Capitolo 11 – Elaborazione di file

Outline

11.1 Introduzione
11.2 La gerarchia dei dati
11.3 File e Stream
11.4 Creare un file ad accesso sequenziale
11.5 Lettura di dati da un file ad accesso sequenziale
11.6 File ad accesso casuale
11.7 Creare un file ad accesso casuale
11.8 Scrittura di dati casuali inn un file ad accesso casuale
11.9 Lettura di dati casuali da un file ad accesso casuale

Obiettivi

- In questo capitolo, apprenderemo a:
  - Creare, leggere, scrivere e modificare i file.
  - Prendere familiarità con i file ad accesso sequenziale.
  - Prendere familiarità con i file ad accesso casuale.

11.1 Introduzione

- I file
  - Possono essere creati, modificati, ed elaborati da programmi scritti in C
  - Sono utilizzati per la memorizzazione permanente dei dati
    - La memorizzazione di dati in variabili ed array è solo temporanea

11.2 La gerarchia dei dati

- Gerarchia dei dati:
  - Bit – il più piccolo
    - Valore `0` o `1`
  - Byte – 8 bits
    - Utilizzato per memorizzare un carattere
      - Cifre decimali, lettere, e simboli speciali
  - Campi – gruppo di caratteri
    - Esempio: your name
  - Record – gruppo di campi
    - Rappresentato da `struct` o `class`
    - Esempio: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.

11.2 La gerarchia dei dati

- Gerarchia dei dati:
  - File – gruppo di record
    - Esempio: payroll file
  - Database – gruppo di files

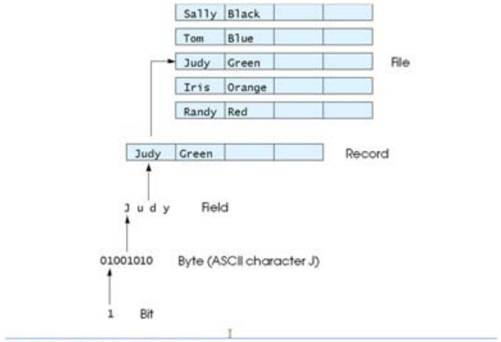


Fig. 11.1 The data hierarchy.

11.2 La gerarchia dei dati

- File
  - Record chiave
    - Identifica un record per facilitare il ritrovamento di uno specifico record da un file
  - File sequenziale
    - I record sono tipicamente ordinati in base alla chiave

11.3 File e Stream

- Il C vede un file come una sequenza di byte
  - I file terminano con un marcatore di fine file
    - O, terminano con uno specifico byte
- Quando si apre un file si crea uno stream
  - Fornisce il canale di comunicazione tra file e programma
  - L'apertura di un file restituisce un puntatore ad una struttura `FILE`
    - Esempi:
    - `stdin` - standard input (keyboard)
    - `stdout` - standard output (screen)
    - `stderr` - standard error (screen)

11.3 File e Stream

- Struttura `FILE`
  - Descrittore di file
    - Index into operating system array called the open file table
  - File Control Block (FCB)
    - Found in every array element, system uses it to administer the file

11.3 File e Stream

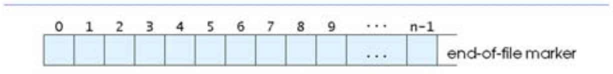


Fig. 11.2 C's view of a file of *n* bytes.

11.3 File e Stream

- Funzioni di lettura/scrittura
  - `fgetc`
    - Legge un carattere da un file
    - Puntatore `FILE` come argomento
    - `fgetc( stdin )` è equivalente a `getchar()`
  - `fputc`
    - Scrive un carattere su un file
    - Puntatore `FILE` e un carattere come argomento
    - `fputc( 'a', stdout )` è equivalente a `putchar( 'a' )`
  - `fgets`
    - Legge una linea da un file
  - `fputs`
    - Scrive una linea su un file
  - `fscanf` / `fprintf`
    - Versioni per file equivalenti di `scanf` e `printf`

Outline

fig11_03.c (1 of 2)

```c
/* Fig. 11.3: fig11_03.c
   Create a sequential file */
#include <stdio.h>

int main()
{
   int account;     /* account number */
   char name[ 30 ]; /* account name */
   double balance;  /* account balance */

   FILE *cfPtr;     /* cfPtr = clients.dat file pointer */

   /* fopen opens file. Exit program if unable to create file  */
   if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
      printf( "File could not be opened\n" );
   } /* end if */
   else {
      printf( "Enter the account, name, and balance.\n");
      printf( "Enter EOF to end input.\n" );
      printf( "? " );
      scanf( "%d%s%lf", &account, name, &balance );

```

Outline

fig11_03.c (2 of 2)

```c
      /* write account, name and balance into file with fprintf */
      while ( !feof( stdin ) ) {
         fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
         printf( "? " );
         scanf( "%d%s%lf", &account, name, &balance );
      } /* end while */

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

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output

```
Enter the account, name, and balance.
Enter EOF to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

11.4 Creazione di un file ad accesso sequenziale

- Il C non impone una struttura file
  - Non esiste la nozione di record in un file
  - Il programmatore deve fornire la struttura
- Creare un file
  - `FILE *cfPtr;`
    - Crea un puntatore `FILE` chiamato `cfPtr`
  - `cfPtr = fopen(“clients.dat", “w”);`
    - La funzione `fopen` restituisce un puntatore `FILE` al file specificato
    - Ha due argomenti – il file da aprire e la modalità d’apertura
    - Se l’apertura fallisce viene restituito `NULL`

11.4 Creazione di un file ad accesso sequenziale

Computer system	Key combination
UNIX systems	*<return> <ctrl> d*
IBM PC and compatibles	*<ctrl> z*
Macintosh	*<ctrl> d*

Fig. 11.4 End-of-file key combinations for various popular computer systems.

11.4 Creazione di un file ad accesso sequenziale

- `fprintf`
  - Utilizzato per stampare in un file
  - Come la `printf`, eccetto che il primo argomento è un puntatore `FILE` (puntatore al file in cui si vuole stampare)
- `feof(` *`FILE` pointer* `)`
  - Restituisce l’indicatore end-of-file
- `fclose(` *`FILE` pointer* `)`
  - Chiude il file specificato
  - Eseguito automaticamente quando termina il programma
  - È buona pratica chiudere il file esplicitamente
- Dettagli
  - I programmi possono operare su un file, su più file o su nessun file
  - Ogni file deve avere un nome unico ed il suo rispettivo puntatore

11.4 Creazione di un file ad accesso sequenziale

Mode	Description
`r`	Open a file for reading.
`w`	Create a file for writing. If the file already exists, discard the current contents.
`a`	Append; open or create a file for writing at end of file.
`r+`	Open a file for update (reading and writing).
`w+`	Create a file for update. If the file already exists, discard the current contents.
`a+`	Append; open or create a file for update; writing is done at the end of the file.
`rb`	Open a file for reading in binary mode.
`wb`	Create a file for writing in binary mode. If the file already exists, discard the current contents.
`ab`	Append; open or create a file for writing at end of file in binary mode.
`rb+`	Open a file for update (reading and writing) in binary mode.
`wb+`	Create a file for update in binary mode. If the file already exists, discard the current contents.
`ab+`	Append; open or create a file for update in binary mode; writing is done at the end of the file.

Fig. 11.6 File open modes.

11.5 Lettura da un file ad accesso sequenziale

- Lettura
  - Crea un puntatore `FILE`, collegarlo al file da leggere
    `cfPtr = fopen( “clients.dat", "r" );`
  - Utilizzare la `fscanf` per leggere dal file
    - Come la `scanf`, eccetto che il primo argomento è un puntatore `FILE`
    `fscanf( cfPtr, "%d%s%f", &accounnt, name, &balance );`
  - Dati letti dall’inizio alla fine
  - Punatatore di posizione del file
    - Indica il numero del prossimo byte da leggere/scivere
    - Non è veramente un puntatore, ma un valore intero (specifica la locazione in byte)
    - Detto anche byte offset
  - `rewind( cfPtr )`
    - Riposizionamento del punatatore posizione all’inizio del file (byte `0`)

Outline

fig11_07.c (1 of 2)

```c
/* Fig. 11.7: fig11_07.c
   Reading and printing a sequential file */
#include <stdio.h>

int main()
{
   int account;     /* account number */
   char name[ 30 ]; /* account name */
   double balance;  /* account balance */

   FILE *cfPtr;     /* cfPtr = clients.dat file pointer */

   /* fopen opens file; exits program if file cannot be opened */
   if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
      printf( "File could not be opened\n" );
   } /* end if */
   else { /* read account, name and balance from file */
      printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
      fscanf( cfPtr, "%d%s%lf", &account, name, &balance );

      /* while not end of file */
      while ( !feof( cfPtr ) ) {
         printf( "%-10d%-13s%7.2f\n", account, name, balance );
         fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
      } /* end while */

```

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

   return 0; /* indicates successful termination */

} /* end main */
```

Outline

fig11_07.c (2 of 2)

```
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
300       White           0.00
400       Stone         -42.16
500       Rich          224.62
```

Outline

fig11_08.c (1 of 5)

```c
/* Fig. 11.8: fig11_08.c
   Credit inquiry program */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int request;     /* request number */
   int account;     /* account number */
   double balance;  /* account balance */
   char name[ 30 ]; /* account name */
   FILE *cfPtr;     /* clients.dat file pointer */

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

      /* display request options */
      printf( "Enter request\n"
              " 1 - List accounts with zero balances\n"
              " 2 - List accounts with credit balances\n"
              " 3 - List accounts with debit balances\n"
              " 4 - End of run\n? " );
      scanf( "%d", &request );

   /* process user's request */
   while ( request != 4 ) {

      /* read account, name and balance from file */
      fscanf( cfPtr, "%d%s%lf", &account, name, &balance );

      switch ( request ) {

         case 1:
            printf( "\nAccounts with zero balances:\n" );

            /* read file contents (until eof) */
            while ( !feof( cfPtr ) ) {

               if ( balance == 0 ) {
                  printf( "%-10d%-13s%7.2f\n",
                          account, name, balance );
               } /* end if */

               /* read account, name and balance from file */
               fscanf( cfPtr, "%d%s%lf",
                       &account, name, &balance );
            } /* end while */

```

fig11_08.c (2 of 5)

```c
            break;

         case 2:
            printf( "\nAccounts with credit balances:\n" );

            /* read file contents (until eof) */
            while ( !feof( cfPtr ) ) {

               if ( balance < 0 ) {
                  printf( "%-10d%-13s%7.2f\n",
                          account, name, balance );
               } /* end if */

               /* read account, name and balance from file */
               fscanf( cfPtr, "%d%s%lf",
                       &account, name, &balance );
            } /* end while */

            break;

         case 3:
            printf( "\nAccounts with debit balances:\n" );

```

fig11_08.c (3 of 5)

```c
            /* read file contents (until eof) */
            while ( !feof( cfPtr ) ) {

               if ( balance > 0 ) {
                  printf( "%-10d%-13s%7.2f\n",
                          account, name, balance );
               } /* end if */

               /* read account, name and balance from file */
               fscanf( cfPtr, "%d%s%lf",
                       &account, name, &balance );
            } /* end while */

            break;

      } /* end switch */

      rewind( cfPtr ); /* return cfPtr to beginning of file */

      printf( "\n? " );
      scanf( "%d", &request );
   } /* end while */

```

fig11_08.c (4 of 5)

```c
      printf( "End of run.\n" );
      fclose( cfPtr ); /* fclose closes the file */
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */
```

fig11_08.c (5 of 5)

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1

Accounts with zero balances:
300       White           0.00

? 2

Accounts with credit balances:
400       Stone         -42.16

? 3

Accounts with debit balances:
100       Jones          24.98
200       Doe           345.67
500       Rich          224.62

? 4
End of run.
```

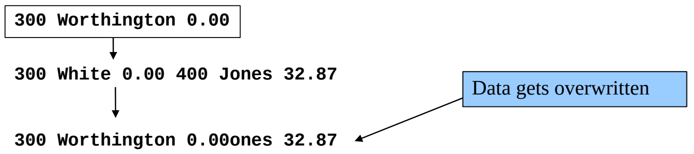
Program Output

11.5 Lettura da un file ad accesso sequenziale

- File ad accesso sequenziale
  - Non può essere modificato senza il rischio di distruggere i dati
  - I campi possono variare in dimensione
    - Diverse rappresentazioni nei file
    - `1`, `34`, `-890` sono tutti `int`, ma hanno diverse dimensioni su disco

```
300 White 0.00 400 Jones 32.87    (old data in file)
```

If we want to change White's name to Worthington,

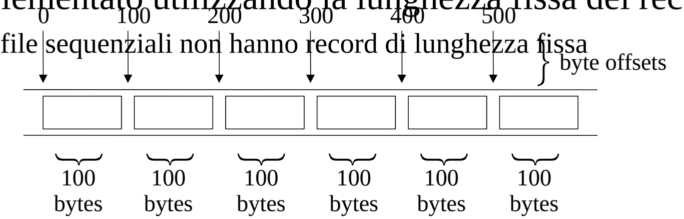


```
300 Worthington 0.00

300 White 0.00 400 Jones 32.87

300 Worthington 0.00ones 32.87
```

Data gets overwritten

11.6 File ad accesso casuale

- File ad accesso casuale
  - L'accesso ai singoli record senza effettuare la ricerca attraverso gli altri record
  - Accesso istantaneo ai record nel file
  - I dati possono essere inseriti senza distruggere gli altri dati
  - I dati precedentemente memorizzati possono essere aggiornati o cancellati senza sovrascrivere
- Implementato utilizzando la lunghezza fissa dei record
  - I file sequenziali non hanno record di lunghezza fissa



11.7 Creazione di un file ad accesso casuale

- Dati in un file ad accesso casuale
  - Non formattati (memorizzati come "raw bytes")
    - Tutti i dati delle stesso tipo (**`int`**, per esempio) utilizzano la stessa quantità di memoria
    - Tutti i record dello stesso tipo hanno una lunghezza fissa
    - I dati non sono human readable

11.7 Creazione di un file ad accesso casuale

- Funzioni I/O non formattate
  - `fwrite`
    - Trasferisce i byte da una locazione di memoria al file
  - `fread`
    - Trasferisce i byte da file ad una locazione di memoria
  - Esempio:

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

- `&number` – Locazione dei byte da trasferire
- `sizeof( int )` – Numero di byte da trasferire
- `1` – per gli array, numero di elementi da trasferire
  - In questo caso, “un elemento" di un array da trasferire
- `myPtr` – file da trasferire da o a

11.7 Creazione di un file ad accesso casuale

- Scrivere `struct`
  `fwrite( &myObject, sizeof (struct myStruct), 1, myPtr );`
  - `sizeof` – restituisce la dimensione in byte dell'oggetto in parentesi
- Per scrivere più elelmenti di un array
  - Puntatore all'array come primo argomento
  - Numero di elementi da scrivere come terzo argomento

Outline

fig11_11.c (1 of 2)

```c
/* Fig. 11.11: fig11_11.c
   Creating a randomly accessed file sequentially */
#include <stdio.h>

/* 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 */

int main()
{
   int i; /* counter */

   /* create clientData with no 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" );
   } /* end if */
```

Outline

fig11_11.c (2 of 2)

```c
   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 */
```

11.8 Scrittura su un file ad accesso casuale

- `fseek`
  - Setta il puntatore della posizione del file ad una specifica posizione
  - `fseek(` *pointer*, *offset*, *symbolic_constant* `);`
    - *pointer* – puntatore al file
    - *offset* – puntatore della posizione sul file (0 è la prima locazione)
    - *symbolic_constant* – specifica da dove leggere nel file
    - `SEEK_SET` – seek starts at beginning of file
    - `SEEK_CUR` – seek starts at current location in file
    - `SEEK_END` – seek starts at end of file

fig11_12.c (1 of 3)

```
/* Fig. 11.12: fig11_12.c
   Writing to a random access file */
#include <stdio.h>

/* 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 */

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

   /* create clientData with no 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" );
   } /* end if */
   else {

```

fig11_12.c (2 of 3)

```
      /* require user to specify account number */
      printf( "Enter account number"
              " ( 1 to 100, 0 to end input )\n? " );
      scanf( "%d", &client.acctNum );

      /* user enters information, which is copied into file */
      while ( client.acctNum != 0 ) {

         /* user enters last name, first name and balance */
         printf( "Enter lastname, firstname, balance\n? " );

         /* set record lastName, firstName and balance value */
         fscanf( stdin, "%s%s%lf", client.lastName,
                 client.firstName, &client.balance );

         /* seek position in file of user-specified record */
         fseek( cfPtr, ( client.acctNum - 1 ) *
                sizeof( struct clientData ), SEEK_SET );

         /* write user-specified information in file */
         fwrite( &client, sizeof( struct clientData ), 1, cfPtr );

         /* enable user to specify another account number */
         printf( "Enter account number\n? " );
         scanf( "%d", &client.acctNum );
```

fig11_12.c (3 of 3)

```
      } /* end while */

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

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output

```
Enter account number ( 1 to 100, 0 to end input )
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
Enter account number
? 29
Enter lastname, firstname, balance
? Brown Nancy -24.54
Enter account number
? 96
Enter lastname, firstname, balance
? Stone Sam 34.98
Enter account number
? 88
Enter lastname, firstname, balance
? Smith Dave 258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn Stacey 314.33
Enter account number
? 0
```

11.8 Scrittura su un file ad accesso casuale

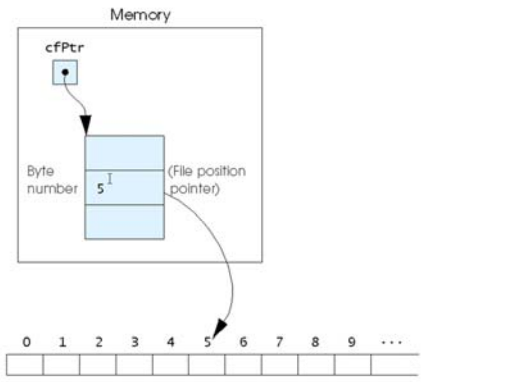


Fig. 11.14 The file position pointer indicating an offset of 5 bytes from the beginning of the file.

11.9 Lettura da un file ad accesso casuale

- `fread`
  - Legge un numero specificato di byte dal file in memoria

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

  - Può leggere più elementi di lunghezza-fissa di un array
    - Fornisce il puntatore all’ array
    - Indica il numero di elementi da leggere
  - Per leggere più elementi, si specifica il terzo elemento

```c
/* Fig. 11.15: fig11_15.c
   Reading a random access file sequentially */
#include <stdio.h>

/* 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 */

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

   /* create clientData with no 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" );
   } /* end if */
```

▲ Outline ▼

fig11_15.c (1 of 2)

```c
   else {
      printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
              "First Name", "Balance" );

      /* read all records from file (until eof) */
      while ( !feof( cfPtr ) ) {
         fread( &client, sizeof( struct clientData ), 1, cfPtr );

         /* display record */
         if ( client.acctNum != 0 ) {
            printf( "%-6d%-16s%-11s%10.2f\n",
                    client.acctNum, client.lastName,
                    client.firstName, client.balance );
         } /* end if */

      } /* end while */

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

   return 0; /* indicates successful termination */

} /* end main */
```

▲ Outline ▼

fig11_15.c (2 of 2)

```
Acct  Last Name       First Name    Balance
29    Brown           Nancy          -24.54
33    Dunn            Stacey         314.33
37    Barker          Doug             0.00
88    Smith           Dave           258.34
96    Stone           Sam             34.98
```

▲ Outline ▼

Program Output

11.10 Case Study: A Transaction Processing Program

- This program
  - Demonstrates using random access files to achieve instant access processing of a bank’s account information
- We will
  - Update existing accounts
  - Add new accounts
  - Delete accounts
  - Store a formatted listing of all accounts in a text file

Outline

fig11_16.c (1 of 11)

```
/* Fig. 11.16: fig11_16.c
   This program reads a random access file sequentially, updates data
   already written to the file, creates new data to be placed in the
   file, and deletes data previously in the file. */
#include <stdio.h>

/* 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 );

int main()
{
   FILE *cfPtr; /* credit.dat file pointer */
   int choice;  /* user's choice */

```

Outline

fig11_16.c (2 of 11)

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

      /* enable user to specify action */
      while ( ( choice = enterChoice() ) != 5 ) {

         switch ( choice ) {

            /* create text file from record file */
            case 1:
               textFile( cfPtr );
               break;

            /* update record */
            case 2:
               updateRecord( cfPtr );
               break;

```

Outline

fig11_16.c (3 of 11)

```
            /* create record */
            case 3:
               newRecord( cfPtr );
               break;

            /* delete existing record */
            case 4:
               deleteRecord( cfPtr );
               break;

            /* display message if user does not select valid choice */
            default:
               printf( "Incorrect choice\n" );
               break;

         } /* end switch */

      } /* end while */

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

   return 0; /* indicates successful termination */

} /* end main */

```

```
/* create formatted text file for printing */
void textFile( FILE *readPtr )
{
   FILE *writePtr; /* accounts.txt file pointer */

   /* create clientData with no information */
   struct clientData client = { 0, "", "", 0.0 };

   /* fopen opens the file; exits if file cannot be opened */
   if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
   } /* end if */
   else {
      rewind( readPtr ); /* sets pointer to beginning of record file */
      fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
               "Acct", "Last Name", "First Name","Balance" );

      /* copy all records from record file into text file */
      while ( !feof( readPtr ) ) {
         fread( &client, sizeof( struct clientData ), 1, readPtr );

```

Outline

fig11_16.c (4 of 11)

```
         /* write single record to text file */
         if ( client.acctNum != 0 ) {
            fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
                     client.acctNum, client.lastName,
                     client.firstName, client.balance );
         } /* end if */

      } /* end while */

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

} /* end function textFile */

/* update balance in record */
void updateRecord( FILE *fPtr )
{
   int account;        /* account number */
   double transaction; /* account transaction */

   /* create clientData with no information */
   struct clientData client = { 0, "", "", 0.0 };

```

Outline

fig11_16.c (5 of 11)

```
   /* obtain number of account to update */
   printf( "Enter account to update ( 1 - 100 ): " );
   scanf( "%d", &account );

   /* move file pointer to correct record in file */
   fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
          SEEK_SET );

   /* read record from file */
   fread( &client, sizeof( struct clientData ), 1, fPtr );

   /* display error if account does not exist */
   if ( client.acctNum == 0 ) {
      printf( "Acount #%d has no information.\n", account );
   } /* end if */
   else { /* update record */
      printf( "%-6d%-16s%-11s%10.2f\n\n",
              client.acctNum, client.lastName,
              client.firstName, client.balance );

      /* request user to specify transaction */
      printf( "Enter charge ( + ) or payment ( - ): " );
      scanf( "%lf", &transaction );
      client.balance += transaction;/* update record balance */

```

Outline

fig11_16.c (6 of 11)

```
      printf( "%-6d%-16s%-11s%10.2f\n",
              client.acctNum, client.lastName,
              client.firstName, client.balance );

      /* move file pointer to correct record in file */
      fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
             SEEK_SET );

      /* write updated record over old record in file */
      fwrite( &client, sizeof( struct clientData ), 1, fPtr );
   } /* end else */

} /* end function updateRecord */

/* delete an existing record */
void deleteRecord( FILE *fPtr )
{
   /* create two clientDatas and initialize blankClient */
   struct clientData client;
   struct clientData blankClient = { 0, "", "", 0 };

   int accountNum; /* account number */

```

Outline

fig11_16.c (7 of 11)

```
   /* obtain number of account to delete */
   printf( "Enter account number to delete ( 1 - 100 ): " );
   scanf( "%d", &accountNum );

   /* move file pointer to correct record in file */
   fseek( fPtr, ( accountNum   1 ) * sizeof( struct clientData ),
          SEEK_SET );

   /* read record from file */
   fread( &client, sizeof( struct clientData ), 1, fPtr );

   /* display error if record does not exist */
   if ( client.acctNum == 0 ) {
      printf( "Account %d does not exist.\n", accountNum );
   } /* end if */
   else { /* delete record */

      /* move file pointer to correct record in file */
      fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
         SEEK_SET );

      /* replace existing record with blank record */
      fwrite( &blankClient,
              sizeof( struct clientData ), 1, fPtr );
   } /* end else */

```

Outline

fig11_16.c (8 of 11)

```
} /* end function deleteRecord */

/* create and insert record */
void newRecord( FILE *fPtr )
{
   /* create clientData with no information */
   struct clientData client = { 0, "", "", 0.0 };

   int accountNum; /* account number */

   /* obtain number of account to create */
   printf( "Enter new account number ( 1 - 100 ): " );
   scanf( "%d", &accountNum );

   /* move file pointer to correct record in file */
   fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
          SEEK_SET );

   /* read record from file */
   fread( &client, sizeof( struct clientData ), 1, fPtr );

```

Outline

fig11_16.c (9 of 11)

```
   /* display error if account previously exists */
   if ( client.acctNum != 0 ) {
      printf( "Account #%d already contains information.\n",
              client.acctNum );
   } /* end if */
   else { /* create record */

      /* user enters last name, first name and balance */
      printf( "Enter lastname, firstname, balance\n? " );
      scanf( "%s%s%lf", &client.lastName, &client.firstName,
             &client.balance );

      client.acctNum = accountNum;

      /* move file pointer to correct record in file */
      fseek( fPtr, ( client.acctNum - 1 ) *
             sizeof( struct clientData ), SEEK_SET );

      /* insert record in file */
      fwrite( &client,
              sizeof( struct clientData ), 1, fPtr );
   } /* end else */

} /* end function newRecord */

```

Outline

fig11_16.c (10 of 11)

```
/* enable user to input menu choice */
int enterChoice( void )
{
   int menuChoice; /* variable to store user's choice */

   /* display available options */
   printf( "\nEnter your choice\n"
           "1 - store a formatted text file of acounts called\n"
           "    \"accounts.txt\" for printing\n"
           "2 - update an account\n"
           "3 - add a new account\n"
           "4 - delete an account\n"
           "5 - end program\n? " );

   scanf( "%d", &menuChoice ); /* receive choice from user */

   return menuChoice;

} /* end function enterChoice */
```

Outline

fig11_16.c (11 of 11)

Outline

Program Output

```
After choosing option 1 accounts.txt contains:

Acct  Last Name       First Name    Balance
29    Brown           Nancy          -24.54
33    Dunn            Stacey         314.33
37    Barker          Doug             0.00
88    Smith           Dave           258.34
96    Stone           Sam             34.98
```

```
After choosing option 2 accounts.txt contains:

Enter account to update ( 1 - 100 ): 37
37    Barker          Doug             0.00

Enter charge ( + ) or payment ( - ): +87.99
37    Barker          Doug            87.99
```

```
After choosing option 3 accounts.txt contains:

Enter new account number ( 1 - 100 ): 22
Enter lastname, firstname, balance
? Johnston Sarah 247.45
```