National University
of computer and emerging sciences

CL-1004

Object Oriented Programming- Lab

Spring' 2023

BS-SE

Lab Manual 13

Binary File Handling

Binary File Handling is a process in which we create a file and store data in its original format. It means that if we stored an integer value in a binary file, the value will be treated as an integer rather than text. Binary files are mainly used for storing records just as we store records in a database. It makes it easier to access or modify the records easily.

Binary File Opening Modes

Mode	Description
ios::binary | ios::in	This mode is used to open a file in binary mode for reading. The file pointer is placed at the beginning of the file. It is the default mode when a file is opened without any specified mode.
ios::binary | ios::out	This mode is used to open a file in binary mode for writing. If the file does not exist, it will be created. If the file already exists, it will be overwritten. The file pointer is placed at the beginning of the file
ios::binary | ios::app	This mode is used to open a file in binary mode for appending. If the file does not exist, it will be created. If the file already exists, all new data will be added to the end of the file.
ios::binary | ios::in | ios::out	This mode is used to open a file in binary mode for both reading and writing. The file pointer is placed at the beginning of the file.
ios::binary | ios::in | ios::out | ios::trunc	This mode is used to open a file in binary mode for both reading and writing, and truncates the file to zero length if it already exists. The file pointer is placed at the beginning of the file.
ios::binary | ios::in | ios::out | ios::app	This mode is used to open a file in binary mode for both reading and writing, and all new data will be added to the end of the file.
ios::binary | ios::in | ios::out | ios::ate	This mode is used to open a file in binary mode for both reading and writing, and the file pointer is placed at the end of the file.

Write primitive data, array and structure in a binary file

The program given below demonstrates how we can write primitive data, array and structure in a binary file.

```
#include <iostream>
#include <fstream>
```

```
#include <string.h>

using namespace std;

struct record
{
   int roll;
   char name[20];
} record;

int main()
{
   // primitive data types
   int i=56;
   float f=45.256;
   double d=12.54863;
   char c='d';

   // array
   int arr[3]= {15,74,29};

   // structure
   record x;
   x.roll=1;
   strcpy(x.name,"Peter");

   fstream fs;
   fs.open("e:/data.txt", ios::binary | ios::out);

   if(fs.is_open()==0)
   {
      cout<<"Cannot open file";
   }
   else
   {
      // writing primitive data types to binary file
      fs.write((char*) &i, sizeof(i));
      fs.write((char*) &f, sizeof(f));
      fs.write((char*) &d, sizeof(d));
      fs.write((char*) &c, sizeof(c));

      // writing an array to binary file
      fs.write((char*) &arr, sizeof(arr));
```

```
        // writing a structure to binary file
        fs.write((char*) &x, sizeof(x));
    }

    fs.close();
    return 0;
}
```

write((char*) data_address, data_size);

- (char*): Type cast the data to be written in the file as an array of characters.
- data_address: Points to the memory address of the data items to be written in a binary file.
- data_size: It specifies the number of bytes of the data item to be written in a binary file.

Read primitive data, array and structure from a binary file

The program given below demonstrates how we can read primitive data, array and structure that we have written previously in a binary file.

```
#include <iostream>
#include <fstream>

using namespace std;

struct record
{
    int roll;
    char name[20];
} record;

int main()
{
    // primitive data types
    int i;
    float f;
    double d;
    char c;

    // array
    int arr[3];

    // structure
    record x;

    fstream fs;
```

```
   fs.open("e:/data.txt", ios::binary | ios::in);

   if(fs.is_open()==0)
   {
      cout<<"Cannot open file";
   }
   else
   {
      // writing primitive data types to binary file
      fs.read((char*) &i, sizeof(i));
      fs.read((char*) &f, sizeof(f));
      fs.read((char*) &d, sizeof(d));
      fs.read((char*) &c, sizeof(c));

      // writing an array to binary file
      fs.read((char*) &arr, sizeof(arr));

      // writing a structure to binary file
      fs.read((char*) &x, sizeof(x));

      cout<<"Printing the primivite data"<<endl;
      cout<<"i="<<i<<endl;
      cout<<"f="<<f<<endl;
      cout<<"d="<<d<<endl;
      cout<<"c="<<c<<endl;
      cout<<"Printing the array"<<endl;
      cout<<arr[0]<<" "<<arr[1]<<" "<<arr[2]<<endl;
      cout<<"Printing the sturcture"<<endl;
      cout<<"roll="<<x.roll<<endl;
      cout<<"name="<<x.name<<endl;
   }

   fs.close();
   return 0;
}
```

Lab Task

Problem 1:

Create a program for writing records of 10 students in a file named as Record. Store Reg no, Full name, Program, and contact of student. Then reads records of all students in a file. Program must read roll no, name, Program and Contact of student. Program must display the records on the console from the file.

Problem 2:

Write a program that consist of two functions. The Input function takes the input from user about the specific topic entered by the user and at least ten lines about it and store the information into the file named as “info”. The output function counts the number of words in each line and show the count of lines having less than 10 words.

Problem 3:

Write a program to implement a telephone directory using formatted I/O file. You should write a class PhoneDirectory for storing the FullName, Number and email of the person. Your program should be capable of writing name, telephone number and email address into a data file phonedir. Then write class functions searchByName: which should read the file and must return the Phone number of searched person and searchByemail: which should read the file and must return the Phone number of person holding the searched email address. Also write a class function printDirectory which must read the complete directory and print all the records of the directory.

Problem 4:

Write a program that reads and prints a joke and its punch line from two different files. The first file contains a joke, but not its punch line. The second file has the punch line as its last line, preceded by “garbage.” The main function of your program should open the two files and then call two functions, passing each one the file it needs. The first function should read and display each line in the file it is passed (the joke file). The second function should display only the last line of the file it is passed (the punch line file). It should find this line by seeking to the end of the file and then backing up to the beginning of the last line. Data to test your program can be found in the joke and punchline files.

Problem 5:

Given a binary file f with size n, write a function calculate_entropy (f) that calculates the entropy H (f) of the file's contents. The entropy is a measure of the randomness of the file's data and is defined as:

$$H(f) = -\text{sum}[p(x) * \log2(p(x))]$$

Where p(x) is the probability of byte value x occurring in the file's data. The entropy is measured in bits and ranges from 0 (for completely predictable data) to log2(n) (for completely random data).Your function should take the file as input, read in the data, calculate the frequency of each

byte value, and then use the above formula to calculate the entropy of the file. The function should return the entropy as a double value.
For example, given a binary file with the following byte values:

```
0x41 0x41 0x42 0x43 0x43 0x43
```

The probability of each byte value occurring is:

```
p(0x41) = 2/6 = 1/3
p(0x42) = 1/6
p(0x43) = 3/6 = 1/2
```

So the entropy of the file is:

```
H(f) = -[(1/3) * log2(1/3) + (1/6) * log2(1/6) + (1/2) * log2(1/2)]
       = 1.45914791703 bits
```