Lecture 25

Files

Outline

- File handling in C - opening and closing.
- Reading from and writing to files.
- How we SHOULD read input from the user.

Introduction

- Data storages of computers
- 1- Main memory (RAM)
  - It is volatile
  - Read / Write data using variables
- 2-Secondary storage (Hard Disk)
  - It is not volatile
  - Read / Write data using files

What is a File?

- A *file* is a collection of related data that a computers treats as a single unit.
- When a computer reads a file, it copies the file from the storage device to memory; when it writes to a file, it transfers data from memory to the storage device.
- C uses a structure called **`FILE`** (defined in **`stdio.h`**) to store the attributes of a file.

Text & Binary Files

- How does computer store data?
  - They are coded
- When data are stored in main memory
  - It is variable
  - Coding is specified by the type: int, char, …
- When data are stored in secondary memory
  - It is file
  - Coding is specified by the file type: Text & Binary

Text Files

- ASCII encoding
- Each line is a string
- Each line is terminated by \n
- Human-readable files
  - Editable by text editor (e.g. Notepad)
- Examples
  - C source files
  - Every .txt files

Binary Files

- Binary encoding
  - int, double, float, struct, … are directly (as 0,1) stored in the file
- Human unreadable files
  - Is not editable by text editor
    - Needs special editor which understands the file
- Examples
  - .exe files
  - Media files such as .mp3
  - Picture files such as .bmp, .jpg

Working with Files

- Until now
  - We read/write data from/to terminal (console)

- In C
  - We can read data from file
  - We can write data to file

Working with Files

- Main steps in working with files
- 1) Open file
  - Get a file handler from Operating System
- 2) Read/Write
  - Use the handler
- 3) Close file
  - Free the handler
- 4) Other operations
  - Check end of file, …

Opening Files

- Function fopen opens files

```
#include <stdio.h>

FILE * fopen(char *name, char *mode);
```

- FILE * is struct
  - Saves information about file.
  - We don't need to know about it.
- If cannot open file, fopen returns NULL.
- name is the name of file:
  - Absolute name: C:\prog\test.txt
  - Relative name: Mytest.txt

Opening Files: Modes

- r: open for read. We cannot write to the file.
- w: open for write. Create new file. We cannot read form the file. If file exist, its content will be destroyed.
- a: open for write. We cannot read form the file. If file exist, its content wont be destroyed. We write at end of file.
- r+, w+, a+ : same to r, w, a but we can read and write.

Mode	Meaning	fopen Returns if FILE-	
		Exists	Not Exists
r	Reading	–	NULL
w	Writing	Over write on Existing	Create New File
a	Append	–	Create New File
r+	Reading + Writing	New data is written at the beginning overwriting existing data	Create New File
w+	Reading + Writing	Over write on Existing	Create New File
a+	Reading + Appending	New data is appended at the end of file	Create New File

Opening Files: Modes

- Files are
  - Text: Some strings
  - Binary: Image file, Video file, …

- To open binary file, we should add `b` to the mode.
  - `rb` : open binary file for read
  - `w+b`: create new binary file for read and write

Opening Files: Examples

```
FILE *fp;

fp = fopen("c:\test.txt", "r");

if(fp == NULL){

    printf("Cannot open file\n");

    return -1;

}
```

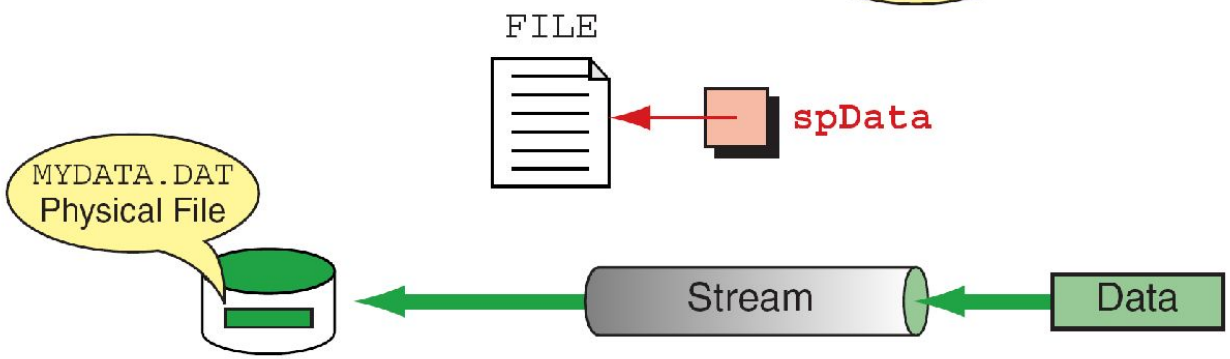
- Open file c:\test.txt for read

More on **fopen**

```
#include <stdio.h>
…
{
int main (void)
  FILE* spData;
  …
  spData = fopen("MYDATA.DAT", "w");
  …
} // main
```

Internal File Variable

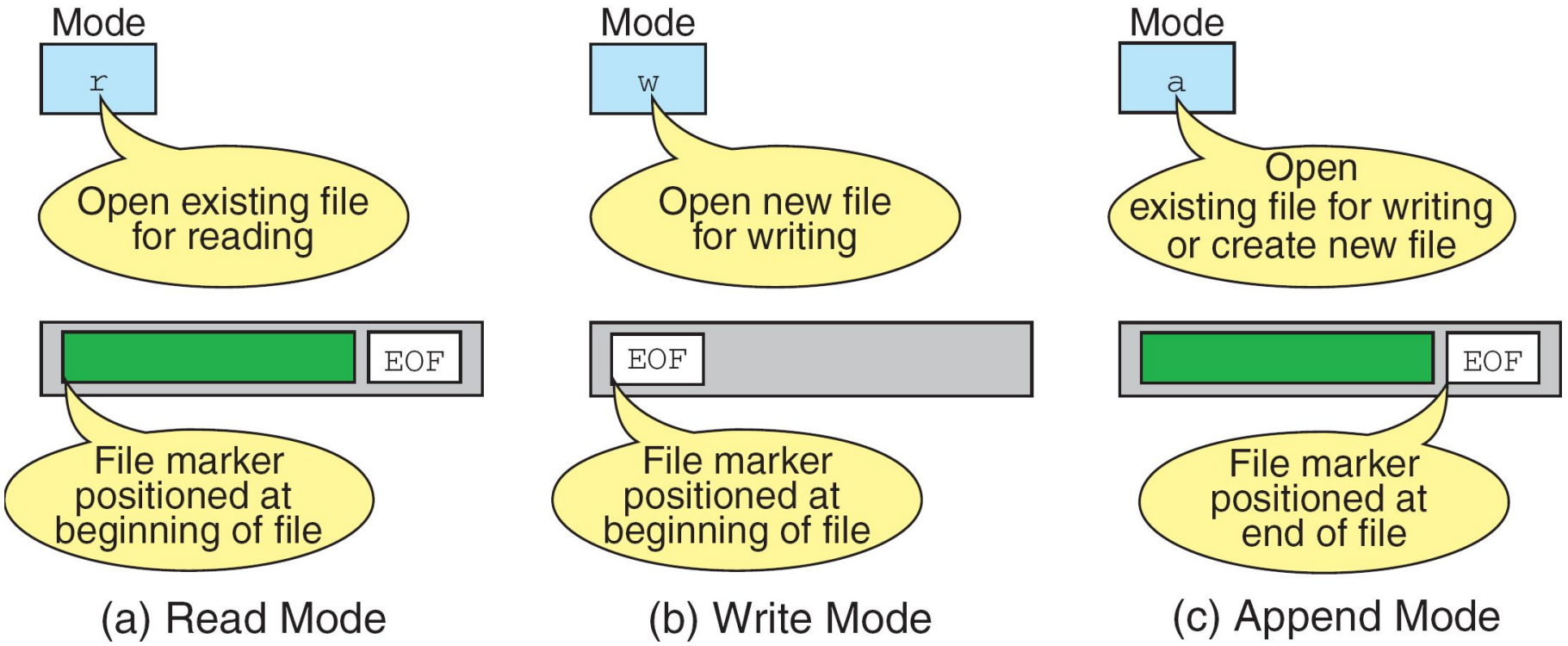
External File Name



File-Position pointer(FPP)

- File-Position Pointer
  - A pointer in file
  - Points to current location of read and write
- When file is open
  - File-Position Pointer is set to start of file
- When you read/write from/to file
  - The File-Position Pointer advance according to the size of data
    - If you read 2 bytes, it moves 2 bytes
    - If you write 50 bytes, it advances 50 bytes

More on File Open Modes



Closing Files

- Each opened file should be closed.
- If we write to a file and don't close it, some of data will be LOST
- To close the file

```
fclose(FILE *fp);
```

Reading/Writing Text File

- fscanf reads from file
  - fscanf is same to scanf. Return EOF if reached
- fprintf writes to file
  - fprintf is same to printf.

```
int fscanf(FILE *fp,"format", parameters);

int fprintf(FILE *fp,"format", parameters);
```

Text File: Example

- We have file in this format

<Number of students>

<id of student 1> <grade of student 1>

<id of student 2> <grade of student 2>

…

<id of student n> <grade of student n>

برنامه‌اي كه شماره و نمره دانشجويان را از فايل بخواند و ميانگين را محاسبه كند.

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
   FILE *fpin;
   char inname[20];
   int num, i, id;
   float sum, average, grade;

   printf("Enter the name of input file: ");
   scanf("%s", inname);

   fpin = fopen(inname, "r");
   if(fpin == NULL){
    printf("Cannot open %s\n", inname);
    return -1;
   }
```

```
   /* Read the number of students */
   fscanf(fpin,"%d", &num);

   /* Read the id and grade from file */
   sum = 0;
   for(i = 0; i < num; i++){
    fscanf(fpin, "%d %f", &id, &grade);
    sum += grade;
   }

   average = sum / num;
   printf("Average = %f\n", average);

   fclose(fpin);
   return 0;
}
```

برنامه‌اي كه شماره و نمره دانشجويان را از فايل بخواند و ليست دانشجوياني كه نمره آنها بيشتر از ميانگين است را در فايل ديگري بنويسد.

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
   FILE *fpin, *fpout;
   char inname[20], outname[20];
   int num, i, id;
   float sum, average, grade;

   printf("Enter the name of input file: ");
   scanf("%s", inname);

   printf("Enter the name of output file: ");
   scanf("%s", outname);

   fpin = fopen(inname, "r");
   if(fpin == NULL){
    printf("Cannot open %s\n", inname);
    return -1;
   }
```

```
fpout = fopen(outname, "w");
if(fpout == NULL){
 printf("Cannot open %s\n", outname);
 return -1;
}

/* Read the number of students */
fscanf(fpin,"%d", &num);

/* Read the id and grade from file */
sum = 0;
for(i = 0; i < num; i++){
 fscanf(fpin, "%d %f", &id, &grade);
 sum += grade;
}

average = sum / num;
```

```
  fclose(fpin);
  fpin = fopen(inname, "r");
  fscanf(fpin,"%d", &num);
  fprintf(fpout, "%f\n", average);
  for(i = 0; i < num; i++){
   fscanf(fpin, "%d %f", &id, &grade);
   if(grade >= average)
      fprintf(fpout, "%d: %s\n", id, "passed");
   else
      fprintf(fpout, "%d: %s\n", id, "failed");
  }
  fclose(fpin);
  fclose(fpout);
  return 0;
}
```

Reading/Writing Characters (Text Files)

- To write a character to file

```
fputc(char c, FILE *fp)
```

- To read a char from file

```
char fgetc(FILE *fp);
```

  - Returns EOF if reaches to End of File

برنامه‌اي كه اسم يك فايل ورودي و خروجي را از كاربر بگيرد و فايل ورودي را در خروجي كپي كند.

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

  FILE *fpin, *fpout;
  char inname[20], outname[20];
  char c;

  printf("Enter the name of input file: ");
  scanf("%s", inname);

  printf("Enter the name of output file: ");
  scanf("%s", outname);
  fpin = fopen(inname, "r");
  if(fpin == NULL){
   printf("Cannot open %s\n", inname);
   return -1;
  }
```

```
  fpout = fopen(outname, "w");
  if(fpout == NULL){
   printf("Cannot open %s\n", outname);
   return -1;
  }

  while((c = fgetc(fpin)) != EOF)
   fputc(c, fpout);

  fclose(fpin);
  fclose(fpout);

  return 0;
}
```

Checking End of File

- Each file has two indicators
  - End of file indicator
  - Error indicator
- These indicators are set when we want to read but there is not enough data or there is an error
- How to use
  - Try to read
  - If the number of read object is less than expected
    - Check end of file **`feof`**
    - Check error of file **`ferror`**
- **`feof`** checks whether the end-of-File indicator associated with stream is set and returns a value different from zero if it is.

Checking End of File

- Previous example with **`feof`**

```
while(1){
 c = fgetc(fpin);
 if(feof(fpin))
   break;
 fputc(c, fpout);
}
```

feof example: byte counter

```
#include <stdio.h>
int main () {
      FILE * pFile;
      int n = 0;
      pFile = fopen ("ss.txt","r");
      while (fgetc(pFile) != EOF) {
            ++n;
      }
      if (feof(pFile)) {
            puts ("End-of-File reached.");
            printf ("Total number of bytes read: %d\n", n);
      }
      fclose (pFile);
      return 0;
}
```

Read/Write a Line (Text File)

- We can read a line of file

```
char * fgets(char *buff, int
maxnumber , FILE *fp);
```

- Read at most `maxnumber-1` chars
- Reading stops after EOF or \n, if a \n is read it is stored in buffer
- Add '\0' to the end of string
- If reach to end of file without reading any character, return NULL

Read/Write a Line (Text File)

- We can write a line to file

```
int fputs(char *buff, FILE *fp);
```

- Write the string buff to file
- Does NOT add \n at the end

Example: Count the number of lines

```
char buf[500]; // 500 > every line

fpin = fopen(inname, "r");
if(fpin == NULL){
 printf("Cannot open %s\n", inname);
 return -1;
}

while(fgets(buf, 499, fpin) != NULL)
 count++;

printf("Number of Lines = %d\n", count);
```

برنامه‌اي كه اسم يك فايل ورودي و خروجي را از كاربر بگيرد و فايل ورودي را در خروجي كپي كند.

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

   FILE *fpin, *fpout;
   char inname[20], outname[20];
   char buf[1000];

   printf("Enter the name of input file: ");
   scanf("%s", inname);

   printf("Enter the name of output file: ");
   scanf("%s", outname);
   fpin = fopen(inname, "r");
   if(fpin == NULL){
    printf("Cannot open %s\n", inname);
    return -1;
   }
```

```
  fpout = fopen(outname, "w");
  if(fpout == NULL){
   printf("Cannot open %s\n", outname);
   return -1;
  }

  while(fgets(buf, 1000, fpin) != NULL)
   fputs(fpout, buf);

  fclose(fpin);
  fclose(fpout);

  return 0;
}
```

تابعي كه اطلاعات دو فايل را بگيرد و فايل اول را به صورت برعكس در فايل دوم بنويسيد.

تعداد خطها و حداكثر طول هر خط فايل اول مشخص شده است.

File 1:

```
3 30
1 2 3 4 5 6 7
12 34 56 78 90
123 456
```

File 2:

```
654 321
09 87 65 43 21
7 6 5 4 3 2 1
```

```
void reverse_copy(FILE *fpin, FILE *fpout){
     int lines, max_len, i = 0, j;
     fscanf(fpin, "%d %d\n", &lines, &max_len);
     char arr[lines * max_len];
     do{
        char c = fgetc(fpin);
        if(feof(fpin))
           break;
        arr[i++] = c;
     }while(1);

     for(j = i - 1; j > -1; j--)
           fputc(arr[j], fpout);
}
```

Binary Files: A Different File Format

- Data in binary files are
  - Not encoded in ASCII format
  - Encoded in binary format

- We must use different functions to read/write from/to binary files
  - Why?
  - Because, data should not be converted to/from ASCII encoding in writing/reading the files

No Conversion to ASCII

- In text files, everything is saved as ASCII codes
- In binary files, there is not any binary to text conversion, everything is read/write in binary format

Reading from Binary Files

```
int fread(void *buf, int size, int num,
  FILE *fp)
```

- Reads **num** objects from file **fp** to **buf**. Size of each object is **size**. Returns the number of read objects.
- If (return val < num)
  - There is an error
  - Or EOF Check with **feof**

Writing to Binary Files

```
int fwrite(void *buf, int size, int num,
  FILE *fp)
```

- Writes **num** objects from **buf** to **fp**. Size of each object is **size**. Returns the number of written objects.
- If (return val < num)
  - There is an error

`fread`: Examples

- Reading 1 int from binary file fp

```
int i;

fread(&i, sizeof(int), 1, fp);
```

- This means
  - Read 1 object from file **`fp`**. Save result in **`&i`**. The size of the object is **`sizeof(int)`**
- It reads 4 bytes from file and saves in **`&i`**
  - We read an integer from file and save it in **`i`**

`fread`: Examples

- Read five floats

```
float farr[5];

fread(farr, sizeof(float), 5, fp);
```

- This means
  - Read 5 objects from file **`fp`**. Save result in **`farr`**. The size of each object is **`sizeof(float)`**
- It reads 20 bytes from file and saves in **`farr`**
  - We read 5 floats from file and save them in **`farr`**

`fwrite`: Examples

- Writing 1 char to binary file **`fp`**

```
char c = 'A';

fwrite(&c, sizeof(char), 1, fp);
```

- This means
  - Write 1 object from **`&c`** into file **`fp`**. Size of the object is **`sizeof(char)`**
- It writes 1 byte from address **`&c`** and saves result in file
  - We write char `c` to the file

`fwrite`: Examples

- Writing 4 doubles to binary file **`fp`**

```
double darr[4];

fwrite(darr, sizeof(double),4,fp);
```

- This means
  - Write 4 object from **`darr`** into file **`fp`**. Size of the object is **`sizeof(double)`**
- It writes 32 bytes from address **`darr`** and saves result in file
  - We write the array of double to the file

برنامه‌اي كه x و 5 y نقطه را از كاربر مي‌گيرد و آنها را در يك فايل باينري ذخيره مي‌كند.

```
#include <stdio.h>
struct point{
   int x, y;
};
int main(void){
   FILE *fp;
   struct point p;
   int i;
   fp = fopen("c:\\point.bin", "wb");
   if(fp == NULL){
    printf("Cannot create file\n");
    return -1;
   }
   for(i = 0; i < 5; i++){
    printf("Enter X and Y: ");
    scanf("%d %d", &p.x, &p.y);
    fwrite(&p, sizeof(p), 1, fp);
   }
   fclose(fp);
   return 0;
}
```

برنامه‌اي كه اطلاعات نقطه‌هاي كه با مثال قبلي در فايل ذخيره شده است را خوانده و نمايش مي‌دهد.

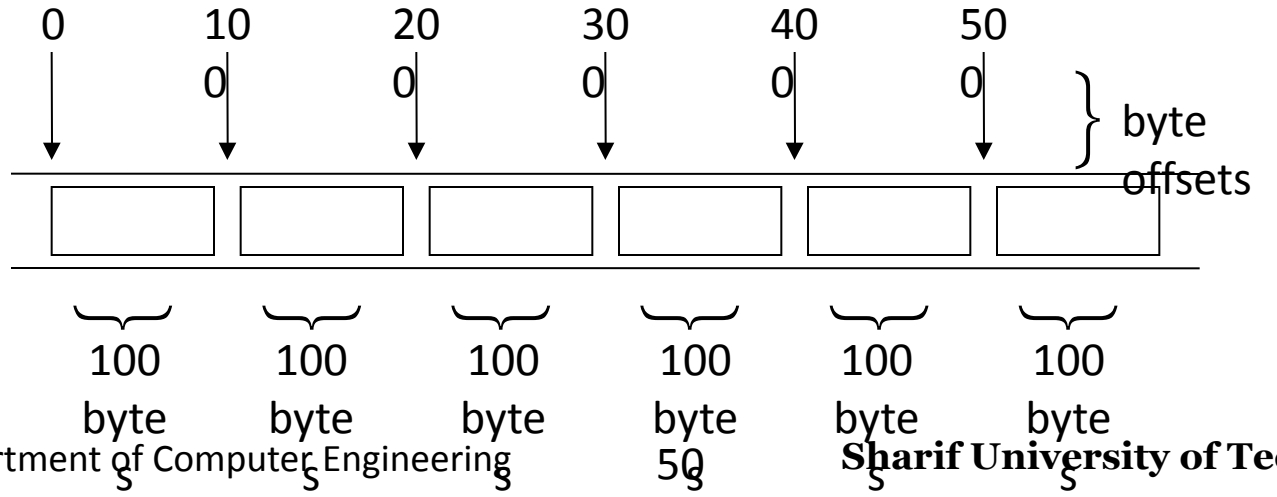
```
#include <stdio.h>
struct point{
   int x, y;
};
int main(void){
   FILE *fp;
   struct point p;
   int i;
   fp = fopen("point.bin", "rb");
   if(fp == NULL){
    printf("Cannot read from file\n");
    return -1;
   }
   while(1){
    if(fread(&p, sizeof(p), 1, fp) < 1)
        break;
    printf("X = %d, and Y = %d\n", p.x, p.y);
   }
   fclose(fp);
   return 0;
}
```

Sequential and Random Accesses

- The access to file is sequential if
  - If we don't move the FPP manually
  - FPP advances through read and write
- File processing can uses *Random* access
  - We *can also* move the FPP manually

Random Access Files

- Random access files
  - Access individual records without searching through other records
  - Instant access to records in a file
  - Data can be inserted without destroying other data
  - Data previously stored can be updated or deleted without overwriting
- Implemented using fixed length records
  - Sequential files do not have fixed length records



Moving FPP, Why?

- To access randomly
- Consider very large file (information about all students in the university)
- Change the name of 5000th student
  - If it is saved in text file
    - Read 4999 lines, skip them and change the 5000th
  - If it is saved in binary file and each object has the same size
    - Jump to the 5000th object by **`fseek`**

Moving FPP

```
int fseek(FILE *fp, long offset, int
  org)
```

- Set FPP in the **offset** respect to **org**
- org:
  - **SEEK_SET**: start of file
  - **SEEK_CUR**: current FPP
  - **SEEK_END**: End of file
- Returns nonzero if it is unsuccessful

فرض كنيد در يك فايل باينري اطلاعات نقاط زير به ترتيب نوشته شده است.
(5,5)(4,4)(3,3)(2,2)(1,1)

```
fp = fopen("point.bin", "rb");

fread(&p, sizeof(p), 1, fp);
printf("%d %d\n", p.x, p.y); 1 1

fseek(fp, 2 * sizeof(p), SEEK_SET);
fread(&p, sizeof(p), 1, fp);
printf("%d %d\n", p.x, p.y); 3 3

fseek(fp, -3 * sizeof(p), SEEK_END);
fread(&p, sizeof(p), 1,fp);
printf("%d %d\n", p.x, p.y); 3 3

fseek(fp, 1 * sizeof(p), SEEK_CUR);
fread(&p, sizeof(p), 1, fp);
printf("%d %d\n", p.x, p.y); 5 5
```

Other FPP related functions

- Find out where is the FPP

```
int ftell(FILE *fp)
```

- `ftell` returns the current FPP
  - With respect to SEEK_SET

- Reset the FPP to the start of file

```
void rewind(FILE *fp)
```

برنامه‌اي كه شماره يك نقطه و x و Y جديد را از كاربر مي‌گيرد و مختصات نقطه تعيين شده را در فايل عوض مي‌كند

```
#include <stdio.h>
struct point{
   int x, y;
};
int main(void){
   FILE *fp;
   struct point p;
   int num;
   fp = fopen("point.bin", "rb+");
   if(fp == NULL){
    printf("Cannot read from file\n");
    return -1;
   }
   printf("Enter the number of points: ");
   scanf("%d", &num);
   printf("Enter new X and Y: ");
   scanf("%d %d", &(p.x), &(p.y));
   fseek(fp, (num - 1) * sizeof(p) , SEEK_SET);
   fwrite(&p, sizeof(p), 1, fp);
   fclose(fp);
   return 0;
}
```

`fseek` in Text files

- Not very useful
- Offset counts the number of characters including '\n'
- Typical useful versions
  - **`fseek(fp, 0, SEEK_SET)`**
    - Go to the start of file
  - **`fseek(fp, 0, SEEK_END)`**
    - Go to the end of file

The basic file operations are

- fopen - open a file- specify how its opened (read/write) and type (binary/text)
- fclose - close an opened file
- fscanf- read from a file
- fprintf – write to a file
- fread - read from a file
- fwrite - write to a file
- fseek/fsetpos - move a file pointer to somewhere in a file.
- ftell/fgetpos - tell you where the file pointer is located.
- fgetc/fputc- read and write a char

Common Bugs and Avoiding Them

- Take care about mode in **`fopen`**
  - w & w+: all data in file will be lost
  - r: you cannot write. **`fprintf`** does not do any thing
- Take care about text or binary
  - fscanf/fprintf don't do meaningful job in binary files
- Check the successful open: **`fp != NULL`**
- Check EOF as much as possible.
- Close the open files.