```c
leaf * treeCreate (int level, char * data)
{
    leaf * current = NULL;
    current = malloc (sizeof (struct TreeNode));
    current->data = NULL;
    current->left = NULL;
    current->right = NULL;

    if (level == 1) {
        return current;
    }

    current->data = treeCreate (level - 1, data);
    current->left = treeCreate (level - 1, data);
    current->right = treeCreate (level - 1, data);
}
```
void treePrint (leaf * count)
{
    printf("%c", count -> data);
    if (count -> left) {
        treePrint (count -> left);
    }
    if (count -> right) {
        treePrint (count -> right);
    }
}

GETZ LUN
- File I/O
- Software Inquiry / Module C / H.S.M.
- Bank Books  
  Circle Buffer
  NUMER:0183
  FIFO Buffer

Gabriel Hugh Elkaim – Spring 2013
CMPE-013/L: "C" Programming
\[(++ b \rightarrow) \text{while} (\% = b \rightarrow \text{sign})\]

```

while (\%)
    \% = \% + 1

\%

\% = \% + 1
```

```
Svable = \% and
```

```
\% = \% + \text{sign} (\%); b \rightarrow \text{index} + 1;
if (\% < \text{index}) \%
    \% = \text{index}:
else
    \% = 0;
```
```c
for(i = 0; i < 10; i++)
{
    ch1sum = ch1sum + (n[i] - '0');
    Nsum += n[i];
}
```
CMPE-013/L

File I/O

Gabriel Hugh Elkaim
Spring 2013
File Processing

Outline

• Files and Streams
• Open and Close Files
• Read and Write Sequential Files
• Read and Write Random Access Files
• Read and Write Random Access Files with Structures

File Processing

Definition

FILE is just a data stream like any other, it is defined by a typedef in STDIO and is a pointer to a file that has several bits of information specific to the file that is in use (read/write access, buffer, current location within the file).

• Nothing special about FILE
• Just another pointer
• You’ve already been using STDIN and STDOUT implicitly
• C opens three files on program startup
  – STDIN (keyboard input)
  – STDOUT (screen)
  – STDERR (screen)
Files and Streams

- C files
  C views each file simply as a sequential stream of bytes as shown below.
  It ends as if there is an end-of-file marker.

```
   0
   1
   2
   ...
   n-1
   EOF
```

A file with n bytes.

- Stream
  - Provide communication channel between files and programs
  - Stream created when a file is opened
  - Function `feof()` tests a stream for end-of-file.

  ```c
  int feof(FILE *stream);
  ```
  returns nonzero if and only if the end-of-file indicator is set for `stream`.  

Open and Close Files

- A file is the most common I/O facility that can be used as a stream.
- Data type FILE, defined in stdio.h, maintains information about the stream.
- An object of type FILE *, created by calling a function such as fopen(), is used to access the file by other file manipulation functions such as fprintf() and fscanf().
- Function fopen() is prototyped as

  FILE *fopen(const char *filename, const char *mode);

  filename is the name of the file which will be opened and associated with a stream. Argument mode specifies the meaning of opening a file. The valid values for this argument are described in the following table.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Open a text file for reading.</td>
</tr>
<tr>
<td>w</td>
<td>Truncate to zero length or create a text file for writing.</td>
</tr>
<tr>
<td>a</td>
<td>Append; open or create a text file for writing at the end-of-file.</td>
</tr>
<tr>
<td>rb</td>
<td>Open a binary file for reading.</td>
</tr>
<tr>
<td>wb</td>
<td>Truncate to zero length or create a binary file for writing.</td>
</tr>
<tr>
<td>ab</td>
<td>Append; open or create a binary file for writing at the end-of-file.</td>
</tr>
<tr>
<td>r+</td>
<td>Open a text file for read/write.</td>
</tr>
<tr>
<td>w+</td>
<td>Truncate to zero length or create a text file for read/write.</td>
</tr>
<tr>
<td>a+</td>
<td>Append; open or create a text file for read/write. You can read data anywhere in the file, but you can write data only at the end-of-file.</td>
</tr>
<tr>
<td>r+b</td>
<td>Open a binary file for read/write.</td>
</tr>
<tr>
<td>w+b</td>
<td>Truncate to zero length or create a binary file for read/write.</td>
</tr>
<tr>
<td>a+b</td>
<td>Append; open or create a binary file for read/write. You can read data anywhere in the file, but you can write only at the end-of-file.</td>
</tr>
</tbody>
</table>
Open and Close Files

- If file cannot be opened, function `fopen()` returns `NULL`.
- All files which are opened and associated with streams should be closed before a program terminates.
- Function `fclose()` shall be used to close a file opened by function `fopen()`. It is prototyped as

  ```c
  int fclose(FILE *stream);
  ```

  - `fclose()` causes the stream pointed to by `stream` to be flushed and the associated file to be closed. It returns 0 when successful. Otherwise, it returns non-zero value.

Example:

```c
#include <stdio.h>
FILE *fpt1, *fpt2;
/* create file named "filename" */
if((fpt1 = fopen("filename", "w")) == NULL) {
    printf("Cannot create or open the file\n");
}
/* open file "filename" for reading, starting at the beginning. */
if((fpt2 = fopen("filename", "r")) == NULL) {
    printf("Cannot open the file\n");
}
...
fclose(fpt1);
fclose(fpt2);
```
### STDIN, STDOUT, STDERR

- Three special files – *standard input, standard output, and standard error*.
- Above three files and their associated streams are automatically opened when the program execution starts and closed when the program terminates. The file pointers for these three streams are as below.
  - **stdin** - standard input (keyboard)
  - **stdout** - standard output (screen)
  - **stderr** - standard error (screen)

- The standard input stream enables a program to read data from the keyboard, and standard output stream enables a program to print data on the screen.
- By default, the standard error stream is directed to the screen.

---

### Read and Write Characters

- **fgets** - reads one character from a file
  ```
  int fgets(FILE *stream);
  
  • **fgets(stdin)** equivalent to getchar()
  ```

- **fputc** - writes one character to a file
  ```
  int fputc(int c, FILE *stream);
  
  • **fputc('a', stdout)** equivalent to putchar('a')
  ```
Copy a file character by character

```c
/* File: copyfile.chf for copying a file char by char */
#include <stdio.h>
int copyfile(const char *inputfile, const char *outputfile) {
    FILE *fp1, *fp2;
    char c;
    if((fp1 = fopen(inputfile, "rb")) == NULL)
        return -1;
    if((fp2 = fopen(outputfile, "wb")) == NULL) {
        fclose(fp1);
        return -1;
    }
    c  = fgetc(fp1);
    while(!feof(fp1)) {
        fputc(c, fp2);
        c  = fgetc(fp1);
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

Read and Write Lines

- `fgets` - read a line from a file
  
  ```c
  char *fgets(char *s, int n, FILE *stream);
  ```

- `fputs` - write a line to a file
  
  ```c
  int fputs(const char *s, FILE *stream);
  ```
Copy a file line by line

/* File: copyfile2.chf for copying a file line by line */
#include <stdio.h>
#define BUFSIZE 1024

int copyfile2(const char *inputfile, const char *outputfile) {
    FILE *fp1, *fp2;
    char line[BUFSIZE];

    if((fp1 = fopen(inputfile, "rb")) == NULL)
        return -1;
    if((fp2 = fopen(outputfile, "wb")) == NULL) {
        fclose(fp1);
        return -1;
    }
    fgets(line, BUFSIZE, fp1);
    while(!feof(fp1)) {
        fputs(line, fp2);
        fgets(line, BUFSIZE, fp1);
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}

Formatted Read and Write to Files

- fscanf/fprintf - file processing equivalents of scanf and printf

int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);

Function calls
    printf(format, arglist);
    scanf(format, arglist);
are equivalent to
    fprintf(stdout, format, arglist);
    fscanf(stdin, format, arglist);
Example Program

/* File: accel.c */
Write accelerations into file accel.txt */
#include <stdio.h> /* for fopen(), fclose(), fprintf(), printf() */
#include <stdlib.h> /* for exit() */
#define M_G 9.81 /* gravitational acceleration constant g */

int main() {
  /* declaration and initialization */
  double a, mu=0.2, m=5.0, t0=0.0, tf=10.0, step=1.0, p, t;
  int i, n;
  FILE *stream; /* a file stream */
  /* open file accel.txt for writing */
  stream = fopen("accel.txt", "w");
  if(stream == NULL) {
    printf("Error: cannot open 'accel.txt'
");
    exit(EXIT_FAILURE);
  }
  n = (tf - t0)/step + 1; /* number of points, 11 */
  for(i=0; i<n; i++) {
    t = t0 + i*step; /* calculate value t */
    p = 4*sin(t-3)+20; /* use t to calculate p */
    a = (p-mu*m*M_G)/m; /* calculate a */
    fprintf(stream, "%f\n", a); /* write acceleration into the file */
  }
  fclose(stream); /* close the file stream */
  printf("Done! Finished writing data to file accel.txt\n");
  return 0;
}

Contents of file accel.txt:

-0.362000
0.438000
1.238000
2.038000
2.838000
3.638000
4.438000
5.238000
6.038000
6.838000
7.638000
Problem Statement

Write a program to read the accelerations from file `accel.txt` and print them out.

/* File: acceli.c
Read accelerations from file accel.txt */
#include <stdio.h> /* for fopen(), fclose(), fscanf(), printf() */
#include <stdlib.h> /* for exit() */

int main() {
    double a;       /* acceleration */
    FILE *stream;   /* a file stream */
    /* open file accel.txt for reading */
    stream = fopen("accel.txt", "r");
    if(stream == NULL) {
        printf("Error: cannot open "accel.txt":\n");
        exit(EXIT_FAILURE);
    }
    fscanf(stream, "%lf", &a); /* read an acceleration */
    while(!feof(stream)) { /* while not end-of-file */
        printf("%lf\n", a); /* print the acceleration */
        fscanf(stream, "%lf", &a); /* read the next acceleration */
    }
    fclose(stream); /* close the file stream */
    return 0;
}
**Output:**

-0.362000
0.438000
1.238000
2.038000
2.838000
3.638000
4.438000
5.238000
6.038000
6.838000
7.638000

**Problem Statement:**

For the GPA library, student information is contained in an application program. If the information of identification number, name, birth day, birth month, birth year, and GPA for a student is kept in a file “RWstudent.data” shown below

101 John 12 11 1985 3.33

read this information into a program and save the information in a new file “RWstudentNew.data”.
/* File: rwstudent.c
   Read student information from file "rwstudent.data" and
   write it into another file "rwstudentNew.data" */
#include <stdio.h> /* for fopen(), fclose(), fprintf(),
                fscanf(), printf() */
struct Birthday {            /* define structure for birthday */
    short day, month, year;
};
typedef struct Student {     /* define structure for student */
    int id;
    char name[32];
    struct Birthday birthday;
    double gpa;
} student_t;

/* read student information in filename into the buffer pointed by sp */
int readData(student_t *sp, const char *filename) {
    FILE *stream;
    stream = fopen(filename, "r");
    if (stream == NULL) {
        fprintf(stderr, "Error: cannot open file '%s' for reading
", filename);
        return -1;
    }
    fscanf(stream, "%d%s%hd%hd%hd%g\n", &sp->id, sp->name,
          &sp->birthday.day, &sp->birthday.month,
          &sp->birthday.year, &sp->gpa);
    fclose(stream);
    return 0;
}

/* write student information in the buffer pointed by sp into filename */
int writeData(const student_t *sp, const char *filename) {
    FILE *stream;
    stream = fopen(filename, "w");
    if (stream == NULL) {
        fprintf(stderr, "Error: cannot open file '%s' for writing\n", filename);
        return -1;
    }
    fprintf(stream, "%d %s %hd %hd %hd %g\n", sp->id, sp->name,
            sp->birthday.day, sp->birthday.month, sp->birthday.year, sp->gpa);
    fclose(stream);
    return 0;
}
```c
int main() {
    const char *rfilename = "rwstudent.data";    /* read this data file */
    const char *wfilename = "rwstudentNew.data"; /* write into this file */
    student_t s;    /* declare a student_t to hold the information */
    int status;     /* status for writing */

    readData(&s, rfilename);           /* read data in rfilename into s */
    status = writeData(&s, wfilename); /* write data from s into wfilename */
    /* demonstrate how a return value should be used, give a message in stdout */
    if (status)
        printf("writeData() failed\n");
    else
        printf("writeData() is successful\n");
    return 0;
}
```

Read and Write 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
Random Access File

- If a file supports random access, a **file position indicator** can be used to determine the position to read or write the next item.
- By default, the file position indicator points to the beginning of a file when it is opened.
- The reading or writing functions `fread()` and `fwrite()` will read or write items from the position pointed by the file position indicator and then increment the indicator accordingly.
- The function `fseek()` can be used to set the indicator to anywhere in the files supporting random access.

Binary Writing

- `fwrite`
  - Write a specified number of bytes from a location in memory to a file.
  - Function prototype
    ```c
    size_t fwrite(const void *ptr, size_t size, size_t nitems, FILE *stream);
    ```
    - **ptr** - Location of written bytes from
    - **size** - Number of bytes for each item
    - **nitems** - Number of items to write
    - **stream** - File to write
  - Can write several array elements
    - Provide pointer to array as first argument (ptr)
    - Indicate number of elements to write as third argument (nitems)
  - The return type `size_t` is defined in header file `stdio.h`.
Binary Reading

- **fread**
  - Reads a specified number of bytes from a file into memory
  - Function prototype
    
    ```c
    size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
    ```
  - **ptr** - Location of read bytes to
  - **size** - Number of bytes for each item
  - **nitems** - Number of items to read
  - **stream** - File to read
  - Can read several fixed-size array elements
    - Provide pointer to array (ptr)
    - Indicate number of elements to read (nitems)

Moving within a FILE

- **fseek**
  - Sets *file position indicator* to a specific position
  - Function prototype
    
    ```c
    int fseek(FILE *stream, long int offset, int whence);
    ```
  - **stream** - pointer to file
  - **offset** - number of bytes from location *whence* in the file pointed to by *stream*.
  - **whence** - has one of following three values
    - SEEK_SET -- seek starts at beginning of file
    - SEEK_CUR -- seek starts at current location in file
    - SEEK_END -- seek starts at end of file
Moving within a FILE

- This function sets the file position indicator for the stream pointed to by argument `stream`.
- For a binary stream, the new position, measured in characters from the beginning of the file, is obtained by adding `offset` to the position specified by `whence`.
- For a text stream, `offset` should be either zero or a value returned by an earlier successful call to function `ftell()` on a stream associated with the same file, and `whence` should be `SEEK_SET`.
- Function `ftell()` obtains the current value of the file position indicator for the stream.

Read and Write
Random Access Files With Structures

A structure is a data object. The values of a structure can be written to a file or read from a file by function `fwrite()` and `fread()`. Function `fwrite()` and `fread()` are used to write and read a block of data. The prototype for `fwrite()` and `fread()` are

```
size_t fwrite(void *ptr, size_t size, size_t nitems, FILE *stream);
size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
```

- `ptr` – A pointer to a memory location. For function `fwrite()`, the data written to a file are read from this location. For `fread()`, the data read from a file will be stored in this location.
- `size` - Number of bytes for each item
- `nitems` - Number of items to write or read
- `stream` - File pointer
Read and Write Random Access Files With Structures

In order to write the data of a structure to a file, the first argument can be specified by the address of a structure and the second argument is the size of the structure. It can be calculated by the `sizeof()` operator. The program below illustrates how to use function `fwrite()` and `fread()` to write and read the values of a structure to or from a file.

```c
/* File: fwrites.c */
#include <stdio.h>
#include <stdlib.h>

struct Student {
    int id;
    char name[32];
};

int main() {
    struct Student s1 = {101, "John"}, s2;
    FILE *stream;

    stream = fopen("student.bin", "wb");
    if(stream == NULL) {
        printf("Error: cannot open 'student.bin' for writing.
");
        exit(EXIT_FAILURE);
    }
    fwrite(&s1, sizeof(struct Student), 1, stream);
    fclose(stream);

    stream = fopen("student.bin", "rb");
    if(stream == NULL) {
        printf("Error: cannot open 'student.bin' for reading.
");
        exit(EXIT_FAILURE);
    }
    fread(&s2, sizeof(struct Student), 1, stream);
    fclose(stream);

    printf("s.id   = %d\n", s2.id);
    printf("s.name = %s\n", s2.name);
    remove("student.bin");
    return 0;
}
```

Output:

```
s.id   = 101
s.name = John
```
Questions?