Topic: Basic I/O in C

- C standard I/O library
  - Standard input & output
  - I/O redirection
  - Formatted input & output
  - File access
- Misc.
  - Error handling
  - Command line arguments (K&R 5.10)
  - Storage management
  - String handling, random numbers

(Reading: K&R Ch. 7 and Ch. 5.10)

The C Standard Library

- C standard library
  - provides common functions we don't need to write ourselves
  - example functions provided in C standard I/O library
    - printf, scanf, getc, putc, etc.
  - defined in the header files, e.g., <stdio.h>, <string.h>, etc.
  - our programs need to link to the headers
    - Include appropriate header files in your programs
    - E.g., "man 3 printf" on linuxlab show that <stdio.h> is needed
  - K&R Appendix B has more information on functions included the standard library and the header files
- Using functions in the C standard library provides portability of C programs

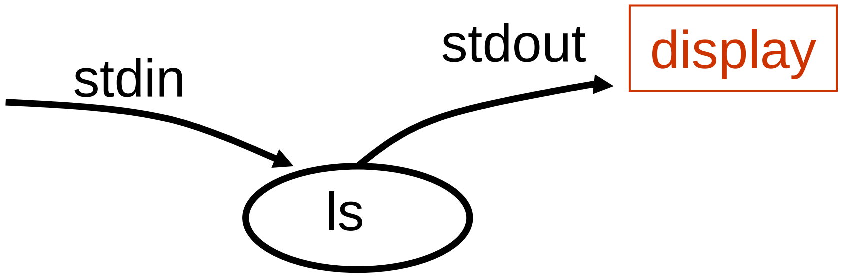
C Standard I/O library

- Examples functions defined in <stdio.h>
  - int getchar(void); //return next character from standard input
  - int putchar(int); //print the given character on standard output
  - int printf(char *format, arg1, arg2, ...)
    //format and print its arguments on standard output
  - int scanf(char *format, ...)
    // read characters from standard input according to format
- Standard I/O devices
  - Standard input: normally keyboard
  - Standard output: normally display
- Standard input and output can be redirected
  - Example: ls –l > outfile
  - Example: prog < infile
    (prog: e.g., the program from Example 3.1 in 1b.C slides in week 1)

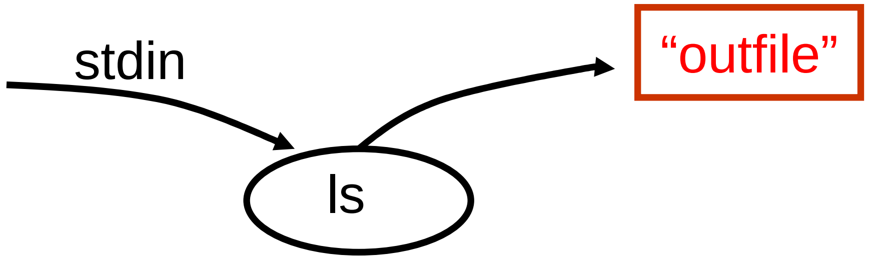
I/O Redirection

- Output redirection example: ls –l > outfile

Before output redirection:



After output redirection:



  - All results that are printed on the standard output are redirected to the output file specified

    (the programs do not know that their outputs are redirected)

- Unix pipes:
  - Puts the standard output of one program into the standard input of another program
  - Example: ls -l | grep "tar"

    (Results of the first command is piped to the second command)

Formatted Output

- Several example functions that print formatted strings
  - int printf(char *format, ...);
    - Sends output to standard output

      (often more convenient and flexible than putchar)
  - int sprintf(char *str, char *format, ...);
    - Send output to a string variable
  - int fprintf(FILE *fp, char *format, ...);
    - Send output to a file stream specified by *fp

      (the file needs to be opened first by fopen)
- Note: the above functions have a variable-length argument list
  - (topic of K&R Ch. 7.3 -- perhaps, we'll revisit this later this term)
- Return value (more detail: "man 3 printf")
  - Upon successful return, return # of characters printed (not including the trailing \0 used to end output to strings).
  - If an output error is encountered, a negative value is returned.

Format Directives

- Some of fprintf format directives (there are more)
  - Example:

char	%c
char*	%s
int	%d
float	%f

- Examples of formatting a char*

```
:%s:          :Hello, World!:
:%10s:        :Hello, World!:    //right justified
:%.10s:       :Hello, Wor:       //cut off of 10 characters
:%.15s:       :Hello, World!:
:%-15s:       :Hello, World!  :  //left justified with space
:%15.10s:     :     Hello, Wor:
:%-15.10s:    :Hello, Wor     :
```

Formatted Input

- Several example functions that read formatted strings
  - int scanf(char *format, ...);
    - Read formatted input from standard input (often more useful than getchar)
  - int sscanf(char *str, char *format, ...);
    - Read from a string (not standard input)
  - int fscanf(FILE *fp, char *format, ...);
    - Read formatted input from a file *fp
- Return value (more detail: “man scanf”)
  - these functions return the number of input items assigned, which can be fewer than provided for, or even zero, in the event of a matching failure
  - You can check the return value for errors

Example 1

```
int main()
{
        int a, b, c;
        printf("char = %c\n", x);
        printf("Enter the first value:");
        scanf("%d", &a);
        printf("Enter the second value:");
        scanf("%d", &b);
        c = a + b;
        printf("%d + %d = %d\n", a, b, c);
        return 0;
}
```

- Note: scanf requires pointer arguments (e.g., &a)

  (More about pointers later in this term)

File Access

- A file needs to be opened before the first access
- Several functions in C library for accessing files
  - To open a file: `FILE *fopen(char *name, char *mode);`
    - mode specifies read, write, etc.
    - returns a pointer to a file
  - To close a file: `int fclose(FILE *fp);`
    - Flushes any output associated with *fp to disk
  - Read from a file with a particular format

    `int fscanf(FILE *fp, char *format, …);`
  - Output to a file using a particular format

    `int fprintf(FILE *fp, char *format, …);`
- There are different styles of file I/O

  (perhaps later in the term)

Example 2: open and print to a text file

```
#include <stdio.h>
#define MAX 10

int main()
{
        FILE *fp;
        int x;
        fp= fopen("out", “w");
        if (!fp) {  // can’t open the file for write
                return 1;
        }
        for (x = 1; x <= MAX; x++) {
                fprintf(fp, "%d\n", x);
        }
        fclose(fp);
        return 0;
}
```

If there is an error opening the file,
fp would contain zero,
which is False.

I/O a Line at a Time

- fgets()
  - `char *fgets(char *line, int n, FILE *fp);`
  - Reads up to (n -1) characters from file *fp
  - *line will be null terminated ('\0')
- fputs()
  - `int fputs(char *line, FILE *fp);`

Example 3: getline

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

/* getline: read a line from stdin, return the length */
int getline(char *line, int max)
{
      if (fgets(line, max, stdin) == NULL) { return 0; }
      else { return strlen(line); }
}
int main()
{
    const int MAX = 128;
    char line[MAX];

    int n = getline(line, MAX);
    printf("get %d characters: %s\n", n, line);
    return 0;
}
```

Example 4: fgets from a file

```
#include <stdio.h>

int main()
{
        FILE *fp;
        char s[1000];

        fp = fopen("infile", "r");
        if (!fp) {
            return 1;
        }
        while (fgets(s, 1000, fp) != NULL) {
            printf("%s", s);
        }
        fclose(fp);
        return 0;
}
```

Error Handling

- How do programs signal errors?
  - Print error messages (to where?)
  - Use exit or return values (typically a non-zero value for an error)
- Standard error (stderr)
  - Similar to stdin and stdout, but for printing error messages

    E.g., fprintf(stderr, “getline: error getting input\n”);

    ```
    perror(“getline”);      // this will print the error encountered
                            // in the last system or library call
    ```
  - errno (defined in <errno.h>, more info: man 3 errno)
    - Used by perror
    - An integer indicating the type of error
  - By default, stderr is the same as stdout (i.e., display)
  - If stdout is redirected to an output file, the error messages printed to stderr will still be shown on the display

Command Line Arguments

- int main(int argc, char* argv[])
  - argc: number of arguments (including program name)
  - argv
    - Array of char*s (that is, an array of 'C' strings)
    - argv[0]: = program name
    - argv[1]: = first argument, … , argv[argc-1]: last argument
- Example:

```
int main(int argc, char* argv[])
{
      int i;
      printf("%d arguments\n", argc);
      for (i = 0; i < argc; i++) {  printf("  %d: %s\n", i, argv[i]); }
      return 0;
}
```

Example 5 -- Putting it together

```
/* files.c: read or append to a file
 * <this_file> -option <filename>
 */
#include <stdio.h>
#define PRT_MSG(message, value)  {  perror(message);  exit(value);  }

void doRead(char *fname);
void doAppend(char *fname);

int main(int argc, char **argv)
{
        char c;
        if (argc !=  3) {
                fprintf(stderr, "Usage: %s -option file (option = r, a) \n", argv[0]);
                exit(1);
        }
        c = argv[1][1];  // skip first character '-' to get the option
        if (c == 'r') { doRead(argv[2]); }
        else if (c == 'a') { doAppend(argv[2]); }
        return 0;
}                                                    // (continuous on next two slides)
```

Example 5 (cont.)

```
/* doRead: open file for reading */
void doRead(char *fname)
{
        FILE *fp;
        char msg[128];

        if ( (fp = fopen(fname, "r")) == NULL) {
                sprintf(msg, "%s - read error ", fname);
                PRT_MSG(msg, 2);
        }
        /* close the file */
        if (fclose(fp) == EOF)  { PRT_MSG("error closing file ", 5); }
        exit(0);
}
```

Question: what change(s) would you recommend to make the program more readable?

Example 5 (cont.)

```
/* doAppend: open file for appending */
void doAppend(char *fname)
{
        FILE *fp;
        char msg[128];
        char c;

        if ( (fp = fopen(fname, "a")) == NULL) {
                sprintf(msg, "%s - append error ", fname);
                PRT_MSG(msg, 4);
        }
        fputs("Enter text (terminate with ^D) \n", stdout);
        while ( (c = getc(stdin)) != EOF) { putc(c, fp); }

        /* close the file */
        if (fclose(fp) == EOF) { PRT_MSG("error closing file ", 5); }
        exit(0);
}
```

Storage Management

- Memory can be dynamically allocated
  - Examples:
    - Allocate an integer

      ```
      int* pt_i =
              (int*) malloc(sizeof(int));
      ```

    - Allocate a structure

      ```
      struct table* pt_table =
              (struct table*)malloc(sizeof(struct table));
      ```

  - Note: return values must be casted to the appropriate type
  - pt_i and pt_table: pointers, holding an address in memory
- Free dynamically allocated memory
  - Pass in a pointer that was returned by **`malloc`**
  - Example: **`free(pt_i);`**
  - Note: never use memory once it's freed

Miscellaneous Functions

- String operations: see “man 3 string” for more
  - strlen(s)
  - strcpy(s, t), strncpy(s, t, n)
  - strcat(s, t), strncat(s, t, n)
  - strcmp(s, t), strncmp(s, t, n)
  - strchr(s, c), strrchr(s, c), strtok(s, t), etc.
- Random number generations
  - rand(), random(), srand(), srandom()
    - Not as random as one would usually like
    - Compilation: need use ‘-lm’ linker switch
  - More detail: “man 4 random” on Linux or “man –s 7d random” on Solaris