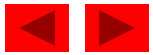
Chapter 11 – File Processing

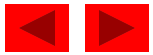
Outline

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



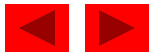
11.1 Introduction

- 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



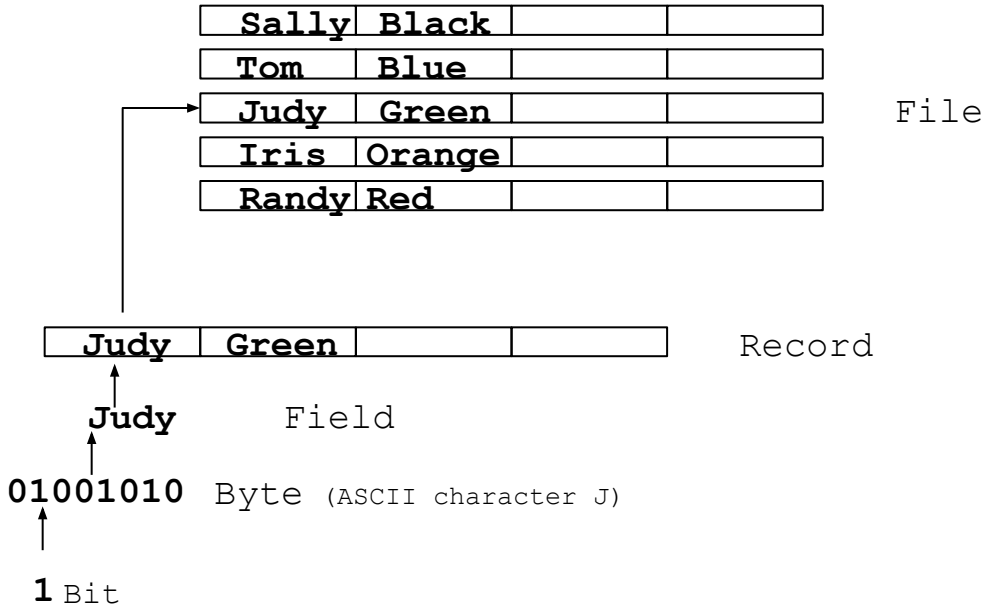
11.2 The Data Hierarchy

- 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.



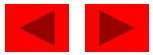
11.2 The Data Hierarchy

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



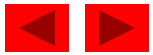
11.2 The Data Hierarchy

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



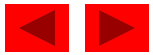
11.3 Files and Streams

- 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



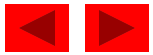
11.3 Files and Streams

- 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

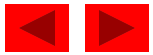
- C imposes no file structure
  - No notion of records in a file
  - Programmer must provide file structure
- Creating a File
  - **`FILE *myPtr;`**
    - Creates a **`FILE`** pointer called **`myPtr`**
  - **`myPtr = fopen("myFile.dat", openmode);`**
    - Function **`fopen`** returns a **`FILE`** pointer 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

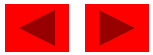
- Details
  - Programs may process no files, one file, or many files
  - Each file must have a unique name and should have its own pointer



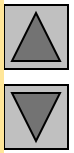
11.4 Creating a Sequential Access File

- 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

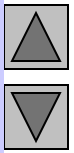
1. Initialize variables and `FILE` pointer

1.1 Link the pointer to a file

2. Input data

2.1 Write to file (`fprintf`)

3. Close file



Outline

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
?
```

11.5 Reading Data from a Sequential Access File

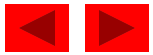
- Reading a sequential access file
  - Create a **FILE** pointer, link it to the file to read

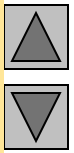
```
myPtr = fopen( "myFile.dat", "r" );
```

  - 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**)





Outline

1. Initialize variables

1.1 Link pointer to file

2. Read data (`fscanf`)

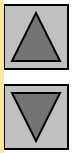
2.1 Print

3. Close file

```
1   /* Fig. 11.7: fig11_07.c
2      Reading and printing a sequential file */
3   #include <stdio.h>
4
5   int main()
6   {
7      int account;
8      char name[ 30 ];
9      double balance;
10     FILE *cfPtr;   /* cfPtr = clients.dat file pointer */
11
12     if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL )
13        printf( "File could not be opened\n" );
14     else {
15        printf( "%-10s%-13s%s\n", "Account", "Name", "Balance"
16        fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
17
18        while ( !feof( cfPtr ) ) {
19           printf( "%-10d%-13s%7.2f\n", account, name, balance
20           fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
21        }
22
23        fclose( cfPtr );
24     }
25
26     return 0;
27  }
```

Program Output

```
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

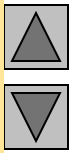
2. Open file

2.1 Input choice

2.2 Scan files

3. Print

```
1   /* Fig. 11.8: fig11 08.c
2      Credit inquiry program */
3   #include <stdio.h>
4
5   int main()
6   {
7      int request, account;
8      double balance;
9      char name[ 30 ];
10     FILE *cfPtr;
11
12     if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL )
13        printf( "File could not be opened\n" );
14     else {
15        printf( "Enter request\n"
16                " 1 - List accounts with zero balances\n"
17                " 2 - List accounts with credit balances\n"
18                " 3 - List accounts with debit balances\n"
19                " 4 - End of run\n? " );
20        scanf( "%d", &request );
21
22        while ( request != 4 ) {
23           fscanf( cfPtr, "%d%s%lf", &account, name,
24                   &balance );
25
26           switch ( request ) {
27              case 1:
28                 printf( "\nAccounts with zero "
29                         "balances:\n" );
30
31                 while ( !feof( cfPtr ) ) {
32
```

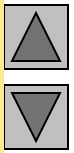


Outline

2.2 Scan files

3. Print

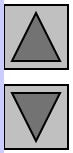
```
33                   if ( balance == 0 )
34                      printf( "%-10d%-13s%7.2f\n",
35                              account, name, balance );
36
37                   fscanf( cfPtr, "%d%s%lf",
38                           &account, name, &balance );
39                }
40
41                break;
42             case 2:
43                printf( "\nAccounts with credit "
44                        "balances:\n" );
45
46                while ( !feof( cfPtr ) ) {
47
48                   if ( balance < 0 )
49                      printf( "%-10d%-13s%7.2f\n",
50                              account, name, balance );
51
52                   fscanf( cfPtr, "%d%s%lf",
53                           &account, name, &balance );
54                }
55
56                break;
57             case 3:
58                printf( "\nAccounts with debit "
59                        "balances:\n" );
60
61                while ( !feof( cfPtr ) ) {
62
63                   if ( balance > 0 )
64                      printf( "%-10d%-13s%7.2f\n",
```



Outline

3.1 Close file

```
65                                account, name, balance );
66
67                         fscanf( cfPtr, "%d%s%lf",
68                                 &account, name, &balance );
69                      }
70
71                      break;
72                }
73
74                rewind( cfPtr );
75                printf( "\n? " );
76                scanf( "%d", &request );
77             }
78
79             printf( "End of run.\n" );
80             fclose( cfPtr );
81          }
82
83          return 0;
84       }
```



Outline

Program Output

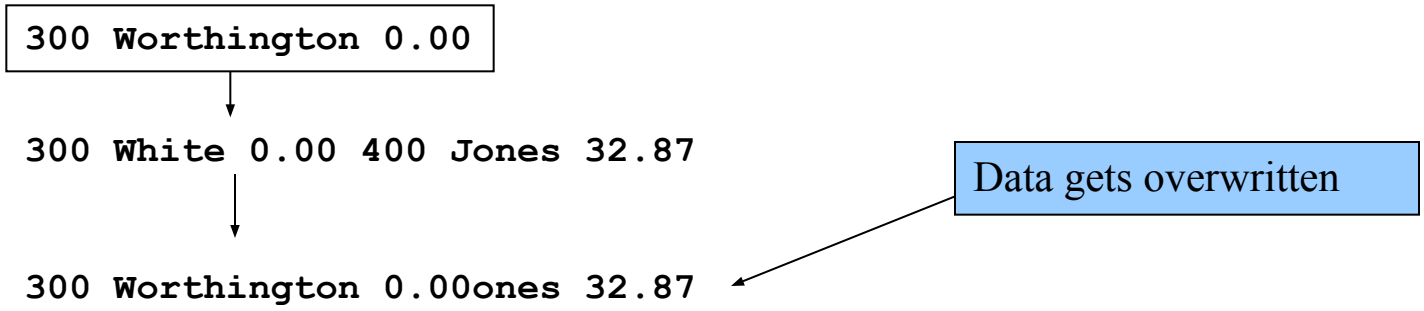
```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1

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.
```

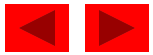
11.5 Reading Data from a Sequential Access File

- 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)
```

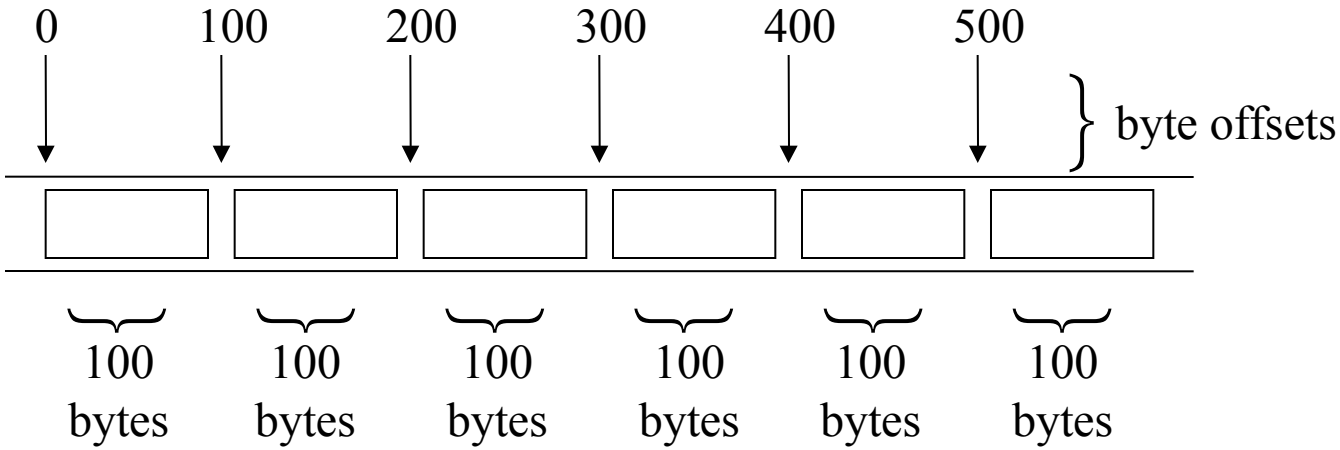
If we want to change White's name to Worthington,





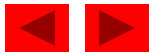
11.6 Random Access Files

- 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

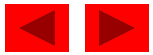


11.7 Creating a Random Access File

- 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

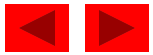


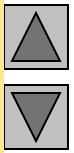
11.7 Creating a Random Access File

- 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





Outline

1. Define struct

1.1 Initialize variable

1.2 Initialize struct

2. Open file

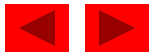
2.1 Write to file using unformatted output

3. Close file

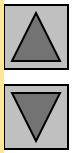
```
1   /* Fig. 11.11: fig11_11.c
2      Creating a randomly accessed file sequentially */
3   #include <stdio.h>
4
5   struct clientData {
6      int acctNum;
7      char lastName[ 15 ];
8      char firstName[ 10 ];
9      double balance;
10  };
11
12  int main()
13  {
14     int i;
15     struct clientData blankClient = { 0, "", "", 0.0 };
16     FILE *cfPtr;
17
18     if ( ( cfPtr = fopen( "credit.dat", "w" ) ) == NULL )
19        printf( "File could not be opened.\n" );
20     else {
21
22        for ( i = 1; i <= 100; i++ )
23           fwrite( &blankClient,
24                   sizeof( struct clientData ), 1, cfPtr );
25
26        fclose( cfPtr );
27     }
28
29     return 0;
30  }
```

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

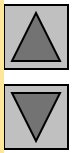
1.1 Initialize variables

2. Open file

2.1 Input data

2.2 Write to file

```
33             scanf( "%d", &client.acctNum );
34          }
35
36          fclose( cfPtr );
37       }
38
39       return 0;
40    }
```

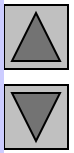


Outline

3. Close file

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
```



Outline

Program Output

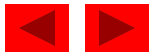
```
Enter account number
? 88
Enter lastname, firstname, balance
? Smith Dave 258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn Stacey 314.33
Enter account number
? 0
```

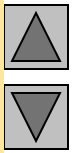
11.9 Reading Data Sequentially from a Random Access File

- **`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





Outline

1. Define struct

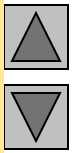
1.1 Initialize variables

2. Read (`fread`)

2.1 Print

```
1   /* Fig. 11.15: fig11 15.c
2      Reading a random access file sequentially */
3   #include <stdio.h>
4
5   struct clientData {
6      int acctNum;
7      char lastName[ 15 ];
8      char firstName[ 10 ];
9      double balance;
10  };
11
12  int main()
13  {
14     FILE *cfPtr;
15     struct clientData client = { 0, "", "", 0.0 };
16
17     if ( ( cfPtr = fopen( "credit.dat", "r" ) ) == NULL )
18        printf( "File could not be opened.\n" );
19     else {
20        printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
21                "First Name", "Balance" );
22
23        while ( !feof( cfPtr ) ) {
24           fread( &client, sizeof( struct clientData ), 1,
25                  cfPtr );
26
27           if ( client.acctNum != 0 )
28              printf( "%-6d%-16s%-11s%10.2f\n",
29                      client.acctNum, client.lastName,
30                      client.firstName, client.balance );
31        }
32
```

```
33          fclose( cfPtr );
34       }
35
36       return 0;
37    }
```



Outline

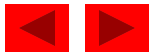
3. Close file

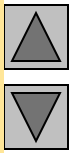
Program Output

```
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
```

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

1. Define struct

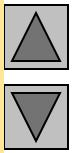
1.1 Function prototypes

1.2 Initialize variables

1.3 Link pointer and open file

2. Input choice

```
1   /* Fig. 11.16: fig11 16.c
2      This program reads a random access file sequentially,
3      updates data already written to the file, creates new
4      data to be placed in the file, and deletes data
5      already in the file.                                  */
6   #include <stdio.h>
7
8   struct clientData {
9      int acctNum;
10     char lastName[ 15 ];
11     char firstName[ 10 ];
12     double balance;
13  };
14
15  int enterChoice( void );
16  void textFile( FILE * );
17  void updateRecord( FILE * );
18  void newRecord( FILE * );
19  void deleteRecord( FILE * );
20
21  int main()
22  {
23     FILE *cfPtr;
24     int choice;
25
26     if ( ( cfPtr = fopen( "credit.dat", "r+" ) ) == NULL )
27        printf( "File could not be opened.\n" );
28     else {
29
30        while ( ( choice = enterChoice() ) != 5 ) {
31
32           switch ( choice ) {
```



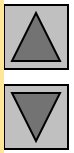
Outline

2.1 Perform action

3. Close file

3.1 Function definitions

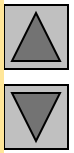
```
33             case 1:
34                textFile( cfPtr );
35                break;
36             case 2:
37                updateRecord( cfPtr );
38                break;
39             case 3:
40                newRecord( cfPtr );
41                break;
42             case 4:
43                deleteRecord( cfPtr );
44                break;
45          }
46       }
47
48       fclose( cfPtr );
49    }
50
51    return 0;
52 }
53
54 void textFile( FILE *readPtr )
55 {
56    FILE *writePtr;
57    struct clientData client = { 0, "", "", 0.0 };
58
59    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL )
60       printf( "File could not be opened.\n" );
61    else {
62       rewind( readPtr );
63       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
64                "Acct", "Last Name", "First Name","Balance" );
```



Outline

3.1 Function definitions

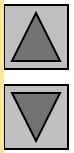
```
65
66          while ( !feof( readPtr ) ) {
67             fread( &client, sizeof( struct clientData ), 1,
68                    readPtr );
69
70             if ( client.acctNum != 0 )
71                fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
72                        client.acctNum, client.lastName,
73                        client.firstName, client.balance );
74          }
75
76          fclose( writePtr );
77       }
78
79    }
80
81    void updateRecord( FILE *fPtr )
82    {
83       int account;
84       double transaction;
85       struct clientData client = { 0, "", "", 0.0 };
86
87       printf( "Enter account to update ( 1 - 100 ): " );
88       scanf( "%d", &account );
89       fseek( fPtr,
90              ( account - 1 ) * sizeof( struct clientData ),
91              SEEK SET );
92       fread( &client, sizeof( struct clientData ), 1, fPtr );
93
94       if ( client.acctNum == 0 )
95          printf( "Acount #%d has no information.\n", account );
96       else {
```



Outline

3.1 Function definitions

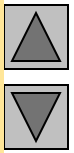
```
97          printf( "%-6d%-16s%-11s%10.2f\n\n",
98                  client.acctNum, client.lastName,
99                  client.firstName, client.balance );
100         printf( "Enter charge ( + ) or payment ( - ): " );
101         scanf( "%lf", &transaction );
102         client.balance += transaction;
103         printf( "%-6d%-16s%-11s%10.2f\n",
104                 client.acctNum, client.lastName,
105                 client.firstName, client.balance );
106         fseek( fPtr,
107                 ( account - 1 ) * sizeof( struct clientData ),
108                 SEEK_SET );
109         fwrite( &client, sizeof( struct clientData ), 1,
110                  fPtr );
111      }
112   }
113
114   void deleteRecord( FILE *fPtr )
115   {
116      struct clientData client,
117                        blankClient = { 0, "", "", 0 };
118      int accountNum;
119
120      printf( "Enter account number to "
121              "delete ( 1 - 100 ): " );
122      scanf( "%d", &accountNum );
123      fseek( fPtr,
124             ( accountNum - 1 ) * sizeof( struct clientData ),
125             SEEK_SET );
126      fread( &client, sizeof( struct clientData ), 1, fPtr );
```



Outline

3.1 Function definitions

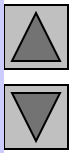
```
127
128    if ( client.acctNum == 0 )
129       printf( "Account %d does not exist.\n", accountNum );
130    else {
131       fseek( fPtr,
132          ( accountNum - 1 ) * sizeof( struct clientData ),
133          SEEK_SET );
134       fwrite( &blankClient,
135               sizeof( struct clientData ), 1, fPtr );
136    }
137 }
138
139 void newRecord( FILE *fPtr )
140 {
141    struct clientData client = { 0, "", "", 0.0 };
142    int accountNum;
143    printf( "Enter new account number ( 1 - 100 ): " );
144    scanf( "%d", &accountNum );
145    fseek( fPtr,
146           ( accountNum - 1 ) * sizeof( struct clientData ),
147           SEEK_SET );
148    fread( &client, sizeof( struct clientData ), 1, fPtr );
149
150    if ( client.acctNum != 0 )
151       printf( "Account #%d already contains information.\n",
152               client.acctNum );
153    else {
154       printf( "Enter lastname, firstname, balance\n? " );
155       scanf( "%s%s%lf", &client.lastName, &client.firstName,
156              &client.balance );
```



Outline

3.1 Function definitions

```
157         client.acctNum = accountNum;
158         fseek( fPtr, ( client.acctNum - 1 ) *
159               sizeof( struct clientData ), SEEK_SET );
160         fwrite( &client,
161                 sizeof( struct clientData ), 1, fPtr );
162      }
163   }
164
165   int enterChoice( void )
166   {
167      int menuChoice;
168
169      printf( "\nEnter your choice\n"
170         "1 - store a formatted text file of acounts called\n"
171         "    \"accounts.txt\" for printing\n"
172         "2 - update an account\n"
173         "3 - add a new account\n"
174         "4 - delete an account\n"
175         "5 - end program\n? " );
176      scanf( "%d", &menuChoice );
177      return menuChoice;
178   }
```



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
```

```
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
```