File I/O: Standard I/O

Three default streams

- stdin (scanf, fscanf)
- stdout (printf, fprintf)
- stderr (fprintf(stderr, <format string>, <data>))

stdin, **stdout**, **stderr** are tied to files with descriptors 0, 1, and 2 respectively

At the command line we can redirect these streams to files

- ./a.out < infile.txt
- ./a.out > outfile.txt
- ./a.out &> allfile.txt (all output)
- ./a.out 2> errfile.txt (error output only)
- ./a.out 1> outfile.txt is the same as ./a.out > outfile.txt

File I/O

Reading/writing files is done using a **file pointer**

```
FILE *infile; FILE *outfile;

infile = fopen("input.txt", "r");
if (infile == NULL) {
   printf("Error: unable to open file %s\n", "input.txt");
   exit(1);
}

outfile = fopen("output.txt", "w");
if (outfile == NULL) {
   printf("Error: unable to open outfile\n");
   exit(1);
}

fclose(infile);
fclose(outfile);
```

Standard I/O (C)

```
int a;
 char word[32];
 printf("Enter an integer: ");
 scanf(" %d", &a);

 printf("Enter a string: ");
 scanf(" %s", word);

 printf("\nYou entered: %d %s\n",
              a, word);
 fprintf(stderr, "Test printing an error\n");
```

```
int a;
 char word[32];
 fprintf(stdout, "Enter an integer: ");
 fscanf(stdin, " %d", &a);

 fprintf(stdout, "Enter a string: ");
 fscanf(stdin, " %s", word);

 fprintf(stdout, "\nYou entered: %d %s\n",
              a, word);
 fprintf(stderr, "Test printing an error\n");
```

printf(…) is the same as fprintf(stdout, …)

When we redirect output/input, they are fed into/out of these commands (demo)

Example: Reading Text Files (C)

```
const char* filename = "grades.txt";
FILE* fp = fopen(filename, "r");
if (!fp) // fp is NULL on error
{
   printf("Cannot load file: %s\n", filename);
   return 1;
}
int num = 0;
float sum = 0;
char line[1024];
while (fgets(line, 1024, fp))
{
   int grade;
   sscanf(line, " %d", &grade);
   sum += grade;
   num++;
}
printf("The average is %.2f\n", sum/num);
fclose(fp);
```

```
99
79
51
92
56
89
63
```

Example: Reading Text Files (C)

```
const char* filename = "grades.txt";
FILE* fp = fopen(filename, "r");
if (!fp) // fp is NULL on error
{
   printf("Cannot load file: %s\n", filename);
   return 1;
}
int num = 0;
float sum = 0;
char line[1024];
while (fscanf(fp, “ %d”, &grade) != EOF)
{
   sum += grade;
   num++;
}
printf("The average is %.2f\n", sum/num);
fclose(fp);
```

```
99
79
51
92
56
89
63
```

Example: Reading/Writing Text Files (C)

```
FILE* infp = fopen(“input.txt”, "r");
 if (!infp) // fp is NULL on error
 {
    printf("Cannot load file: %s\n", filename);
    return 1;
 }
 FILE* outfp = fopen(“output.txt”, “w");
 if (!outfp) // fp is NULL on error
 {
    printf("Cannot load file: %s\n", filename);
    return 1;
 }
 int c = fgetc(infp);
 while (c != EOF)
 {
    fputc(c, outfp);
    c = fgetc(infp);
 }
fclose(infp);
fclose(outfp);
```

Helpful text file commands

- fprintf, fscanf
- fgets, fputs (gets/puts a line of input)
- fgetc, fputc (gets/puts a single character)

Helpfull string commands

- strlen
- strcpy
- strncmp
- strcmp
- strtok

Example: strtok

```
void print(char** list, int n) {
  for (int i = 0; i < n; i++)   {
    printf("%s, ", list[i]);
  }
  printf("\n");
}
```

```
int main() {
   char line[1024];
   strcpy(line, "cat bear dog");

   int n = 0;
   char* list[16]; // max of 16 words can be put in the list
   char* token = strtok(line, delims);
   while (token && n < 16)     {
      list[n] = token;
      token = strtok(NULL, delims);
      n++;
   }
   print(list, n);
}
```

Example: strtok stack diagram

Moving the file pointer

Usually we read files sequentially. What if we want to

- skip ahead,
- get the file length,
- return to the beginning of the file?

fseek allows you to move the file pointer

```
int fseek(FILE *stream, long offset, int whence);
```

Examples:

- `fseek(fp, 40L, SEEK_CUR); // skip ahead 40 bytes from current pos`
- `fseek(fp, 0L, SEEK_SET); // go to beginning`
- `fseek(fp, 0L, SEEK_END); // go to end`
  `size_t sz = ftell(fp); // get file size`

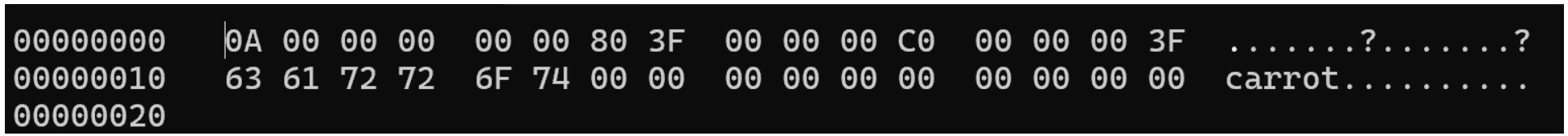
Discuss: How is redirected input/output different from reading/writing a file?

How is it similar?

Writing Binary Files

```
#include <stdio.h>
struct Data {
  int size;
  float p[3];
  char buff[16];
};

int main()
{
  struct Data data = {10, 1.0, -2.0, 0.5, "carrot"};
  FILE* fp = fopen("data.bin", "wb");
  fwrite(&data, sizeof(struct Data), 1, fp);
  fclose(fp);
}
```



Reading Binary Files

```
#include <stdio.h>
struct Data {
  int size;
  float p[3];
  char buff[16];
};

int main()
{
  struct Data data;
  FILE* fp = fopen("data.bin", "rb");
  fread(&data, sizeof(struct Data), 1, fp);
  fclose(fp);
}
```

```
00000000   0A 00 00 00  00 00 80 3F  00 00 00 C0  00 00 00 3F  .......?.......?
00000010   63 61 72 72  6F 74 00 00  00 00 00 00  00 00 00 00  carrot..........
00000020
```

Helpful binary file commands

FILE* fp = fopen(filename, “rb”);

// Write binary data

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)

// Read binary data

size_t fread(const void *ptr, size_t size, size_t nmemb, FILE *stream)

Comparison: Text versus binary files

```
struct point {
  int x;
  int y;
};

int main() {
  struct point p[3];
  for (int i = 0; i < 3; i++) {
    p[i].x = i;
    p[i].y = 2*i;
  }

  FILE* fp = fopen("points.txt", "w");
  fprintf(“%d %d %d\n”, p[0][0], p[0][1]);
  fprintf(“%d %d %d\n”, p[1][0], p[1][1]);
  fprintf(“%d %d %d\n”, p[2][0], p[2][1]);
  fclose(fp);
}
```

```
struct point {
  int x;
  int y;
};

int main() {
  struct point p[3];
  for (int i = 0; i < 3; i++) {
    p[i].x = i;
    p[i].y = 2*i;
  }

  FILE* fp = fopen("points.bin", "wb");
  fwrite(p, sizeof(struct point), 3, fp);
  fclose(fp);
}
```

Comparison: Text versus binary files

Text File:

```
0 0
1 2
2 4
```

Binary File:

```
00000000  00 00 00 00  00 00 00 00  01 00 00 00  02 00 00 00  02 00 00 00  ..................
00000014  04 00 00 00                                                      ....
```