Introduction to Programming

Lecture 12:

Files

What We Will Learn

- Introduction
  - Text vs. Binary files
- Text File Operations
  - Open/Close
  - Read/Write
- Binary File Operations
  - Open/Close
  - Read/Write
- Bugs and avoiding them

What We Will Learn

- Introduction
  - Text vs. Binary files
- Text File Operations
  - Open/Close
  - Read/Write
- Binary File Operations
  - Open/Close
  - Read/Write
- Bugs and avoiding them

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

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, skip in 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.

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: Example

```
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

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

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);
```

What We Will Learn

- Introduction
  - Text vs. Binary files
- Text File Operations
  - Open/Close
  - Read/Write
- Binary File Operations
  - Open/Close
  - Read/Write
- Bugs and avoiding them

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 fie 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`** tells that an attempt has been made to read past the end of the file, which is not the same as that we just read the last data item from a file. We have to read one past the last data item for feof to return nonzero.

Checking End of File

- Previous example with **`feof`**

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

Read/Write a Line (Text File)

- We can read a line of file
  - **`fscanf`** reads until the first free space

```
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_copy1(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);
}
```

What happen if input file is to large?!!
Huge memory allocation!
May not feasible

```
void reverse_copy2(char *inname, char *outname){
    FILE * fpin = fopen(inname, "r"); FILE * fpout = fopen(outname, "w");
    if((fpin == NULL) || (fpout == NULL)){ printf("Error");  exit(-1); }

     int lines, max_len, i, j, k;
     fscanf(fpin, "%d %d\n", &lines, &max_len);
     fclose(fpin);
     char arr[max_len];
     for(i = 0; i < lines; i++){
       int tmp1, tmp2;
        FILE * fpin = fopen(inname, "r");
        fscanf(fpin, "%d %d\n", &tmp1, &tmp2);

        for(j = 0; j < lines - i; j++)
            fgets(arr, max_len, fpin);

        fclose(fpin);

        for(k = strlen(arr) - 1; k >= 0; k--)
            fputc(arr[k], fpout);
   }
    fclose(fpout);
}
```

So many open/close
Lot of dummy read

What We Will Learn

- Introduction
  - Text vs. Binary files
- Text File Operations
  - Open/Close
  - Read/Write
- Binary File Operations
  - Open/Close
  - Read/Write
- Bugs and avoiding them

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
  - fprintf(fp, “%d”, 10)
  - Saves 2 bytes in the file: ASCII ‘1’ ASCII ‘0’
    - 00110001 00110000
  - fscanf(fp, “%d”, &i)
  - Read 2 bytes from file (ASCII ‘1’ ASCII ‘0’) and convert it to base 2 which mean integer number 10
- In binary files, there is not any binary to text conversion, everything is read/write in binary format
  - int i = 10; fwrite(&i, sizeof(int), 1, fp)
  - Saves 4 bytes in the file: The code of 10 in base 2
    - 00000000 00000000 00000000 00001010
  - fread(&i, sizeof(int), 1, fp)
  - Reads 4 bytes from file into i (without any conversion)

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

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**

`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

برنامه‌ای که x و y ۵ نقطه را از کاربر می‌گیرد و آنها را در یک فایل باینری ذخیره می‌کند.

```
#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;
}
```

برنامه‌ای که اطلاعات نقطه‌های که با مثال قبلی در فایل ذخیره شده است را خوانده و نمایش می‌دهد .

```
#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
- Text file processing usually uses sequential access (why?)
- The access to file is Random
  - FPP advances through read and write
  - We *can also* move the FPP manually
- File processing can uses *Random* access

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 کنید در یک فایل باینری اطلاعات نقاط زیر به ترتیب نوشته شده است .
(1,1)(2,2)(3,3)(4,4)(5,5)

```
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)
```

برنامه‌ای که شماره یک نقطه و X و Y جدید را از کاربر می‌گیرد و مختصات نقطه تعیین شده را در فایل عوض می‌کند

```
#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

تابعي كه دو File Handler 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_copy3(FILE *fpin, FILE *fpout){
      int lines, max_len;
      fscanf(fpin, "%d %d\n", &lines, &max_len);
      do{
         char c = fgetc(fpin);
         rewind(fpout);
         fputc(c, fpout);
      }while(!feof(fpin));
}
```

This is a wrong version!!!

```
void reverse_copy4(FILE *fpin, FILE *fpout){
      int lines, max_len, i, j, k;
      fscanf(fpin, "%d %d\n", &lines, &max_len);
      char arr[max_len];

      for(i = 0; i < lines; i++){
            fseek(fpin, 0, SEEK_SET);
            fscanf(fpin, "%d %d\n", &lines, &max_len);
            for(j = 0; j < lines - i; j++)
                  fgets(arr, max_len, fpin);
            for(k = strlen(arr) - 1; k >= 0; k--)
                  fputc(arr[k],fpout);
      }
}
```

High overhead, a lot of reading to seek!!

```
void reverse_copy5(FILE *fpin, FILE *fpout){
     int lines, max_len, i, j;
     fscanf(fpin, "%d %d\n", &lines, &max_len);
     i = 1;  j = 1;
     while(1){
        fseek(fpin, -1 * i, SEEK_END);
        char c = fgetc(fpin);
        i++;
        fputc(c, fpout);
        if(c == '\n'){
           i++; //this is due to Windows, \n is saved as "\n\r" !!!
           j++;
        }
        if(j > lines)
             break;
     }
}
```

Good, but we have to seek from end for each read → High overhead

```
void reverse_copy6(FILE *fpin, FILE *fpout){
      int lines, max_len, i, j;
      fscanf(fpin, "%d %d\n", &lines, &max_len);
      j = 1;
      fseek(fpin, -1, SEEK_END);
      while(1){
         char c = fgetc(fpin);
        fputc(c, fpout);
         i = 2;
         if(c == '\n'){
            i++; // This is due to Windows
            j++;
         }
         fseek(fpin, -1 * i, SEEK_CUR);
         if(j > lines)
              break;
      }
}
```

Good enough ☺

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.

Reference

- Reading Assignment: Chapter 11 of “C How to Program”

Homework

No Homework

There will be some sample questions with answer