**Tutorial - Week 9 Solution**

**Objectives:** To practice with

* Text files
* Binary files

1. **What is the difference between a file and a file stream?**

**A file** is an external collection of data **that is not a part of a C program.** C must provide a linkage between the external file and the program. **The linkage is provided by File Streams**. A File Stream is an interface between a program and a physical device It separates the program from the physical device by **assigning each file a logical structure**

1. **Open file report.txt for reading. Use appropriate error recovery in case of failure.**

We’ll assume that the file is a text file given its extension; therefore, the access mode is “r”

**FILE\*** inFile;

**char** fileName[] = “report.txt”;

inFile = **fopen**( fileName, “r” ); **if**( inFile == **NULL** )

{

**fprintf**( **stderr**, “Error opening %s”, fileName ); **return -1;**

}

1. Write a fragment of a program that counts and displays the number of characters stored in a file. You can assume that the file has been opened successfully.

Answer:

As there is no indication if the file is text or binary, we’ll use the function fgetc()

**int** character, count = 0;

**while**((character = **fgetc(**infile)) != **EOF** ) count++;

**if**(!**ferror**(inFile))

**printf**(“Num of char = %d “, count);

**else**

printf(“File Read Error”);

The C library function **int fgetc(FILE \*stream)** gets the next character (an unsigned char) from the specified stream and advances the position indicator for the stream.

This function returns the character read as an unsigned char cast to an int or EOF on end of file or error.

The C library function int ferror (FILE \*stream) tests the error indicator for the given stream.

1. **Open a binary file results.dat for output. If the file with such a name does not exist, it is created. If the file already exists, its content is preserved. Use appropriate error recovery in case of failure.**

**FILE** \*outFile;

**char** filename[] = “results.dat”;

outFile = **fopen**( filename, "**ab**" ); **if**( outFile == **NULL** )

{

/\* error recovery action \* /

}

1. **Write a fragment of a program, which reads all experiment data stored in a binary file block-by-block, calculates an average value for a block of data and prints it with 2 decimal point precision. All data are stored as type float. The block size is 128. You can assume that the file has been opened successfully.**

#define SIZE 128

FILE \* inData;

……

float average, sum, buffer[SIZE]; **int** i;

while(**fread**( buffer, **sizeof**(float), SIZE, inData ) == SIZE)

{

sum=0.0;

for(i=0; i<SIZE; i++)

sum += buffer[i];

**printf**("Block average = %.2f", sum/SIZE);

}

**if**(!**feof**(inData))

**printf**(“beware!!! Not all the file content has been read”);

**if**(**ferror**(inData))

**printf**(“beware!!! Hw Failure during the processing”);

1. **A color image file image.raw sequentially stores a block of red color samples, a block of green color samples followed by a block of blue color samples. Each block is of size 76800 elements of type unsigned char. Write a fragment of code that opens binary file image.raw for reading. Use appropriate error recovery in case of failure. Set the file position indicator to the beginning of the green color block. Use appropriate error recovery in case** **of failure.**

|  |  |  |
| --- | --- | --- |
|  |  |  |

#define BLOCK\_SIZE 76800

fseek function returns zero if successful, or else it returns a non-zero value.

**FILE** \*imageFile;

**char** filename[] = “image.raw”;

imageFile = **fopen**( filename, "rb" );

**if**(imageFile == NULL )

return –1;

**if**( **fseek**(imageFile, BLOCK\_SIZE, **SEEK\_SET** ) != 0 ) **return** –2;

1. **Define a function** saveRecord() **that stores one structured variable into a binary file. You can assume that the file position indicator has already been set. Use the following structure definition.**

**typedef** **struct**

{

**char** level; **int** code;

}element\_t;

/\*---function definition --\*/

**bool** saveRecord( **FILE**\* outFile, element\_t record )

{

**if**( **fwrite**( &record, **sizeof**(element\_t), 1, outFile) == 1 )

**return true**;

**else**

**return false;**

}

1. **Assume the environment shown, and complete the statements that follow so that they are valid:**

#define NAME\_LEN 50

#define SIZE 30

**typedef struct** {

**char** name[NAME\_LEN];

**int** age;

**double** income;

} person\_t;

. . .

**int** num\_err[SIZE];

**person\_t** exec;

/\* binary files \*/

**FILE** \*nums\_inp, \*psn\_inp, \*psn\_outp, \*nums\_outp;

/\* text files \*/

**FILE** \*nums\_txt\_inp, \*psn\_txt\_inp, \*psn\_txt\_outp;

nums\_inp = **fopen**("nums.bin", "rb");

nums\_txt\_inp = **fopen**("nums.txt", "r");

psn\_inp = **fopen**("persons.bin", "rb");

psn\_txt\_inp = **fopen**("persons.txt", "r");

psn\_outp = **fopen**("persout.bin", "wb");

psn\_txt\_outp = **fopen**("persout.txt", "w");

nums\_outp = **fopen**("numsout.bin", "wb");

1. **fscanf(psn\_txt\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, "%s", \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);**
2. **fwrite(num\_err, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,**

**nums\_outp);**

1. **fprintf(psn\_txt\_outp, "%s %d %f\n", \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);**

1. **fscanf**(psn\_txt\_inp, "%s", exec.name);
2. **fwrite**(num\_err, **sizeof** (int), SIZE, nums\_outp);
3. **fprintf**(psn\_txt\_outp, "%s %d %f\n", exec.name, exec.age, exec.income);
4. **Write a void function make\_product\_file that would convert a text file containing product information to a binary file of product\_t structures. The function’s parameters are file pointers to the text input and binary output files.**

**typedef struct** { /\* product structure type \*/

**int** stock\_num; /\* stock number \*/

**char** category[STR\_SIZ];

**char** tech\_descript[STR\_SIZ];

**double** price;

} product\_t;

/\*

\* Converts a text file of product data to a binary file of

product structures.

\*/

**void**

make\_product\_file(**FILE** \*fpin, /\* file pointer to text

input file \*/

**FILE** \*fpout) /\* file pointer to binary

ouput file \*/

{

**product\_t** p;

**int** status;

**while**(**fscanf**(fpin, "%d%s%s%lf", &p.stock\_num,p.category, p.tech\_descript, &p.price)==4)

{

status=**fwrite**(&p, **sizeof**(p), 1, fpout);

**if**(status<0)

{

printf(“ error during the writing);

break;

}

}

}

if(!feof(fpin))

printf(“ BEWARE!!! The input file was not read fully \n“);

if(ferror(fpin)|| ferror(fpout))

printf(“ BEWARE!!! There was a Hw Failure during the file processing \n“);

1. **Consider a file empstat.txt that contains employee records. The data for each employee consist of the employee’s name (up to 20 characters), social security number (up to 11 characters), gross pay for the week (double), taxes deducted (double), and net pay (double) for the week. Each record is a separate text line in file empstat.txt. Write a program that will create a text file report.txt with the heading line:**

**NAME SOC.SEC.NUM GROSS TAXES NET**

**followed by two blank lines and the pertinent information under each column heading. The program should also produce a binary file version of empstat.txt named empstat.bin.**

#include <stdio.h>

#define NAME\_LEN 21

#define SOCSEC\_LEN 12

**typedef struct** {

**char**  name[NAME\_LEN];

**char**  socsec[SOCSEC\_LEN];

**double** gross, taxes, net;

} emppay\_t;

**int main(void)**

{

**FILE** \*fpin, \*fpout, \*fpbin;

**emppay\_t** emp;

**int** status;

**if** ((fpin = **fopen**("empstat.txt", "r")) == **NULL**)

**printf**("Unable to open empstat.txt.\n");

**else if** ((fpout = **fopen**("report.txt", "w")) == **NULL**)

**printf**("Unable to open report.txt.\n");

**else if** ((fpbin = **fopen**("empstat.bin", "w")) == **NULL**)

**printf**("Unable to open empstat.bin.\n");

**else** {

**fprintf**(fpout, "NAME SOC.SEC.NUM GROSS TAXES NET**\n\n**");

**while** (**fscanf**(fpin, "%s%s%lf%lf%lf", emp.name,emp.socsec, &emp.gross, &emp.taxes,&emp.net)==5))

{

**fprintf**(fpout, "%‑20s%‑lls %8.2f %8.2f %8.2f\n",

emp.name, emp.socsec, emp.gross, emp.taxes,

emp.net);

**fwrite**(&emp, sizeof (emppay\_t), 1, fpbin);

}

}

**if** (fpin != NULL) **fclose**(fpin);

**if** (fpout != NULL) **fclose**(fpout);

**if** (fpbin != NULL) **fclose**(fpbin);

return (0);

}

Alternatively, you could have for each file stream , a dedicated fopen, then error recovery in case the file stream was not set up, i.e. remained NULL.