Binary Files in In C (01-11-19)

01 November 2019 22:00

BINARY FILES IN C :-

Disadvantages of Text files :-

1) They are in human readable format and easily modifiable, and thus should not be used for storing records.

2) They need to be parsed by the programs to correctly read them.

Advantages of Binary Files :-

1) They are present in computer readable format, hence not directly understood by humans and does not facilitate easy modification like text files, hence can be used for storing records.

2) Any kind of data can be written to, and read from a binary file. starting from arrays, to variables, structures etc.

FILE MODES FOR A BINARY FILE :-

We have six file file modes for binary file :-

1. rb - for opening a file in read mode
2. wb - for opening a file in write mode
3. ab - for opening a file in append mode
4. rb+ - for opening a file in r/w mode
5. wb+ - " and same as write mode

6. a b + — " and same as append mode

BINARY FILE HANDLING FUNCTIONS :-

General syntax :-

fwrite (addressData, sizeData, numberData, file pointer)

fread (addressData, sizeData, numberData, file pointer)

Where,

addressData - The address of the data to be written in the disk

sizeData - The size of one data element to be written in the disk

numberData - The number of data items, we want to write in the disk

file pointer - The file pointer representing the output/input file.

WRITING A VARIABLE TO A FILE :-

```
FILE *fp = NULL;
fp = fopen("DUMMY.BIN", "wb");
```

→ Extensions for a binary file are .BIN, .DAT etc.

```
int num = 14;
```

→ integer to be written in the file

.BIN, .DAT etc.

```
int num = 14;
```

→ integer to be written in the file

```
fwrite(&num, sizeof(num), 1, fp);
```

sizeof(int) ↑

```
fclose(fp);
```

READING THE VARIABLE FROM THE FILE:-

```
FILE *fp = NULL;
fp = fopen("DUMMY.BIN", "rb");

int num;
```

→ 14

```
fread(&num, sizeof(num), 1, fp);

fclose(fp);
```

WRITING AN ARRAY TO A FILE:-

```
FILE *fp = NULL;
fp = fopen("DUMMY.BIN", "wb");

int arr[] = {1, 2, 3, 4, 5};
```

→ Array to be written in the file

```
fwrite(arr, sizeof(int), 5, fp);

fclose(fp);
```

NOTE: Instead of sizeof(int), we could have also written sizeof(arr[0])

READING AN ARRAY FROM A FILE:-

```
FILE *fp = NULL;
fp = fopen("DUMMY.BIN", "rb");

int arr[5];
```

→ 1, 2, 3, 4, 5,

→ The size of the array can be

```
int arr[5];
```
→ 1, 2, 3, 4, 5,

→ The size of the array can be larger than the array being read from the file

```
fread(arr, sizeof(int), 5, fp);

fclose(fp);
```

Q) Write a program in C to store Student records in a binary file.

Each Student should contain the following fields: roll, age, standard and name.

Implement the following operations.
(i) Creation of records
(ii) Displaying of records
(iii) Updation of records
(iv) Deletion of records
(v) Searching if a record exists

Structure definition :-

```
typedef struct{
    int roll;
    int age;
    int std;
    char name[1024];
}Student;
```

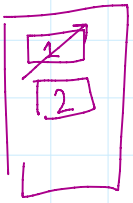
Taking input in a structure variable :-

```
void input(Student *s){
    printf("\nEnter your roll: ");
    scanf("%d", &(s->roll));
    printf("Enter your age: ");
    scanf("%d", &(s->age));
    printf("Enter your class: ");
    scanf("%d", &(s->std));
    printf("Enter your name: ");
→   fflush(stdin);
    gets(s->name);
}
```

Displaying the content of a structure variable :-

```
void display(Student *s){
    printf("%-5d %-5d %-5d %-s\n",
                s->roll,
                s->age,
                s->std,
                s->name);
}
```

Creating a record and saving it in the file :-



- File needs to be opened in append mode
- fwrite() is to be used.

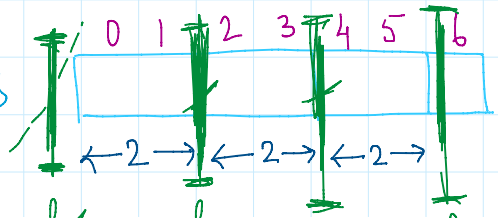
```
void writeRecord(char filename[],  Student *s){
    // Opening the file in append mode
    // such that we do not overwrite
    // existing records
    FILE *fp = fopen(filename, "ab");
    fwrite(s, sizeof(Student), 1, fp);
    fclose(fp);
}
```

Displaying all the records present in the file :-

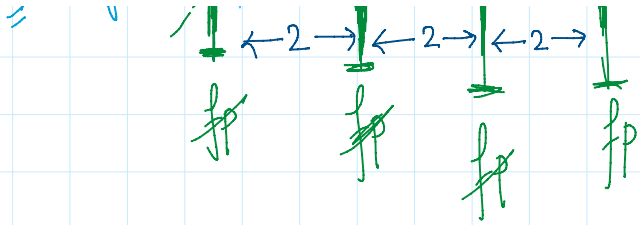
- File to be opened in read mode
- Read a structure from the file using fread()
- call display() with the read structure to display its contents.
- We continue until we have read all records.

How to determine that we have read all records :-

size = 6 bytes



ftell(fp)	(ftell(fp) < size)
0	0 < 6 – T



0	0 < 6 - T
2	2 < 6 - T
4	4 < 6 - T
6	6 < 6 - F

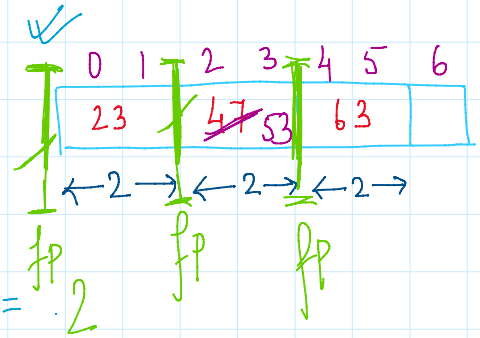
- Calculate size of file, by seeking to the end using fseek() and calling ftell() to get the size of the file in bytes.

```
void displayRecords(char filename[]){
    long size;
    Student temp;
    // Read Mode
    FILE *fp = fopen(filename, "rb");
    // Move to the end of the file
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    rewind(fp);
    printf("\n%-5s %-5s %-5s %-s\n", "ROLL", "AGE", "CLASS", "NAME");
    // We run the loop till the file pointer
    // is less than the total size of the file
    while(ftell(fp) < size){
        fread(&temp, sizeof(Student), 1, fp);
        display(&temp);
    }
    fclose(fp);
}
```

Updating a record in a file :-

- Open the file in read/write mode
- Read each record using fread().
- Check to see if the currently read record matches with the record that needs to be updated.
- Move the file pointer/cursor to the beginning of the record that needs to be updated and write the updated

that needs to be updated. and write the updated structure over it using fwrite()



To update roll 47 and change it to 53

sizeof(Student) = 2

size of a single record

ftell(fp)
0
2
4 - 2 = 2

fseek(fp, ftell(fp) - sizeof(Student), 0);

```
void updateRecord(char filename[], int roll, Student *s){
    long size;
    Student temp;
    FILE *fp = fopen(filename, "rb+");
    // Calculate the size of the file
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    rewind(fp);
    while(ftell(fp) < size){
        fread(&temp, sizeof(Student), 1, fp);
        if(temp.roll == roll){
            fseek(fp, ftell(fp) - sizeof(Student), SEEK_SET);
            fwrite(s, sizeof(Student), 1, fp);
            break;
        }
    }
    fclose(fp);
}
```

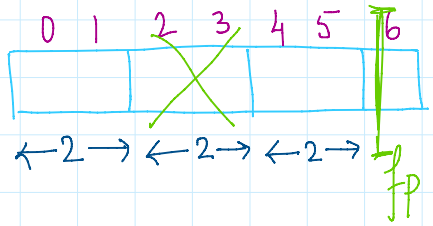
5

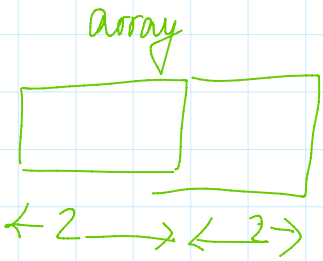
roll	5	→	roll = 5
age	15		age = 16
std	10		std = 10
name	Ayan		name = Ayan

Deleting a record from a file :-

- Open the file in read mode.
- Store all the records of the file except the record that you want to delete.

How to calculate the size of the array where the records will be stored?





sizeof(Student) = 2

size = 6 bytes

Number of records = [size / sizeof(Student)] − 1

- Close the file, and re-open in write mode.
- Write the content of the array to the file, using fwrite()

```
void deleteRecord(char filename[], int roll){
    long size;
    int total_records;
    Student temp;
    Student *arr = NULL;
    int index = 0;
    FILE *fp = fopen(filename, "rb");
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    rewind(fp);
    total_records = size / sizeof(Student);
    arr = (Student*) malloc(sizeof(Student) * (total_records - 1));
    while(ftell(fp) < size){
        fread(&temp, sizeof(Student), 1, fp);
        if(temp.roll != roll){
            arr[index] = temp;
            index++;
        }
    }
    fclose(fp);
    fp = fopen(filename, "wb");
```

```
    fwrite(arr, sizeof(Student), total_records - 1, fp);
    fclose(fp);
    free(arr);
}
```

Searching for a record in the file :-

- Open the file in read mode
- Like display, but after reading each record we check to see if the record we are looking for is found
  - If found, display "Record exists"
  - If not found, display "Record does not exist"

```
void searchRecords(char filename[], int roll){
    long size;
    int found = 0;
    Student temp;
    FILE *fp = fopen(filename, "rb");
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    rewind(fp);
    while(ftell(fp) < size){
        fread(&temp, sizeof(Student), 1, fp);
        if(temp.roll == roll){
            found = 1;
            break;
        }
    }
    fclose(fp);
    if(found){
        printf("Record exists!!\n");
    }
    else{
        printf("Record not found!\n");
    }
}
```

Finally, the main method :-

```c
int main(void)
{
    Student temp;
    int choice, roll;
    char filename[1024];
    clrscr();
    printf("Enter the name of the dbfile: ");
    gets(filename);
    do{
        printf("\nPress 1 to insert a Student record\n");
        printf("Press 2 to display the records\n");
        printf("Press 3 to update an existing record\n");
        printf("Press 4 to delete an existing record\n");
        printf("Press 5 to search for an record\n");
        printf("Press 6 to quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                input(&temp);
                writeRecord(filename, &temp);
                break;
            case 2:
                displayRecords(filename);
                break;
            case 3:
                printf("\nEnter Student roll to update: ");
                scanf("%d", &roll);
                printf("Enter the updated records\n");
                input(&temp);
                updateRecord(filename, roll, &temp);
                break;
            case 4:
                printf("\nEnter Student roll to delete: ");
                scanf("%d", &roll);
                deleteRecord(filename, roll);
                break;
            case 5:
                printf("\nEnter Student roll to search: ");
                scanf("%d", &roll);
                searchRecords(filename, roll);
                break;
            case 6:
                break;
            default:
                printf("Invalid choice!");
```

```
        }
    }
    while(choice != 6);
    return 0;
}
```

DRAWBACKS OF STORING FILES IN THIS FASHION :-

- Adhoc or purpose built database
- Each operation needs to be seperately implemented by writing new methods.
- Changing data model, requires re-writing of entire program
- Does not implement transactions, data integrity and security.
- Enter DBMS, Oracle DB, MySQL, PostgreSQL, etc.