RANDOM ACCESS FILE

A random-access data file enables you to read or write information anywhere in the file. Random access allows you to access any record directly at any position in the file. Individual records of a random-access file are normally fixed in length and may be accessed directly without searching through other records. This makes random-access files appropriate for airline reservation systems, banking systems, point-of-sale systems, and other kinds of transaction- processing systems.

Random access is sometimes called direct access. C supports the following functions for random access file processing of a binary file:

1. fseek()
2. ftell()
3. rewind()
4. fgetpos()
5. fsetpos()

fseek()

This function is used for setting the file pointer at the specified byte. On successful operation, fseek() returns zero. Otherwise it returns a non-zero value.

Syntax:

int fseek(FILE *fp, long displacement, int origin);

Here,

fp – file pointer

displacement – denotes the number of bytes which are moved backward or forward in the file. It can be positive or negative.

origin – denotes the position relative to which the displacement takes place. It takes one of the following three values.

Constant	**Value**	**Position**
SEEK_SET	0	Beginning of file
SEEK_CURRENT	1	Current position
SEEK_END	2	End of file

Origin field in fseek() function

ftell()

This function returns the current position of the file position pointer. The value is counted from the beginning of the file. If successful, ftell() returns the current file position. In case of error, it returns -1.

Syntax:

long ftell(FILE *fp);

Here, fp – file pointer.

rewind()

This function is used to move the file pointer to the beginning of the file. This function is useful when we open file for update.

Syntax:

void rewind(FILE *fp);

Here, fp – file pointer.

It is impossible to determine if rewind() was successful or not.

fgetpos()

The fgetpos() is used to determine the current position of the stream. The fgetpos function stores the current position of stream into the object pointed to by pos. The fgetpos function returns zero if successful. If an error occurs, it will return a nonzero value.

Syntax:

int fgetpos(FILE *stream, fpos_t *pos);

Here,

stream - The stream whose current position is to be determined.

pos - The current position of stream to be stored.

fsetpos()

The fsetpos() function moves the file position indicator to the location specified by the 'pos' returned by the fgetpos() function. If fsetpos() function successful, it return zero otherwise returns nonzero value.

Syntax:

int fsetpos(FILE *stream, const fpos_t *pos);

Here,

stream − This is the pointer to a FILE object that identifies the stream.

pos − This is the pointer to a fpos_t object containing a position previously obtained with fgetpos.

Difference Between Sequential Access Files And Random Access Files

Sequential Access Files	**Random Access Files**
Sequential Access to a data file means that the computer system reads or writes information to the file sequentially, starting from the beginning of the file and proceeding step by step.	Random Access to a file means that the computer system can read or write information anywhere in the data file.
When you access information in the same order all the time, sequential access is faster than random access.	In a random access file, you can search through it and find the data you need more easily.
To read the last record of the file, we have to read all the records from the beginning.	To read the last record of the file, we can read the last record directly.
Example: Tape drives	Example: Hard drives

EXAMPLE PROGRAM: Transaction Processing Using Random Access Files

```
struct clientData
{
        unsigned int acctNum;
        char lastName[ 15 ];
        char firstName[ 10 ];
        double balance;
};
unsigned int enterChoice( void );
void textFile( FILE *readPtr );
void updateRecord( FILE *fPtr );
void newRecord( FILE *fPtr );
void deleteRecord( FILE *fPtr );
```

```
int main( void )
{
        FILE *cfPtr;
        unsigned int choice;
        if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL )
        {
                puts( "File could not be opened." );
        }
        else
        {
                while ( ( choice = enterChoice() ) != 5 )
                {
                        switch ( choice )
                        {
                                // create text file from record file
                                case 1:
                                        textFile( cfPtr );
                                        break;
                                // update record
                                case 2:
                                updateRecord( cfPtr );
                                break;
                                // create record
                                case 3:
```

```
                    newRecord( cfPtr );
                    break;
                    // delete existing record
                    case 4:
                        deleteRecord( cfPtr );
                        break;
                    default:
                        puts( "Incorrect choice" );
                        break;
                }
            }
            fclose( cfPtr );
        }
}
void textFile( FILE *readPtr )
{
        FILE *writePtr;
        int result;
        struct clientData client = { 0, "", "", 0.0 };
        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" );
            while ( !feof( readPtr ) )
            {
                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 );
        }
}
void updateRecord( FILE *fPtr )
{
        unsigned int account;
        double transaction;
        struct clientData client = { 0, "", "", 0.0 };
        printf( "%s", "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( "Account #%d has no information.\n", account );
        }
        else
        {
                printf( "%-6d%-16s%-11s%10.2f\n\n",client.acctNum, client.lastName,
                client.firstName, client.balance );
                printf( "%s", "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 );
                    // write updated record over old record in file
                    fwrite( &client, sizeof( struct clientData ), 1, fPtr );
          }
}
void deleteRecord( FILE *fPtr )
{
          struct clientData client;
          struct clientData blankClient = { 0, "", "", 0 };
          unsigned int accountNum;
          printf( "%s", "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 );
          if ( client.acctNum == 0 )
          {
                              printf( "Account %d does not exist.\n", accountNum );
          }
          else
          {
                    fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),SEEK_SET );
                    // replace existing record with blank record
                    fwrite( &blankClient,sizeof( struct clientData ), 1, fPtr );
          }
}
void newRecord( FILE *fPtr )
{
          struct clientData client = { 0, "", "", 0.0 };
          unsigned int accountNum;
```

```
        printf( "%s", "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( "%s", "Enter lastname, firstname, balance\n? " );
                scanf( "%14s%9s%lf", &client.lastName, &client.firstName,
                &client.balance );
                client.acctNum = accountNum;
               fseek( fPtr, ( client.acctNum - 1 ) *sizeof( struct clientData ), SEEK_SET );
                fwrite( &client,sizeof( struct clientData ), 1, fPtr );
        }
}

unsigned int enterChoice( void )
{
        unsigned int menuChoice;
        printf( "%s", "\nEnter your choice\n"
        "1 - Display a formatted text file of accounts\n"
        "2 - update an account\n"
        "3 - add a new account\n"
        "4 - delete an account\n"
        "5 - end program\n? " );
        scanf( "%u", &menuChoice );
        return menuChoice;
}
```

Output:

```
Enter your choice
1 - Display a formatted text file of accounts
2 - update an account
3 - add a new account
4 - delete an account
5 - end program
?1
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 your choice
1 - Display a formatted text file of accounts
2 - update an account
3 - add a new account
4 - delete an account
5 - end program
?2
Enter account to update ( 1 - 100 ): 37
37 Barker     Doug    0.00
Enter charge ( + ) or payment (-): +87.99
37 Barker     Doug    87.99
Enter your choice
1 - Display a formatted text file of accounts
2 - update an account
3 - add a new account
4 - delete an account
```

```
5 - end program
?3
Enter new account number ( 1 - 100 ): 22
Enter lastname, firstname, balance ?
Johnston Sarah 247.45
Enter your choice
1 - Display a formatted text file of accounts
2 - update an account
3 - add a new account
4 - delete an account
5 - end program
?4
Enter account number to delete ( 1 - 100 ): 29
Enter your choice
1 - Display a formatted text file of accounts
2 - update an account
3 - add a new account
4 - delete an account
5 - end program
?5
```