File Processing

Outline

11.1 Introduction
11.2 The Data Hierarchy
11.3 Files and Streams
11.4 Creating a Sequential Access File
11.5 Reading Data from a Sequential Access File
11.6 Updating Sequential Access Files
11.7 Random Access Files
11.8 Creating a Random Access File
11.9 Writing Data Randomly to a Random Access File
11.10 Reading Data Sequentially from a Random Access File
11.11 Case Study: A Transaction Processing Program
11.12 Input/Output of Objects

11.1 Introduction

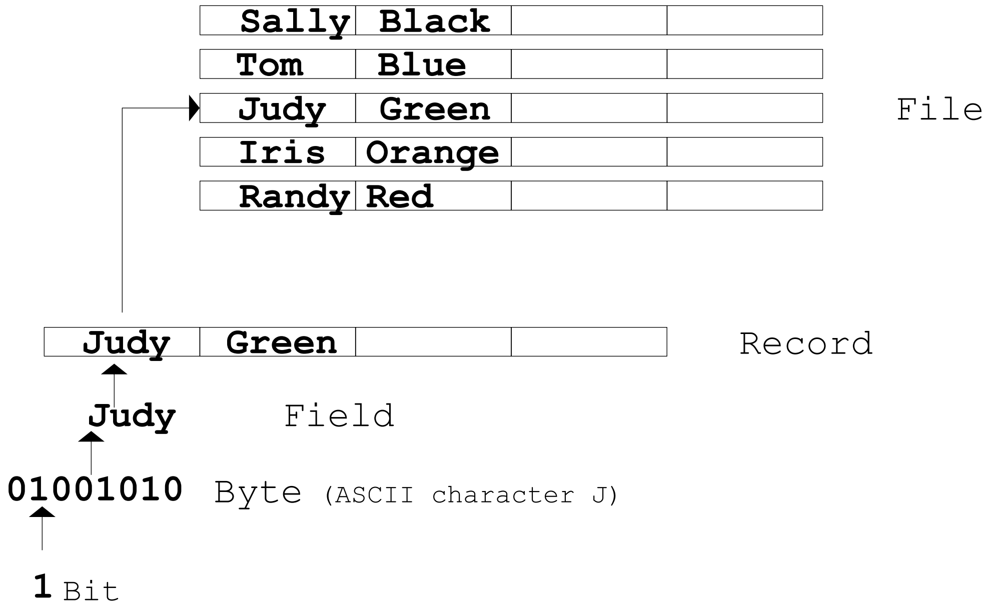
- Data files
  - Can be created, updated, and processed by C programs
  - Are used for permanent storage of large amounts of data
    - Storage of data in variables and arrays is only temporary

11.2 The Data Hierarchy

- Data Hierarchy:
  - Bit – smallest data item
    - Value of `0` or `1`
  - Byte – 8 bits
    - Used to store a character
      - Decimal digits, letters, and special symbols
  - Field – group of characters conveying meaning
    - Example: your name
  - Record – group of related fields
    - Represented by a `struct` or a `class`
    - Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.

11.2 The Data Hierarchy

- Data Hierarchy (continued):
  - File – group of related records
    - Example: payroll file
  - Database – group of related files



11.2 The Data Hierarchy

- Data files
  - Record key
    - Identifies a record to facilitate the retrieval of specific records from a file
  - Sequential file
    - Records typically sorted by key

11.3 Files and Streams

- C views each file as a sequence of bytes
  - File ends with the *end-of-file marker*
    - Or, file ends at a specified byte
- Stream created when a file is opened
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a **`FILE`** structure
    - Example file pointers:
    - **`stdin`** - standard input (keyboard)
    - **`stdout`** - standard output (screen)
    - **`stderr`** - standard error (screen)
- **`FILE`** structure
  - File descriptor
    - 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 Files and Streams

- Read/Write functions in standard library
  - **`fgetc`**
    - Reads one character from a file
    - Takes a **`FILE`** pointer as an argument
    - **`fgetc( stdin )`** equivalent to **`getchar()`**
  - **`fputc`**
    - Writes one character to a file
    - Takes a **`FILE`** pointer and a character to write as an argument
    - **`fputc( 'a', stdout )`** equivalent to **`putchar( 'a' )`**
  - **`fgets`**
    - Reads a line from a file
  - **`fputs`**
    - Writes a line to a file
  - **`fscanf`** / **`fprintf`**
    - File processing equivalents of **`scanf`** and **`printf`**

11.4 Creating a Sequential Access File

- C imposes no file structure
  - No notion of records in a file
  - Programmer must provide file structure
- Creating a File
  - **`FILE *myPtr;`**
    - Creates a **`FILE`** pointer called **`myPtr`**
  - **`myPtr = fopen("myFile.dat", openmode);`**
    - Function **`fopen`** returns a **`FILE`** pointer to file specified
    - Takes two arguments – file to open and file open mode
    - If open fails, **`NULL`** returned
  - **`fprintf`**
    - Used to print to a file
    - Like printf, except first argument is a **`FILE`** pointer (pointer to the file you want to print in)

11.4 Creating a Sequential Access File

- `feof(` ***`FILE`*** *pointer* `)`
  - Returns true if end-of-file indicator (no more data to process) is set for the specified file
- `fclose(` ***`FILE`*** *pointer* `)`
  - Closes specified file
  - Performed automatically when program ends
  - Good practice to close files explicitly

- Details
  - Programs may process no files, one file, or many files
  - Each file must have a unique name and should have its own pointer

11.4 Creating a Sequential Access File

- Table of file open modes:

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.

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

int main()
{
   int account;
   char name[ 30 ];
   double balance;
   FILE *cfPtr;   /* cfPtr = clients.dat file pointer */

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

      while ( !feof( stdin ) ) {
         fprintf( cfPtr, "%d %s %.2f\n",
                  account, name, balance );
         printf( "? " );
         scanf( "%d%s%lf", &account, name, &balance );
      }

      fclose( cfPtr );
   }

   return 0;
}
```

Outline

- 1. Initialize variables and `FILE` pointer
- 1.1 Link the pointer to a file
- 2. Input data
- 2.1 Write to file

```
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
?
```

- Program Output

11.5 Reading Data from a Sequential Access File

- Reading a sequential access file
  - Create a **`FILE`** pointer, link it to the file to read

    ```
    myPtr = fopen( "myFile.dat", "r" );
    ```
  - Use **`fscanf`** to read from the file
    - Like **`scanf`**, except first argument is a **`FILE`** pointer

    ```
    fscanf( myPtr, "%d%s%f", &myInt, &myString,
       &myFloat );
    ```
  - Data read from beginning to end
  - File position pointer
    - Indicates number of next byte to be read / written
    - Not really a pointer, but an integer value (specifies byte location)
    - Also called byte offset
  - **`rewind( myPtr )`**
    - Repositions file position pointer to beginning of file (byte **`0`**)

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

int main()
{
   int account;
   char name[ 30 ];
   double balance;
   FILE *cfPtr;   /* cfPtr = clients.dat file pointer */

   if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL )
      printf( "File could not be opened\n" );
   else {
      printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
      fscanf( cfPtr, "%d%s%lf", &account, name, &balance );

      while ( !feof( cfPtr ) ) {
         printf( "%-10d%-13s%7.2f\n", account, name, balance );
         fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
      }

      fclose( cfPtr );
   }

   return 0;
}
```

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

Outline

- 1. Initialize variables
- 1.1 Link pointer to file
- 2. Read data (`fscanf`)
- 2.1 Print
- 3. Close file

```c
/* Fig. 11.8: fig11 08.c
   Credit inquiry program */
#include <stdio.h>

int main()
{
   int request, account;
   double balance;
   char name[ 30 ];
   FILE *cfPtr;

   if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL )
      printf( "File could not be opened\n" );
   else {
      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 );

      while ( request != 4 ) {
         fscanf( cfPtr, "%d%s%lf", &account, name,
                 &balance );

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

               while ( !feof( cfPtr ) ) {

```

Outline

- 1. Initialize variables
- 2. Open file
- 2.1 Input choice
- 2.2 Scan files
- 3. Print

```
               if ( balance == 0 )
                  printf( "%-10d%-13s%7.2f\n",
                         account, name, balance );

               fscanf( cfPtr, "%d%s%lf",
                      &account, name, &balance );
            }

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

            while ( !feof( cfPtr ) ) {

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

               fscanf( cfPtr, "%d%s%lf",
                      &account, name, &balance );
            }

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

            while ( !feof( cfPtr ) ) {

               if ( balance > 0 )
                  printf( "%-10d%-13s%7.2f\n",
```

Outline

- 2.2 Scan files
- 3. Print

```
                                       account, name, balance );

                          fscanf( cfPtr, "%d%s%lf",
                                 &account, name, &balance );
                       }

                       break;
               }

               rewind( cfPtr );
               printf( "\n? " );
               scanf( "%d", &request );
            }

            printf( "End of run.\n" );
            fclose( cfPtr );
         }

         return 0;
      }
```

- 3.1 Close file

```
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.
```

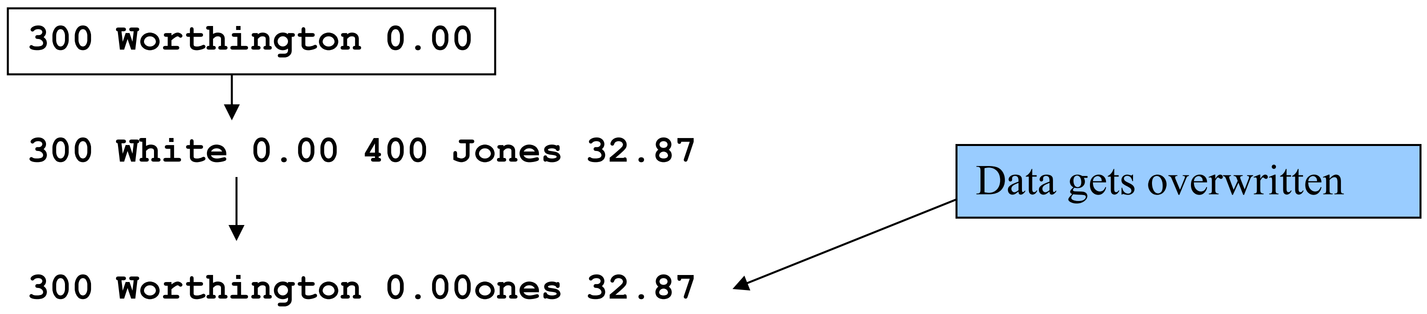
- Program Output

11.5 Reading Data from a Sequential Access File

- Sequential access file
  - Cannot be modified without the risk of destroying other data
  - Fields can vary in size
    - Different representation in files and screen than internal representation
    - **`1`**, **`34`**, **`-890`** are all **`int`**s, but have different sizes on disk

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

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



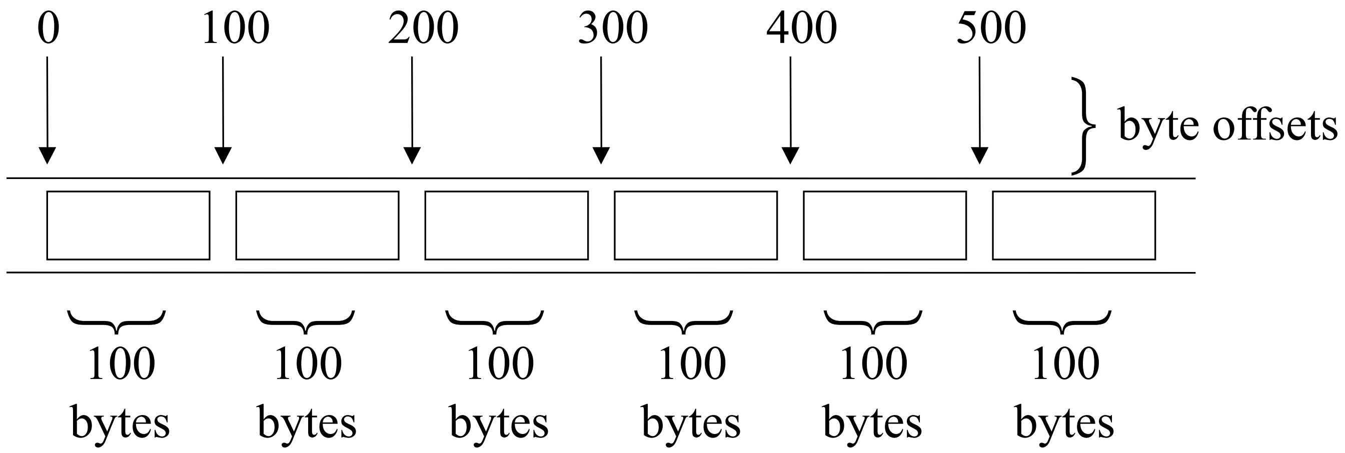
```
300 Worthington 0.00

300 White 0.00 400 Jones 32.87

300 Worthington 0.00ones 32.87
```

11.6 Random Access Files

- Random access files
  - Access individual records without searching through other records
  - Instant access to records in a file
  - Data can be inserted without destroying other data
  - Data previously stored can be updated or deleted without overwriting
- Implemented using fixed length records
  - Sequential files do not have fixed length records



11.7 Creating a Random Access File

- Data in random access files
  - Unformatted (stored as "raw bytes")
    - All data of the same type (**`int`**s, for example) uses the same amount of memory
    - All records of the same type have a fixed length
    - Data not human readable

11.7 Creating a Random Access File

- Unformatted I/O functions
  - `fwrite`
    - Transfer bytes from a location in memory to a file
  - `fread`
    - Transfer bytes from a file to a location in memory
  - Example:

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

    - `&number` – Location to transfer bytes from
    - `sizeof( int )` – Number of bytes to transfer
    - `1` – For arrays, number of elements to transfer
      - In this case, "one element" of an array is being transferred
    - `myPtr` – File to transfer to or from

11.7 Creating a Random Access File

- Writing structs

  ```
  fwrite( &myObject, sizeof (struct myStruct),
    1, myPtr );
  ```

  – **`sizeof`** – returns size in bytes of object in parentheses
- To write several array elements
  – Pointer to array as first argument
  – Number of elements to write as third argument

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

struct clientData {
   int acctNum;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;
};

int main()
{
   int i;
   struct clientData blankClient = { 0, "", "", 0.0 };
   FILE *cfPtr;

   if ( ( cfPtr = fopen( "credit.dat", "w" ) ) == NULL )
      printf( "File could not be opened.\n" );
   else {

      for ( i = 1; i <= 100; i++ )
         fwrite( &blankClient,
                 sizeof( struct clientData ), 1, cfPtr );

      fclose( cfPtr );
   }

   return 0;
}
```

Outline

- 1. Define struct
- 1.1 Initialize variable
- 1.2 Initialize struct
- 2. Open file
- 2.1 Write to file using

11.8 Writing Data Randomly to a Random Access File

- **`fseek`**
  - Sets file position pointer to a specific position
  - **`fseek(`** *pointer, offset, symbolic_constant* **`);`**
    - *pointer* – pointer to file
    - *offset* – file position pointer (0 is first location)
    - *symbolic_constant* – specifies where in file we are reading from
    - **`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

```c
/* Fig. 11.12: fig11 12.c
   Writing to a random access file */
#include <stdio.h>

struct clientData {
   int acctNum;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;
};

int main()
{
   FILE *cfPtr;
   struct clientData client = { 0, "", "", 0.0 };

   if ( ( cfPtr = fopen( "credit.dat", "r+" ) ) == NULL )
      printf( "File could not be opened.\n" );
   else {
      printf( "Enter account number"
              " ( 1 to 100, 0 to end input )\n? " );
      scanf( "%d", &client.acctNum );

      while ( client.acctNum != 0 ) {
         printf( "Enter lastname, firstname, balance\n? " );
         fscanf( stdin, "%s%s%lf", client.lastName,
                client.firstName, &client.balance );
         fseek( cfPtr, ( client.acctNum - 1 ) *
                sizeof( struct clientData ), SEEK SET );
         fwrite( &client, sizeof( struct clientData ), 1,
                 cfPtr );
         printf( "Enter account number\n? " );
```

Outline

- 1. Define struct
- 1.1 Initialize variables
- 2. Open file
- 2.1 Input data
- 2.2 Write to file

```c
            scanf( "%d", &client.acctNum );
         }

         fclose( cfPtr );
      }

      return 0;
   }
```

```
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
```

Outline

- 3. Close file

- Program Output

```
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
```

Outline

- Program Output

11.9 Reading Data Sequentially from a Random Access File

- **`fread`**
  - Reads a specified number of bytes from a file into memory

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

  - Can read several fixed-size array elements
    - Provide pointer to array
    - Indicate number of elements to read
  - To read multiple elements, specify in third argument

```c
/* Fig. 11.15: fig11 15.c
   Reading a random access file sequentially */
#include <stdio.h>

struct clientData {
   int acctNum;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;
};

int main()
{
   FILE *cfPtr;
   struct clientData client = { 0, "", "", 0.0 };

   if ( ( cfPtr = fopen( "credit.dat", "r" ) ) == NULL )
      printf( "File could not be opened.\n" );
   else {
      printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
             "First Name", "Balance" );

      while ( !feof( cfPtr ) ) {
         fread( &client, sizeof( struct clientData ), 1,
                cfPtr );

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

```

Outline

- 1. Define struct
- 1.1 Initialize variables
- 2. Read (`fread`)
- 2.1 Print

```c
      fclose( cfPtr );
   }

   return 0;
}
```

```
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

- 3. Close file
- 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

```c
/* 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
   already in the file.                                    */
#include <stdio.h>

struct clientData {
   int acctNum;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;
};

int enterChoice( void );
void textFile( FILE * );
void updateRecord( FILE * );
void newRecord( FILE * );
void deleteRecord( FILE * );

int main()
{
   FILE *cfPtr;
   int choice;

   if ( ( cfPtr = fopen( "credit.dat", "r+" ) ) == NULL )
      printf( "File could not be opened.\n" );
   else {

      while ( ( choice = enterChoice() ) != 5 ) {

         switch ( choice ) {
```

<u>Outline</u>

- 1. Define struct
- 1.1 Function prototypes
- 1.2 Initialize variables
- 1.3 Link pointer and open file

```
               case 1:
                  textFile( cfPtr );
                  break;
               case 2:
                  updateRecord( cfPtr );
                  break;
               case 3:
                  newRecord( cfPtr );
                  break;
               case 4:
                  deleteRecord( cfPtr );
                  break;
            }
         }

      fclose( cfPtr );
   }

   return 0;
}

void textFile( FILE *readPtr )
{
   FILE *writePtr;
   struct clientData client = { 0, "", "", 0.0 };

   if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL )
      printf( "File could not be opened.\n" );
   else {
      rewind( readPtr );
      fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
               "Acct", "Last Name", "First Name","Balance" );
```

- 2.1 Perform action
- 3. Close file
- 3.1 Function definitions

```c

      while ( !feof( readPtr ) ) {
         fread( &client, sizeof( struct clientData ), 1,
                readPtr );

         if ( client.acctNum != 0 )
            fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
                    client.acctNum, client.lastName,
                    client.firstName, client.balance );
      }

      fclose( writePtr );
   }

}

void updateRecord( FILE *fPtr )
{
   int account;
   double transaction;
   struct clientData client = { 0, "", "", 0.0 };

   printf( "Enter account to update ( 1 - 100 ): " );
   scanf( "%d", &account );
   fseek( fPtr,
          ( account - 1 ) * sizeof( struct clientData ),
          SEEK_SET );
   fread( &client, sizeof( struct clientData ), 1, fPtr );

   if ( client.acctNum == 0 )
      printf( "Acount #%d has no information.\n", account );
   else {
```

Outline

- 3.1 Function definitions

```
      printf( "%-6d%-16s%-11s%10.2f\n\n",
              client.acctNum, client.lastName,
              client.firstName, client.balance );
      printf( "Enter charge ( + ) or payment ( - ): " );
      scanf( "%lf", &transaction );
      client.balance += transaction;
      printf( "%-6d%-16s%-11s%10.2f\n",
              client.acctNum, client.lastName,
              client.firstName, client.balance );
      fseek( fPtr,
              ( account - 1 ) * sizeof( struct clientData ),
              SEEK_SET );
      fwrite( &client, sizeof( struct clientData ), 1,
               fPtr );
   }
}

void deleteRecord( FILE *fPtr )
{
   struct clientData client,
                     blankClient = { 0, "", "", 0 };
   int accountNum;

   printf( "Enter account number to "
           "delete ( 1 - 100 ): " );
   scanf( "%d", &accountNum );
   fseek( fPtr,
          ( accountNum - 1 ) * sizeof( struct clientData ),
          SEEK_SET );
   fread( &client, sizeof( struct clientData ), 1, fPtr );
```

Outline

- 3.1 Function definitions

```

   if ( client.acctNum == 0 )
      printf( "Account %d does not exist.\n", accountNum );
   else {
      fseek( fPtr,
         ( accountNum - 1 ) * sizeof( struct clientData ),
         SEEK_SET );
      fwrite( &blankClient,
              sizeof( struct clientData ), 1, fPtr );
   }
}

void newRecord( FILE *fPtr )
{
   struct clientData client = { 0, "", "", 0.0 };
   int accountNum;
   printf( "Enter new account number ( 1 - 100 ): " );
   scanf( "%d", &accountNum );
   fseek( fPtr,
          ( accountNum - 1 ) * sizeof( struct clientData ),
          SEEK_SET );
   fread( &client, sizeof( struct clientData ), 1, fPtr );

   if ( client.acctNum != 0 )
      printf( "Account #%d already contains information.\n",
              client.acctNum );
   else {
      printf( "Enter lastname, firstname, balance\n? " );
      scanf( "%s%s%lf", &client.lastName, &client.firstName,
             &client.balance );
```

- 3.1 Function definitions

```
      client.acctNum = accountNum;
      fseek( fPtr, ( client.acctNum - 1 ) *
            sizeof( struct clientData ), SEEK_SET );
      fwrite( &client,
              sizeof( struct clientData ), 1, fPtr );
   }
}

int enterChoice( void )
{
   int menuChoice;

   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 );
   return menuChoice;
}
```

Outline

- 3.1 Function definitions

```
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
```

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

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

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

- Program Output