fig12_13.c

```c
struct queueNode {
    char data; /* define data as a char */
    struct queueNode *nextPtr; /* queueNode pointer */
}; /* end structure queueNode */

typedef struct queueNode QueueNode;
typedef QueueNode *QueueNodePtr;

/* function prototypes */
void printQueue( QueueNodePtr currentPtr );
int isEmpty( QueueNodePtr headPtr );
char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr );
void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value );
void instructions( void );
```

Queue

- Example: fig12_13.c

```c
struct queueNode {
    char data; /* define data as a char */
    struct queueNode *nextPtr; /* queueNode pointer */
}; /* end structure queueNode */

typedef struct queueNode QueueNode;
typedef QueueNode *QueueNodePtr;

/* function prototypes */
void printQueue( QueueNodePtr currentPtr );
int isEmpty( QueueNodePtr headPtr );
char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr );
void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value );
void instructions( void );
```

Queue

- Example: fig12_13.c

```
23 int main( void ) {
24     QueueNodePtr headPtr = NULL; /* initialize headPtr */
25     QueueNodePtr tailPtr = NULL; /* initialize tailPtr */
26     int choice; /* user's menu choice */
27     char item; /* char 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             /* enqueue value */
38             case 1:
39                 printf( "Enter a character: " );
40                 scanf( "\n%c", &item );
41                 enqueue( &headPtr, &tailPtr, item );
42                 printQueue( headPtr );
43                 break;
```

Queue

- Example: fig12_13.c

```
23 int main( void ) {
24     QueueNodePtr headPtr = NULL; /* initialize headPtr */
25     QueueNodePtr tailPtr = NULL; /* initialize tailPtr */
26     int choice; /* user's menu choice */
27     char item; /* char 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             /* enqueue value */
38             case 1:
39                 printf( "Enter a character: " );
40                 scanf( "\n%c", &item );
41                 enqueue( &headPtr, &tailPtr, item );
42                 printQueue( headPtr );
43                 break;
```

Queue

- Example: fig12_13.c

```
23 int main( void ) {
24     QueueNodePtr headPtr = NULL; /* initialize headPtr */
25     QueueNodePtr tailPtr = NULL; /* initialize tailPtr */
26     int choice; /* user's menu choice */
27     char item; /* char 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             /* enqueue value */
38             case 1:
39                 printf( "Enter a character: " );
40                 scanf( "\n%c", &item );
41                 enqueue( &headPtr, &tailPtr, item );
42                 printQueue( headPtr );
43                 break;
```

Queue

- Example: fig12_13.c

```
45              case 2:
46                  /* if queue is not empty */
47                  if ( !isEmpty( headPtr ) ) {
48                      item = dequeue( &headPtr, &tailPtr );
49                      printf( "%c has been dequeued.\n", item );
50                  } /* end if */
51
52                  printQueue( headPtr );
53                  break;
54              default:
55                  printf( "Invalid choice.\n\n" );
56                  instructions();
57                  break;
58          } /* end switch */
59
60          printf( "? " );
61          scanf( "%d", &choice );
62      } /* end while */
63
64      printf( "End of run.\n" );
65      return 0; /* indicates successful termination */
66  } /* end main */
```

Queue

- Example: fig12_13.c

```
45              case 2:
46                  /* if queue is not empty */
47                  if ( !isEmpty( headPtr ) ) {
48                      item = dequeue( &headPtr, &tailPtr );
49                      printf( "%c has been dequeued.\n", item );
50                  } /* end if */
51
52                  printQueue( headPtr );
53                  break;
54              default:
55                  printf( "Invalid choice.\n\n" );
56                  instructions();
57                  break;
58          } /* end switch */
59
60          printf( "? " );
61          scanf( "%d", &choice );
62      } /* end while */
63
64      printf( "End of run.\n" );
65      return 0; /* indicates successful termination */
66  } /* end main */
```

Queue

- Example: fig12_13.c

```
77 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value ) {
78     QueueNodePtr newPtr; /* pointer to new node */
79
80     newPtr = malloc( sizeof( QueueNode ) );
81
82     if ( newPtr != NULL ) { /* is space available */
83         newPtr->data = value;
84         newPtr->nextPtr = NULL;
85
86         /* if empty, insert node at head */
87         if ( isEmpty( *headPtr ) ) {
88             *headPtr = newPtr;
89         } /* end if */
90         else {
91             ( *tailPtr )->nextPtr = newPtr;
92         } /* end else */
93
94         *tailPtr = newPtr;
95     } /* end if */
96     else {
97         printf( "%c not inserted. No memory available.\n", value );
98     } /* end else */
99 } /* end function enqueue */
```

Queue

- Example: fig12_13.c

```
77 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value ) {
78     QueueNodePtr newPtr; /* pointer to new node */
79
80     newPtr = malloc( sizeof( QueueNode ) );
81
82     if ( newPtr != NULL ) { /* is space available */
83         newPtr->data = value;
84         newPtr->nextPtr = NULL;
85
86         /* if empty, insert node at head */
87         if ( isEmpty( *headPtr ) ) {
88             *headPtr = newPtr;
89         } /* end if */
90         else {
91             ( *tailPtr )->nextPtr = newPtr;
92         } /* end else */
93
94         *tailPtr = newPtr;
95     } /* end if */
96     else {
97         printf( "%c not inserted. No memory available.\n", value );
98     } /* end else */
99 } /* end function enqueue */
```

Queue

- Example: fig12_13.c

```
77 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value ) {
78     QueueNodePtr newPtr; /* pointer to new node */
79
80     newPtr = malloc( sizeof( QueueNode ) );
81
82     if ( newPtr != NULL ) { /* is space available */
83         newPtr->data = value;
84         newPtr->nextPtr = NULL;
85
86         /* if empty, insert node at head */
87         if ( isEmpty( *headPtr ) ) {
88             *headPtr = newPtr;
89         } /* end if */
90         else {
91             ( *tailPtr )->nextPtr = newPtr;
92         } /* end else */
93
94         *tailPtr = newPtr;
95     } /* end if */
96     else {
97         printf( "%c not inserted. No memory available.\n", value );
98     } /* end else */
99 } /* end function enqueue */
```

Queue

- Example: fig12_13.c

```
77 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value ) {
78     QueueNodePtr newPtr; /* pointer to new node */
79
80     newPtr = malloc( sizeof( QueueNode ) );
81
82     if ( newPtr != NULL ) { /* is space available */
83         newPtr->data = value;
84         newPtr->nextPtr = NULL;
85
86         /* if empty, insert node at head */
87         if ( isEmpty( *headPtr ) ) {
88             *headPtr = newPtr;
89         } /* end if */
90         else {
91             ( *tailPtr )->nextPtr = newPtr;
92         } /* end else */
93
94         *tailPtr = newPtr;
95     } /* end if */
96     else {
97         printf( "%c not inserted. No memory available.\n", value );
98     } /* end else */
99 } /* end function enqueue */
```

Queue

- Example: fig12_13.c

```
77 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value ) {
78     QueueNodePtr newPtr; /* pointer to new node */
79
80     newPtr = malloc( sizeof( QueueNode ) );
81
82     if ( newPtr != NULL ) { /* is space available */
83         newPtr->data = value;
84         newPtr->nextPtr = NULL;
85
86         /* if empty, insert node at head */
87         if ( isEmpty( *headPtr ) ) {
88             *headPtr = newPtr;
89         } /* end if */
90         else {
91             ( *tailPtr )->nextPtr = newPtr;
92         } /* end else */
93
94         *tailPtr = newPtr;
95     } /* end if */
96     else {
97         printf( "%c not inserted. No memory available.\n", value );
98     } /* end else */
99 } /* end function enqueue */
```

Queue

- Example: fig12_13.c

```
102 char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr ) {
103     char value; /* node value */
104     QueueNodePtr tempPtr; /* temporary node pointer */
105 
106     value = ( *headPtr )->data;
107     tempPtr = *headPtr;
108     *headPtr = ( *headPtr )->nextPtr;
109 
110     /* if queue is empty */
111     if ( *headPtr == NULL ) {
112         *tailPtr = NULL;
113     } /* end if */
114 
115     free( tempPtr );
116     return value;
117 } /* end function dequeue */
```

Queue

- Example: fig12_13.c

```
102 char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr ) {
103     char value; /* node value */
104     QueueNodePtr tempPtr; /* temporary node pointer */
105
106     value = ( *headPtr )->data;
107     tempPtr = *headPtr;
108     *headPtr = ( *headPtr )->nextPtr;
109
110     /* if queue is empty */
111     if ( *headPtr == NULL ) {
112         *tailPtr = NULL;
113     } /* end if */
114
115     free( tempPtr );
116     return value;
117 } /* end function dequeue */
```

Queue

- Example: fig12_13.c

```
120 int isEmpty( QueueNodePtr headPtr ) {
121     return headPtr == NULL;
122 } /* end function isEmpty */
```

```
125 void printQueue( QueueNodePtr currentPtr ) {
126     /* if queue is empty */
127     if ( currentPtr == NULL ) {
128         printf( "Queue is empty.\n\n" );
129     } /* end if */
130     else {
131         printf( "The queue is:\n" );
132 
133         /* while not end of queue */
134         while ( currentPtr != NULL ) {
135             printf( "%c --> ", currentPtr->data );
136             currentPtr = currentPtr->nextPtr;
137         } /* end while */
138 
139         printf( "NULL\n\n" );
140     } /* end else */
141 } /* end function printQueue */
```

Queue

- Example: fig12_13.c

```
120 int isEmpty( QueueNodePtr headPtr ) {
121    return headPtr == NULL;
122 } /* end function isEmpty */
```

```
125 void printQueue( QueueNodePtr currentPtr ) {
126    /* if queue is empty */
127    if ( currentPtr == NULL ) {
128       printf( "Queue is empty.\n\n" );
129    } /* end if */
130    else {
131       printf( "The queue is:\n" );
132
133       /* while not end of queue */
134       while ( currentPtr != NULL ) {
135          printf( "%c --> ", currentPtr->data );
136          currentPtr = currentPtr->nextPtr;
137       } /* end while */
138
139       printf( "NULL\n\n" );
140    } /* end else */
141 } /* end function printQueue */
```

Queue

- Example: fig12_13.c

```
120 int isEmpty( QueueNodePtr headPtr ) {
121    return headPtr == NULL;
122 } /* end function isEmpty */
```

```
125 void printQueue( QueueNodePtr currentPtr ) {
126    /* if queue is empty */
127    if ( currentPtr == NULL ) {
128       printf( "Queue is empty.\n\n" );
129    } /* end if */
130    else {
131       printf( "The queue is:\n" );
132
133       /* while not end of queue */
134       while ( currentPtr != NULL ) {
135          printf( "%c --> ", currentPtr->data );
136          currentPtr = currentPtr->nextPtr;
137       } /* end while */
138
139       printf( "NULL\n\n" );
140    } /* end else */
141 } /* end function printQueue */
```

Queue

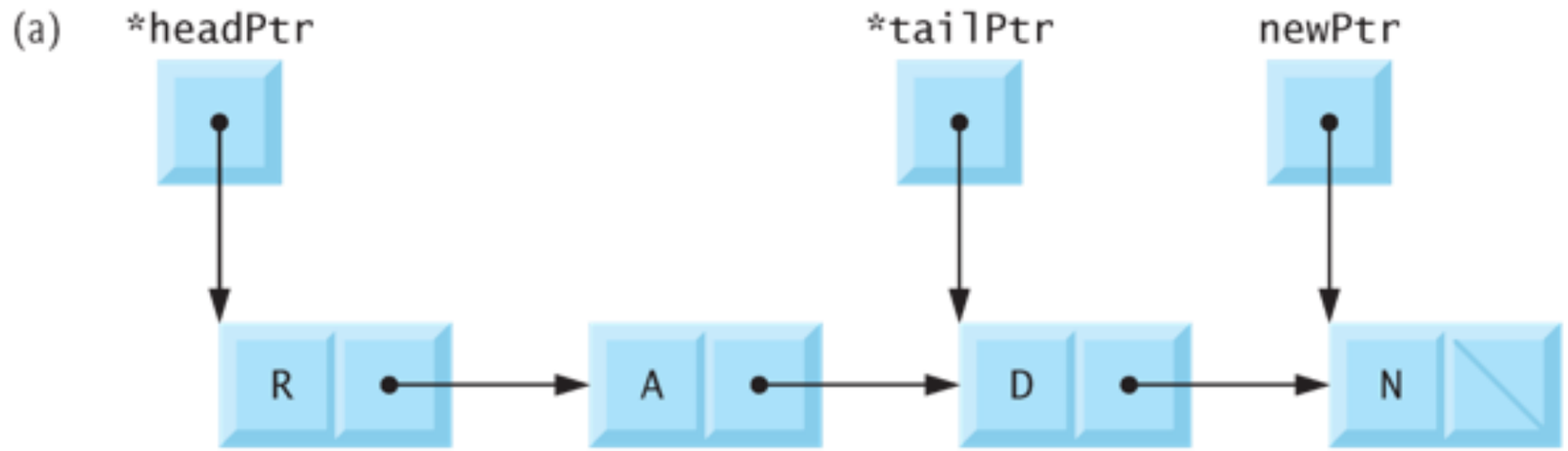

Fig. 12.15 | enqueue operation.

Queue

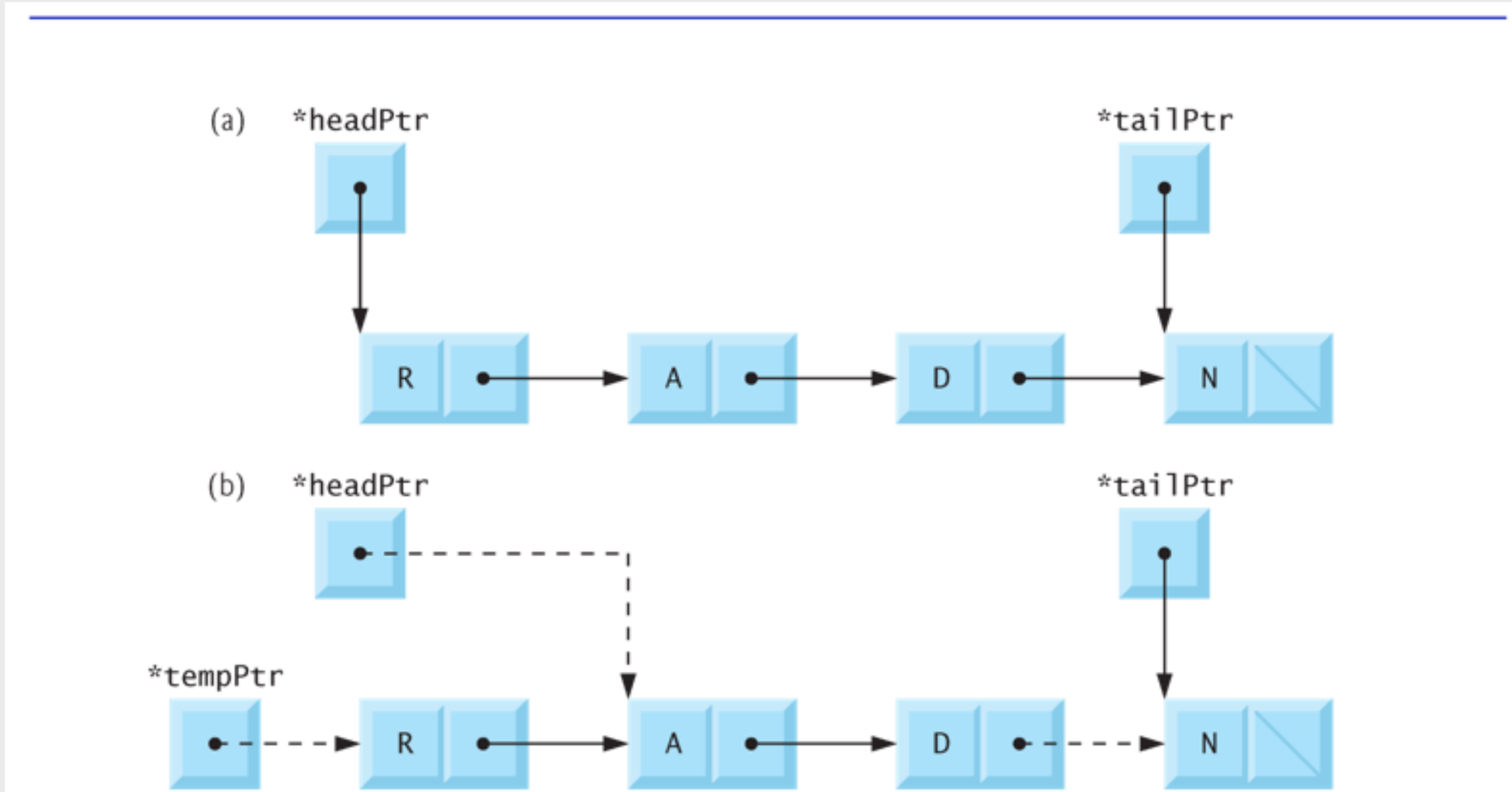

Fig. 12.16 | dequeue operation.

Trees

- Linked lists, stacks and queues are linear data structures.
- A tree is a nonlinear, two-dimensional data structure with special properties.
- Tree nodes contain two or more links.

Trees

- This section discusses binary trees—trees whose nodes all contain two links (none, one, or both of which may be NULL).
- The root node is the first node in a tree.
- Each link in the root node refers to a child.
- The left child is the first node in the left subtree, and the right child is the first node in the right subtree.
- The children of a node are called siblings.
- A node with no children is called a leaf node.

Trees

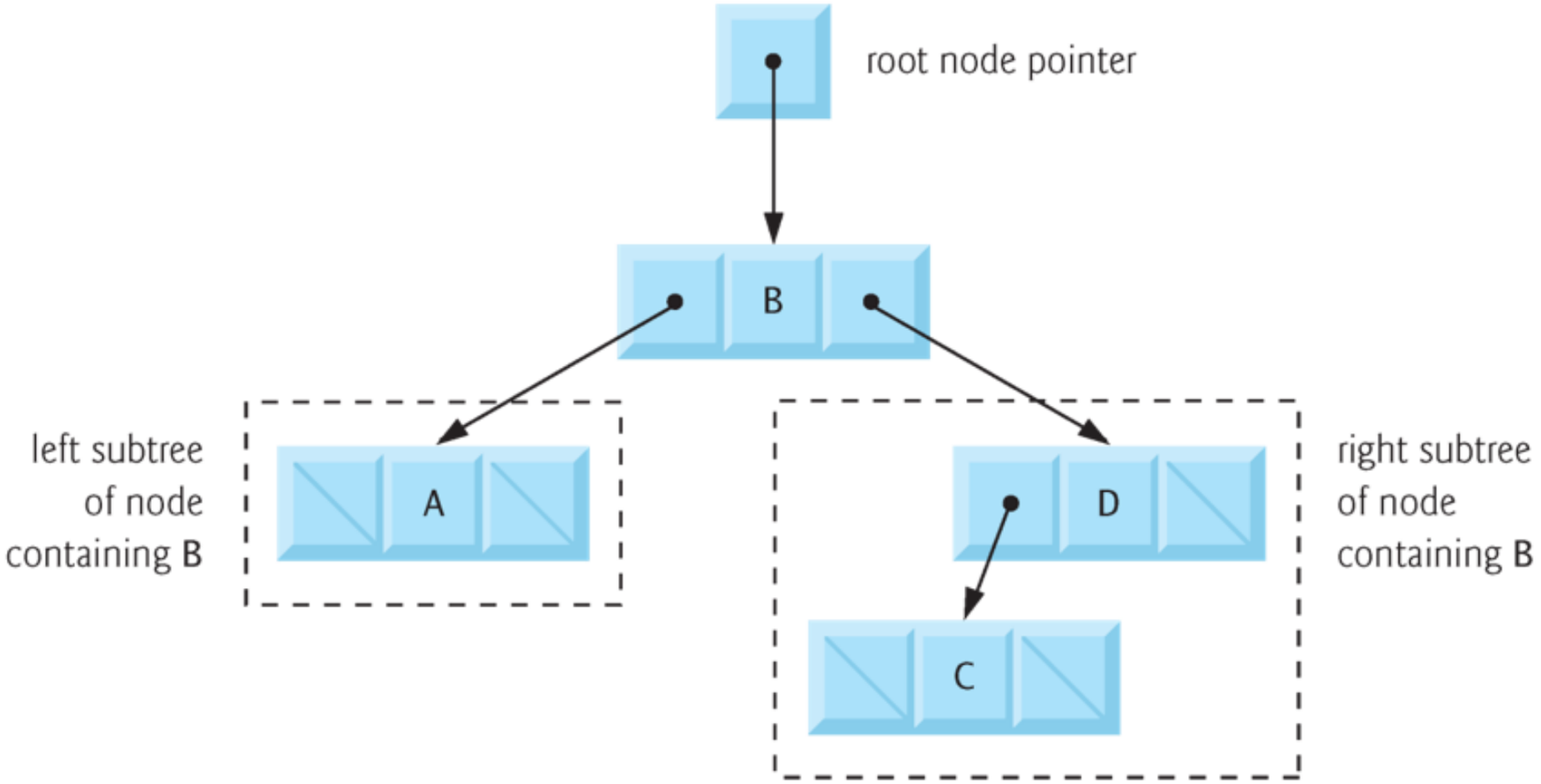

Fig. 12.17 | Binary tree graphical representation.

Trees

- In this section, a special binary tree called a binary search tree is created.
- A binary search tree (with no duplicate node values) has the characteristic that the values in any left subtree are less than the value in its parent node, and the values in any right subtree are greater than the value in its parent node.

Trees

Common Programming Error 12.8

Not setting to NULL the links in leaf nodes of a tree can lead to runtime errors.

Trees

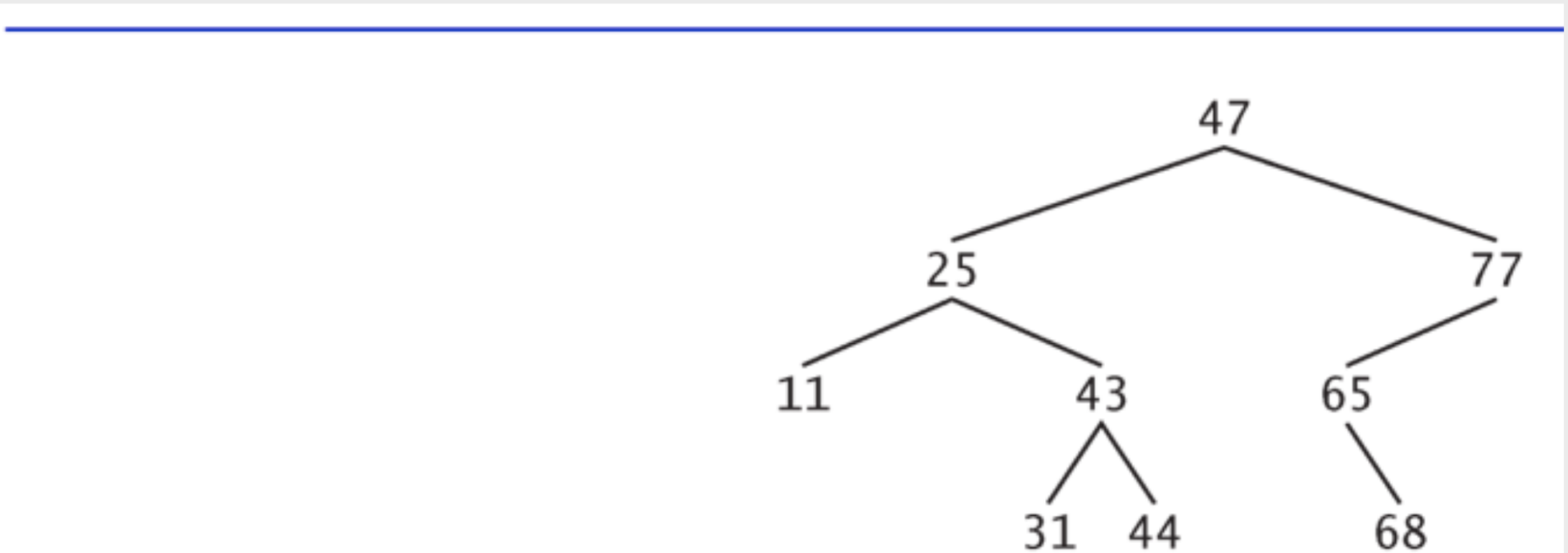

Fig. 12.18 | Binary search tree.

Trees

- Below we show a binary search tree and traverse it in three ways—inorder, preorder and postorder.

Trees

- Example: fig12_19.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* self-referential structure */
struct treeNode {
   struct treeNode *leftPtr; /* pointer to left subtree */
   int data; /* node value */
   struct treeNode *rightPtr; /* pointer to right subtree */
}; /* end structure treeNode */

typedef struct treeNode TreeNode; /* synonym for struct treeNode */
typedef TreeNode *TreeNodePtr; /* synonym for TreeNode* */

/* prototypes */
void insertNode( TreeNodePtr *treePtr, int value );
void inOrder( TreeNodePtr treePtr );
void preOrder( TreeNodePtr treePtr );
void postOrder( TreeNodePtr treePtr );
```

Trees

- Example: fig12_19.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* self-referential structure */
struct treeNode {
   struct treeNode *leftPtr; /* pointer to left subtree */
   int data; /* node value */
   struct treeNode *rightPtr; /* pointer to right subtree */
}; /* end structure treeNode */

typedef struct treeNode TreeNode; /* synonym for struct treeNode */
typedef TreeNode *TreeNodePtr; /* synonym for TreeNode* */

/* prototypes */
void insertNode( TreeNodePtr *treePtr, int value );
void inOrder( TreeNodePtr treePtr );
void preOrder( TreeNodePtr treePtr );
void postOrder( TreeNodePtr treePtr );
```

Trees

- Example: fig12_19.c

```
25 int main( void ) {
26    int i; /* counter to loop from 1-10 */
27    int item; /* variable to hold random values */
28    TreeNodePtr rootPtr = NULL; /* tree initially empty */
29
30    srand( time( NULL ) );
31    printf( "The numbers being placed in the tree are:\n" );
32
33    /* insert random values between 0 and 14 in the tree */
34    for ( i = 1; i <= 10; i++ ) {
35       item = rand() % 15;
36       printf( "%3d", item );
37       insertNode( &rootPtr, item );
38    } /* end for */
39
40    /* traverse the tree preOrder */
41    printf( "\n\nThe preOrder traversal is:\n" );
42    preOrder( rootPtr );
43
44    /* traverse the tree inOrder */
45    printf( "\n\nThe inOrder traversal is:\n" );
46    inOrder( rootPtr );
```

Trees

- Example: fig12_19.c

```
25 int main( void ) {
26    int i; /* counter to loop from 1-10 */
27    int item; /* variable to hold random values */
28    TreeNodePtr rootPtr = NULL; /* tree initially empty */
29
30    srand( time( NULL ) );
31    printf( "The numbers being placed in the tree are:\n" );
32
33    /* insert random values between 0 and 14 in the tree */
34    for ( i = 1; i <= 10; i++ ) {
35       item = rand() % 15;
36       printf( "%3d", item );
37       insertNode( &rootPtr, item );
38    } /* end for */
39
40    /* traverse the tree preOrder */
41    printf( "\n\nThe preOrder traversal is:\n" );
42    preOrder( rootPtr );
43
44    /* traverse the tree inOrder */
45    printf( "\n\nThe inOrder traversal is:\n" );
46    inOrder( rootPtr );
```

Trees

- Example: fig12_19.c

```
25 int main( void ) {
26    int i; /* counter to loop from 1-10 */
27    int item; /* variable to hold random values */
28    TreeNodePtr rootPtr = NULL; /* tree initially empty */
29
30    srand( time( NULL ) );
31    printf( "The numbers being placed in the tree are:\n" );
32
33    /* insert random values between 0 and 14 in the tree */
34    for ( i = 1; i <= 10; i++ ) {
35       item = rand() % 15;
36       printf( "%3d", item );
37       insertNode( &rootPtr, item );
38    } /* end for */
39
40    /* traverse the tree preOrder */
41    printf( "\n\nThe preOrder traversal is:\n" );
42    preOrder( rootPtr );
43
44    /* traverse the tree inOrder */
45    printf( "\n\nThe inOrder traversal is:\n" );
46    inOrder( rootPtr );
```

Trees

- Example: fig12_19.c

```
25 int main( void ) {
26    int i; /* counter to loop from 1-10 */
27    int item; /* variable to hold random values */
28    TreeNodePtr rootPtr = NULL; /* tree initially empty */
29
30    srand( time( NULL ) );
31    printf( "The numbers being placed in the tree are:\n" );
32
33    /* insert random values between 0 and 14 in the tree */
34    for ( i = 1; i <= 10; i++ ) {
35       item = rand() % 15;
36       printf( "%3d", item );
37       insertNode( &rootPtr, item );
38    } /* end for */
39
40    /* traverse the tree preOrder */
41    printf( "\n\nThe preOrder traversal is:\n" );
42    preOrder( rootPtr );
43
44    /* traverse the tree inOrder */
45    printf( "\n\nThe inOrder traversal is:\n" );
46    inOrder( rootPtr );
```

Trees

- Example: fig12_19.c

```
48    /* traverse the tree postOrder */
49    printf( "\n\nThe postOrder traversal is:\n" );
50    postOrder( rootPtr );
51    printf("\n");
52    return 0; /* indicates successful termination */
53 } /* end main */
54
55 /* insert node into tree */
56 void insertNode( TreeNodePtr *treePtr, int value )
57 {
58    /* if tree is empty */
59    if ( *treePtr == NULL ) {...
60       *treePtr = malloc( sizeof( TreeNode ) );
61
62       /* if memory was allocated then assign data */
63       if ( *treePtr != NULL ) {
64          ( *treePtr )->data = value;
65          ( *treePtr )->leftPtr = NULL;
66          ( *treePtr )->rightPtr = NULL;
67       } /* end if */
68       else {
69          printf( "%d not inserted. No memory available.\n", value );
70       } /* end else */
71    } /* end if */
```

Trees

- Example: fig12_19.c

```
48    /* traverse the tree postOrder */
49    printf( "\n\nThe postOrder traversal is:\n" );
50    postOrder( rootPtr );
51    printf("\n");
52    return 0; /* indicates successful termination */
53 } /* end main */
54
55 /* insert node into tree */
56 void insertNode( TreeNodePtr *treePtr, int value )
57 {
58    /* if tree is empty */
59    if ( *treePtr == NULL ) {...
60       *treePtr = malloc( sizeof( TreeNode ) );
61
62       /* if memory was allocated then assign data */
63       if ( *treePtr != NULL ) {
64          ( *treePtr )->data = value;
65          ( *treePtr )->leftPtr = NULL;
66          ( *treePtr )->rightPtr = NULL;
67       } /* end if */
68       else {
69          printf( "%d not inserted. No memory available.\n", value );
70       } /* end else */
71    } /* end if */
```

Trees

- Example: fig12_19.c

```
72    else { /* tree is not empty */
73       /* data to insert is less than data in current node */
74       if ( value < ( *treePtr )->data ) {
75          insertNode( &( ( *treePtr )->leftPtr ), value );
76       } /* end if */
77
78       /* data to insert is greater than data in current node */
79       else if ( value > ( *treePtr )->data ) {
80          insertNode( &( ( *treePtr )->rightPtr ), value );
81       } /* end else if */
82       else { /* duplicate data value ignored */
83          printf( "dup" );
84       } /* end else */
85    } /* end else */
86 } /* end function insertNode */
87
```

Trees

- Example: fig12_19.c

```
72    else { /* tree is not empty */
73       /* data to insert is less than data in current node */
74       if ( value < ( *treePtr )->data ) {
75          insertNode( &( ( *treePtr )->leftPtr ), value );
76       } /* end if */
77
78       /* data to insert is greater than data in current node */
79       else if ( value > ( *treePtr )->data ) {
80          insertNode( &( ( *treePtr )->rightPtr ), value );
81       } /* end else if */
82       else { /* duplicate data value ignored */
83          printf( "dup" );
84       } /* end else */
85    } /* end else */
86 } /* end function insertNode */
87
```

Trees

- Example: fig12_19.c

```
72    else { /* tree is not empty */
73       /* data to insert is less than data in current node */
74       if ( value < ( *treePtr )->data ) {
75          insertNode( &( ( *treePtr )->leftPtr ), value );
76       } /* end if */
77
78       /* data to insert is greater than data in current node */
79       else if ( value > ( *treePtr )->data ) {
80          insertNode( &( ( *treePtr )->rightPtr ), value );
81       } /* end else if */
82       else { /* duplicate data value ignored */
83          printf( "dup" );
84       } /* end else */
85    } /* end else */
86 } /* end function insertNode */
87
```

Trees

- Example: fig12_19.c

```
88 /* begin inorder traversal of tree */
89 void inOrder( TreeNodePtr treePtr )
90 {
91    /* if tree is not empty then traverse */
92    if ( treePtr != NULL ) {
93       inOrder( treePtr->leftPtr );
94       printf( "%3d", treePtr->data );
95       inOrder( treePtr->rightPtr );
96    } /* end if */
97 } /* end function inOrder */
98
99 /* begin preorder traversal of tree */
100 void preOrder( TreeNodePtr treePtr )
101 {
102    /* if tree is not empty then traverse */
103    if ( treePtr != NULL ) {
104       printf( "%3d", treePtr->data );
105       preOrder( treePtr->leftPtr );
106       preOrder( treePtr->rightPtr );
107    } /* end if */
108 } /* end function preOrder */
```

Trees

- Example: fig12_19.c

```
88 /* begin inorder traversal of tree */
89 void inOrder( TreeNodePtr treePtr )
90 {
91    /* if tree is not empty then traverse */
92    if ( treePtr != NULL ) {
93       inOrder( treePtr->leftPtr );
94       printf( "%3d", treePtr->data );
95       inOrder( treePtr->rightPtr );
96    } /* end if */
97 } /* end function inOrder */
98
99 /* begin preorder traversal of tree */
100 void preOrder( TreeNodePtr treePtr )
101 {
102    /* if tree is not empty then traverse */
103    if ( treePtr != NULL ) {
104       printf( "%3d", treePtr->data );
105       preOrder( treePtr->leftPtr );
106       preOrder( treePtr->rightPtr );
107    } /* end if */
108 } /* end function preOrder */
```

Trees

- Example: fig12_19.c

```
88 /* begin inorder traversal of tree */
89 void inOrder( TreeNodePtr treePtr )
90 {
91    /* if tree is not empty then traverse */
92    if ( treePtr != NULL ) {
93       inOrder( treePtr->leftPtr );
94       printf( "%3d", treePtr->data );
95       inOrder( treePtr->rightPtr );
96    } /* end if */
97 } /* end function inOrder */
98
99 /* begin preorder traversal of tree */
100 void preOrder( TreeNodePtr treePtr )
101 {
102    /* if tree is not empty then traverse */
103    if ( treePtr != NULL ) {
104       printf( "%3d", treePtr->data );
105       preOrder( treePtr->leftPtr );
106       preOrder( treePtr->rightPtr );
107    } /* end if */
108 } /* end function preOrder */
```

Trees

- Example: fig12_19.c

```
110 /* begin postorder traversal of tree */
111 void postOrder( TreeNodePtr treePtr )
112 {
113    /* if tree is not empty then traverse */
114    if ( treePtr != NULL ) {
115       postOrder( treePtr->leftPtr );
116       postOrder( treePtr->rightPtr );
117       printf( "%3d", treePtr->data );
118    } /* end if */
119 } /* end function postOrder */
```

Trees

- Example: fig12_19.c

```
110 /* begin postorder traversal of tree */
111 void postOrder( TreeNodePtr treePtr )
112 {
113    /* if tree is not empty then traverse */
114    if ( treePtr != NULL ) {
115       postOrder( treePtr->leftPtr );
116       postOrder( treePtr->rightPtr );
117       printf( "%3d", treePtr->data );
118    } /* end if */
119 } /* end function postOrder */
```

Trees

- Example: fig12_19.c

```
110 /* begin postorder traversal of tree */
111 void postOrder( TreeNodePtr treePtr )
112 {
113    /* if tree is not empty then traverse */
114    if ( treePtr != NULL ) {
115       postOrder( treePtr->leftPtr );
116       postOrder( treePtr->rightPtr );
117       printf( "%3d", treePtr->data );
118    } /* end if */
119 } /* end function postOrder */
```

```
The numbers being placed in the tree are:
 11  8  5 10 10dup  3  0  9 10dup 13

The preOrder traversal is:
 11  8  5  3  0 10  9 13

The inOrder traversal is:
  0  3  5  8  9 10 11 13

The postOrder traversal is:
  0  3  5  9 10  8 13 11
```

Trees

- The steps for an inOrder traversal are:
  - Traverse the left subtree inOrder.
  - Process the value in the node.
  - Traverse the right subtree inOrder.
- The value in a node is not processed until the values in its left subtree are processed.

Trees

- The steps for an preOrder traversal are:
  - Process the value in the node.
  - Traverse the left subtree preOrder.
  - Traverse the right subtree preOrder.
- The value in each node is processed as the node is visited.

Trees

- The steps for a postOrder traversal are:
  - Traverse the left subtree postOrder.
  - Traverse the right subtree postOrder.
  - Process the value in the node.
- The value in each node is not printed until the values of its children are printed.

Trees

- Example

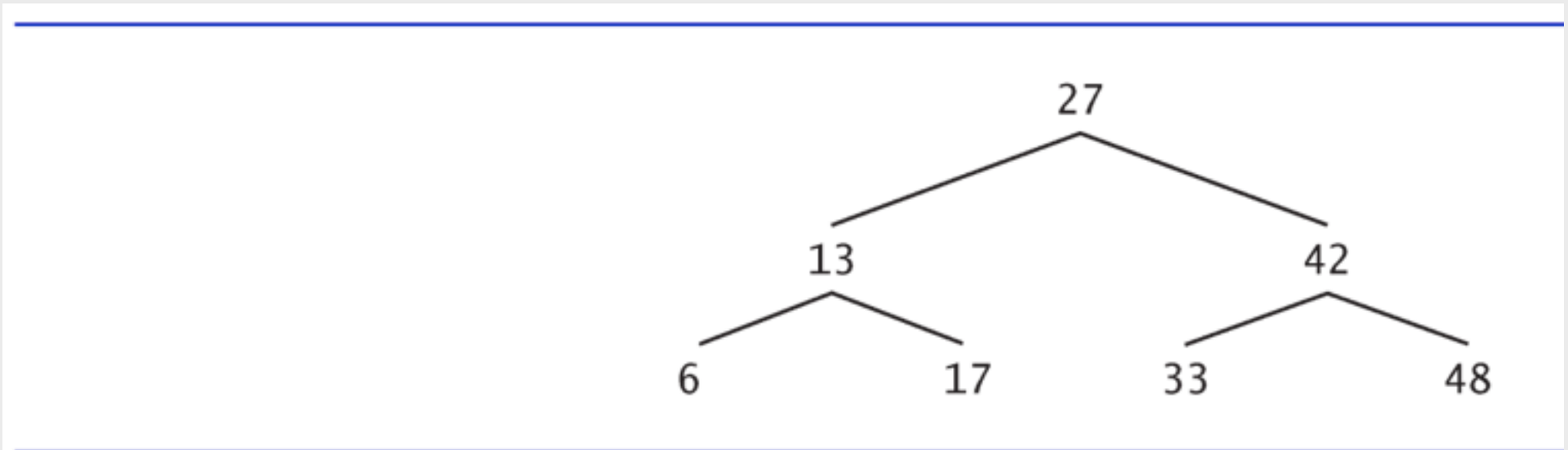

Fig. 12.21 | Binary search tree with seven nodes.

- inOrder: 6 13 17 27 33 42 48
- preOrder: 27 13 6 17 42 33 48
- postOrder: 6 17 13 33 48 42 27

Trees

- Searching a binary tree for a value that matches a key value is also fast.
- When searching a (tightly packed) 1,000,000 element binary search tree, no more than 20 comparisons need to be made because 2^20 > 1,000,000.
- This means, for example, that when searching a (tightly packed) 1000-element binary search tree, no more than 10 comparisons need to be made because 2^10 > 1000.

C Preprocessors

Objectives

- In this chapter, you'll learn
  - To use **`#include`** to develop large programs
  - To use **`#define`** to create macros and macros with arguments
  - To use conditional compilation
  - To display error messages during conditional compilation
  - To use assertion to test if the values of expressions are correct

Introduction

- The C preprocessor executes before a program is compiled
- Some actions can be defined in preprocessor
  - symbolic constants and macros
  - conditional compilation of program
  - conditional execution of preprocessor directives
- Preprocessor directives begin with #

`#include` Preprocessor Directive

- The `#include` **directive causes** a copy of a specified file to be included in place of the directive.
- The two forms of the #include directive are:

```
#include <filename>
#include "filename"
```

  - If the file name is enclosed in quotes, the preprocessor starts searches in the same directory as the file being compiled

`#include` Preprocessor Directive (Cont.)

- If the file name is enclosed in angle brackets (< and >)—used for standard library headers —the search is performed in an implementation-dependent manner, normally through predesignated compiler and system directories (e.g., /usr/include)

`#include` Preprocessor Directive (Cont.)

- A header containing declarations common to the separate program files is often created and included in the file.
- Examples of such declarations are structure and union declarations, enumerations and function prototypes.

`#define` Preprocessor Directive Symbolic Constants

- The **`#define`** directive creates symbolic constants —constants represented as symbols—and macros —operations defined as symbols.
- The **`#define`** directive format is

```
#define identifier replacement-text
```

- For example
  - **`#define PI  3.14159`**

❗ **`#define PI = 3.14159`**

`#define` Preprocessor Directive Symbolic Constants

- The **`#define`** directive creates symbolic constants —constants represented as symbols—and macros —operations defined as symbols.
- The **`#define`** directive format is

```
#define identifier replacement-text
```

- For example
  - `#define PI  3.14159`

❗ ~~`#define PI = 3.14159`~~

`#define` Preprocessor Directive Symbolic Constants (Cont.)

Good Programming Practice 13.1

Using meaningful names for symbolic constants helps make programs more self-documenting.

Good Programming Practice 13.2

By convention, symbolic constants are defined using only uppercase letters and underscores.

`#define` Preprocessor Directive: Macros

- A macro is an identifier defined in a **`#define`** preprocessor directive.
- As with symbolic constants, the macro-identifier is replaced in the program with the replacement-text before the program is compiled.
- In a macro with arguments, the arguments are substituted in the replacement text, then the macro is expanded

`#define` Preprocessor Directive: Macros (Cont.)

- Consider the following macro definition with one argument for the area of a circlee

```
#define CIRCLE_AREA(x) ((PI)*(x)*(x))
```

- For example, the statement

```
area = CIRCLE_AREA(4);
```

is expanded to

```
area = ((3.14159)*(4)*(4));
```

`#define` Preprocessor Directive: Macros (Cont.)

- The parentheses around each **`x`** in the replacement text force the proper order of evaluation when the macro argument is an expression.
- For example, the statement

```
area = CIRCLE_AREA(c + 2);
```

is expanded to

```
area = ((3.14159)*(c + 2)*(c + 2));
```

which evaluates correctly because the parentheses force the proper order of evaluation.

`#define` Preprocessor Directive: Macros (Cont.)

Common Programming Error 13.1

Forgetting to enclose macro arguments in parentheses in the replacement text can lead to logic errors.

`#define` Preprocessor Directive: Macros (Cont.)

- Macro **`CIRCLE_AREA`** could be defined as a function.
- The advantages of macro **`CIRCLE_AREA`** are that macros insert code directly in the program—avoiding function call overhead.

`#define` Preprocessor Directive: Macros (Cont.)

- The following is a macro definition with two arguments for the area of a rectangle:

```
#define RECTANGLE_AREA( x, y )  ((x)*(y))
```

- For example, the statement

```
rectArea = RECTANGLE_AREA(a+4, b+7);
```

is expanded to

```
rectArea = ((a+4)*(b+7));
```

`#define` Preprocessor Directive: Macros (Cont.)

- Symbolic constants and macros can be discarded by using the `#undef` preprocessor directive.
- Directive `#undef` "undefines" a symbolic constant or macro name.
- The scope of a symbolic constant or macro is from its definition until it is undefined with `#undef`, or until the end of the file.

`#define` Preprocessor Directive: Macros (Cont.)

- A macro commonly defined in the **`<stdio.h>`** header is

  ```
  #define getchar() getc(stdin)
  ```

- The macro definition of **`getchar`** uses function **`getc`** to get one character from the standard input stream.

- Expressions with side effects (i.e., variable values are modified) should not be passed to a macro because macro arguments may be evaluated more than once.

Conditional Compilation

- Conditional compilation enables you to control the execution of preprocessor directives and the compilation of program code.
- The conditional preprocessor construct is much like the if selection statement.
- Each of the conditional preprocessor directives evaluates a constant integer expression.

Conditional Compilation (Cont.)

- Consider the following preprocessor code:

```
#if !defined(MY_CONSTANT)
    #define MY_CONSTANT 0
#endif
```

- These directives determine if **`MY_CONSTANT`** is defined.

Conditional Compilation (Cont.)

- Every **`#if`** construct ends with **`#endif`**
- Directives **`#ifdef`** and **`#ifndef`** are shorthand for **`#if defined(name)`** and **`#if !defined(name)`**
- A multiple-part conditional preprocessor construct may be tested by using the **`#elif`** (the equivalent of **`else if`** in an **`if`** statement) and the **#else** (the equivalent of **`else`** in an **`if`** statement) directives.
- These directives are frequently used to prevent header files from being included multiple times in the same source file.

Conditional Compilation (Cont.)

- During program development, it is often helpful to "comment out" portions of code to prevent it from being compiled.
- You can use the following preprocessor construct:

```
#if 0
   code prevented from compiling
#endif
```

- To enable the code to be compiled, replace the 0 in the preceding construct with 1.

Conditional Compilation (Cont.)

- Conditional compilation is commonly used as a debugging aid.
- For example,

```
#ifdef DEBUG
   printf( "Variable x = %d\n", x );
#endif
```

causes a **`printf`** statement to be compiled in the program if the symbolic constant **`DEBUG`** has been defined (**`#define DEBUG`**) before directive **`#ifdef DEBUG`**.

Conditional Compilation (Cont.)

- When debugging is completed, the **`#define`** directive is removed from the source file (or commented out) and the **`printf`** statements inserted for debugging purposes are ignored during compilation.
- In larger programs, it may be desirable to define several different symbolic constants that control the conditional compilation in separate sections of the source file.

and ## Operators

- The # operator causes a replacement text token to be converted to a string surrounded by quotes.
- Consider the following macro definition:

```
#define HELLO(x) printf("Hello, " #x "\n");
```

- When **HELLO(John)** appears in a program file, it is expanded to

```
printf("Hello, " "John" "\n");
```

- The string "**John**" replaces **#x** in the replacement text.
- Strings separated by white space are concatenated during preprocessing, so the preceding statement is equivalent to

```
printf( "Hello, John\n" );
```

and ## Operators (Cont.)

- The ## operator concatenates two tokens.
- Consider the following macro definition:

```
#define TOKENCONCAT(x, y)  x ## y
```

- When **TOKENCONCAT** appears in the program, its arguments are concatenated and used to replace the macro.
- For example, **TOKENCONCAT(O, K)** is replaced by **OK** in the program.

Line Numbers

- The **`#line` preprocessor directive** causes the subsequent source code lines to be renumbered starting with the specified constant integer value.
- The directive

```
#line 100
```

starts line numbering from 100 beginning with the next source code line.

Line Numbers (Cont.)

- A file name can be included in the **`#line`** directive.
- The directive

```
#line 100 "file1.c"
```

  indicates that lines are numbered from 100 beginning with the next source code line and that the name of the file for the purpose of any compiler messages is "**`file1.c`**".
- The directive normally is used to help make the messages produced by syntax errors and compiler warnings more meaningful.

Assertions

- The **assert macro**—defined in the **`<assert.h>`** header—tests the value of an expression.
- If the value of the expression is false **`(0)`**, **assert** prints an error message and calls function **abort** (of the general utilities library—**`<stdlib.h>`**) to terminate program execution.
- This is a useful debugging tool for testing if a variable has a correct value.

Assertions (Cont.)

- For example, suppose variable **x** should never be larger than **10** in a program.
- An assertion may be used to test the value of **x** and print an error message if the value of **x** is incorrect.
- The statement would be

```
assert( x <= 10 );
```

- If **x** is greater than **10** when the preceding statement is encountered in a program, an error message containing the line number and file name is printed and the program terminates.

Assertions (Cont.)

Software Engineering Observation 13.1

Assertions are not meant as a substitute for error handling during normal runtime conditions. Their use should be limited to finding logic errors.

Other C Topics

Objectives

- In this chapter, you'll learn
  - To redirect keyboard input to come from a file
  - To redirect screen output to be placed in a file
  - To write functions that use variable-length argument lists
  - To process command-line arguments
  - To assign specific types to numeric constants
  - To use temporary files
  - To process external asynchronous events in a program

Redirecting I/O

- Redirect input symbol (<) indicates that the data in file input is to be used as input by the program
- A pipe (|) causes the output of one program to be redirected as the input to another program
- Program output can be redirected to a file by using the redirect output symbol (>)
- Program output can be appended to the end of an existing file by using the append output symbol (>>)

Standard streams

- Standard Streams
  - **`stdin`**: an input stream associated with a device - your keyboard; **`scanf(..)`**
  - **`stdout`**: an output stream associated with a device - your terminal; **`printf(..)`**
  - **`stderr`**: an output stream associated with your terminal just like stdout; **`fprintf(stderr, "string\n")`**
  - **`dev/null`**: a output stream associated with no device. This stream is generally used to make textual output disappear and not show up anywhere.

Redirections (1)

- redirection operations can be parenthesized
  - **e.g., (./a.out < input.txt > out.txt) >& err.txt**
- Redirecting output
  - use the “**>**” symbol to redirect the output of a command
- Appending to a file
  - use the “**>>**” to append standard output to a file
- Redirecting input
  - use the “**<**” symbol to redirect the input of a command
- Redirect output and error message
  - use teh “**>&**” to redirect stderr to a file

Pipes

- Use the vertical bar "|" to pipe outputs to another command
  - **`command1 | command2`**
- use pipes to generate complex commands
  - e.g., **`who | wc -l`**

Pipes

- Use the vertical bar "|" to pipe outputs to another command
  - **`command1 | command2`**
- use pipes to generate complex commands
  - e.g., **`who | wc -l`**

```
% who > names.txt
% sort < names.txt
```

Pipes

- Use the vertical bar "|" to pipe outputs to another command
  - **`command1 | command2`**
- use pipes to generate complex commands
  - e.g., **`who | wc -l`**

```
% who > names.txt
% sort < names.txt
```

equivalent to

Pipes

- Use the vertical bar "|" to pipe outputs to another command
  - **`command1 | command2`**
- use pipes to generate complex commands
  - e.g., **`who | wc -l`**

```
% who > names.txt
% sort < names.txt
```

equivalent to

```
% who | sort
```

Variable-Length Argument Lists

- It's possible to create functions that receive an unspecified number of arguments.
- The function prototype for **`printf`** is

```
int printf(const char *format, ...);
```

- The ellipsis (`...`) in the prototype indicates that the function receives a variable number of arguments of any type.

Variable-Length Argument Lists (Cont.)

- The ellipsis must always be placed at the end of the parameter list.
- The macros and definitions of the variable arguments headers **`<stdarg.h>`** provide the capabilities necessary to build functions with variable-length argument lists.

Variable-Length Argument Lists (Cont.)

Identifier	Explanation
`va_list`	A type suitable for holding information needed by macros `va_start`, `va_arg` and `va_end`. To access the arguments in a variable-length argument list, an object of type `va_list` must be defined.
`va_start`	A macro that is invoked before the arguments of a variable-length argument list can be accessed. The macro initializes the object declared with `va_list` for use by the `va_arg` and `va_end` macros.
`va_arg`	A macro that expands to an expression of the value and type of the next argument in the variable-length argument list. Each invocation of `va_arg` modifies the object declared with `va_list` so that the object points to the next argument in the list.
`va_end`	A macro that facilitates a normal return from a function whose variable-length argument list was referred to by the `va_start` macro.

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```c
#include <stdio.h>
#include <stdarg.h>

double average( int i, ... ); /* prototype */

int main( void ) {
    double w = 37.5;
    double x = 22.5;
    double y = 1.7;
    double z = 10.2;

    printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",
            "w = ", w, "x = ", x, "y = ", y, "z = ", z );
    printf( "%s%.3f\n%s%.3f\n%s%.3f\n",
            "The average of w and x is ", average( 2, w, x ),
            "The average of w, x, and y is ", average( 3, w, x, y ),
            "The average of w, x, y, and z is ",
            average( 4, w, x, y, z ) );
    return 0; /* indicates successful termination */
} /* end main */
```

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```c
#include <stdio.h>
#include <stdarg.h>

double average( int i, ... ); /* prototype */

int main( void ) {
    double w = 37.5;
    double x = 22.5;
    double y = 1.7;
    double z = 10.2;

    printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",
            "w = ", w, "x = ", x, "y = ", y, "z = ", z );
    printf( "%s%.3f\n%s%.3f\n%s%.3f\n",
            "The average of w and x is ", average( 2, w, x ),
            "The average of w, x, and y is ", average( 3, w, x, y ),
            "The average of w, x, y, and z is ",
            average( 4, w, x, y, z ) );
    return 0; /* indicates successful termination */
} /* end main */
```

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```c
#include <stdio.h>
#include <stdarg.h>

double average( int i, ... ); /* prototype */

int main( void ) {
    double w = 37.5;
    double x = 22.5;
    double y = 1.7;
    double z = 10.2;

    printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",
            "w = ", w, "x = ", x, "y = ", y, "z = ", z );
    printf( "%s%.3f\n%s%.3f\n%s%.3f\n",
            "The average of w and x is ", average( 2, w, x ),
            "The average of w, x, and y is ", average( 3, w, x, y ),
            "The average of w, x, y, and z is ",
            average( 4, w, x, y, z ) );
    return 0; /* indicates successful termination */
} /* end main */
```

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```c
#include <stdio.h>
#include <stdarg.h>

double average( int i, ... ); /* prototype */

int main( void ) {
    double w = 37.5;
    double x = 22.5;
    double y = 1.7;
    double z = 10.2;

    printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",
            "w = ", w, "x = ", x, "y = ", y, "z = ", z );
    printf( "%s%.3f\n%s%.3f\n%s%.3f\n",
            "The average of w and x is ", average( 2, w, x ),
            "The average of w, x, and y is ", average( 3, w, x, y ),
            "The average of w, x, y, and z is ",
            average( 4, w, x, y, z ) );
    return 0; /* indicates successful termination */
} /* end main */
```

define a function prototype with variable-length arguments

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```c
#include <stdio.h>
#include <stdarg.h>

double average( int i, ... ); /* prototype */

int main( void ) {
    double w = 37.5;
    double x = 22.5;
    double y = 1.7;
    double z = 10.2;

    printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",
            "w = ", w, "x = ", x, "y = ", y, "z = ", z );
    printf( "%s%.3f\n%s%.3f\n%s%.3f\n",
            "The average of w and x is ", average( 2, w, x ),
            "The average of w, x, and y is ", average( 3, w, x, y ),
            "The average of w, x, y, and z is ",
            average( 4, w, x, y, z ) );
    return 0; /* indicates successful termination */
} /* end main */
```

define a function prototype with variable-length arguments

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26     double total = 0; /* initialize total */
27     int j; /* counter for selecting arguments  */
28     va_list ap; /* stores information needed by va_start and va_end */
29
30     va_start( ap, i ); /* initializes the va_list object */
31
32     /* process variable length argument list */
33     for ( j = 1; j <= i; j++ ) {
34         total += va_arg( ap, double );
35     } /* end for */
36
37     va_end( ap ); /* clean up variable-length argument list */
38     return total / i; /* calculate average */
39 } /* end function average */
```

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```c
double average( int i, ... ) {
    double total = 0; /* initialize total */
    int j; /* counter for selecting arguments  */
    va_list ap; /* stores information needed by va_start and va_end */

    va_start( ap, i ); /* initializes the va_list object */

    /* process variable length argument list */
    for ( j = 1; j <= i; j++ ) {
        total += va_arg( ap, double );
    } /* end for */

    va_end( ap ); /* clean up variable-length argument list */
    return total / i; /* calculate average */
} /* end function average */
```

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26    double total = 0; /* initialize total */
27    int j; /* counter for selecting arguments  */
28    va_list ap; /* stores information needed by va_start and va_end */
29
30    va_start( ap, i ); /* initializes the va_list object */
31
32    /* process variable length argument list */
33    for ( j = 1; j <= i; j++ ) {
34       total += va_arg( ap, double );
35    } /* end for */
36
37    va_end( ap ); /* clean up variable-length argument list */
38    return total / i; /* calculate average */
39 } /* end function average */
```

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26    double total = 0; /* initialize total */
27    int j; /* counter for selecting arguments  */
28    va_list ap; /* stores information needed by va_start and va_end */
29
30    va_start( ap, i ); /* initializes the va_list object */
31
32    /* process variable length argument list */
33    for ( j = 1; j <= i; j++ ) {
34       total += va_arg( ap, double );
35    } /* end for */
36
37    va_end( ap ); /* clean up variable-length argument list */
38    return total / i; /* calculate average */
39 } /* end function average */
```

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26    double total = 0; /* initialize total */
27    int j; /* counter for selecting arguments  */
28    va_list ap; /* stores information needed by va_start and va_end */
29
30    va_start( ap, i ); /* initializes the va_list object */
31
32    /* process variable length argument list */
33    for ( j = 1; j <= i; j++ ) {
34       total += va_arg( ap, double );
35    } /* end for */
36
37    va_end( ap ); /* clean up variable-length argument list */
38    return total / i; /* calculate average */
39 } /* end function average */
```

initializes the va_list object

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26    double total = 0; /* initialize total */
27    int j; /* counter for selecting arguments  */
28    va_list ap; /* stores information needed by va_start and va_end */
29
30    va_start( ap, i ); /* initializes the va_list object */
31
32    /* process variable length argument list */
33    for ( j = 1; j <= i; j++ ) {
34       total += va_arg( ap, double );
35    } /* end for */
36
37    va_end( ap ); /* clean up variable-length argument list */
38    return total / i; /* calculate average */
39 } /* end function average */
```

initializes the va_list object

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26    double total = 0; /* initialize total */
27    int j; /* counter for selecting arguments  */
28    va_list ap; /* stores information needed by va_start and va_end */
29
30    va_start( ap, i ); /* initializes the va_list object */
31
32    /* process variable length argument list */
33    for ( j = 1; j <= i; j++ ) {
34       total += va_arg( ap, double );
35    } /* end for */
36
37    va_end( ap ); /* clean up variable-length argument list */
38    return total / i; /* calculate average */
39 } /* end function average */
```

initializes the va_list object

process each argument

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26    double total = 0; /* initialize total */
27    int j; /* counter for selecting arguments  */
28    va_list ap; /* stores information needed by va_start and va_end */
29
30    va_start( ap, i ); /* initializes the va_list object */
31
32    /* process variable length argument list */
33    for ( j = 1; j <= i; j++ ) {
34       total += va_arg( ap, double );
35    } /* end for */
36
37    va_end( ap ); /* clean up variable-length argument list */
38    return total / i; /* calculate average */
39 } /* end function average */
```

initializes the va_list object

process each argument

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26    double total = 0; /* initialize total */
27    int j; /* counter for selecting arguments  */
28    va_list ap; /* stores information needed by va_start and va_end */
29
30    va_start( ap, i ); /* initializes the va_list object */
31
32    /* process variable length argument list */
33    for ( j = 1; j <= i; j++ ) {
34       total += va_arg( ap, double );
35    } /* end for */
36
37    va_end( ap ); /* clean up variable-length argument list */
38    return total / i; /* calculate average */
39 } /* end function average */
```

initializes the va_list object

process each argument

clean up the list

Variable-Length Argument Lists (Cont.)

- Example: fig14_02.c

```
25 double average( int i, ... ) {
26    double total = 0; /* initialize total */
27    int j; /* counter for selecting arguments  */
28    va_list ap; /* stores information needed by va_start and va_end */
29
30    va_start( ap, i ); /* initializes the va_list object */
31
32    /* process variable length argument list */
33    for ( j = 1; j <= i; j++ ) {
34       total += va_arg( ap, double );
35    } /* end for */
36
37    va_end( ap ); /* clean up variable-length argument list */
38    return total / i; /* calculate average */
39 } /* end function average */
```

initializes the va_list object

process each argument

clean up the list

```
w = 37.5
x = 22.5
y = 1.7
z = 10.2

The average of w and x is 30.000
The average of w, x, and y is 20.567
The average of w, x, y, and z is 17.975
```

Variable-Length Argument Lists (Cont.)

- Object **ap**, of type **va_list**, is used by macros **va_start**, **va_arg** and **va_end** to process the variable-length argument list of function **average**.
- The function begins by invoking macro **va_start** to initialize object **ap** for use in **va_arg** and **va_end**.
- Macro **va_arg** receives two arguments—object **ap** and the type of the value expected in the argument list—**double** in this case.
- Macro **va_end** with object **ap** to finalize the object

Variable-Length Argument Lists (Cont.)

Common Programming Error 14.1

Placing an ellipsis in the middle of a function parameter list is a syntax error. An ellipsis may only be placed at the end of the parameter list.

Using Command-Line Arguments

- On many systems, it's possible to pass arguments to **`main`** from a command line by including parameters **`int argc`** and **`char *argv[]`** in the parameter list of main.
  - Parameter **`argc`** receives the number of command-line arguments.
  - Parameter **`argv`** is an array of strings in which the actual command-line arguments are stored.

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
3  #include <stdio.h>
4
5  int main( int argc, char *argv[] ) {
6      FILE *inFilePtr; /* input file pointer */
7      FILE *outFilePtr; /* output file pointer */
8      int c; /* define c to hold characters input by user */
9
10     /* check number of command-line arguments */
11     if ( argc != 3 ) {
12         printf( "Usage: mycopy infile outfile\n" );
13     } /* end if */
```

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
3  #include <stdio.h>
4
5  int main( int argc, char *argv[] ) {
6      FILE *inFilePtr; /* input file pointer */
7      FILE *outFilePtr; /* output file pointer */
8      int c; /* define c to hold characters input by user */
9
10     /* check number of command-line arguments */
11     if ( argc != 3 ) {
12         printf( "Usage: mycopy infile outfile\n" );
13     } /* end if */
```

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
3  #include <stdio.h>
4
5  int main( int argc, char *argv[] ) {
6      FILE *inFilePtr; /* input file pointer */
7      FILE *outFilePtr; /* output file pointer */
8      int c; /* define c to hold characters input by user */
9
10     /* check number of command-line arguments */
11     if ( argc != 3 ) {
12         printf( "Usage: mycopy infile outfile\n" );
13     } /* end if */
```

main() with the argc and argv arguments

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
3  #include <stdio.h>
4
5  int main( int argc, char *argv[] ) {
6      FILE *inFilePtr; /* input file pointer */
7      FILE *outFilePtr; /* output file pointer */
8      int c; /* define c to hold characters input by user */
9
10     /* check number of command-line arguments */
11     if ( argc != 3 ) {
12         printf( "Usage: mycopy infile outfile\n" );
13     } /* end if */
```

main() with the argc and argv arguments

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
3  #include <stdio.h>
4
5  int main( int argc, char *argv[] ) {
6      FILE *inFilePtr; /* input file pointer */
7      FILE *outFilePtr; /* output file pointer */
8      int c; /* define c to hold characters input by user */
9
10     /* check number of command-line arguments */
11     if ( argc != 3 ) {
12         printf( "Usage: mycopy infile outfile\n" );
13     } /* end if */
```

main() with the argc and argv arguments

if # of args < 3, display the usage

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
14    else {
15       /* if input file can be opened */
16       if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {
17          /* if output file can be opened */
18          if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {
19             /* read and output characters */
20             while ( ( c = fgetc( inFilePtr ) ) != EOF ) {
21                fputc( c, outFilePtr );
22             } /* end while */
23          } /* end if */.
24          else { /* output file could not be opened */
25             printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
26          } /* end else */
27       } /* end if */
28       else { /* input file could not be opened */
29          printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
30       } /* end else */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
14    else {
15       /* if input file can be opened */
16       if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {
17          /* if output file can be opened */
18          if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {
19             /* read and output characters */
20             while ( ( c = fgetc( inFilePtr ) ) != EOF ) {
21                fputc( c, outFilePtr );
22             } /* end while */
23          } /* end if */
24          else { /* output file could not be opened */
25             printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
26          } /* end else */
27       } /* end if */
28       else { /* input file could not be opened */
29          printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
30       } /* end else */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
14    else {
15       /* if input file can be opened */
16       if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {
17          /* if output file can be opened */
18          if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {
19             /* read and output characters */
20             while ( ( c = fgetc( inFilePtr ) ) != EOF ) {
21                fputc( c, outFilePtr );
22             } /* end while */
23          } /* end if */
24          else { /* output file could not be opened */
25             printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
26          } /* end else */
27       } /* end if */
28       else { /* input file could not be opened */
29          printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
30       } /* end else */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

open source file

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
14    else {
15       /* if input file can be opened */
16       if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {
17          /* if output file can be opened */
18          if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {
19             /* read and output characters */
20             while ( ( c = fgetc( inFilePtr ) ) != EOF ) {
21                fputc( c, outFilePtr );
22             } /* end while */
23          } /* end if */
24          else { /* output file could not be opened */
25             printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
26          } /* end else */
27       } /* end if */
28       else { /* input file could not be opened */
29          printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
30       } /* end else */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

open source file

Using Command-Line Arguments (Cont.)

- Example: fig14_03.c

```
14    else {
15       /* if input file can be opened */
16       if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {
17          /* if output file can be opened */
18          if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {
19             /* read and output characters */
20             while ( ( c = fgetc( inFilePtr ) ) != EOF ) {
21                fputc( c, outFilePtr );
22             } /* end while */
23          } /* end if */
24          else { /* output file could not be opened */
25             printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
26          } /* end else */
27       } /* end if */
28       else { /* input file could not be opened */
29          printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
30       } /* end else */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

open source file

open destination file

Program Termination with **`exit`** and **`atexit`**

- The general utilities library (**`<stdlib.h>`**) provides methods of terminating program execution by means other than a conventional return from function main.
- Function **`exit`** forces a program to terminate as if it executed normally.
- Function **`atexit`** registers a function that should be called upon successful termination of the program.

Program Termination with **`exit`** and **`atexit`** (Cont.)

- Example: fig14_04.c

```
 8 int main( void ) {
 9     int answer; /* user's menu choice */
10
11     atexit( print ); /* register function print */
12     printf( "Enter 1 to terminate program with function exit"
13             "\nEnter 2 to terminate program normally\n" );
14     scanf( "%d", &answer );
15
16     /* call exit if answer is 1 */
17     if ( answer == 1 ) {
18         printf( "\nTerminating program with function exit\n" );
19         exit( EXIT_SUCCESS );
20     } /* end if */
21
22     printf( "\nTerminating program by reaching the end of main\n" );
23     return 0; /* indicates successful termination */
24 } /* end main */
25
26 /* display message before termination */
27 void print( void ) {
28     printf( "Executing function print at program "
29             "termination\nProgram terminated\n" );
30 } /* end function print */
```

Program Termination with **`exit`** and **`atexit`** (Cont.)

- Example: fig14_04.c

```
 8 int main( void ) {
 9    int answer; /* user's menu choice */
10
11    atexit( print ); /* register function print */
12    printf( "Enter 1 to terminate program with function exit"
13            "\nEnter 2 to terminate program normally\n" );
14    scanf( "%d", &answer );
15
16    /* call exit if answer is 1 */
17    if ( answer == 1 ) {
18       printf( "\nTerminating program with function exit\n" );
19       exit( EXIT_SUCCESS );
20    } /* end if */
21
22    printf( "\nTerminating program by reaching the end of main\n" );
23    return 0; /* indicates successful termination */
24 } /* end main */
25
26 /* display message before termination */
27 void print( void ) {
28    printf( "Executing function print at program "
29            "termination\nProgram terminated\n" );
30 } /* end function print */
```

Program Termination with **`exit`** and **`atexit`** (Cont.)

- Example: fig14_04.c

```
 8 int main( void ) {
 9    int answer; /* user's menu choice */
10
11    atexit( print ); /* register function print */
12    printf( "Enter 1 to terminate program with function exit"
13            "\nEnter 2 to terminate program normally\n" );
14    scanf( "%d", &answer );
15
16    /* call exit if answer is 1 */
17    if ( answer == 1 ) {
18       printf( "\nTerminating program with function exit\n" );
19       exit( EXIT_SUCCESS );
20    } /* end if */
21
22    printf( "\nTerminating program by reaching the end of main\n" );
23    return 0; /* indicates successful termination */
24 } /* end main */
25
26 /* display message before termination */
27 void print( void ) {
28    printf( "Executing function print at program "
29            "termination\nProgram terminated\n" );
30 } /* end function print */
```

register print()

Program Termination with **`exit`** and **`atexit`** (Cont.)

- Example: fig14_04.c

```
 8 int main( void ) {
 9    int answer; /* user's menu choice */
10
11    atexit( print ); /* register function print */
12    printf( "Enter 1 to terminate program with function exit"
13            "\nEnter 2 to terminate program normally\n" );
14    scanf( "%d", &answer );
15
16    /* call exit if answer is 1 */
17    if ( answer == 1 ) {
18       printf( "\nTerminating program with function exit\n" );
19       exit( EXIT_SUCCESS );
20    } /* end if */
21
22    printf( "\nTerminating program by reaching the end of main\n" );
23    return 0; /* indicates successful termination */
24 } /* end main */
25
26 /* display message before termination */
27 void print( void ) {
28    printf( "Executing function print at program "
29            "termination\nProgram terminated\n" );
30 } /* end function print */
```

register print()

Program Termination with **`exit`** and **`atexit`** (Cont.)

- Example: fig14_04.c

```
 8 int main( void ) {
 9    int answer; /* user's menu choice */
10
11    atexit( print ); /* register function print */
12    printf( "Enter 1 to terminate program with function exit"
13            "\nEnter 2 to terminate program normally\n" );
14    scanf( "%d", &answer );
15
16    /* call exit if answer is 1 */
17    if ( answer == 1 ) {
18       printf( "\nTerminating program with function exit\n" );
19       exit( EXIT_SUCCESS );
20    } /* end if */
21
22    printf( "\nTerminating program by reaching the end of main\n" );
23    return 0; /* indicates successful termination */
24 } /* end main */
25
26 /* display message before termination */
27 void print( void ) {
28    printf( "Executing function print at program "
29            "termination\nProgram terminated\n" );
30 } /* end function print */
```

register print()

exit normally

Program Termination with **`exit`** and **`atexit`** (Cont.)

- Function **`atexit`** takes as an argument a pointer to a function (i.e., the function name).
- Function **`exit`** normally takes the symbolic constant **`EXIT_SUCCESS`** or the symbolic constant **`EXIT_FAILURE`**.

More on Files

- The standard library also provides function tmpfile() that opens a temporary file.
- Although this is a binary file mode ("**`wb+`**"), some systems process temporary files as text files.
- A temporary file exists until it's closed with **`fclose`**, or until the program terminates.

More on Files (Cont.)

- Example: fig14_06.c

```
 5 int main( void ) {
 6     FILE *filePtr; /* pointer to file being modified */
 7     FILE *tempFilePtr; /* temporary file pointer */
 8     int c; /* define c to hold characters read from a file */
 9     char fileName[ 30 ]; /* create char array */
10
11     printf( "This program changes tabs to spaces.\n"
12             "Enter a file to be modified: " );
13     scanf( "%29s", fileName );
14
15     /* fopen opens the file */
16     if ( ( filePtr = fopen( fileName, "r+" ) ) != NULL ) {
17        /* create temporary file */
18        if ( ( tempFilePtr = tmpfile() ) != NULL ) {
19           printf( "\nThe file before modification is:\n" );
```

More on Files (Cont.)

- Example: fig14_06.c

```
 5 int main( void ) {
 6     FILE *filePtr; /* pointer to file being modified */
 7     FILE *tempFilePtr; /* temporary file pointer */
 8     int c; /* define c to hold characters read from a file */
 9     char fileName[ 30 ]; /* create char array */
10
11     printf( "This program changes tabs to spaces.\n"
12             "Enter a file to be modified: " );
13     scanf( "%29s", fileName );
14
15     /* fopen opens the file */
16     if ( ( filePtr = fopen( fileName, "r+" ) ) != NULL ) {
17        /* create temporary file */
18        if ( ( tempFilePtr = tmpfile() ) != NULL ) {
19           printf( "\nThe file before modification is:\n" );
```

More on Files (Cont.)

- Example: fig14_06.c

```
 5 int main( void ) {
 6     FILE *filePtr; /* pointer to file being modified */
 7     FILE *tempFilePtr; /* temporary file pointer */
 8     int c; /* define c to hold characters read from a file */
 9     char fileName[ 30 ]; /* create char array */
10
11     printf( "This program changes tabs to spaces.\n"
12             "Enter a file to be modified: " );
13     scanf( "%29s", fileName );
14
15     /* fopen opens the file */
16     if ( ( filePtr = fopen( fileName, "r+" ) ) != NULL ) {
17        /* create temporary file */
18        if ( ( tempFilePtr = tmpfile() ) != NULL ) {
19           printf( "\nThe file before modification is:\n" );
```

create temp file

More on Files (Cont.)

- Example: fig14_06.c

```
22          while ( ( c = getc( filePtr ) ) != EOF ) {
23              putchar( c );
24              putc( c == '\t' ? ' ': c, tempFilePtr );
25          } /* end while */
26
27          rewind( tempFilePtr );
28          rewind( filePtr );
29          printf( "\n\nThe file after modification is:\n" );
30
31          /* read from temporary file and write into original file */
32          while ( ( c = getc( tempFilePtr ) ) != EOF ) {
33              putchar( c );
34              putc( c, filePtr );
35          } /* end while */
36      } /* end if */
37      else { /* if temporary file could not be opened */
38          printf( "Unable to open temporary file\n" );
39      } /* end else */
40  } /* end if */
41  else { /* if file could not be opened */
42      printf( "Unable to open %s\n", fileName );
43  } /* end else */
44
45  return 0; /* indicates successful termination */
46 } /* end main */
```

More on Files (Cont.)

- Example: fig14_06.c

```
22          while ( ( c = getc( filePtr ) ) != EOF ) {
23              putchar( c );
24              putc( c == '\t' ? ' ': c, tempFilePtr );
25          } /* end while */
26
27          rewind( tempFilePtr );
28          rewind( filePtr );
29          printf( "\n\nThe file after modification is:\n" );
30
31          /* read from temporary file and write into original file */
32          while ( ( c = getc( tempFilePtr ) ) != EOF ) {
33              putchar( c );
34              putc( c, filePtr );
35          } /* end while */
36      } /* end if */
37      else { /* if temporary file could not be opened */
38          printf( "Unable to open temporary file\n" );
39      } /* end else */
40  } /* end if */
41  else { /* if file could not be opened */
42      printf( "Unable to open %s\n", fileName );
43  } /* end else */
44
45  return 0; /* indicates successful termination */
46 } /* end main */
```

More on Files (Cont.)

- Example: fig14_06.c

```
22          while ( ( c = getc( filePtr ) ) != EOF ) {
23              putchar( c );
24              putc( c == '\t' ? ' ': c, tempFilePtr );
25          } /* end while */
26
27          rewind( tempFilePtr );
28          rewind( filePtr );
29          printf( "\n\nThe file after modification is:\n" );
30
31          /* read from temporary file and write into original file */
32          while ( ( c = getc( tempFilePtr ) ) != EOF ) {
33              putchar( c );
34              putc( c, filePtr );
35          } /* end while */
36      } /* end if */
37      else { /* if temporary file could not be opened */
38          printf( "Unable to open temporary file\n" );
39      } /* end else */
40  } /* end if */
41  else { /* if file could not be opened */
42      printf( "Unable to open %s\n", fileName );
43  } /* end else */
44
45  return 0; /* indicates successful termination */
46 } /* end main */
```

change '\t' to ' ' and write to the temp file

More on Files (Cont.)

- Example: fig14_06.c

```
22          while ( ( c = getc( filePtr ) ) != EOF ) {
23              putchar( c );
24              putc( c == '\t' ? ' ': c, tempFilePtr );
25          } /* end while */
26
27          rewind( tempFilePtr );
28          rewind( filePtr );
29          printf( "\n\nThe file after modification is:\n" );
30
31          /* read from temporary file and write into original file */
32          while ( ( c = getc( tempFilePtr ) ) != EOF ) {
33              putchar( c );
34              putc( c, filePtr );
35          } /* end while */
36      } /* end if */
37      else { /* if temporary file could not be opened */
38          printf( "Unable to open temporary file\n" );
39      } /* end else */
40  } /* end if */
41  else { /* if file could not be opened */
42      printf( "Unable to open %s\n", fileName );
43  } /* end else */
44
45  return 0; /* indicates successful termination */
46 } /* end main */
```

change '\t' to ' ' and write to the temp file

More on Files (Cont.)

- Example: fig14_06.c

```
22          while ( ( c = getc( filePtr ) ) != EOF ) {
23              putchar( c );
24              putc( c == '\t' ? ' ': c, tempFilePtr );
25          } /* end while */
26
27          rewind( tempFilePtr );
28          rewind( filePtr );
29          printf( "\n\nThe file after modification is:\n" );
30
31          /* read from temporary file and write into original file */
32          while ( ( c = getc( tempFilePtr ) ) != EOF ) {
33              putchar( c );
34              putc( c, filePtr );
35          } /* end while */
36      } /* end if */
37      else { /* if temporary file could not be opened */
38          printf( "Unable to open temporary file\n" );
39      } /* end else */
40  } /* end if */
41  else { /* if file could not be opened */
42      printf( "Unable to open %s\n", fileName );
43  } /* end else */
44
45  return 0; /* indicates successful termination */
46 } /* end main */
```

change '\t' to ' ' and write to the temp file

read each character from the temp file, and put it to input file

More on Files (Cont.)

- Example: fig14_06.c

```
^_^ mftsai@MBP [~/Classes/CP1_1001/16/codes] ./a.out
This program changes tabs to spaces.
Enter a file to be modified: file.txt

The file before modification is:
a	b
a	b
a	b
a	b
a	b


The file after modification is:
a b
a b
a b
a b
a b
```

Signal Handling

- An external asynchronous event, or signal, can cause a program to terminate prematurely.
- Some events include interrupts (e.g., **`Ctrl+c`**), illegal instructions, segmentation violations, and floating-point exceptions (division by zero or multiplying large floating-point values).

Signal Handling (Cont.)

- The signal handling library (**`<signal.h>`**) provides the capability to trap unexpected events with function signal.
- Function signal receives two arguments—an integer signal number and a pointer to the signal handling function.
- Signals can be generated by function raise which takes an integer signal number as an argument.

Signal Handling (Cont.)

Signal	Explanation
SIGABRT	Abnormal termination of the program (such as a call to function abort).
SIGFPE	An erroneous arithmetic operation, such as a divide by zero or an operation resulting in overflow.
SIGILL	Detection of an illegal instruction.
SIGINT	Receipt of an interactive attention signal.
SIGSEGV	An invalid access to storage.
SIGTERM	A termination request set to the program.

Signal Handling (Cont.)

- Example: fig14_08.c

```c
void signalHandler( int signalValue ); /* prototype */

int main( void ) {
    int i; /* counter used to loop 100 times */
    int x; /* variable to hold random values between 1-50 */

    signal( SIGINT, signalHandler ); /* register signal handler */
    srand( time( NULL ) );

    /* output numbers 1 to 100 */
    for ( i = 1; i <= 100; i++ ) {
        x = 1 + rand() % 50; /* generate random number to raise SIGINT */

        /* raise SIGINT when x is 25 */
        if ( x == 25 ) {
            raise( SIGINT );
        } /* end if */

        printf( "%4d", i );

        /* output \n when i is a multiple of 10 */
        if ( i % 10 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Signal Handling (Cont.)

- Example: fig14_08.c

```
 8 void signalHandler( int signalValue ); /* prototype */
 9
10 int main( void ) {
11    int i; /* counter used to loop 100 times */
12    int x; /* variable to hold random values between 1-50 */
13
14    signal( SIGINT, signalHandler ); /* register signal handler */
15    srand( time( NULL ) );
16
17    /* output numbers 1 to 100 */
18    for ( i = 1; i <= 100; i++ ) {
19       x = 1 + rand() % 50; /* generate random number to raise SIGINT */
20
21       /* raise SIGINT when x is 25 */
22       if ( x == 25 ) {
23          raise( SIGINT );
24       } /* end if */
25
26       printf( "%4d", i );
27
28       /* output \n when i is a multiple of 10 */
29       if ( i % 10 == 0 ) {
30          printf( "\n" );
31       } /* end if */
32    } /* end for */
33
34    return 0; /* indicates successful termination */
35 } /* end main */
```

Signal Handling (Cont.)

- Example: fig14_08.c

```c
void signalHandler( int signalValue ); /* prototype */

int main( void ) {
    int i; /* counter used to loop 100 times */
    int x; /* variable to hold random values between 1-50 */

    signal( SIGINT, signalHandler ); /* register signal handler */
    srand( time( NULL ) );

    /* output numbers 1 to 100 */
    for ( i = 1; i <= 100; i++ ) {
        x = 1 + rand() % 50; /* generate random number to raise SIGINT */

        /* raise SIGINT when x is 25 */
        if ( x == 25 ) {
            raise( SIGINT );
        } /* end if */

        printf( "%4d", i );

        /* output \n when i is a multiple of 10 */
        if ( i % 10 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

define a function prototype

Signal Handling (Cont.)

- Example: fig14_08.c

```c
void signalHandler( int signalValue ); /* prototype */

int main( void ) {
    int i; /* counter used to loop 100 times */
    int x; /* variable to hold random values between 1-50 */

    signal( SIGINT, signalHandler ); /* register signal handler */
    srand( time( NULL ) );

    /* output numbers 1 to 100 */
    for ( i = 1; i <= 100; i++ ) {
        x = 1 + rand() % 50; /* generate random number to raise SIGINT */

        /* raise SIGINT when x is 25 */
        if ( x == 25 ) {
            raise( SIGINT );
        } /* end if */

        printf( "%4d", i );

        /* output \n when i is a multiple of 10 */
        if ( i % 10 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

define a function prototype

Signal Handling (Cont.)

- Example: fig14_08.c

```c
void signalHandler( int signalValue ); /* prototype */

int main( void ) {
    int i; /* counter used to loop 100 times */
    int x; /* variable to hold random values between 1-50 */

    signal( SIGINT, signalHandler ); /* register signal handler */
    srand( time( NULL ) );

    /* output numbers 1 to 100 */
    for ( i = 1; i <= 100; i++ ) {
        x = 1 + rand() % 50; /* generate random number to raise SIGINT */

        /* raise SIGINT when x is 25 */
        if ( x == 25 ) {
            raise( SIGINT );
        } /* end if */

        printf( "%4d", i );

        /* output \n when i is a multiple of 10 */
        if ( i % 10 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

define a function prototype

register signal handler

Signal Handling (Cont.)

- Example: fig14_08.c

```
 8 void signalHandler( int signalValue ); /* prototype */
 9
10 int main( void ) {
11    int i; /* counter used to loop 100 times */
12    int x; /* variable to hold random values between 1-50 */
13
14    signal( SIGINT, signalHandler ); /* register signal handler */
15    srand( time( NULL ) );
16
17    /* output numbers 1 to 100 */
18    for ( i = 1; i <= 100; i++ ) {
19       x = 1 + rand() % 50; /* generate random number to raise SIGINT */
20
21       /* raise SIGINT when x is 25 */
22       if ( x == 25 ) {
23          raise( SIGINT );
24       } /* end if */
25
26       printf( "%4d", i );
27
28       /* output \n when i is a multiple of 10 */
29       if ( i % 10 == 0 ) {
30          printf( "\n" );
31       } /* end if */
32    } /* end for */
33
34    return 0; /* indicates successful termination */
35 } /* end main */
```

define a function prototype

register signal handler

Signal Handling (Cont.)

- Example: fig14_08.c

```
 8 void signalHandler( int signalValue ); /* prototype */
 9
10 int main( void ) {
11    int i; /* counter used to loop 100 times */
12    int x; /* variable to hold random values between 1-50 */
13
14    signal( SIGINT, signalHandler ); /* register signal handler */
15    srand( time( NULL ) );
16
17    /* output numbers 1 to 100 */
18    for ( i = 1; i <= 100; i++ ) {
19       x = 1 + rand() % 50; /* generate random number to raise SIGINT */
20
21       /* raise SIGINT when x is 25 */
22       if ( x == 25 ) {
23          raise( SIGINT );
24       } /* end if */
25
26       printf( "%4d", i );
27
28       /* output \n when i is a multiple of 10 */
29       if ( i % 10 == 0 ) {
30          printf( "\n" );
31       } /* end if */
32    } /* end for */
33
34    return 0; /* indicates successful termination */
35 } /* end main */
```

define a function prototype

register signal handler

raise SIGINT

Signal Handling (Cont.)

- Example: fig14_08.c

```
38 void signalHandler( int signalValue ) {
39     int response; /* user's response to signal (1 or 2) */
40
41     printf( "%s%d%s\n%s",
42             "\nInterrupt signal ( ", signalValue, " ) received.",
43             "Do you wish to continue ( 1 = yes or 2 = no )? " );
44
45     scanf( "%d", &response );
46
47     /* check for invalid responses */
48     while ( response != 1 && response != 2 ) {
49         printf( "( 1 = yes or 2 = no )? " );
50         scanf( "%d", &response );
51     } /* end while */
52
53     /* determine if it is time to exit */
54     if ( response == 1 ) {
55         /* reregister signal handler for next SIGINT */
56         signal( SIGINT, signalHandler );
57     } /* end if */
58     else {
59         exit( EXIT_SUCCESS );
60     } /* end else */
61 } /* end function signalHandler */
```

Signal Handling (Cont.)

- Example: fig14_08.c

```
38 void signalHandler( int signalValue ) {
39    int response; /* user's response to signal (1 or 2) */
40
41    printf( "%s%d%s\n%s",
42            "\nInterrupt signal ( ", signalValue, " ) received.",
43            "Do you wish to continue ( 1 = yes or 2 = no )? " );
44
45    scanf( "%d", &response );
46
47    /* check for invalid responses */
48    while ( response != 1 && response != 2 ) {
49       printf( "( 1 = yes or 2 = no )? " );
50       scanf( "%d", &response );
51    } /* end while */
52
53    /* determine if it is time to exit */
54    if ( response == 1 ) {
55       /* reregister signal handler for next SIGINT */
56       signal( SIGINT, signalHandler );
57    } /* end if */
58    else {
59       exit( EXIT_SUCCESS );
60    } /* end else */
61 } /* end function signalHandler */
```

Signal Handling (Cont.)

- Example: fig14_08.c

```
38 void signalHandler( int signalValue ) {
39    int response; /* user's response to signal (1 or 2) */
40
41    printf( "%s%d%s\n%s",
42            "\nInterrupt signal ( ", signalValue, " ) received.",
43            "Do you wish to continue ( 1 = yes or 2 = no )? " );
44
45    scanf( "%d", &response );
46
47    /* check for invalid responses */
48    while ( response != 1 && response != 2 ) {
49       printf( "( 1 = yes or 2 = no )? " );
50       scanf( "%d", &response );
51    } /* end while */
52
53    /* determine if it is time to exit */
54    if ( response == 1 ) {
55       /* reregister signal handler for next SIGINT */
56       signal( SIGINT, signalHandler );
57    } /* end if */
58    else {
59       exit( EXIT_SUCCESS );
60    } /* end else */
61 } /* end function signalHandler */
```

handle signal

Signal Handling (Cont.)

- Example: fig14_08.c

```
38 void signalHandler( int signalValue ) {
39    int response; /* user's response to signal (1 or 2) */
40
41    printf( "%s%d%s\n%s",
42            "\nInterrupt signal ( ", signalValue, " ) received.",
43            "Do you wish to continue ( 1 = yes or 2 = no )? " );
44
45    scanf( "%d", &response );
46
47    /* check for invalid responses */
48    while ( response != 1 && response != 2 ) {
49       printf( "( 1 = yes or 2 = no )? " );
50       scanf( "%d", &response );
51    } /* end while */
52
53    /* determine if it is time to exit */
54    if ( response == 1 ) {
55       /* reregister signal handler for next SIGINT */
56       signal( SIGINT, signalHandler );
57    } /* end if */
58    else {
59       exit( EXIT_SUCCESS );
60    } /* end else */
61 } /* end function signalHandler */
```

handle signal

Signal Handling (Cont.)

- Example: fig14_08.c

```
38 void signalHandler( int signalValue ) {
39    int response; /* user's response to signal (1 or 2) */
40
41    printf( "%s%d%s\n%s",
42            "\nInterrupt signal ( ", signalValue, " ) received.",
43            "Do you wish to continue ( 1 = yes or 2 = no )? " );
44
45    scanf( "%d", &response );
46
47    /* check for invalid responses */
48    while ( response != 1 && response != 2 ) {
49       printf( "( 1 = yes or 2 = no )? " );
50       scanf( "%d", &response );
51    } /* end while */
52
53    /* determine if it is time to exit */
54    if ( response == 1 ) {
55       /* reregister signal handler for next SIGINT */
56       signal( SIGINT, signalHandler );
57    } /* end if */
58    else {
59       exit( EXIT_SUCCESS );
60    } /* end else */
61 } /* end function signalHandler */
```

handle signal

reregister signal handler for next SIGINT

Signal Handling (Cont.)

- Example: fig14_08.c

```
   1   2   3   4   5   6   7
Interrupt signal ( 2 ) received.
Do you wish to continue ( 1 = yes or 2 = no )? 1
   8   9  10
  11  12  13  14  15  16  17  18  19  20
  21  22  23  24  25  26  27  28  29  30
  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  46  47  48  49  50
  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70
  71  72  73  74  75  76  77  78  79  80

Interrupt signal ( 2 ) received.
Do you wish to continue ( 1 = yes or 2 = no )? 1
  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100
```

Unconditional Branching with `goto`

- Another instance of unstructured programming is the `goto` statement—an unconditional branch.
- The result of the goto statement is a change in the flow of control of the program to the first statement after the `label` specified in the goto statement.

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

label start

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

label start

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

label start

goto end label

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

label start

goto end label

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

label start

goto end label

goto start label

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

label start

goto end label

goto start label

Unconditional Branching with **goto** (Cont.)

- Example: fig14_09.c

```
 5 int main( void ) {
 6     int count = 1; /* initialize count */
 7
 8 start: /* label */
 9
10     if ( count > 10 ) {
11         goto end;
12     } /* end if */
13
14     printf( "%d  ", count );
15     count++;
16
17     goto start; /* goto start on line 9 */
18
19 end: /* label */
20     putchar( '\n' );
21
22     return 0; /* indicates successful termination */
23 } /* end main */
```

label start

goto end label

goto start label

label end

Unconditional Branching with **goto** (Cont.)

Software Engineering Observation 14.3

The `goto` *statement should be used only in performance-oriented applications. The* `goto` *statement is unstructured and can lead to programs that are more difficult to debug, maintain and modify.*