i/o

```
#include <st dio.h>
```

TAGS	
"r" / "rb"	Read existing text/binary file.
"w" / "wb"	Write new/over existing text/binary file.
"a" / "ab"	Write new/append to existing text/binary file.
"r+" / "r+b" / "rb+"	Read and write existing text/binary file.
"w+" / "w+b" / "wb+"	Read and write new/over existing text/binary file.
"a+" / "a+b" / "ab+"	Read and write new/append to existing text/binary file.
RANDOM-ACCESS	
fread(void *buf, sizeof(element), number, fptr)	Reads a number of elements from fptr to array *ptr. (safe)
fwrite(void *buf, sizeof(element), number, fptr)	Writes a number of elements to file fptr from array *ptr.
SEQUENCIAL	
fscanf(fptr, format, [...])	Same as scanf with additional file pointer parameter. (unsafe)
fprintf(fptr, format, [...])	Same as printf with additional file pointer parameter.
fgets(strName, length, fptr); sscanf(strName, "%d", &x);	Uses fgets to limit the input length, then uses sscanf to read the resulting string in place of scanf. (safe)
NAVIGATION	
fseek(fptr, offset, origin);	Sets current file position. Returns false is successful, true otherwise. The offset is a long integer type.
ftell(fptr)	Return current file position as a long integer.
SEEK_SET	Beginning of file.
SEEK_CUR	Current position in file.
SEEK_END	End of file.
feof(fptr)	Tests end-of-file indicator.

read line one by one

```
#include <stdio.h>
FILE *h;
char line[100];
h = fopen("filename", "rb");
if (h == NULL) {
    exit(1);
}
while (fgets(line, sizeof line, h)) {
   / deal with line /
}
/ if needed test why last read failed /
if (feof(h) || ferror(h)) / whatever /;
fclose(h);
```

makefile

RULES	
all: <rule>...	executes all specified rules
clean: rm <file>...	removes specified files
.c to .o	<filename>.o: <filename>.c
FLAGS	
$@	current target
$^	all sources for $@
$<	leftmost source for $@

memory

```
#include <st dli b.h>
```

malloc(sizeof(type) * length);	allocates memory, returns location.
realloc(ptrName, size);	reallocates memory, returns location.
free(ptrName);	decallocates memory.

operators

BIT	
a << b;	shift a to the left for b bits
a >> b;	shift a to the right for b bits
a & b;	bitwise and
a ^ b;	bitwise xor
a | b;	bitwise or
~a	bitwise complement

structs

BITFIELD	
struct{char a:4, b:4} x;	Declares x with two members a and b , both four bits in size (0 to 15.)
ENUM	
enum bool { false, true };	A custom data type bool with two possible states: false or true .

j

character

```
#include <ct  yp e.h>
```

tolower(char)	lowercase modification
toupper(char)	lowercase modification
isalpha(char)	alphabet check
islower(char)	lowercase check
isupper(char)	uppercase check
isnumber(char)	number check
isblank(char)	whitespace check

string

```
#include <st rin g.h>
```

DEFAULT	
strlen(a)	return length
strcpy(a, b)	copy a to b
strcat(a, b)	concatenate a to b
strcmp(a, b)	compares a to b (FALSE IF SAME)
strstr(a, b)	finds first str b in a, returns ptr.
strchr(a, b)	finds first char b in a, returns ptr.
strrchr(a,b)	finds last char b in a, return ptr.
strpbrk(a,b)	finds first char in a that's also in b, return ptr.
strspn(a, b)	returns range from a to char from b
strcspn(a, b)	returns range from a to char NOT from b
MEMORY	