Week 9 Lecture 3

Binary Files

Reading and Writing Binary Files

Binary Files

- It is possible to write the contents of memory directly to a file.
  - The bits need to be interpreted on input
- Possible to write out content like images.

Binary set with mode

- Add b to normal modes
  - Read in binary: rb
  - Write in binary: wb
  - Append in binary: ab
  - Update in binary: r+b

ASCII vs Binary Files

- ASCII files represent everything as a sequence of characters
  - We can print the contents of an ASCII file on the screen.
  - ASCII needs character representation for line control.
- Binary files are the bit values from memory.
  - Printing to the screen produces nothing legible.

Text vs Binary mode

- Newline
  - Text writes '/n' as <cr><lf>; binary writes as <cr>
- EOF
  - Text files put EOF in the file; binary files use the file size to determine end-of-file
- Numbers
  - Numbers are stored as characters in text mode; they are stored in binary in binary mode.
  - Therefore binary files must be read in binary.

Write Records with fwrite

- fwrite(const void *record, int size,
  int n_recs, FILE *fp);
  - record: pointer to a record to write
  - size: size of the record to write
  - n_recs: number of records to write
  - fp: file pointer to file to write
- The records are written starting from the current location of the file pointer.

Write Example

- Declare FILE *
- Open file append binary
- Write a record
  - Appended to end because of mode
- Close and return

```
int add(attend_rec ar)
{
  FILE *fptr = NULL;

  // Open file for append
  fptr = fopen(DB_FILE, "ab");
  if (fptr == NULL) {
    return FALSE;
  }

  // Write record at end of file
  fwrite(&ar, sizeof(attend_rec), 1, fptr);

  // Close file
  fclose(fptr);
  return TRUE;
}
```

Read Records with fread

- fread(const void *record, int size,
  int n_recs, FILE *fp);
  - record: pointer to a record to write
  - size: size of the record to write
  - n_recs: number of records to write
  - fp: file pointer to file to write
- The records are read starting from the current location of the file pointer.

Read Example

- Open file
- While record read
  - Print record
  - Read next
- Close file

```
void list()
{
  FILE *fptr = NULL;
  attend_rec r;
  size_t records_read = 0;

  fptr = fopen(DB_FILE, "rb");
  if (fptr == NULL) {
    fprintf(stderr, "%s: unable to open file\n", DB_FILE);
    return;
  }

  printf_attend_header();
  records_read = fread(&r, rec_size, 1, fptr);
  while (records_read == 1) {
    printf_attend(r);
    records_read = fread(&r, rec_size, 1, fptr);
  }
  printf("\n");
  fclose(fptr);
}
```

Read Detail

- fread parameters
  - Pointer to a record r
  - Size of record pointed to
  - Number of records requested
  - File pointer
- fread returns the number of records read.
- Print then read until no record is read
  - I.e., records_read < Number of records requested

```
records_read = fread(&r, rec_size, 1, fptr);
while (records_read == 1) {
  printf_attend(r);
  records_read = fread(&r, rec_size, 1, fptr);
}
```

Moving with fseek

- fseek(FILE *fp, int offset, int start);
  - fp: The file pointer whose index we are changing
  - offset: how far into that file we are moving
  - start: where we start counting offset
    - SEEK_SET: beginning of file
    - SEEK_CUR: Current position of file pointer
- fseek(fp, sizeof(rec), SEEK_SET);
  - Skips one “rec”;

Seek Example

- find_index() returns position of record
- Open file for update (r+)
- fseek() moves to that position
  - Returns negative if not found
- fwrite() writes at that position
  - Overwrites current contents
- Close whether found or not

```
int modify(attend_rec ar)
{
  FILE *fptr = NULL;
  size_t record_pos = 0;

  record_pos = find_index(ar.sno);

  fptr = fopen(DB_FILE, "r+b");
  if (fptr == NULL) {
    printf("%s: File open failed\n", DB_FILE);
    return FALSE;
  }

  if (record_pos >= 0) {
    fseek(fptr, record_pos * rec_size, SEEK_SET);
    fwrite(&ar, rec_size, 1, fptr);
    fclose(fptr);
    return TRUE;
  } else {
    fclose(fptr);
    return FALSE;
  }
}
```

Find Index in File

- Count the number of records before end-of-file
  - Return number counted if found
  - Return -1 if not found.

```
int find_index(int sno)
{
  FILE *fptr = NULL;
  attend_rec r;
  size_t records_read = 0;
  int records_seen = 0;

  fptr = fopen(DB_FILE, "rb");
  if (fptr == NULL) {
    printf("%s: File open failed\n", DB_FILE);
    return -1;
  }

  records_read = fread(&r, rec_size, 1, fptr);
  while (records_read == 1) {
    if (sno == r.sno) {
      fclose(fptr);
      return records_seen;
    }
    records_read = fread(&r, rec_size, 1, fptr);
    records_seen++;
  }
  fclose(fptr);
  return -1;
}
```

Seek Details

- fseek() parameters
  - File pointer
  - File position
  - Where to start
    - SEEK_SET starts at beginning

```
fseek(fptr, record_pos, SEEK_SET);
fwrite(&ar, sizeof(struct attendance_rec), 1, fptr);
fclose(fptr);
return TRUE;
```

Database Function

- Add: add a record to the end of the file
  - Seek to end of file; write a record
- List: display existing records
  - Seek to beginning of file; read and display records until end of file
- Modify: change a particular record
  - Read records until correct one found; modify record; seek back one record; write modified record
- Delete: remove a particular record
  - Read records writing into temporary file, skipping record to be deleted; rename temporary file to database file.

Tips

- The file pointer is set to beginning of file in “w” and “r” modes
- The file pointer is set to the end of the file in “a” mode
- The fread and fwrite function move the pointer the size of one record
  - They read or write the record at the current pointer
- On closing the file pointer is deactivated

Binary Write

```c
#include <stdio.h>
int main(int argc, char *argv[] )
{
  FILE *fp;
  char another = 'Y';
  struct emp
  {
    char name[40];
    int age;
    float bs;
  };

  struct emp e;

  fp = fopen("EMP.DAT", "wb");

  if (fp == NULL) {
    puts ("Cannot open file");
    return 1;
  }

  while (another == 'Y') {
    printf("\nEnter name, age and basic salary: ");
    scanf("%s %d %f\n", e.name, &e.age, &e.bs);
    fwrite(&e, sizeof(e), 1, fp);
    printf("Add another record (Y/N) ");
    fflush(stdin);
    another = getchar();
  }
  fclose(fp);
  return 0;
}
```

Binary Read

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  FILE *fp ;
  struct emp
  {
    char name[40];
    int age;
    float bs;
  };
  struct emp e;

  fp = fopen("EMP.DAT", "rb");
  if (fp == NULL) {
    puts("Cannot open file");
    return 1;
  }

  while (fread(&e, sizeof(e), 1, fp ) == 1) {
    printf("%s %d %f\n", e.name, e.age, e.bs);
  }
  fclose (fp);
}
```