CMSC 11 Handout – (S-4L and GH-4L)
Prepared by: MAADCLARIÑO

FILES

- **text file (user-readable) – sequence of bytes which can be converted to characters**

- **binary file – sequence of bytes that may or may not be characters**
  – no character translations needed

file pointer – a pointer to a file (FILE *)
FILE *fp; /*fp is a file pointer*/

OPENING A FILE

FILE *fopen(char *fname, char *mode);
----> Absolute filename: Starting from the root directory
----> Relative filename: Does not have to start from the root

mode format(string)

<file mode> [+] [file type]

file mode:
"r" - open a file for reading
"w" - open a new file for writing
"a" - open a file for appending
"t" - open for reading and writing

file type:
"t" - text file
"b" - binary file

Example: msg.txt is opened for reading and is being pointed to by *fp* pointer.

```
FILE *fp;
fp=fopen("msg.txt","r")
```

CLOSING A FILE

```
int fclose(FILE *fp);

fclose(fp);
```

0 if successful, EOF if error

READING AND WRITING BYTES

int getc(FILE *fp);
-retrieves the byte value (which can be a character) currently pointed to by the access pointer
-byte retrieved is returned as an integer-ASCII VALUE (which can also be a character)

```
FILE *fp;
char ch;

ch=getc(fp);
```

//EOF is returned if getc attempts to read at the end of the file

---

```
FILE *fp;
int x;

if((fp=fopen("msg.txt","r"))==NULL){
        printf("error:unable to open file\n");
        exit(1);
}

while ((x=getc(fp))!=EOF){
        printf("%c", (char)x);
}

fclose(fp);
```

---

int putc(int c, FILE *fp);
-writes a byte value to file fp points

putc('h',fp);

STRING INPUT AND OUTPUT

char *fgets(char *s, int n, FILE *fp);

-reads a sequence of characters starting at the access pointer, reading n-1 characters and places it to string s plus a null character '\0'

-after reading, access pointer points after the last character read

-also stops reading when an EOF or a newline is encountered

-returns s or NULL(when an EOF) if the end of the file has been reached

```
char str[10];
fgets(str, 10, fp);
```
//reads 9 characters from file pointed by fp & stores to str

int fputs(char *s, FILE *fp);

-writes string s to file pointed by fp

-returns a nonnegative integer or EOF if an error occurs

```
fputs("hey!", fp);
```
//this statement replaces the series of putc

FORMATTED INPUT AND OUTPUT

fprintf(FILE *fp, <format string>,<item>,<item>,...>;
fscanf(FILE *fp, <format string>,<item>,<item>,...>;

fprintf – output is written on the file pointed by fp

```
int x=10;
float f=1.5;
fp=fopen("data.txt", "w");
fprintf(fp, "%d %f\n", x, f);
```

fscanf – input is read from a text file

```
int num;
float r;
fp=fopen("data.txt", "r");
fscanf(fp, "%d", &num); //stores 10 in num
fscanf(fp, "%f", &r); //stores 1.5 in r
```

-fscanf scans the characters starting at the access pointer until it encounters either a space, a tab or a new line character

INPUT AND OUTPUT OF BINARY DATA

size_t fread(void *ptr, size_t size, size_t n, FILE *fp);
size_t fwrite(void *ptr, size_t size, size_t n, FILE *fp);

size_t – integer data type capable of storing very large integer values
n – number of elements
size – size of each element

fread - reads n items of size bytes each
- returns the number of items (not bytes) read

fwrite - writes n items of size bytes each
- returns the number of items (not bytes) actually written

```
int A[10];
int x;
fp=fopen("binary.dat","wb");
fwrite(A, sizeof(int), 10, fp); //writes contents of A to file

fwrite(&x, sizeof(int), 1, fp); //writes content of x to file

fclose(fp);

int B[10];
int y;
fp=fopen("binary.dat", "rb");
fread(B, sizeof(int), 10, fp);//reads 10 integers from file and stores to B

fread(&y, sizeof(int), 1, fp); //reads an integer and stores it in x

fclose(fp);
```

***NOTE: fwrite and fread can be used with text files.**