Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Files

Random-Access Files

- A random-access file is normally fixed in length and may be accessed directly (and thus quickly) without searching through other records.
- This makes random-access files appropriate for transaction processing systems that require rapid access to specific data.

Random-Access Files (Cont.)

- Because every record in a random-access file normally has the same length, the exact location of a record relative to the beginning of the file can be calculated as a function of the record key.
- We'll soon see how this facilitates immediate access to specific records, even in large files.

Random-Access Files (Cont.)

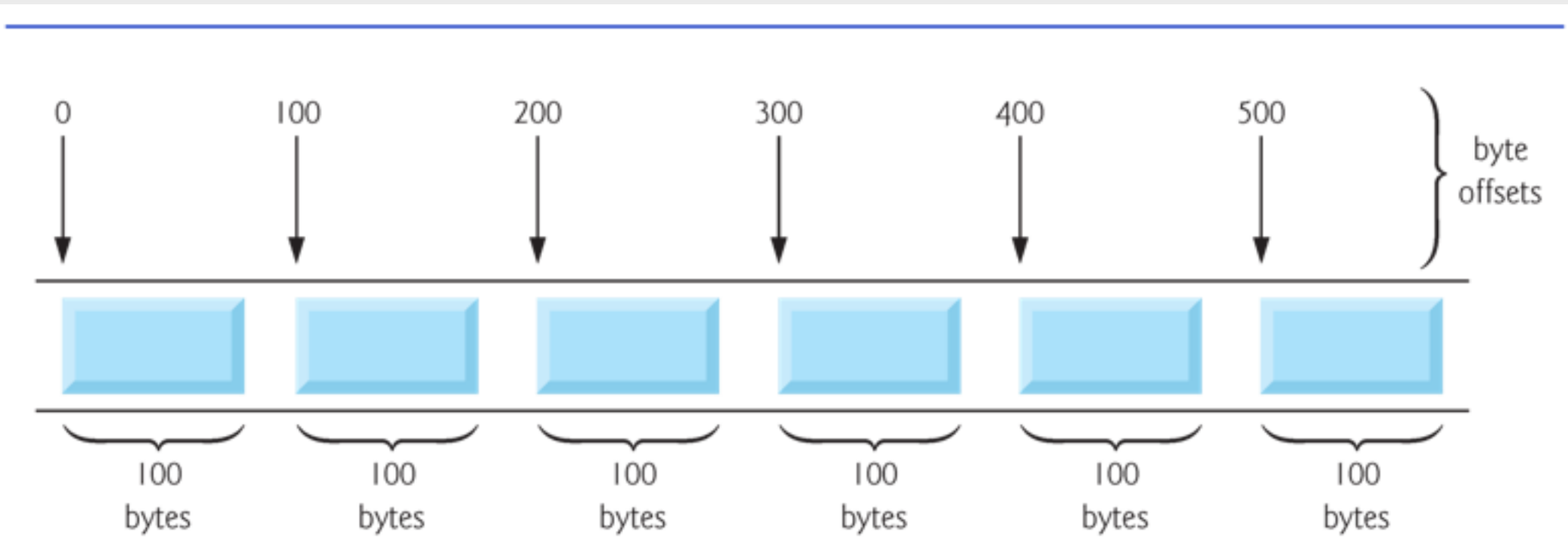


Fig. 11.10 | C's view of a random-access file.

Random-Access Files (Cont.)

- Fixed-length records enable data to be inserted in a random-access file without destroying other data in the file.
- Data stored previously can also be updated or deleted without rewriting the entire file.

Creating a Random-Access File

- **`fwrite()`**
  - transfers a specified number of bytes beginning at a specified location in memory to a file.
- **`fread()`**
  - transfers a specified number of bytes from the location in the file specified by the file position pointer to an area in memory beginning with a specified address.

Creating a Random-Access File (Cont.)

- When writing an integer, instead of using

```
fprintf(fPtr, "%d", number);
```

we can use

```
fwrite(&number, sizeof(int), 1, fPtr);
```

which always writes 4 bytes from a variable number to the file represented by **`fPtr`**

Creating a Random-Access File (Cont.)

- Functions **`fwrite`** and **`fread`** are capable of reading and writing arrays of data to and from disk.
- The third argument of both **`fread`** and **`fwrite`** is the number of elements in the array that should be read from disk or written to disk.
- The preceding **`fwrite`** function call writes a single integer to disk, so the third argument is **`1`** (as if one element of an array is being written).
- Normally, they write one **`struct`** at a time, as we show in the following examples.

Creating a Random-Access File (Cont.)

- Consider the following problem statement:
  - Create a credit processing system capable of storing up to 100 fixed-length records. Each record should consist of an account number that will be used as the record key, a last name, a first name and a balance. The resulting program should be able to update an account, insert a new account record, delete an account and list all the account records in a formatted text file for printing. Use a random-access file.

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
5  /* clientData structure definition */
6  struct clientData {
7      int acctNum; /* account number */
8      char lastName[ 15 ]; /* account last name */
9      char firstName[ 10 ]; /* account first name */
10     double balance; /* account balance */
11 }; /* end structure clientData */
```

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
5  /* clientData structure definition */
6  struct clientData {
7      int acctNum; /* account number */
8      char lastName[ 15 ]; /* account last name */
9      char firstName[ 10 ]; /* account first name */
10     double balance; /* account balance */
11 }; /* end structure clientData */
```

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
5  /* clientData structure definition */
6  struct clientData {
7      int acctNum; /* account number */
8      char lastName[ 15 ]; /* account last name */
9      char firstName[ 10 ]; /* account first name */
10     double balance; /* account balance */
11 }; /* end structure clientData */
```

define a clientData struct

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```c
int main( void ) {
    int i; /* counter used to count from 1-100 */

    /* create clientData with default information */
    struct clientData blankClient = { 0, "", "", 0.0 };

    FILE *cfPtr; /* credit.dat file pointer */

    /* fopen opens the file; exits if file cannot be opened */
    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
        printf( "File could not be opened.\n" );
    } else {
        /* output 100 blank records to file */
        for ( i = 1; i <= 100; i++ ) {
            fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
        } /* end for */

        fclose ( cfPtr ); /* fclose closes the file */
    } /* end else */

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

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```c
int main( void ) {
    int i; /* counter used to count from 1-100 */

    /* create clientData with default information */
    struct clientData blankClient = { 0, "", "", 0.0 };

    FILE *cfPtr; /* credit.dat file pointer */

    /* fopen opens the file; exits if file cannot be opened */
    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
        printf( "File could not be opened.\n" );
    } else {
        /* output 100 blank records to file */
        for ( i = 1; i <= 100; i++ ) {
            fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
        } /* end for */

        fclose ( cfPtr ); /* fclose closes the file */
    } /* end else */

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

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
13 int main( void ) {
14    int i; /* counter used to count from 1-100 */
15
16    /* create clientData with default information */
17    struct clientData blankClient = { 0, "", "", 0.0 };
18
19    FILE *cfPtr; /* credit.dat file pointer */
20
21    /* fopen opens the file; exits if file cannot be opened */
22    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
23       printf( "File could not be opened.\n" );
24    } else {
25       /* output 100 blank records to file */
26       for ( i = 1; i <= 100; i++ ) {
27          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
28       } /* end for */
29
30       fclose ( cfPtr ); /* fclose closes the file */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

default record

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
13 int main( void ) {
14    int i; /* counter used to count from 1-100 */
15
16    /* create clientData with default information */
17    struct clientData blankClient = { 0, "", "", 0.0 };
18
19    FILE *cfPtr; /* credit.dat file pointer */
20
21    /* fopen opens the file; exits if file cannot be opened */
22    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
23       printf( "File could not be opened.\n" );
24    } else {
25       /* output 100 blank records to file */
26       for ( i = 1; i <= 100; i++ ) {
27          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
28       } /* end for */
29
30       fclose ( cfPtr ); /* fclose closes the file */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

default record

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
13 int main( void ) {
14    int i; /* counter used to count from 1-100 */
15
16    /* create clientData with default information */
17    struct clientData blankClient = { 0, "", "", 0.0 };
18
19    FILE *cfPtr; /* credit.dat file pointer */
20
21    /* fopen opens the file; exits if file cannot be opened */
22    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
23       printf( "File could not be opened.\n" );
24    } else {
25       /* output 100 blank records to file */
26       for ( i = 1; i <= 100; i++ ) {
27          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
28       } /* end for */
29
30       fclose ( cfPtr ); /* fclose closes the file */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

default record

open the file pointer

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
13 int main( void ) {
14     int i; /* counter used to count from 1-100 */
15
16     /* create clientData with default information */
17     struct clientData blankClient = { 0, "", "", 0.0 };
18
19     FILE *cfPtr; /* credit.dat file pointer */
20
21     /* fopen opens the file; exits if file cannot be opened */
22     if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
23         printf( "File could not be opened.\n" );
24     } else {
25         /* output 100 blank records to file */
26         for ( i = 1; i <= 100; i++ ) {
27             fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
28         } /* end for */
29
30         fclose ( cfPtr ); /* fclose closes the file */
31     } /* end else */
32
33     return 0; /* indicates successful termination */
34 } /* end main */
```

default record

open the file pointer

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
13 int main( void ) {
14    int i; /* counter used to count from 1-100 */
15
16    /* create clientData with default information */
17    struct clientData blankClient = { 0, "", "", 0.0 };
18
19    FILE *cfPtr; /* credit.dat file pointer */
20
21    /* fopen opens the file; exits if file cannot be opened */
22    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
23       printf( "File could not be opened.\n" );
24    } else {
25       /* output 100 blank records to file */
26       for ( i = 1; i <= 100; i++ ) {
27          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
28       } /* end for */
29
30       fclose ( cfPtr ); /* fclose closes the file */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

default record

open the file pointer

write 100 records into the file

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
13 int main( void ) {
14    int i; /* counter used to count from 1-100 */
15
16    /* create clientData with default information */
17    struct clientData blankClient = { 0, "", "", 0.0 };
18
19    FILE *cfPtr; /* credit.dat file pointer */
20
21    /* fopen opens the file; exits if file cannot be opened */
22    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
23       printf( "File could not be opened.\n" );
24    } else {
25       /* output 100 blank records to file */
26       for ( i = 1; i <= 100; i++ ) {
27          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
28       } /* end for */
29
30       fclose ( cfPtr ); /* fclose closes the file */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

default record

open the file pointer

write 100 records into the file

Creating a Random-Access File (Cont.)

- Example: fig11_11.c

```
13 int main( void ) {
14    int i; /* counter used to count from 1-100 */
15
16    /* create clientData with default information */
17    struct clientData blankClient = { 0, "", "", 0.0 };
18
19    FILE *cfPtr; /* credit.dat file pointer */
20
21    /* fopen opens the file; exits if file cannot be opened */
22    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
23       printf( "File could not be opened.\n" );
24    } else {
25       /* output 100 blank records to file */
26       for ( i = 1; i <= 100; i++ ) {
27          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
28       } /* end for */
29
30       fclose ( cfPtr ); /* fclose closes the file */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34 } /* end main */
```

default record

open the file pointer

write 100 records into the file

close the file stream

Creating a Random-Access File (Cont.)

- Function **`fwrite`** writes a block (specific number of bytes) of data to a file.
- The operator **`sizeof`** returns the size in bytes of its operand in parentheses (in this case **`struct clientData`**).
- To write several array elements, supply in the call to **`fwrite`** a pointer to an array as the first argument and the number of elements to be written as the third argument.
- Writing a single object is equivalent to writing one element of an array, hence the **`1`** in the **`fwrite`** call.

Writing Data Randomly to a Random-Access File

- Use the combination of **`fseek`** and **`fwrite`** to store data at specific locations in the file.
- Function **`fseek`** sets the file position pointer to a specific position in the file, then **`fwrite`** writes the data.

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
 6 struct clientData {
 7     int acctNum; /* account number */
 8     char lastName[ 15 ]; /* account last name */
 9     char firstName[ 10 ]; /* account first name */
10     double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14     FILE *cfPtr; /* credit.dat file pointer */
15
16     /* create clientData with default information */
17     struct clientData client = { 0, "", "", 0.0 };
18
19     /* fopen opens the file; exits if file cannot be opened */
20     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
21         printf( "File could not be opened.\n" );
```

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
 6 struct clientData {
 7     int acctNum; /* account number */
 8     char lastName[ 15 ]; /* account last name */
 9     char firstName[ 10 ]; /* account first name */
10     double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14     FILE *cfPtr; /* credit.dat file pointer */
15
16     /* create clientData with default information */
17     struct clientData client = { 0, "", "", 0.0 };
18
19     /* fopen opens the file; exits if file cannot be opened */
20     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
21         printf( "File could not be opened.\n" );
```

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
 6 struct clientData {
 7     int acctNum; /* account number */
 8     char lastName[ 15 ]; /* account last name */
 9     char firstName[ 10 ]; /* account first name */
10     double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14     FILE *cfPtr; /* credit.dat file pointer */
15
16     /* create clientData with default information */
17     struct clientData client = { 0, "", "", 0.0 };
18
19     /* fopen opens the file; exits if file cannot be opened */
20     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
21         printf( "File could not be opened.\n" );
```

the same struct

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
 6 struct clientData {
 7     int acctNum; /* account number */
 8     char lastName[ 15 ]; /* account last name */
 9     char firstName[ 10 ]; /* account first name */
10     double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14     FILE *cfPtr; /* credit.dat file pointer */
15
16     /* create clientData with default information */
17     struct clientData client = { 0, "", "", 0.0 };
18
19     /* fopen opens the file; exits if file cannot be opened */
20     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
21         printf( "File could not be opened.\n" );
```

the same struct

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
 6 struct clientData {
 7     int acctNum; /* account number */
 8     char lastName[ 15 ]; /* account last name */
 9     char firstName[ 10 ]; /* account first name */
10     double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14     FILE *cfPtr; /* credit.dat file pointer */
15
16     /* create clientData with default information */
17     struct clientData client = { 0, "", "", 0.0 };
18
19     /* fopen opens the file; exits if file cannot be opened */
20     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
21         printf( "File could not be opened.\n" );
```

the same struct

open the file with "**rb+**"

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
23        /* require user to specify account number */
24        printf( "Enter account number"
25                " ( 1 to 100, 0 to end input )\n? " );
26        scanf( "%d", &client.acctNum );
27
28        /* user enters information, which is copied into file */
29        while ( client.acctNum != 0 ) {
30           /* user enters last name, first name and balance */
31           printf( "Enter lastname, firstname, balance\n? " );
32
33           /* set record lastName, firstName and balance value */
34           fscanf( stdin, "%s%s%lf", client.lastName,
35                   client.firstName, &client.balance );
36
37           /* seek position in file to user-specified record */
38           fseek( cfPtr, ( client.acctNum - 1 ) *
39                   sizeof( struct clientData ), SEEK_SET );
40
41           /* write user-specified information in file */
42           fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
43
44           /* enable user to input another account number */
45           printf( "Enter account number\n? " );
46           scanf( "%d", &client.acctNum );
47        } /* end while */
48
49        fclose( cfPtr ); /* fclose closes the file */
```

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
23        /* require user to specify account number */
24        printf( "Enter account number"
25                " ( 1 to 100, 0 to end input )\n? " );
26        scanf( "%d", &client.acctNum );
27
28        /* user enters information, which is copied into file */
29        while ( client.acctNum != 0 ) {
30           /* user enters last name, first name and balance */
31           printf( "Enter lastname, firstname, balance\n? " );
32
33           /* set record lastName, firstName and balance value */
34           fscanf( stdin, "%s%s%lf", client.lastName,
35                   client.firstName, &client.balance );
36
37           /* seek position in file to user-specified record */
38           fseek( cfPtr, ( client.acctNum - 1 ) *
39                   sizeof( struct clientData ), SEEK_SET );
40
41           /* write user-specified information in file */
42           fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
43
44           /* enable user to input another account number */
45           printf( "Enter account number\n? " );
46           scanf( "%d", &client.acctNum );
47        } /* end while */
48
49        fclose( cfPtr ); /* fclose closes the file */
```

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
23        /* require user to specify account number */
24        printf( "Enter account number"
25                " ( 1 to 100, 0 to end input )\n? " );
26        scanf( "%d", &client.acctNum );
27
28        /* user enters information, which is copied into file */
29        while ( client.acctNum != 0 ) {
30           /* user enters last name, first name and balance */
31           printf( "Enter lastname, firstname, balance\n? " );
32
33           /* set record lastName, firstName and balance value */
34           fscanf( stdin, "%s%s%lf", client.lastName,
35                   client.firstName, &client.balance );
36
37           /* seek position in file to user-specified record */
38           fseek( cfPtr, ( client.acctNum - 1 ) *
39                   sizeof( struct clientData ), SEEK_SET );
40
41           /* write user-specified information in file */
42           fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
43
44           /* enable user to input another account number */
45           printf( "Enter account number\n? " );
46           scanf( "%d", &client.acctNum );
47        } /* end while */
48
49        fclose( cfPtr ); /* fclose closes the file */
```

seek position in file to user-specified record

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
23        /* require user to specify account number */
24        printf( "Enter account number"
25                " ( 1 to 100, 0 to end input )\n? " );
26        scanf( "%d", &client.acctNum );
27
28        /* user enters information, which is copied into file */
29        while ( client.acctNum != 0 ) {
30           /* user enters last name, first name and balance */
31           printf( "Enter lastname, firstname, balance\n? " );
32
33           /* set record lastName, firstName and balance value */
34           fscanf( stdin, "%s%s%lf", client.lastName,
35                   client.firstName, &client.balance );
36
37           /* seek position in file to user-specified record */
38           fseek( cfPtr, ( client.acctNum - 1 ) *
39                   sizeof( struct clientData ), SEEK_SET );
40
41           /* write user-specified information in file */
42           fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
43
44           /* enable user to input another account number */
45           printf( "Enter account number\n? " );
46           scanf( "%d", &client.acctNum );
47        } /* end while */
48
49        fclose( cfPtr ); /* fclose closes the file */
```

seek position in file to user-specified record

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
23        /* require user to specify account number */
24        printf( "Enter account number"
25                " ( 1 to 100, 0 to end input )\n? " );
26        scanf( "%d", &client.acctNum );
27
28        /* user enters information, which is copied into file */
29        while ( client.acctNum != 0 ) {
30           /* user enters last name, first name and balance */
31           printf( "Enter lastname, firstname, balance\n? " );
32
33           /* set record lastName, firstName and balance value */
34           fscanf( stdin, "%s%s%lf", client.lastName,
35                   client.firstName, &client.balance );
36
37           /* seek position in file to user-specified record */
38           fseek( cfPtr, ( client.acctNum - 1 ) *
39                   sizeof( struct clientData ), SEEK_SET );
40
41           /* write user-specified information in file */
42           fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
43
44           /* enable user to input another account number */
45           printf( "Enter account number\n? " );
46           scanf( "%d", &client.acctNum );
47        } /* end while */
48
49        fclose( cfPtr ); /* fclose closes the file */
```

seek position in file to user-specified record

write user-specific information in file

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_12.c

```
Enter account number ( 1 to 100, 0 to end input )
? 10
Enter lastname, firstname, balance
? Wang, Mike, 100.0
Enter account number
? 20
Enter lastname, firstname, balance
? Tan, Tom, 200.0
Enter account number
? 0
```

Writing Data Randomly to a Random-Access File (Cont.)

- The value of this expression is called the offset or the displacement.

```
(client.accountNum - 1 ) *
sizeof(struct clientData)
```

- Because the account number is between 1 and 100 but the byte positions in the file start with 0, 1 is subtracted from the account number when calculating the byte location of the record.

Writing Data Randomly to a Random-Access File (Cont.)

- Thus, for record 1, the file position pointer is set to byte 0 of the file.
- **`SEEK_SET`**
  - The symbolic constant indicates that the file position pointer is positioned relative to the beginning of the file by the amount of the offset.

Writing Data Randomly to a Random-Access File (Cont.)

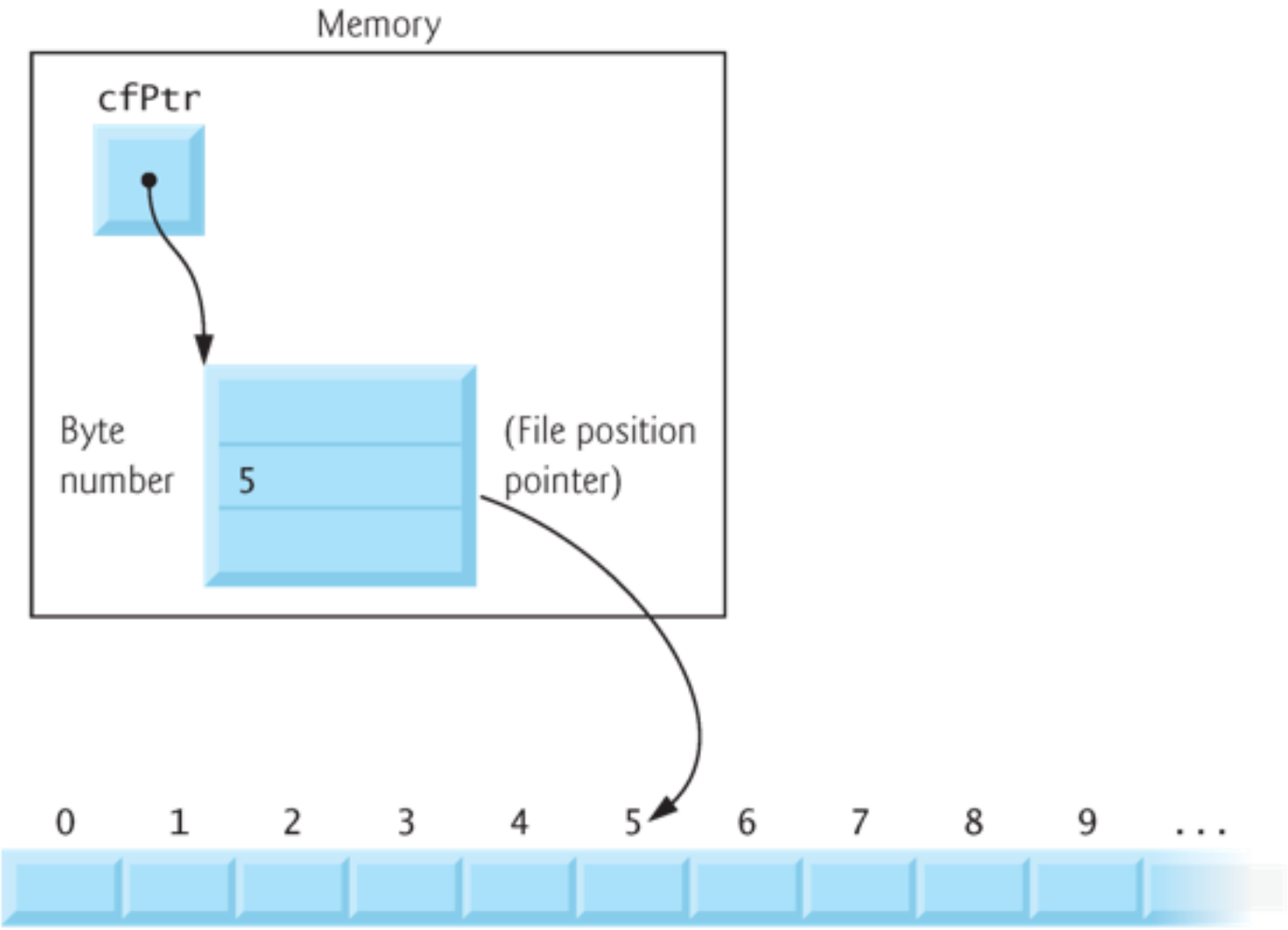


Fig. 11.14 | File position pointer indicating an offset of 5 bytes from the beginning of the file.

Writing Data Randomly to a Random-Access File (Cont.)

- The function prototype for **`fseek`** is

```
int fseek(FILE *stream, long int offset,
          int whence );
```

  where **`offset`** is the number of bytes to seek from location whence in the file pointed to by **`stream`**.

- The argument **`whence`** can have one of three values—**`SEEK_SET`**, **`SEEK_CUR`** or **`SEEK_END`** (all defined in **`<stdio.h>`**)—indicating the location in the file from which the seek begins.

Writing Data Randomly to a Random-Access File (Cont.)

- **`SEEK_SET`**
  - the seek starts at the beginning of the file;
- **`SEEK_CUR`**
  - the seek starts at the current location in the file;
- **`SEEK_END`**
  - the seek starts at the end of the file.

Writing Data Randomly to a Random-Access File (Cont.)

- Function **`fseek`** returns -1 if the seek operation cannot be performed.
- Function **`fwrite`** returns the number of items it successfully output.
  - If this number is less than the third argument in the function call, then a write error occurred.

Writing Data Randomly to a Random-Access File (Cont.)

- Function **`fread`** reads a specified number of bytes from a file into memory.
- For example,

```
fread(&client, sizeof(struct
clientData), 1, cfPtr);
```

- Function **`fread`** can be used to read several fixed-size array elements by providing a pointer to the array in which the elements will be stored and by indicating the number of elements to be read.
- If this number is less than the third argument in the function call, then a read error occurred.

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```
 6 struct clientData {
 7    int acctNum; /* account number */
 8    char lastName[ 15 ]; /* account last name */
 9    char firstName[ 10 ]; /* account first name */
10    double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14    FILE *cfPtr; /* credit.dat file pointer */
15
16    /* create clientData with default information */
17    struct clientData client = { 0, "", "", 0.0 };
18
19    /* fopen opens the file; exits if file cannot be opened */
20    if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
21       printf( "File could not be opened.\n" );
22    } else {
```

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```
6  struct clientData {
7     int acctNum; /* account number */
8     char lastName[ 15 ]; /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14    FILE *cfPtr; /* credit.dat file pointer */
15
16    /* create clientData with default information */
17    struct clientData client = { 0, "", "", 0.0 };
18
19    /* fopen opens the file; exits if file cannot be opened */
20    if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
21       printf( "File could not be opened.\n" );
22    } else {
```

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```c
struct clientData {
   int acctNum; /* account number */
   char lastName[ 15 ]; /* account last name */
   char firstName[ 10 ]; /* account first name */
   double balance; /* account balance */
}; /* end structure clientData */

int main( void ) {
   FILE *cfPtr; /* credit.dat file pointer */

   /* create clientData with default information */
   struct clientData client = { 0, "", "", 0.0 };

   /* fopen opens the file; exits if file cannot be opened */
   if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
   } else {
```

the same struct

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```
6  struct clientData {
7     int acctNum; /* account number */
8     char lastName[ 15 ]; /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14    FILE *cfPtr; /* credit.dat file pointer */
15
16    /* create clientData with default information */
17    struct clientData client = { 0, "", "", 0.0 };
18
19    /* fopen opens the file; exits if file cannot be opened */
20    if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
21       printf( "File could not be opened.\n" );
22    } else {
```

the same struct

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```
 6 struct clientData {
 7    int acctNum; /* account number */
 8    char lastName[ 15 ]; /* account last name */
 9    char firstName[ 10 ]; /* account first name */
10    double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main( void ) {
14    FILE *cfPtr; /* credit.dat file pointer */
15
16    /* create clientData with default information */
17    struct clientData client = { 0, "", "", 0.0 };
18
19    /* fopen opens the file; exits if file cannot be opened */
20    if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
21       printf( "File could not be opened.\n" );
22    } else {
```

the same struct

open the file with "**rb**"

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```
23       printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
24          "First Name", "Balance" );
25
26       /* read all records from file (until eof) */
27       while ( !feof( cfPtr ) ) {
28          fread( &client, sizeof( struct clientData ), 1, cfPtr );
29
30          /* display record */
31          if ( client.acctNum != 0 ) {
32             printf( "%-6d%-16s%-11s%10.2f\n",
33                client.acctNum, client.lastName,
34                client.firstName, client.balance );
35          } /* end if */
36       } /* end while */
37
38       fclose( cfPtr ); /* fclose closes the file */
39    } /* end else */
```

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```
23       printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
24          "First Name", "Balance" );
25
26       /* read all records from file (until eof) */
27       while ( !feof( cfPtr ) ) {
28          fread( &client, sizeof( struct clientData ), 1, cfPtr );
29
30          /* display record */
31          if ( client.acctNum != 0 ) {
32             printf( "%-6d%-16s%-11s%10.2f\n",
33                client.acctNum, client.lastName,
34                client.firstName, client.balance );
35          } /* end if */
36       } /* end while */
37
38       fclose( cfPtr ); /* fclose closes the file */
39    } /* end else */
```

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```
23       printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
24          "First Name", "Balance" );
25
26       /* read all records from file (until eof) */
27       while ( !feof( cfPtr ) ) {
28          fread( &client, sizeof( struct clientData ), 1, cfPtr );
29
30          /* display record */
31          if ( client.acctNum != 0 ) {
32             printf( "%-6d%-16s%-11s%10.2f\n",
33                client.acctNum, client.lastName,
34                client.firstName, client.balance );
35          } /* end if */
36       } /* end while */
37
38       fclose( cfPtr ); /* fclose closes the file */
39    } /* end else */
```

read all records from file (until eof)

Writing Data Randomly to a Random-Access File (Cont.)

- Example: fig11_15.c

```
23       printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
24          "First Name", "Balance" );
25
26       /* read all records from file (until eof) */
27       while ( !feof( cfPtr ) ) {
28          fread( &client, sizeof( struct clientData ), 1, cfPtr );
29
30          /* display record */
31          if ( client.acctNum != 0 ) {
32             printf( "%-6d%-16s%-11s%10.2f\n",
33                client.acctNum, client.lastName,
34                client.firstName, client.balance );
35          } /* end if */
36       } /* end while */
37
38       fclose( cfPtr ); /* fclose closes the file */
39    } /* end else */
```

read all records from file (until eof)

```
Acct  Last Name       First Name    Balance
10    Wang,           Mike,          100.00
20    Tan,            Tom,           200.00
```

Case Study: Transaction-Processing Program

- Present a substantial transaction-processing program using random-access files.
- The program maintains a bank's account information.
- The program updates existing accounts, adds new accounts, deletes accounts and stores a listing of all the current accounts in a text file for printing.

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
 7 /* clientData structure definition */
 8 struct clientData {
 9     int acctNum; /* account number */
10     char lastName[ 15 ]; /* account last name */
11     char firstName[ 10 ]; /* account first name */
12     double balance; /* account balance */
13 }; /* end structure clientData */
14
15 /* prototypes */
16 int enterChoice( void );
17 void textFile( FILE *readPtr );
18 void updateRecord( FILE *fPtr );
19 void newRecord( FILE *fPtr );
20 void deleteRecord( FILE *fPtr );
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
 7 /* clientData structure definition */
 8 struct clientData {
 9    int acctNum; /* account number */
10    char lastName[ 15 ]; /* account last name */
11    char firstName[ 10 ]; /* account first name */
12    double balance; /* account balance */
13 }; /* end structure clientData */
14
15 /* prototypes */
16 int enterChoice( void );
17 void textFile( FILE *readPtr );
18 void updateRecord( FILE *fPtr );
19 void newRecord( FILE *fPtr );
20 void deleteRecord( FILE *fPtr );
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
/* clientData structure definition */
struct clientData {
    int acctNum; /* account number */
    char lastName[ 15 ]; /* account last name */
    char firstName[ 10 ]; /* account first name */
    double balance; /* account balance */
}; /* end structure clientData */

/* prototypes */
int enterChoice( void );
void textFile( FILE *readPtr );
void updateRecord( FILE *fPtr );
void newRecord( FILE *fPtr );
void deleteRecord( FILE *fPtr );
```

define a struct

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
22 int main( void ) {
23     FILE *cfPtr; /* credit.dat file pointer */
24     int choice; /* user's choice */
25
26     /* fopen opens the file; exits if file cannot be opened */
27     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
28         printf( "File could not be opened.\n" );
29     } else {
30         /* enable user to specify action */
31         while ( ( choice = enterChoice() ) != 5 ) {
32             switch ( choice ) {
33                 /* create text file from record file */
34                 case 1:
35                     textFile( cfPtr );
36                     break;
37                     /* update record */
38                 case 2:
39                     updateRecord( cfPtr );
40                     break;
41                     /* create record */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
22 int main( void ) {
23     FILE *cfPtr; /* credit.dat file pointer */
24     int choice; /* user's choice */
25
26     /* fopen opens the file; exits if file cannot be opened */
27     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
28         printf( "File could not be opened.\n" );
29     } else {
30         /* enable user to specify action */
31         while ( ( choice = enterChoice() ) != 5 ) {
32             switch ( choice ) {
33                 /* create text file from record file */
34                 case 1:
35                     textFile( cfPtr );
36                     break;
37                     /* update record */
38                 case 2:
39                     updateRecord( cfPtr );
40                     break;
41                     /* create record */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
22 int main( void ) {
23     FILE *cfPtr; /* credit.dat file pointer */
24     int choice; /* user's choice */
25
26     /* fopen opens the file; exits if file cannot be opened */
27     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
28         printf( "File could not be opened.\n" );
29     } else {
30         /* enable user to specify action */
31         while ( ( choice = enterChoice() ) != 5 ) {
32             switch ( choice ) {
33                 /* create text file from record file */
34                 case 1:
35                     textFile( cfPtr );
36                     break;
37                     /* update record */
38                 case 2:
39                     updateRecord( cfPtr );
40                     break;
41                     /* create record */
```

open the file with "**rb+**"

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
42              case 3:
43                  newRecord( cfPtr );
44                  break;
45                  /* delete existing record */
46              case 4:
47                  deleteRecord( cfPtr );
48                  break;
49                  /* display message if user does not select valid choice */
50              default:
51                  printf( "Incorrect choice\n" );
52                  break;
53          } /* end switch */
54      } /* end while */
55
56      fclose( cfPtr ); /* fclose closes the file */
57  } /* end else */
58
59  return 0; /* indicates successful termination */
60 } /* end main */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
42              case 3:
43                  newRecord( cfPtr );
44                  break;
45                  /* delete existing record */
46              case 4:
47                  deleteRecord( cfPtr );
48                  break;
49                  /* display message if user does not select valid choice */
50              default:
51                  printf( "Incorrect choice\n" );
52                  break;
53          } /* end switch */
54      } /* end while */
55
56      fclose( cfPtr ); /* fclose closes the file */
57  } /* end else */
58
59  return 0; /* indicates successful termination */
60 } /* end main */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
42              case 3:
43                  newRecord( cfPtr );
44                  break;
45                  /* delete existing record */
46              case 4:
47                  deleteRecord( cfPtr );
48                  break;
49                  /* display message if user does not select valid choice */
50              default:
51                  printf( "Incorrect choice\n" );
52                  break;
53          } /* end switch */
54      } /* end while */
55
56      fclose( cfPtr ); /* fclose closes the file */
57  } /* end else */
58
59  return 0; /* indicates successful termination */
60 } /* end main */
```

close the stream

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
63 void textFile( FILE *readPtr ) {
64    FILE *writePtr; /* accounts.txt file pointer */
65
66    /* create clientData with default information */
67    struct clientData client = { 0, "", "", 0.0 };
68
69    /* fopen opens the file; exits if file cannot be opened */
70    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
71       printf( "File could not be opened.\n" );
72    } else {
73       rewind( readPtr ); /* sets pointer to beginning of file */
74       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
75               "Acct", "Last Name", "First Name","Balance" );
76
77       /* copy all records from random-access file into text file */
78       while ( !feof( readPtr ) ) {
79          fread( &client, sizeof( struct clientData ), 1, readPtr );
80
81          /* write single record to text file */
82          if ( client.acctNum != 0 ) {
83             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
84                     client.acctNum, client.lastName,
85                     client.firstName, client.balance );
86          } /* end if */
87       } /* end while */
88
89       fclose( writePtr ); /* fclose closes the file */
90    } /* end else */
91 } /* end function textFile */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
63 void textFile( FILE *readPtr ) {
64    FILE *writePtr; /* accounts.txt file pointer */
65
66    /* create clientData with default information */
67    struct clientData client = { 0, "", "", 0.0 };
68
69    /* fopen opens the file; exits if file cannot be opened */
70    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
71       printf( "File could not be opened.\n" );
72    } else {
73       rewind( readPtr ); /* sets pointer to beginning of file */
74       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
75               "Acct", "Last Name", "First Name","Balance" );
76
77       /* copy all records from random-access file into text file */
78       while ( !feof( readPtr ) ) {
79          fread( &client, sizeof( struct clientData ), 1, readPtr );
80
81          /* write single record to text file */
82          if ( client.acctNum != 0 ) {
83             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
84                     client.acctNum, client.lastName,
85                     client.firstName, client.balance );
86          } /* end if */
87       } /* end while */
88
89       fclose( writePtr ); /* fclose closes the file */
90    } /* end else */
91 } /* end function textFile */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
63 void textFile( FILE *readPtr ) {
64    FILE *writePtr; /* accounts.txt file pointer */
65
66    /* create clientData with default information */
67    struct clientData client = { 0, "", "", 0.0 };
68
69    /* fopen opens the file; exits if file cannot be opened */
70    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
71       printf( "File could not be opened.\n" );
72    } else {
73       rewind( readPtr ); /* sets pointer to beginning of file */
74       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
75                "Acct", "Last Name", "First Name","Balance" );
76
77       /* copy all records from random-access file into text file */
78       while ( !feof( readPtr ) ) {
79          fread( &client, sizeof( struct clientData ), 1, readPtr );
80
81          /* write single record to text file */
82          if ( client.acctNum != 0 ) {
83             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
84                      client.acctNum, client.lastName,
85                      client.firstName, client.balance );
86          } /* end if */
87       } /* end while */
88
89       fclose( writePtr ); /* fclose closes the file */
90    } /* end else */
91 } /* end function textFile */
```

open a file to write text data

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
63 void textFile( FILE *readPtr ) {
64    FILE *writePtr; /* accounts.txt file pointer */
65
66    /* create clientData with default information */
67    struct clientData client = { 0, "", "", 0.0 };
68
69    /* fopen opens the file; exits if file cannot be opened */
70    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
71       printf( "File could not be opened.\n" );
72    } else {
73       rewind( readPtr ); /* sets pointer to beginning of file */
74       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
75               "Acct", "Last Name", "First Name","Balance" );
76
77       /* copy all records from random-access file into text file */
78       while ( !feof( readPtr ) ) {
79          fread( &client, sizeof( struct clientData ), 1, readPtr );
80
81          /* write single record to text file */
82          if ( client.acctNum != 0 ) {
83             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
84                     client.acctNum, client.lastName,
85                     client.firstName, client.balance );
86          } /* end if */
87       } /* end while */
88
89       fclose( writePtr ); /* fclose closes the file */
90    } /* end else */
91 } /* end function textFile */
```

open a file to write text data

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
63 void textFile( FILE *readPtr ) {
64    FILE *writePtr; /* accounts.txt file pointer */
65
66    /* create clientData with default information */
67    struct clientData client = { 0, "", "", 0.0 };
68
69    /* fopen opens the file; exits if file cannot be opened */
70    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
71       printf( "File could not be opened.\n" );
72    } else {
73       rewind( readPtr ); /* sets pointer to beginning of file */
74       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
75             "Acct", "Last Name", "First Name","Balance" );
76
77       /* copy all records from random-access file into text file */
78       while ( !feof( readPtr ) ) {
79          fread( &client, sizeof( struct clientData ), 1, readPtr );
80
81          /* write single record to text file */
82          if ( client.acctNum != 0 ) {
83             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
84                   client.acctNum, client.lastName,
85                   client.firstName, client.balance );
86          } /* end if */
87       } /* end while */
88
89       fclose( writePtr ); /* fclose closes the file */
90    } /* end else */
91 } /* end function textFile */
```

open a file to write text data

set pointer to beginning of file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
63 void textFile( FILE *readPtr ) {
64    FILE *writePtr; /* accounts.txt file pointer */
65
66    /* create clientData with default information */
67    struct clientData client = { 0, "", "", 0.0 };
68
69    /* fopen opens the file; exits if file cannot be opened */
70    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
71       printf( "File could not be opened.\n" );
72    } else {
73       rewind( readPtr ); /* sets pointer to beginning of file */
74       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
75             "Acct", "Last Name", "First Name","Balance" );
76
77       /* copy all records from random-access file into text file */
78       while ( !feof( readPtr ) ) {
79          fread( &client, sizeof( struct clientData ), 1, readPtr );
80
81          /* write single record to text file */
82          if ( client.acctNum != 0 ) {
83             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
84                   client.acctNum, client.lastName,
85                   client.firstName, client.balance );
86          } /* end if */
87       } /* end while */
88
89       fclose( writePtr ); /* fclose closes the file */
90    } /* end else */
91 } /* end function textFile */
```

open a file to write text data

set pointer to beginning of file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
63 void textFile( FILE *readPtr ) {
64    FILE *writePtr; /* accounts.txt file pointer */
65
66    /* create clientData with default information */
67    struct clientData client = { 0, "", "", 0.0 };
68
69    /* fopen opens the file; exits if file cannot be opened */
70    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
71       printf( "File could not be opened.\n" );
72    } else {
73       rewind( readPtr ); /* sets pointer to beginning of file */
74       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
75             "Acct", "Last Name", "First Name","Balance" );
76
77       /* copy all records from random-access file into text file */
78       while ( !feof( readPtr ) ) {
79          fread( &client, sizeof( struct clientData ), 1, readPtr );
80
81          /* write single record to text file */
82          if ( client.acctNum != 0 ) {
83             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
84                   client.acctNum, client.lastName,
85                   client.firstName, client.balance );
86          } /* end if */
87       } /* end while */
88
89       fclose( writePtr ); /* fclose closes the file */
90    } /* end else */
91 } /* end function textFile */
```

open a file to write text data

set pointer to beginning of file

write all records into text file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
94  void updateRecord( FILE *fPtr ) {
95      int account; /* account number */
96      double transaction; /* transaction amount */
97
98      /* create clientData with no information */
99      struct clientData client = { 0, "", "", 0.0 };
100
101     /* obtain number of account to update */
102     printf( "Enter account to update ( 1 - 100 ): " );
103     scanf( "%d", &account );
104
105     /* move file pointer to correct record in file */
106     fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
107             SEEK_SET );
108
109     /* read record from file */
110     fread( &client, sizeof( struct clientData ), 1, fPtr );
111
112     /* display error if account does not exist */
113     if ( client.acctNum == 0 ) {
114         printf( "Acount #%d has no information.\n", account );
115     } else { /* update record */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
94  void updateRecord( FILE *fPtr ) {
95      int account; /* account number */
96      double transaction; /* transaction amount */
97
98      /* create clientData with no information */
99      struct clientData client = { 0, "", "", 0.0 };
100
101     /* obtain number of account to update */
102     printf( "Enter account to update ( 1 - 100 ): " );
103     scanf( "%d", &account );
104
105     /* move file pointer to correct record in file */
106     fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
107            SEEK_SET );
108
109     /* read record from file */
110     fread( &client, sizeof( struct clientData ), 1, fPtr );
111
112     /* display error if account does not exist */
113     if ( client.acctNum == 0 ) {
114         printf( "Acount #%d has no information.\n", account );
115     } else { /* update record */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
94  void updateRecord( FILE *fPtr ) {
95      int account; /* account number */
96      double transaction; /* transaction amount */
97
98      /* create clientData with no information */
99      struct clientData client = { 0, "", "", 0.0 };
100
101     /* obtain number of account to update */
102     printf( "Enter account to update ( 1 - 100 ): " );
103     scanf( "%d", &account );
104
105     /* move file pointer to correct record in file */
106     fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
107            SEEK_SET );
108
109     /* read record from file */
110     fread( &client, sizeof( struct clientData ), 1, fPtr );
111
112     /* display error if account does not exist */
113     if ( client.acctNum == 0 ) {
114         printf( "Acount #%d has no information.\n", account );
115     } else { /* update record */
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
94  void updateRecord( FILE *fPtr ) {
95      int account; /* account number */
96      double transaction; /* transaction amount */
97
98      /* create clientData with no information */
99      struct clientData client = { 0, "", "", 0.0 };
100
101     /* obtain number of account to update */
102     printf( "Enter account to update ( 1 - 100 ): " );
103     scanf( "%d", &account );
104
105     /* move file pointer to correct record in file */
106     fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
107            SEEK_SET );
108
109     /* read record from file */
110     fread( &client, sizeof( struct clientData ), 1, fPtr );
111
112     /* display error if account does not exist */
113     if ( client.acctNum == 0 ) {
114         printf( "Acount #%d has no information.\n", account );
115     } else { /* update record */
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
94  void updateRecord( FILE *fPtr ) {
95      int account; /* account number */
96      double transaction; /* transaction amount */
97
98      /* create clientData with no information */
99      struct clientData client = { 0, "", "", 0.0 };
100
101     /* obtain number of account to update */
102     printf( "Enter account to update ( 1 - 100 ): " );
103     scanf( "%d", &account );
104
105     /* move file pointer to correct record in file */
106     fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
107            SEEK_SET );
108
109     /* read record from file */
110     fread( &client, sizeof( struct clientData ), 1, fPtr );
111
112     /* display error if account does not exist */
113     if ( client.acctNum == 0 ) {
114         printf( "Acount #%d has no information.\n", account );
115     } else { /* update record */
```

move file pointer to correct record in file

read record from file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
116         printf( "%-6d%-16s%-11s%10.2f\n\n",
117                 client.acctNum, client.lastName,
118                 client.firstName, client.balance );
119
120         /* request transaction amount from user */
121         printf( "Enter charge ( + ) or payment ( - ): " );
122         scanf( "%lf", &transaction );
123         client.balance += transaction; /* update record balance */
124
125         printf( "%-6d%-16s%-11s%10.2f\n",
126                 client.acctNum, client.lastName,
127                 client.firstName, client.balance );
128
129         /* move file pointer to correct record in file */
130         fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
131                 SEEK_SET );
132
133         /* write updated record over old record in file */
134         fwrite( &client, sizeof( struct clientData ), 1, fPtr );
135      } /* end else */
136   } /* end function updateRecord */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
116         printf( "%-6d%-16s%-11s%10.2f\n\n",
117                 client.acctNum, client.lastName,
118                 client.firstName, client.balance );
119
120         /* request transaction amount from user */
121         printf( "Enter charge ( + ) or payment ( - ): " );
122         scanf( "%lf", &transaction );
123         client.balance += transaction; /* update record balance */
124
125         printf( "%-6d%-16s%-11s%10.2f\n",
126                 client.acctNum, client.lastName,
127                 client.firstName, client.balance );
128
129         /* move file pointer to correct record in file */
130         fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
131                 SEEK_SET );
132
133         /* write updated record over old record in file */
134         fwrite( &client, sizeof( struct clientData ), 1, fPtr );
135      } /* end else */
136   } /* end function updateRecord */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
116         printf( "%-6d%-16s%-11s%10.2f\n\n",
117                 client.acctNum, client.lastName,
118                 client.firstName, client.balance );
119
120         /* request transaction amount from user */
121         printf( "Enter charge ( + ) or payment ( - ): " );
122         scanf( "%lf", &transaction );
123         client.balance += transaction; /* update record balance */
124
125         printf( "%-6d%-16s%-11s%10.2f\n",
126                 client.acctNum, client.lastName,
127                 client.firstName, client.balance );
128
129         /* move file pointer to correct record in file */
130         fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
131                 SEEK_SET );
132
133         /* write updated record over old record in file */
134         fwrite( &client, sizeof( struct clientData ), 1, fPtr );
135      } /* end else */
136   } /* end function updateRecord */
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
116         printf( "%-6d%-16s%-11s%10.2f\n\n",
117                 client.acctNum, client.lastName,
118                 client.firstName, client.balance );
119
120         /* request transaction amount from user */
121         printf( "Enter charge ( + ) or payment ( - ): " );
122         scanf( "%lf", &transaction );
123         client.balance += transaction; /* update record balance */
124
125         printf( "%-6d%-16s%-11s%10.2f\n",
126                 client.acctNum, client.lastName,
127                 client.firstName, client.balance );
128
129         /* move file pointer to correct record in file */
130         fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
131                 SEEK_SET );
132
133         /* write updated record over old record in file */
134         fwrite( &client, sizeof( struct clientData ), 1, fPtr );
135      } /* end else */
136   } /* end function updateRecord */
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
116         printf( "%-6d%-16s%-11s%10.2f\n\n",
117                 client.acctNum, client.lastName,
118                 client.firstName, client.balance );
119
120         /* request transaction amount from user */
121         printf( "Enter charge ( + ) or payment ( - ): " );
122         scanf( "%lf", &transaction );
123         client.balance += transaction; /* update record balance */
124
125         printf( "%-6d%-16s%-11s%10.2f\n",
126                 client.acctNum, client.lastName,
127                 client.firstName, client.balance );
128
129         /* move file pointer to correct record in file */
130         fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
131                 SEEK_SET );
132
133         /* write updated record over old record in file */
134         fwrite( &client, sizeof( struct clientData ), 1, fPtr );
135      } /* end else */
136   } /* end function updateRecord */
```

move file pointer to correct record in file

write updated record over old record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
139 void deleteRecord( FILE *fPtr ) {
140     struct clientData client; /* stores record read from file */
141     struct clientData blankClient = { 0, "", "", 0 }; /* blank client */
142 
143     int accountNum; /* account number */
144 
145     /* obtain number of account to delete */
146     printf( "Enter account number to delete ( 1 - 100 ): " );
147     scanf( "%d", &accountNum );
148 
149     /* move file pointer to correct record in file */
150     fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
151            SEEK_SET );
152 
153     /* read record from file */
154     fread( &client, sizeof( struct clientData ), 1, fPtr );
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
139 void deleteRecord( FILE *fPtr ) {
140     struct clientData client; /* stores record read from file */
141     struct clientData blankClient = { 0, "", "", 0 }; /* blank client */
142 
143     int accountNum; /* account number */
144 
145     /* obtain number of account to delete */
146     printf( "Enter account number to delete ( 1 - 100 ): " );
147     scanf( "%d", &accountNum );
148 
149     /* move file pointer to correct record in file */
150     fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
151             SEEK_SET );
152 
153     /* read record from file */
154     fread( &client, sizeof( struct clientData ), 1, fPtr );
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
139 void deleteRecord( FILE *fPtr ) {
140     struct clientData client; /* stores record read from file */
141     struct clientData blankClient = { 0, "", "", 0 }; /* blank client */
142
143     int accountNum; /* account number */
144
145     /* obtain number of account to delete */
146     printf( "Enter account number to delete ( 1 - 100 ): " );
147     scanf( "%d", &accountNum );
148
149     /* move file pointer to correct record in file */
150     fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
151            SEEK_SET );
152
153     /* read record from file */
154     fread( &client, sizeof( struct clientData ), 1, fPtr );
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
139 void deleteRecord( FILE *fPtr ) {
140     struct clientData client; /* stores record read from file */
141     struct clientData blankClient = { 0, "", "", 0 }; /* blank client */
142
143     int accountNum; /* account number */
144
145     /* obtain number of account to delete */
146     printf( "Enter account number to delete ( 1 - 100 ): " );
147     scanf( "%d", &accountNum );
148
149     /* move file pointer to correct record in file */
150     fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
151            SEEK_SET );
152
153     /* read record from file */
154     fread( &client, sizeof( struct clientData ), 1, fPtr );
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
139 void deleteRecord( FILE *fPtr ) {
140     struct clientData client; /* stores record read from file */
141     struct clientData blankClient = { 0, "", "", 0 }; /* blank client */
142
143     int accountNum; /* account number */
144
145     /* obtain number of account to delete */
146     printf( "Enter account number to delete ( 1 - 100 ): " );
147     scanf( "%d", &accountNum );
148
149     /* move file pointer to correct record in file */
150     fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
151            SEEK_SET );
152
153     /* read record from file */
154     fread( &client, sizeof( struct clientData ), 1, fPtr );
```

move file pointer to correct record in file

read record from file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
156    /* display error if record does not exist */
157    if ( client.acctNum == 0 ) {
158       printf( "Account %d does not exist.\n", accountNum );
159    } else { /* delete record */
160       /* move file pointer to correct record in file */
161       fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
162              SEEK_SET );
163
164       /* replace existing record with blank record */
165       fwrite( &blankClient,
166               sizeof( struct clientData ), 1, fPtr );
167    } /* end else */
168 } /* end function deleteRecord */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
156    /* display error if record does not exist */
157    if ( client.acctNum == 0 ) {
158       printf( "Account %d does not exist.\n", accountNum );
159    } else { /* delete record */
160       /* move file pointer to correct record in file */
161       fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
162              SEEK_SET );
163
164       /* replace existing record with blank record */
165       fwrite( &blankClient,
166               sizeof( struct clientData ), 1, fPtr );
167    } /* end else */
168 } /* end function deleteRecord */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
156    /* display error if record does not exist */
157    if ( client.acctNum == 0 ) {
158       printf( "Account %d does not exist.\n", accountNum );
159    } else { /* delete record */
160       /* move file pointer to correct record in file */
161       fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
162              SEEK_SET );
163
164       /* replace existing record with blank record */
165       fwrite( &blankClient,
166               sizeof( struct clientData ), 1, fPtr );
167    } /* end else */
168 } /* end function deleteRecord */
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
156    /* display error if record does not exist */
157    if ( client.acctNum == 0 ) {
158       printf( "Account %d does not exist.\n", accountNum );
159    } else { /* delete record */
160       /* move file pointer to correct record in file */
161       fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
162              SEEK_SET );
163
164       /* replace existing record with blank record */
165       fwrite( &blankClient,
166              sizeof( struct clientData ), 1, fPtr );
167    } /* end else */
168 } /* end function deleteRecord */
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
156    /* display error if record does not exist */
157    if ( client.acctNum == 0 ) {
158       printf( "Account %d does not exist.\n", accountNum );
159    } else { /* delete record */
160       /* move file pointer to correct record in file */
161       fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
162              SEEK_SET );
163
164       /* replace existing record with blank record */
165       fwrite( &blankClient,
166               sizeof( struct clientData ), 1, fPtr );
167    } /* end else */
168 } /* end function deleteRecord */
```

move file pointer to correct record in file

replace existing record with blank record

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
171 void newRecord( FILE *fPtr ) {
172    /* create clientData with default information */
173    struct clientData client = { 0, "", "", 0.0 };
174
175    int accountNum; /* account number */
176
177    /* obtain number of account to create */
178    printf( "Enter new account number ( 1 - 100 ): " );
179    scanf( "%d", &accountNum );
180
181    /* move file pointer to correct record in file */
182    fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
183           SEEK_SET );
184
185    /* read record from file */
186    fread( &client, sizeof( struct clientData ), 1, fPtr );
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
171 void newRecord( FILE *fPtr ) {
172    /* create clientData with default information */
173    struct clientData client = { 0, "", "", 0.0 };
174
175    int accountNum; /* account number */
176
177    /* obtain number of account to create */
178    printf( "Enter new account number ( 1 - 100 ): " );
179    scanf( "%d", &accountNum );
180
181    /* move file pointer to correct record in file */
182    fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
183           SEEK_SET );
184
185    /* read record from file */
186    fread( &client, sizeof( struct clientData ), 1, fPtr );
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
171 void newRecord( FILE *fPtr ) {
172    /* create clientData with default information */
173    struct clientData client = { 0, "", "", 0.0 };
174
175    int accountNum; /* account number */
176
177    /* obtain number of account to create */
178    printf( "Enter new account number ( 1 - 100 ): " );
179    scanf( "%d", &accountNum );
180
181    /* move file pointer to correct record in file */
182    fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
183           SEEK_SET );
184
185    /* read record from file */
186    fread( &client, sizeof( struct clientData ), 1, fPtr );
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
171 void newRecord( FILE *fPtr ) {
172    /* create clientData with default information */
173    struct clientData client = { 0, "", "", 0.0 };
174
175    int accountNum; /* account number */
176
177    /* obtain number of account to create */
178    printf( "Enter new account number ( 1 - 100 ): " );
179    scanf( "%d", &accountNum );
180
181    /* move file pointer to correct record in file */
182    fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
183           SEEK_SET );
184
185    /* read record from file */
186    fread( &client, sizeof( struct clientData ), 1, fPtr );
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
171 void newRecord( FILE *fPtr ) {
172     /* create clientData with default information */
173     struct clientData client = { 0, "", "", 0.0 };
174
175     int accountNum; /* account number */
176
177     /* obtain number of account to create */
178     printf( "Enter new account number ( 1 - 100 ): " );
179     scanf( "%d", &accountNum );
180
181     /* move file pointer to correct record in file */
182     fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
183            SEEK_SET );
184
185     /* read record from file */
186     fread( &client, sizeof( struct clientData ), 1, fPtr );
```

move file pointer to correct record in file

read record from file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
188    /* display error if account already exists */
189    if ( client.acctNum != 0 ) {
190       printf( "Account #%d already contains information.\n",
191               client.acctNum );
192    } else { /* create record */
193       /* user enters last name, first name and balance */
194       printf( "Enter lastname, firstname, balance\n? " );
195       scanf( "%s%s%lf", &client.lastName, &client.firstName,
196               &client.balance );
197
198       client.acctNum = accountNum;
199
200       /* move file pointer to correct record in file */
201       fseek( fPtr, ( client.acctNum - 1 ) *
202               sizeof( struct clientData ), SEEK_SET );
203
204       /* insert record in file */
205       fwrite( &client,
206               sizeof( struct clientData ), 1, fPtr );
207    } /* end else */
208 } /* end function newRecord */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
188    /* display error if account already exists */
189    if ( client.acctNum != 0 ) {
190       printf( "Account #%d already contains information.\n",
191               client.acctNum );
192    } else { /* create record */
193       /* user enters last name, first name and balance */
194       printf( "Enter lastname, firstname, balance\n? " );
195       scanf( "%s%s%lf", &client.lastName, &client.firstName,
196               &client.balance );
197
198       client.acctNum = accountNum;
199
200       /* move file pointer to correct record in file */
201       fseek( fPtr, ( client.acctNum - 1 ) *
202               sizeof( struct clientData ), SEEK_SET );
203
204       /* insert record in file */
205       fwrite( &client,
206               sizeof( struct clientData ), 1, fPtr );
207    } /* end else */
208 } /* end function newRecord */
```

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
188    /* display error if account already exists */
189    if ( client.acctNum != 0 ) {
190       printf( "Account #%d already contains information.\n",
191               client.acctNum );
192    } else { /* create record */
193       /* user enters last name, first name and balance */
194       printf( "Enter lastname, firstname, balance\n? " );
195       scanf( "%s%s%lf", &client.lastName, &client.firstName,
196               &client.balance );
197
198       client.acctNum = accountNum;
199
200       /* move file pointer to correct record in file */
201       fseek( fPtr, ( client.acctNum - 1 ) *
202               sizeof( struct clientData ), SEEK_SET );
203
204       /* insert record in file */
205       fwrite( &client,
206               sizeof( struct clientData ), 1, fPtr );
207    } /* end else */
208 } /* end function newRecord */
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
188    /* display error if account already exists */
189    if ( client.acctNum != 0 ) {
190       printf( "Account #%d already contains information.\n",
191               client.acctNum );
192    } else { /* create record */
193       /* user enters last name, first name and balance */
194       printf( "Enter lastname, firstname, balance\n? " );
195       scanf( "%s%s%lf", &client.lastName, &client.firstName,
196               &client.balance );
197
198       client.acctNum = accountNum;
199
200       /* move file pointer to correct record in file */
201       fseek( fPtr, ( client.acctNum - 1 ) *
202               sizeof( struct clientData ), SEEK_SET );
203
204       /* insert record in file */
205       fwrite( &client,
206               sizeof( struct clientData ), 1, fPtr );
207    } /* end else */
208 } /* end function newRecord */
```

move file pointer to correct record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
188    /* display error if account already exists */
189    if ( client.acctNum != 0 ) {
190       printf( "Account #%d already contains information.\n",
191               client.acctNum );
192    } else { /* create record */
193       /* user enters last name, first name and balance */
194       printf( "Enter lastname, firstname, balance\n? " );
195       scanf( "%s%s%lf", &client.lastName, &client.firstName,
196               &client.balance );
197
198       client.acctNum = accountNum;
199
200       /* move file pointer to correct record in file */
201       fseek( fPtr, ( client.acctNum - 1 ) *
202               sizeof( struct clientData ), SEEK_SET );
203
204       /* insert record in file */
205       fwrite( &client,
206               sizeof( struct clientData ), 1, fPtr );
207    } /* end else */
208 } /* end function newRecord */
```

move file pointer to correct record in file

insert record in file

Case Study: Transaction-Processing Program (Cont.)

- Example: fig11_16.c

```
211 int enterChoice( void ) {
212     int menuChoice; /* variable to store user's choice */
213
214     /* display available options */
215     printf( "\nEnter your choice\n"
216             "1 - store a formatted text file of acounts called\n"
217             "    \"accounts.txt\" for printing\n"
218             "2 - update an account\n"
219             "3 - add a new account\n"
220             "4 - delete an account\n"
221             "5 - end program\n? " );
222
223     scanf( "%d", &menuChoice ); /* receive choice from user */
224     return menuChoice;
225 } /* end function enterChoice */
```

C Basic Data Structures

Objectives

- In this chapter, you'll learn
  - To allocate and free memory dynamically for data objects.
  - To form linked data structures using pointers, self-referential structures and recursion.
  - To create and manipulate linked lists, queues, stacks and binary trees.
  - Various important applications of linked data structures.

Introduction

- This chapter introduces dynamic allocation with sizes that grow and shrink at execution time.
- The dynamic allocation can be used for data structures like Linked List, Stack, Queue, and Tree.

Self-Referential Structures

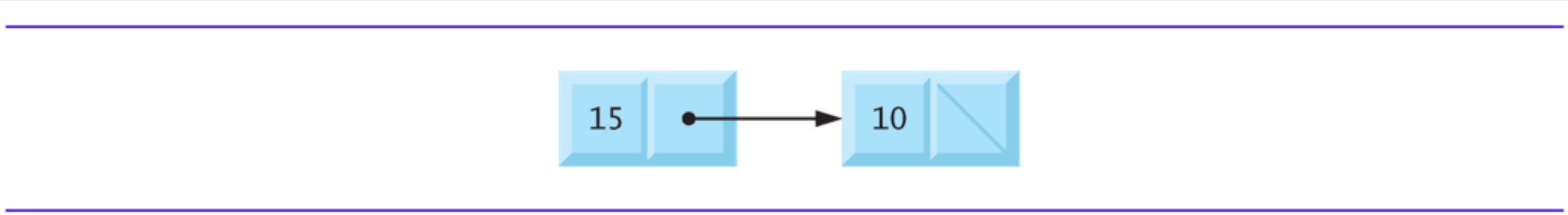
- A self-referential structure contains a pointer member that points to a structure of the same structure type.
- For example, the definition

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

  defines a type, struct node.

- A structure of type struct node has two members—integer member **`data`** and pointer member **`nextPtr`**.

Self-Referential Structures



Dynamic Memory Allocation

- Creating and maintaining dynamic data structures requires dynamic memory allocation—the ability for a program to obtain more memory space at execution time to hold new nodes, and to release space no longer needed.
- Functions **`malloc`** and **`free`**, and operator **`sizeof`**, are essential to dynamic memory allocation.

Dynamic Memory Allocation

- Function **`malloc`** takes as an argument the number of bytes to be allocated and returns a pointer of type **`void *`** (pointer to void) to the allocated memory.
- A **`void *`** is a generic pointer may be assigned to a variable of any pointer type.
- Function **`malloc`** is normally used with the **`sizeof`** operator.

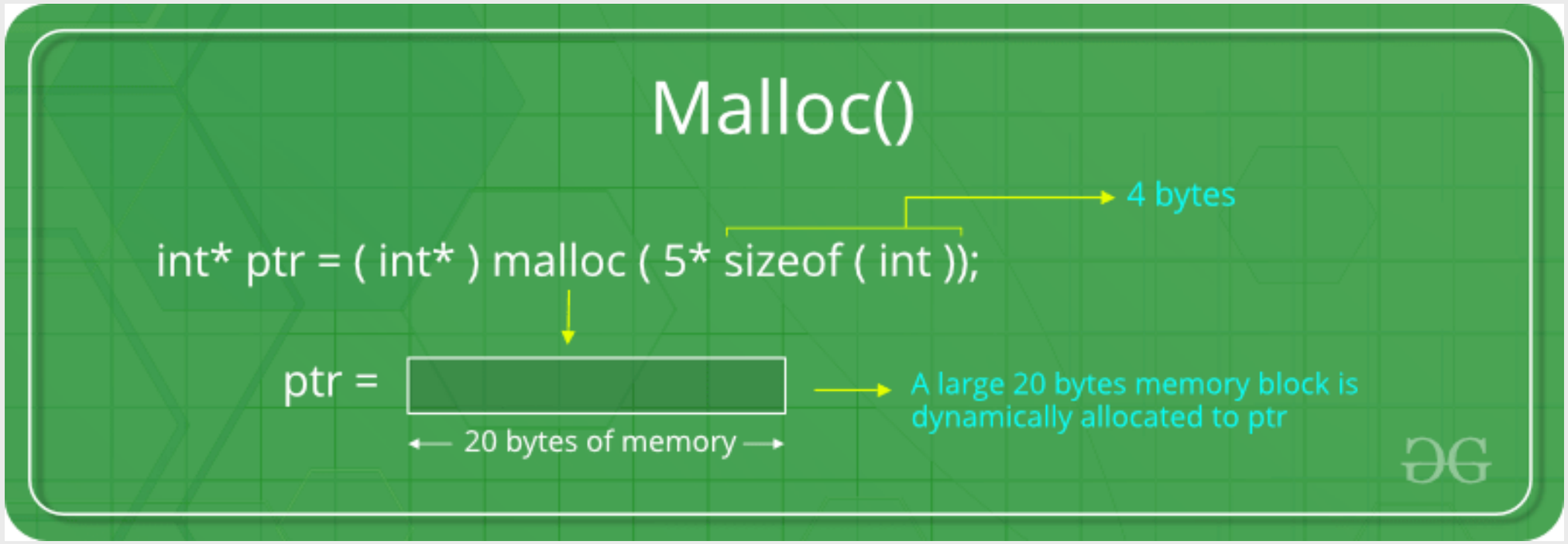
Dynamic Memory Allocation

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

  evaluates **`sizeof( struct node )`** to determine the size in bytes of a structure of type **`struct node`**, allocates a new area in memory of that number of bytes and stores a pointer to the allocated memory in variable **`newPtr`**.

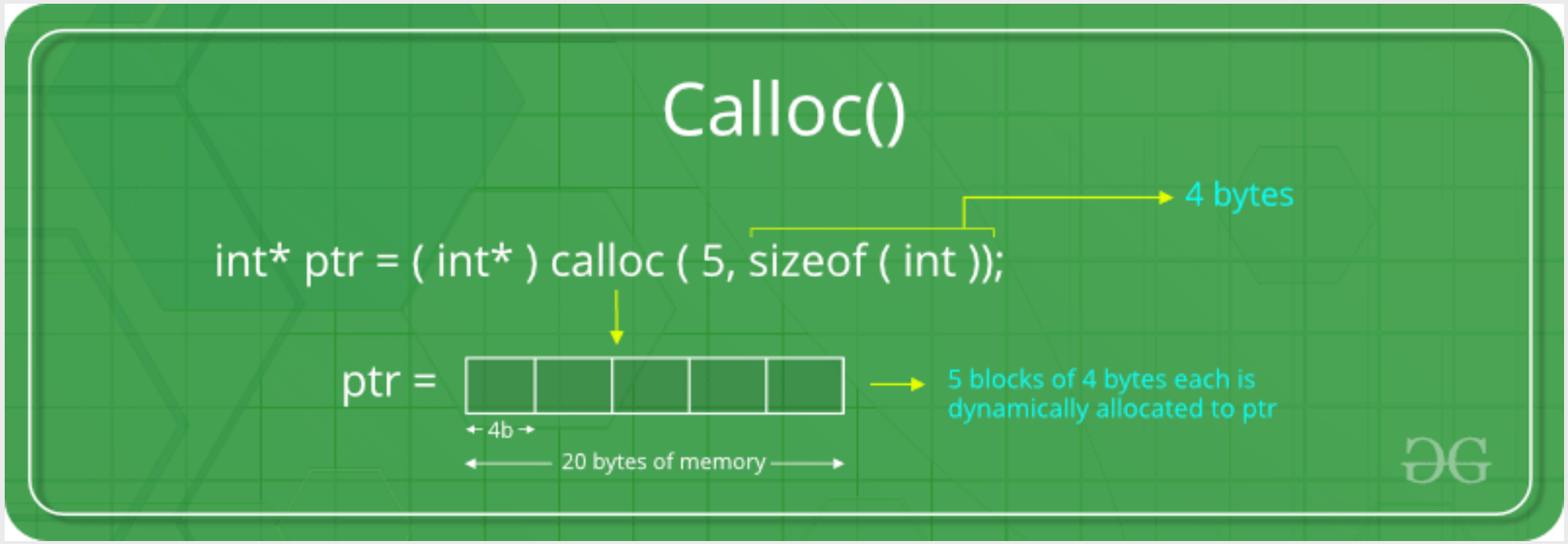
Dynamic Memory Allocation

- Example: fig12_01-1.c



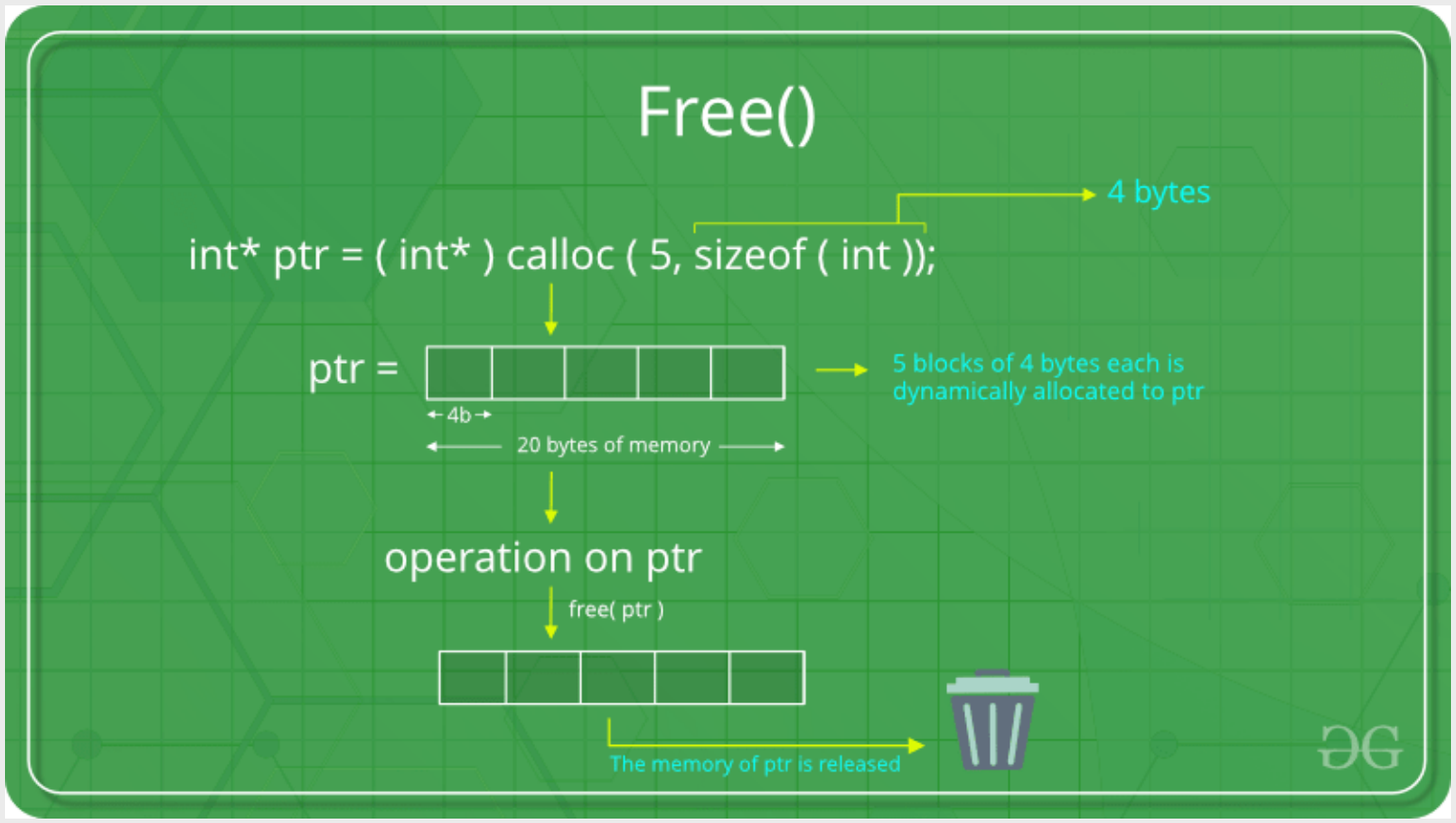
Dynamic Memory Allocation

- Example: fig12_01-2.c



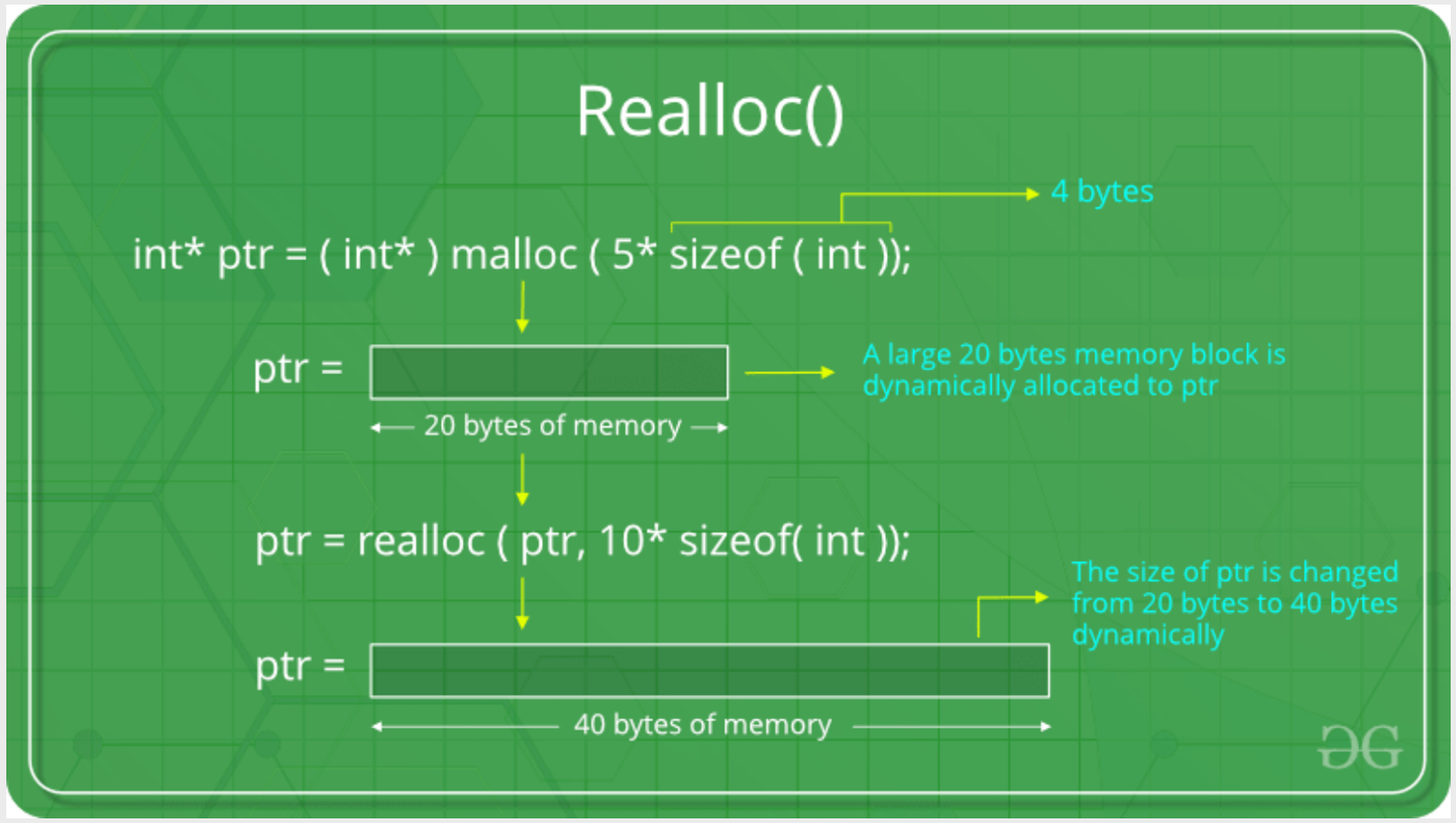
Dynamic Memory Allocation

- Example: fig12_02-1.c



Dynamic Memory Allocation

- Example: fig12_02-2.c



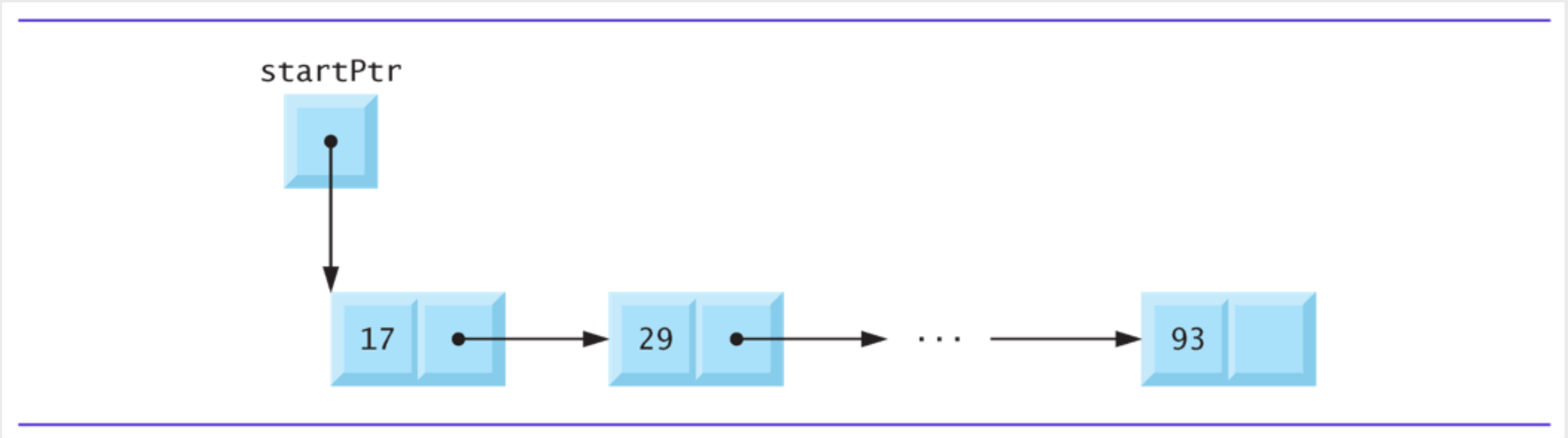
Linked Lists

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

Linked Lists

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

Linked Lists



Linked Lists

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