Binary Files

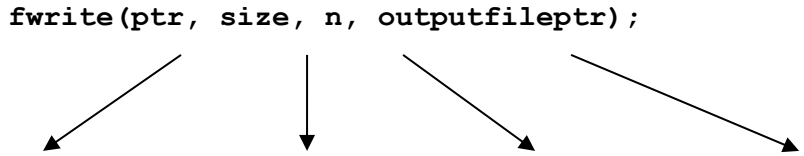
- A *binary file* is a file containing binary numbers (0's and 1's) which represent the computer's internal representation of each file component.
- It can not be created with a text editor. Instead, it can only be created as the output of a program.
- Similar to a text file, first it is necessary to declare a file pointer (using `FILE *`) and open it for input or output (using `fopen`).
- While opening a binary file, we need to append the letter "**b**" to the file opening mode .

The following table shows the *File Opening Modes* you can use in `fopen` statement:

Text	Binary	
`r`	`rb`	: open a file for reading
`w`	`wb`	: create a file for writing. If the file already exists, discard the current contents
`a`	`ab`	: append; open or create a file for writing at the end of file

```
FILE  *binaryp;
binaryp = fopen("data.bin", "wb");
```

- `fwrite` and `fread` functions are used to write to and read from a binary file.



start address of data to be written into the file

size of value to be written

# of values to be written

file pointer to file being created

means write `n` items of data of the size `size` into output file starting from `ptr`. (address of first memory cell to be copied to the file).

Example:

```
int num = 244;
fwrite(&num, sizeof(int), 1, binaryp);
```

writes the integer 244 to the binary file.

Example:

```
int score[10];
…
fwrite(score, sizeof(int), 10, binaryp);
```

writes the entire `score` array to the binary file.

Example: Write a program that creates a binary file called "nums.bin" which contains even integers from 2 to 500.

```
#include <stdio.h>
int main(void)
{
      FILE *binaryp;
      int i;
      binaryp = fopen("nums.bin", "wb");
      for (i = 2; i <= 500; i += 2)
           fwrite(&i, sizeof(int), 1, binaryp);
      fclose(binaryp);
      return (0);
}
```

- Writing the value of an integer variable `i` to a binary file using `fwrite` is faster than writing `i` to a text file using `fprintf`.

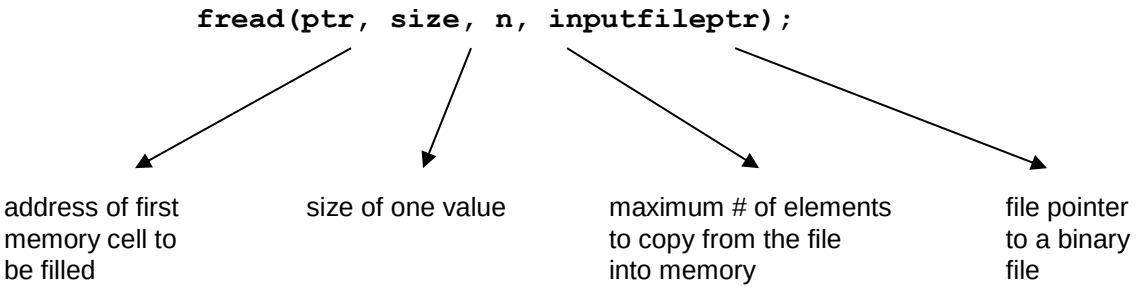
  ```
  i = 183790244;
  fprintf(textp, "%d ", i);
  ```

  writes the value of `i` to the file using ten characters (10 bytes). The computer must first convert the binary number in `i` to the character string "`183790244`" and then write the binary codes for each character (1, 8, ...) and blank to the file.

  ```
  fwrite(&i, sizeof(int), 1, binaryp);
  ```

  copies the internal representation of `183790244` to disk, without making any conversions, and using only 4 bytes. This is faster than the other.

- Each time, you write a double value to a text file, computer must convert it to a character string whose precision is determined by the placeholder. So, a loss of accuracy may occur. The disadvantages of binary file usage are that it is not readable, and it cannot be created or modified in an editor or word processor.



means read at most `n` items of data of the size `size` from the input file into the memory starting from the address `ptr`.

- `fread` returns the number of elements it successfully copies from file. It has to be less than or equal to the third argument.

- `fscanf` or `getc` can read a file that was created using a text editor or using `fprintf` or `putc`. `fread` can only read a file that was created using `fwrite`.

Example: Write a program that displays on the screen the contents of a binary file called "nums.bin" which contains integers.

```
#include <stdio.h>
int main(void)
{
      FILE *binaryp;
      int num;
      binaryp = fopen("nums.bin", "rb");
      while (!feof(binaryp))
      {
           fread(&num, sizeof(int), 1, binaryp);
           printf("%d\n", num);
      }
      fclose(binaryp);
      return (0);
}
```

- If you want to move to the beginning of an input file to be able to read it again, you can close it and reopen it in read mode.

- `rewind (filepointer)` is another function that causes a file pointer to be repositioned to the beginning of the file.

```
rewind(binaryp);
fread(&num, sizeof(int), 1, binaryp);
printf("%d\n", num);
```

reads and displays the first number in the binary file.

Example:

```
int k, ar[50];
rewind(binaryp);
fread(ar, sizeof(int), 50, binaryp);
for (k = 0; k < 50; k++)
      printf("%d\n", ar[k]);
```

reads and displays the first 50 numbers in the binary file.

- In the above example, if there are less than 50 numbers in the file, it does not cause any problem for `fread`. It reads only the existing ones. However, it causes a problem in the for loop, since it tries to display 50 numbers. We can solve this problem as follows:

```
int k, size, ar[50];
rewind(binaryp);
size = fread(ar, sizeof(int), 50, binaryp);
for (k = 0; k < size; k++)
      printf("%d\n", ar[k]);
```

Example: Write a function that takes two binary file pointers as arguments and copies everything from the first file to the second. The first file contains the id and age of some students.

```
typedef struct
{     int  id,
           age;
} std_t;


void copyBinFile(FILE *file1, FILE *file2)
{
      std_t std;
      while (!feof(file1))
      {
           fread(&std, sizeof(std_t), 1, file1);
           fwrite(&std, sizeof(std_t), 1, file2);
      }
}
-------------------------------------------------------------------------

/* Alternative solution */
void copyBinFile(FILE *file1, FILE *file2)
{
      std_t std;
      while (fread(&std, sizeof(std_t), 1, file1) != NULL)
             fwrite(&std, sizeof(std_t), 1, file2);
}
-------------------------------------------------------------------------

/* Alternative solution if the size of the first file is known */
void copyBinFile(FILE *file1, FILE *file2, int size) {
      std_t *std;

      std = (std_t *)malloc(size * sizeof(std_t));
      fread(std, sizeof(std_t), size, file1);
      fwrite(std, sizeof(std_t), size, file2);
}
-------------------------------------------------------------------------

/* Alternative solution if the size of the first file is not known;
   uses more memory */
void copyBinFile(FILE *file1, FILE *file2) {
      std_t std[500];
      int size; // number of students in the first file
      size = fread(std, sizeof(std_t), 500, file1);
      fwrite(std, sizeof(std_t), size, file2);
}
```

Example: Write a main program that makes a back-up copy of a binary file whose name is given as input, using `copyBinFile` function.

```
int main(void)
{
     char in_name[30];         // name of the input file
     char out_name[30]=”bu_”;  // name of the output file
     FILE *inp, *outp;

     printf("Enter the name of the file you want to back up> ");
     scanf("%s", in_name);
     inp = fopen(in_name, "rb");
     while (inp == NULL)
     {
          printf("Can not open the file %s!\n", in_name);
          printf("Re-enter the file name> ");
          scanf("%s", in_name);
          inp = fopen (in_name, "rb");
     }

     // The back-up file will have the prefix "bu_"
     outp = fopen(strcat(out_name, in_name), "wb");
     copyBinFile(inp, outp);

     fclose (inp);
     fclose (outp);
     return(0);
}
```

- It is possible to move the file pointer to a certain position in a file using `fseek` function.

```
fseek(filepointer, offset, whence);
```

repositions the file pointer to a new position that is `offset` bytes from the file location given by `whence`.

`whence` =
- `SEEK_SET` : seek starts at the beginning of the file.
- `SEEK_CUR` : seek starts at the current location of the file.
- `SEEK_END` : seek starts at the end of the file

Example:

```
fseek(binaryp, 5 * sizeof(int), SEEK_SET);
fread(&num, sizeof(int), 1, binaryp);
printf("%d\n", num);
```

reads and displays the sixth number in the binary file. If the `offset` was 0, then it would read and display the first number. If the `offset` was larger than the number of values in the file, the file pointer would move to the end of the file. In the above example, in such a case, it would not read anything, and so would display the last value of `num`.

Example: Write a function that takes a file pointer (to a file containing integers), a character representing the direction (T-Top, B-Bottom, C-Current), and a positive integer `n`, and moves the file pointer to the n^{th} integer, according to the given direction. For example, `move(binaryp, 'T', 5);` will move the file pointer to the fifth integer within the file. `move(binaryp, 'B', 5);` will move it to the fifth integer from the bottom.

```
void move(FILE *b_outp,
          char direction,    //direction as T-Top B-Bottom and C-Current
          int n) {
     switch(direction) {
          // if the direction is T-Top
          case 'T':
          case 't':
                fseek(b_outp, (n - 1) * sizeof(int), SEEK_SET);
                break;
          // if the direction is C-Current
          case 'C':
          case 'c':
                fseek(b_outp, (n - 1) * sizeof(int), SEEK_CUR);
                break;
          // if the direction is B-Bottom
          case 'B':
          case 'b':
                fseek(b_outp, (-n) * sizeof(int), SEEK_END);
          }
}

int main (void) {
     ...
     move(binaryp, 'T', 3);
     fread(&num, sizeof(int), 1, binaryp);
     printf("%d\n", num);
     move(binaryp, 'C', 1);
     fread(&num, sizeof(int), 1, binaryp);
     printf("%d\n", num);
     move(binaryp, 'B', 1);
     fread(&num, sizeof(int), 1, binaryp);
     printf("%d\n", num);
     move(binaryp, 'B', 2);
     fread(&num, sizeof(int), 1, binaryp);
     printf("%d\n", num);
```

reads and displays the third, fourth and last two numbers in the binary file.

- `ftell` function returns the current file position of the given stream.

```
int ftell(file pointer);
```

Example:

```
#include <stdio.h>
int main(void)
{    FILE *fp;
     int len;
     fp = fopen("file.bin", "rb");
     if (fp == NULL)
          printf("\nError opening file");
     else
     {
          fseek(fp, 0, SEEK_END);
          len = ftell(fp);
          fclose(fp);
          printf("Total size of file.bin = %d bytes\n", len);
     }
     return(0);
}
```

Let us assume we have a binary file **file.bin** having the following content:

- `This is CTIS152`

Now let us compile and run the above program that will produce the following result if file has above mentioned content otherwise it will give different result based on the file content:

```
Total size of file.bin = 16 bytes
```

Example

- Assume that the following structure has been defined:

```
typedef struct {
      int id;
      char name[15];
      double cgpa;
}stu_t
```

- Create a binary file with 3 students' information.

```
int main(void)
{  FILE *binaryp;
   stu_t S[3] = {{1111, "AYLIN", 3.00}, {2222, "NAZLI", 3.50},{3333, "CANSU", 4.00}};
   binaryp = fopen("stu.bin", "wb");
   fwrite(S, sizeof(stu_t), 3, binaryp);
   fclose(binaryp);
   return(0);
}
```

- Read unknown number of students' information from a binary file into an array by creating the array dynamically.

```
int main(void)
{  FILE *fp;
   stu_t *sptr;
   int I, size;
   fp = fopen("stu.bin", "rb");
   if (fp == NULL)
      printf("\nError opening file");
   else {
      fseek(fp, 0, SEEK_END);
      size = ftell(fp) / sizeof(stu_t);
      sptr = (stu_t *)malloc(size * sizeof(stu_t));
      rewind(fp);
      fread(sptr, sizeof(stu_t), size, fp);
      for (i=0; i<size; i++)
            printf("\n%d %s %0.2f", (sptr+i)->id, (sptr+i)->name, (sptr+i)->cgpa);
            // printf("\n%d %s %0.2f", sptr[i].id, sptr[i].name, sptr[i].cgpa);
      fclose(fp);
      free(sptr);
   }
   return(0);
}
```

Output:

```
1111 AYLIN 3.00
2222 NAZLI 3.50
3333 CANSU 4.00
```