11.9 Case Study: Transaction-Processing Program

- We now present a substantial transaction-processing program (Fig. 11.15) using random-access files.
- The program maintains a bank's account information—updating existing accounts, adding new accounts, deleting accounts and storing a listing of all the current accounts in a text file for printing.
- We assume that the program of Fig. 11.10 has been executed to create the file `credit.dat`.

11.9 Case Study: Transaction-Processing Program (Cont.)

- The program has five options.
- Option 1 calls function `textFile` to store a formatted list of all the accounts (typically called a report) in a text file called `accounts.txt` that may be printed later.
- The function uses `fread` and the sequential file access techniques used in the program of Fig. 11.14.

11.9 Case Study: Transaction-Processing Program (Cont.)

- Option 2 calls the function `updateRecord` to update an account.
- The function will update only a record that already exists, so the function first checks whether the record specified by the user is empty.
- The record is read into structure `client` with `fread`, then member `acctNum` is compared to 0.
- If it’s 0, the record contains no information, and a message is printed stating that the record is empty.
- Then the menu choices are displayed.
- If the record contains information, function `updateRecord` inputs the transaction amount, calculates the new balance and rewrites the record to the file.

11.9 Case Study: Transaction-Processing Program (Cont.)

- Option 3 calls the function `newRecord` to add a new account to the file.
- If the user enters an account number for an existing account, `newRecord` displays an error message indicating that the record already contains information, and the menu choices are printed again.
- This function uses the same process to add a new account as does the program in Fig. 11.11.

11.9 Case Study: Transaction-Processing Program (Cont.)

- Option 4 calls function `deleteRecord` to delete a record from the file.
- Deletion is accomplished by asking the user for the account number and reinitializing the record.
- If the account contains no information, `deleteRecord` displays an error message indicating that the account does not exist.
- Option 5 terminates program execution.
- The program is shown in Fig. 11.15.
- The file `"credit.dat"` is opened for update (reading and writing) using `"rb+"` mode.

```c
// Fig. 11.15: fig11_15.c
// Transaction-processing 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 stored in the file.
#include <stdio.h>

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

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

```

Fig. 11.15 | Transaction-processing program. (Part 1 of 11.)

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

   // fopen opens the file; exits if file cannot be opened
   if ((cfPtr = fopen("accounts.dat", "rb+")) == NULL) {
      puts("File could not be opened.");
   }
   else {
      unsigned int choice; // user's choice

      // enable user to specify action
      while ((choice = enterChoice()) != 5) {
         switch (choice) {
            // create text file from record file
            case 1:
               textFile(cfPtr);
               break;
```

Fig. 11.15 | Transaction-processing program. (Part 2 of 11.)

```
            // update record
            case 2:
               updateRecord(cfPtr);
               break;
            // 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:
               puts("Incorrect choice");
               break;
         }
      }

      fclose(cfPtr); // fclose closes the file
   }
}

```

Fig. 11.15 | Transaction-processing program. (Part 3 of 11.)

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

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

```

Fig. 11.15 | Transaction-processing program. (Part 4 of 11.)

```c
      // copy all records from random-access file into text file
      while (!feof(readPtr)) {
         // create clientData with default information
         struct clientData client = { 0, "", "", 0.0 };
         int result =
            fread(&client, sizeof(struct clientData), 1, readPtr);

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

      fclose(writePtr); // fclose closes the file
   }
}

```

Fig. 11.15 | Transaction-processing program. (Part 5 of 11.)

```c
// update balance in record
void updateRecord(FILE *fPtr)
{
   // obtain number of account to update
   printf("%s", "Enter account to update (1 - 100): ");
   unsigned int account; // account number
   scanf("%d", &account);

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

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

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

   // display error if account does not exist
   if (client.acctNum == 0) {
      printf("Account #%d has no information.\n", account);
   }
```

Fig. 11.15 | Transaction-processing program. (Part 6 of 11.)

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

       // request transaction amount from user
       printf("%s", "Enter charge (+) or payment (-): ");
       double transaction; // transaction amount
       scanf("%lf", &transaction);
       client.balance += transaction; // update record balance

       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);
    }
 }

```

Fig. 11.15 | Transaction-processing program. (Part 7 of 11.)

```c
// delete an existing record
void deleteRecord(FILE *fPtr)
{
   // obtain number of account to delete
   printf("%s", "Enter account number to delete (1 - 100): ");
   unsigned int accountNum; // account number
   scanf("%d", &accountNum);

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

   struct clientData client; // stores record read from file

   // 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);
   }
```

Fig. 11.15 | Transaction-processing program. (Part 8 of 11.)

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

         struct clientData blankClient = {0, "", "", 0}; // blank client

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

   // create and insert record
   void newRecord(FILE *fPtr)
   {
      // obtain number of account to create
      printf("%s", "Enter new account number (1 - 100): ");
      unsigned int accountNum; // account number
      scanf("%d", &accountNum);

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

Fig. 11.15 | Transaction-processing program. (Part 9 of 11.)

```c

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

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

   // display error if account already exists
   if (client.acctNum != 0) {
      printf("Account #%d already contains information.\n",
         client.acctNum);
   }
   else { // create record
      // user enters last name, first name and balance
      printf("%s", "Enter lastname, firstname, balance\n? ");
      scanf("%14s%9s%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);

```

Fig. 11.15 | Transaction-processing program. (Part 10 of 11.)

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

// enable user to input menu choice
unsigned int enterChoice(void)
{
   // display available options
   printf("%s", "\nEnter your choice\n"
      "1 - store a formatted text file of accounts 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? ");

   unsigned int menuChoice; // variable to store user's choice
   scanf("%u", &menuChoice); // receive choice from user
   return menuChoice;
}
```

Fig. 11.15 | Transaction-processing program. (Part 11 of 11.)