FILE PROCESSING II

CEN116 Algorithms and Programming II

Random-Access Files

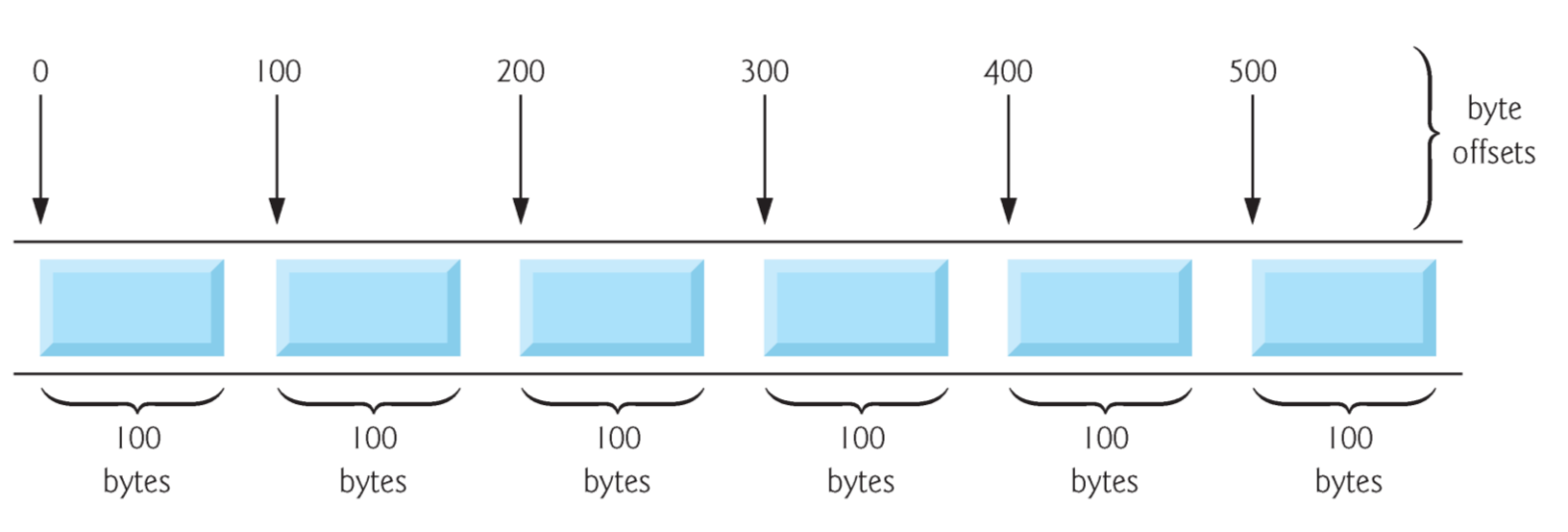
- As we stated previously, records in a file created with the formatted output function fprintf are not necessarily the same length.
- However, individual records of a random-access file are normally fixed in length and may be accessed directly (and thus quickly) without searching through other records.

Random-Access Files

- There are other ways of implementing random-access files, but we'll limit our discussion to this straightforward approach using fixed-length records.
- Because every record in a random-access file normally has the same length, the exact location of a record relative to the beginning of the file can be calculated as a function of the record key.
- We'll soon see how this facilitates immediate access to specific records, even in large files.

Random-Access Files

- Figure illustrates one way to implement a random-access file.
- Such a file is like a freight train with many cars—some empty and some with cargo.



Random-Access Files

- Fixed-length records enable data to be inserted in a random-access file without destroying other data in the file.
- Data stored previously can also be updated or deleted without rewriting the entire file.

Creating a Random-Access File

- Function **fwrite** transfers a specified number of bytes beginning at a specified location in memory to a file.
- The data is written beginning at the location in the file indicated by the file position pointer.
- Function **fread** transfers a specified number of bytes from the location in the file specified by the file position pointer to an area in memory beginning with a specified address.

Creating a Random-Access File

- Now, when writing an integer, instead of using

```
fprintf(fPtr, "%d", number);
```

which could print a single digit or as many as 11 digits (10 digits plus a sign, each of which requires 1 byte of storage) for a four-byte integer, we can use

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

which always writes four bytes on a system with four-byte integers from a variable number to the file represented by fPtr.

Creating a Random-Access File

- Later, fread can be used to read those four bytes into an integer variable number.
- Although fread and fwrite read and write data, such as integers, in fixed-size rather than variable-size format, the data they handle are processed in computer “raw data” format (i.e., bytes of data) rather than in printf’s and scanf’s human-readable text format.
- Because the “raw” representation of data is system dependent, “raw data” may not be readable on other systems, or by programs produced by other compilers or with other compiler options.

Creating a Random-Access File

- Functions fwrite and fread are capable of reading and writing arrays of data to and from disk.
- The third argument of both fread and fwrite is the number of elements in the array that should be read from or written to disk.
- The preceding fwrite function call writes a single integer to disk, so the third argument is 1 (as if one element of an array is being written).
- File-processing programs rarely write a single field to a file.
- Normally, they write one struct at a time, as we show in the following examples.

Creating a Random-Access File

Consider the following problem statement:

- Create a credit-processing system capable of storing up to 100 fixed-length records. Each record should consist of an account number that will be used as the record key, a last name, a first name and a balance. The resulting program should be able to update an account, insert a new account record, delete an account and list all the account records in a formatted text file for printing. Use a random-access file.

Creating a Random-Access File

- Next figure shows how to open a random-access file, define a record format using a struct, write data to the disk and close the file.
- This program initializes all 100 records of the file "credit.dat" with empty structs using the function fwrite.
- Each empty struct contains 0 for the account number, “ " (the empty string) for the last name, “ " for the first name and 0.0 for the balance.
- The file is initialized in this manner to create space on the disk in which the file will be stored and to make it possible to determine whether a record contains data.

```c
#include <stdio.h>

typedef struct {
    int acctNum; // account number
    char lastName[15]; // account last name
    char firstName[10]; // account first name
    float balance; // account balance
}client_t;

int main(void)
{
  FILE *fPtr; // accounts.dat file pointer
  // fopen opens the file; exits if file cannot be opened
  if((fPtr = fopen("accounts.dat", "wb")) == NULL) {
    puts("File could not be opened.");
  }
  else {
    client_t blankClient = {0, "", "", 0.0};
    for (int i = 1; i <= 100; ++i) {
      fwrite(&blankClient, sizeof(client_t), 1, fPtr);
    }
    fclose (fPtr); // fclose closes the file
  }
  return 0;
}
```

Creating a Random-Access File

- Function fwrite writes a block bytes to a file.
- fwrite causes the structure blankClient of size sizeof(struct client_t) to be written to the file pointed to by fPtr.
- The operator sizeof returns the size in bytes of its operand in parentheses (in this case struct client_t).
- Function fwrite can actually be used to write several elements of an array of objects.
- Writing a single object is equivalent to writing one element of an array, hence the 1 in the fwrite call.

Writing Data Randomly to a Random-Access File

- Next example writes data to the file "credit.dat".
- It uses the combination of fseek and fwrite to store data at specific locations in the file.
- Function fseek sets the file position pointer to a specific position in the file, then fwrite writes the data.

```c
#include <stdio.h>

typedef struct {
   int acctNum;
   char lastName[ 15 ];
   char firstName[ 10 ];
   float balance;
} client_t;

int main()
{
   FILE *fPtr;
   client_t client = { 0, "", "", 0.0 };
   if ( (fPtr = fopen( "credit.dat", "rb+" ) ) == 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 %f", client.lastName, client.firstName, &client.balance);
         fseek(fPtr, ( client.acctNum - 1 ) * sizeof(client_t), SEEK_SET);
         fwrite(&client, sizeof( client_t ), 1, fPtr);
         printf("Enter account number\n? ");
         scanf("%d", &client.acctNum);
      }
      fclose(fPtr);
   }
   return 0;
}
```

Sample Execution

```
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
```

Writing Data Randomly to a Random-Access File

- fseek, position the file position pointer for the file referenced by fPtr to the byte location calculated by

  *(client.accountNum - 1) * sizeof(client_t).*

- The value of this expression is called the offset or the displacement.
- Because the account number is between 1 and 100 but the byte positions in the file start with 0, 1 is subtracted from the account number when calculating the byte location of the record.

Writing Data Randomly to a Random-Access File

- Thus, for record 1, the file position pointer is set to byte 0 of the file.
- The symbolic constant SEEK_SET indicates that the file position pointer is positioned relative to the beginning of the file by the amount of the offset.
- As the above statement indicates, a seek for account number 1 in the file sets the file position pointer to the beginning of the file because the byte location calculated is 0.

Writing Data Randomly to a Random-Access File

- The function prototype for fseek is

  int fseek(FILE *stream, int offset, int whence);

- where offset is the number of bytes to seek from whence in the file pointed to by stream—a positive offset seeks forward and a negative one seeks backward.
- Argument whence is one of the values SEEK_SET, SEEK_CUR or SEEK_END (all defined in <stdio.h>), which indicate the location from which the seek begins.

Writing Data Randomly to a Random-Access File

- SEEK_SET indicates that the seek starts at the beginning of the file; SEEK_CUR indicates that the seek starts at the current location in the file; and SEEK_END indicates that the seek starts at the end of the file.
- For simplicity, the programs in this chapter do not perform error checking.
- Industrial-strength programs should determine whether functions such as fscanf, fseek and fwrite operate correctly by checking their return values.

Writing Data Randomly to a Random-Access File

- Function fscanf returns the number of data items successfully read or the value EOF if a problem occurs while reading data.
- Function fseek returns a nonzero value if the seek operation cannot be performed.
- Function fwrite returns the number of items it successfully output.
- If this number is less than the third argument in the function call, then a write error occurred.

Reading Data Randomly to a Random-Access File

- Function fread reads a specified number of bytes from a file into memory.
- For example,

  fread(&client, sizeof(client_t), 1, fPtr);

  reads the number of bytes determined by sizeof(client_t) from the file referenced by fPtr, stores the data in client and returns the number of bytes read.
- The bytes are read from the location specified by the file position pointer.

Reading Data Randomly to a Random-Access File

- Function fread can read several fixed-size array elements by providing a pointer to the array in which the elements will be stored and by indicating the number of elements to be read.
- The preceding statement reads one element should be read.
- To read more than one, specify the number of elements as fread's third argument.
- Function fread returns the number of items it successfully input.

Reading Data Randomly to a Random-Access File

- If this number is less than the third argument in the function call, then a read error occurred.
- Next example reads sequentially every record in the "credit.dat" file, determines whether each record contains data and displays the formatted data for records containing data.
- Function feof determines when the end of the file is reached, and the fread function transfers data from the file to the client_t structure client.

```c
#include <stdio.h>

typedef struct {
  int acctNum;
  char lastName[15];
  char firstName[10];
  float balance;
}client_t;

int main() {
  FILE * fPtr;
  client_t client = { 0,"","",0.0};
  if ((fPtr = fopen("accounts.dat", "rb")) == NULL) {
    printf("File could not be opened.\n");
  }
  else {
    printf("%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
      "First Name", "Balance");
    while (!feof(fPtr)) {
      fread( & client, sizeof(client_t), 1, fPtr);
      if (client.acctNum != 0) {
        printf("%-6d%-16s%-11s%10.2f\n",
          client.acctNum, client.lastName,
          client.firstName, client.balance);
      }
    }
    fclose(fPtr);
  }
  return 0;
}
```

References

- C How to Program, 8ed, by Paul Deitel and Harvey Deitel, Pearson, 2016.