CS 50:
Software Design and Implementation

File I/O

A note about Lab 2

word.c [filename] [filename] …
A dash (–) indicates read from stdin instead of a file

Example:
./words file1 – file2

1. Read file1 and print all the words in the file with one word per line
2. Read from stdin (due to the dash) instead of a file and print all words input, each word on its own line (use control-D to end input from stdin)
3. Read file2 and print all the words in the file with one word per line

Agenda

➡ 1. Write files

2. Read files

3. Activity

stdin, stdout, and stderr are always available for input and output

stdout_stderr.c

```
#include <stdio.h>

int main() {
        int class = 50;
        char department[] = "Computer Science";

        fprintf(stdout, "Course: %s %03d\n", department, class);
        fprintf(stderr, "Message to stderr for class %03d\n",class);

         return 0;
}
```

stdin, stdout, stderr do not need to be opened or closed

"%03d" left pads with 0 for three places
If class == 1 would become 001

See `man 3 fprintf` for more details on format specifiers

```
$ mygcc stdout_stderr.c
$ ./a.out
Course: Computer Science 050
Message to stderr for class 050
```

fprintf prints formatted output to a file

- **stdin, stdout, stderr are treated as files by Linux**
- **fprintf(stdout, "…") is equivalent to printf("…")**
- **printf prints to stdout**
- **fprintf can print to files too**

Use fopen to open a file, make sure to check the operation succeeded!

open_file.c

```
#include<stdio.h>

int main(int argc, char *argv[]) {
        FILE *fp;

        printf("Trying to open %s\n",argv[1]);
        fp = fopen(argv[1],"r");
        if (fp == NULL) {
                fprintf(stderr,"Unable to open %s\n",argv[1]);
                return 1;
        }
        printf(”Successfully opened %s\n", argv[1]);
        fclose(fp);


        return 0;
}
```

Create file pointer fp

Modes:

- **r = read**
- **w=write**
- **r+ = read/write**
- **a=append**

Attempt to open file

Check if operation succeeded

Returns NULL if not

Here we print to stderr if operations fails, then exit with status code 1

Remember status code 0 means successful completion

Don’t forget to close the file when done

Defensive programming, assume operation failed!

fprintf can write to files, here we write to the file name given by argv[1]

write_file.c

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

char class[] = "CS50";

bool check_params(int expected, int received) {
        if (received != expected) {
                fprintf(stderr,"Expecting %i parameters, but got %i\n",expected,received);
                return false;
        }
        return true;
}

int main(int argc, char *argv[]) {
        FILE *fp;

        //check parameters
        if (!check_params(2, argc)) {
                return 1;
        }

        //open file
        fp = fopen(argv[1],"w");
        if (fp == NULL) {
                fprintf(stderr,"Unable to open %s\n",argv[1]);
                return 2;
        }

        //write data
        fprintf(fp,"This is the first line\n");
        fprintf(fp,"This is the second line\n");
        fprintf(fp,"The class name is %s\n",class);

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

stdbool.h gives boolean data types

Check that we got expected number of parameters

Write to stderr if not, then exit

Defensive!

“w” opens for writing

Creates or erases existing file

Return different status codes for different errors

Defensive!

fprintf writes to file pointed to by fp

fprintf can write to files, here we write to the file name given by argv[1]

write_file.c

```c
#include<stdio.h>
#include<stdbool.h>

char class[] = "CS50";

bool check_params(int expected, int received) {
        if (received != expected) {
                fprintf(stderr,"Expecting %i parameters, but got %i\n",expected,received);
                return false;
        }
        return true;
}

int main(int argc, char *argv[]) {
        FILE *fp;

        //check parameters
        if (!check_params(2, argc)) {
                return 1;
        }

        //open file
        fp = fopen(argv[1],"w");
        if (fp == NULL) {
                fprintf(stderr,"Unable to open %s\n",argv[1]);
                return 2;
        }

        //write data
        fprintf(fp,"This is the first line\n");
        fprintf(fp,"This is the second line\n");
        fprintf(fp,"The class name is %s\n",class);

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

```
$ mygcc write_file.c
$ ./a.out test.txt
$ cat test.txt
This is the first line
This is the second line
The class name is CS50
```

Lines written to file with name given by parameter 1

Several functions can write to a file and return the number of characters written

Function	Description
int **fputc**(int *c*, FILE **fp*);	Writes the character c, cast to an unsigned char, to fp

From man [command]

Several functions can write to a file and return the number of characters written

Function	Description
int **fputc**(int _c_, FILE *_fp_);	Writes the character c, cast to an unsigned char, to fp
int **putc**(int c, FILE *fp);	Equivalent to fputc()

From man [command]

Several functions can write to a file and return the number of characters written

Function	Description
int **fputc**(int *c*, FILE **fp*);	Writes the character c, cast to an unsigned char, to fp
int **putc**(int c, FILE *fp);	Equivalent to fputc()
int **fputs**(const char **s*, FILE **fp*);	Writes the string s to fp, without its terminating null byte ('\0')

From man [command]

Several functions can write to a file and return the number of characters written

Function	**Description**
int **fputc**(int *c*, FILE **fp*);	Writes the character c, cast to an unsigned char, to fp
int **putc**(int c, FILE *fp);	Equivalent to fputc()
int **fputs**(const char **s*, FILE **fp*);	Writes the string s to fp, without its terminating null byte ('\0')
int **puts**(const char *s);	Writes the string s and a trailing newline to stdout (similar to System.out.println in Java)

From man [command]

Several functions can write to a file and return the number of characters written

Function	Description
int **fputc**(int *c*, FILE **fp*);	Writes the character c, cast to an unsigned char, to fp
int **putc**(int c, FILE *fp);	Equivalent to fputc()
int **fputs**(const char **s*, FILE **fp*);	Writes the string s to fp, without its terminating null byte ('\0')
int **puts**(const char *s);	Writes the string s and a trailing newline to stdout (similar to System.out.println in Java)
int **fprintf**(FILE *fp, const char *format, ...);	Write output to fp; if fp is stdout, same as printf

From man [command]

Several functions can write to a file and return the number of characters written

Function	**Description**
int **fputc**(int *c*, FILE **fp*);	Writes the character c, cast to an unsigned char, to fp
int **putc**(int c, FILE *fp);	Equivalent to fputc()
int **fputs**(const char **s*, FILE **fp*);	Writes the string s to fp, without its terminating null byte ('\0')
int **puts**(const char *s);	Writes the string s and a trailing newline to stdout (similar to System.out.println in Java)
int **fprintf**(FILE *fp, const char *format, ...);	Write output to fp; if fp is stdout, same as printf
int **printf**(const char *format, ...);	Write output to stdout

From man [command]

Several functions can write to a file and return the number of characters written

Function	Description
int **fputc**(int *c*, FILE **fp*);	Writes the character c, cast to an unsigned char, to fp
int **putc**(int c, FILE *fp);	Equivalent to fputc()
int **fputs**(const char **s*, FILE **fp*);	Writes the string s to fp, without its terminating null byte ('\0')
int **puts**(const char *s);	Writes the string s and a trailing newline to stdout (similar to System.out.println in Java)
int **fprintf**(FILE *fp, const char *format, ...);	Write output to fp; if fp is stdout, same as printf
int **printf**(const char *format, ...);	Write output to stdout
int **snprintf**(char *str, size_t size, const char *format, ...);	Write a maximum of size bytes to the character string str

From man [command]

Agenda

1. Write files
2. Read files
3. Activity

Files can be read line by line, here we read the file name given by argv[1]

read_file_line_by_line.c

```c
const int MAX_SIZE = 100;

bool check_params(int expected, int received) {
        if (received != expected) {
                fprintf(stderr,"Expecting %i parameters, but got %i\n",expected,received);
                return false;
        }
        return true;
}

int main(int argc, char *argv[]) {
        FILE* fp;
        char buffer[MAX_SIZE];

        //check parameters
        if (!check_params(2,argc)) {
                return 1;
        }

        //open file
        fp = fopen(argv[1], "r");
        if (fp == NULL) {
                perror("Error opening file");
                return 2;
        }

        //read line by line
        while(fgets(buffer, MAX_SIZE, fp) != NULL) {
                printf("%s",buffer);
        }

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

Buffer size in characters

FOR GOODNESS SAKE DO NOT USE gets()!
Why not?
Possible buffer overflow because size not checked

Open file for reading with “r”
Check for error

perror will print your message, plus a description of the error (ex. file not found)

Read fp until:

- **end of line**
- **end of file**
- **MAX_SIZE characters read**

Files can be read line by line, here we read the file name given by argv[1]

read_file_line_by_line.c

```c
const int MAX_SIZE = 100;
```

Buffer size in characters

```c
bool check_params(int expected, int received) {
        if (received != expected) {
                fprintf(stderr,"Expecting %i parameters, but got %i\n",expected,received);
                return false;
        }
        return true;
}

int main(int argc, char *argv[]) {
        FILE* fp;
        char buffer[MAX_SIZE];

        //check parameters
        if (!check_params(2,argc)) {
                return 1;
        }

        //open file
        fp = fopen(argv[1], "r");
        if (fp == NULL) {
                perror("Error opening file");
                return 2;
        }

        //read line by line
        while(fgets(buffer, MAX_SIZE, fp) != NULL) {
                printf("%s",buffer);
        }

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

```
$ mygcc read_file_by_line.c
$ ./a.out test.txt
This is the first line
This is the second line
The class name is CS50
```

Open file for reading with “r”
Check for error

perror will print your message, plus a description of the error (ex. file not found)

Read fp until:

- **end of line**
- **end of file**
- **MAX_SIZE characters read**

Files can also be read char by char

read_file_character_by_character.c

```
<snip>
int main(int argc, char *argv[]) {
        FILE* fp;

        //check parameters
        if (!check_params(2,argc)) {
                return 1;
        }

        //open file
        fp = fopen(argv[1], "r");
        if (fp == NULL) {
                perror("Error opening file");
                return 2;
        }

        //read char by char
        while(!feof(fp)) {
                printf("%c",(char)fgetc(fp));
        }

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

Open file and check for errors

Loop until end of file

Read int and cast to char

No need to set buffer size here, reading one char at a time

See lecture extra on course web page

Formatted data can be read using fscanf

read_formatted_data.c

```c
int main(int argc, char *argv[]) {
    FILE* fp;
    int height, width;

    //check parameters
    if (!check_params(2,argc)) {
        return 1;
    }

    //open file
    fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror("Error opening file");
        return 2;
    }
    //read header until new line or EOF
    char c = fgetc(fp);
    while (!feof(fp) && c != '\n') {
        putc(c,stdout);
        c = fgetc(fp);
    }
    printf("\n");

    //read formatted data
    int count = fscanf(fp,"%d,%d",&height, &width);
    while(count == 2) {
        printf("%d %d\n",height, width);
        count = fscanf(fp,"%d,%d",&height, &width);
    }

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

Make data file with header row then two variables per line in csv format

```
$ cat > data.csv
Height,Width
10,5
15,7
20,3
$ cat data.csv
Height,Width
10,5
15,7
20,3
```

Check file contain what we expect

Formatted data can be read using fscanf

```c
int main(int argc, char *argv[]) {
  FILE* fp;
  int height, width;

  //check parameters
  if (!check_params(2,argc)) {
    return 1;
  }

  //open file
  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    perror("Error opening file");
    return 2;
  }
  //read header until new line or EOF
  char c = fgetc(fp);
  while (!feof(fp) && c != '\n') {
    putc(c,stdout);
    c = fgetc(fp);
  }
  printf("\n");

  //read formatted data
  int count = fscanf(fp,"%d,%d",&height, &width);
  while(count == 2) {
    printf("%d %d\n",height, width);
    count = fscanf(fp,"%d,%d",&height, &width);
  }

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

Last class we used sscanf to scan a string, today we use fscanf to scan a file

Open file and check for errors

Read and print header row

fscanf reads fp and converts input to two comma separated integers here

fscan returns the number of successful conversions and EOF at end of file

Pass address of variables to be changed with &

read_formatted_data.c

```
$ cat > data.csv
Height,Width
10,5
15,7
20,3
$ cat data.csv
Height,Width
10,5
15,7
20,3
```

Formatted data can be read using fscanf

read_formatted_data.c

```c
int main(int argc, char *argv[]) {
   FILE* fp;
   int height, width;

    //check parameters
    if (!check_params(2,argc)) {
        return 1;
    }

    //open file
    fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror("Error opening file");
        return 2;
    }
    //read header until new line or EOF
    char c = fgetc(fp);
    while (!feof(fp) && c != '\n') {
        putc(c,stdout);
        c = fgetc(fp);
    }
    printf("\n");

    //read formatted data
    int count = fscanf(fp,"%d,%d",&height, &width);
    while(count == 2) {
        printf("%d %d\n",height, width);
        count = fscanf(fp,"%d,%d",&height, &width);
    }

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

Last class we used sscanf to scan a string, today we use fscanf to scan a file

Open file and check for errors

Read and print header row

fscanf reads fp and converts input to two comma separated integers here

fscan returns the number of successful conversions and EOF at end of file

Pass address of variables to be changed with &

```
$ cat > data.csv
Height,Width
10,5
15,7
20,3
$ cat data.csv
Height,Width
10,5
15,7
20,3
$ mygcc read_formatted_data.c
$ ./a.out data.csv
Height,Width
10 5
15 7
20 3
```

Several functions can read from a file

Function	Description
int **fgetc**(FILE *fp);	Reads the next character from fp and returns it as an unsigned char cast to an int, or EOF on end of file or error

From man [command]

Several functions can read from a file

Function	Description
int **fgetc**(FILE *fp);	Reads the next character from fp and returns it as an unsigned char cast to an int, or EOF on end of file or error
int **getc**(FILE *fp);	Equivalent to fgetc()

From man [command]

Several functions can read from a file

Function	Description
int **fgetc**(FILE *fp);	Reads the next character from fp and returns it as an unsigned char cast to an int, or EOF on end of file or error
int **getc**(FILE *fp);	Equivalent to fgetc()
char ***fgets**(char *s, int size, FILE *fp);	Reads in at most one less than size characters from fp and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer

From man [command]

Several functions can read from a file

Function	Description
int **fgetc**(FILE *fp);	Reads the next character from fp and returns it as an unsigned char cast to an int, or EOF on end of file or error
int **getc**(FILE *fp);	Equivalent to fgetc()
char ***fgets**(char *s, int size, FILE *fp);	Reads in at most one less than size characters from fp and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer
int **scanf**(const char *format, ...);	Reads formatted input from stdin

From man [command]

Several functions can read from a file

Function	Description
int **fgetc**(FILE *fp);	Reads the next character from fp and returns it as an unsigned char cast to an int, or EOF on end of file or error
int **getc**(FILE *fp);	Equivalent to fgetc()
char ***fgets**(char *s, int size, FILE *fp);	Reads in at most one less than size characters from fp and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer
int **scanf**(const char *format, ...);	Reads formatted input from stdin
int **fscanf**(FILE *fp, const char *format, ...);	Reads formatted input from fp

From man [command]

Several functions can read from a file

Function	Description
int **fgetc**(FILE *fp);	Reads the next character from fp and returns it as an unsigned char cast to an int, or EOF on end of file or error
int **getc**(FILE *fp);	Equivalent to fgetc()
char ***fgets**(char *s, int size, FILE *fp);	Reads in at most one less than size characters from fp and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer
int **scanf**(const char *format, ...);	Reads formatted input from stdin
int **fscanf**(FILE *fp, const char *format, ...);	Reads formatted input from fp
int **sscanf**(const char *str, const char *format, ...);	Reads formatted input from the string pointed to by str

From man [command]

Agenda

1. Write files

2. Read files

3. Activity