```
#include <stdio.h>
#include <stdlib.h> // For exit()
#include <string.h> // for strncpy
#include <limits.h> // for PATH_MAX
#include <ctype.h> // for isprint()

///< A program to print a file - like our own version of 'cat'
///< It's good to include some comment at leat
//

// provide some help if needed
void usage(char *prog)
{
	fprintf(stderr,"usage: %s [filename]\n",prog);
	exit(1);
}

int main(int argc,char*argv[])
{
	FILE *fptr;
	char filename[PATH_MAX]; // 100 isn't a good value - long enough to pass tests, short enough to fail
easily
				// PATH_MAX exists for reasons and is fairly portable
	char c; // mixed declarations on one line are bad news - easily cause bugs

	if (argc==2) {
		strncpy(filename,argv[1],PATH_MAX); // strncpy is safer than strcpy
		if (filename[0]=='\0') { // check odd cases
			fprintf(stderr,"Error reading name (2)\n"); // using an errno like method would be
better
			exit(2);
		}
	} else if (argc==1) {
		printf("Enter the filename to open \n");
		// scanf("%s", filename); - scanf can't really be safe
		// ok the code below is fairly OTT, but hey prompting is dangerous!!!
		char *rv=fgets(filename,PATH_MAX,stdin); // use fgets and check results
		if (rv==NULL || filename[0]=='\0') {
			fprintf(stderr,"Error reading name (3)\n");
			exit(3);
		}
		// strip CR/LF from end of string
		size_t flen=strlen(filename);
		if (flen==0) {
			fprintf(stderr,"Error reading name (4)\n");
			exit(5);
		}
		// strip CR/LF from end of string
		while (flen && (filename[flen-1]=='\n' || filename[flen-1]=='\r')) {
			filename[flen-1]='\0';
			flen--;
		}
		if (flen==0) {
			fprintf(stderr,"Error reading name (5)\n");
			exit(5);
		}

	} else {
		usage(argv[0]);
	}

	// Open file
	fptr = fopen(filename, "r");
	if (fptr == NULL) { // the "==" vs. "=" here means not crashing every time
		printf("Cannot open file: |%s|\n",filename); // provide feedbac, could be typo'd spaces at end
		exit(5); // don't return zero - that's used for 'success' by convention
	}

	// Read contents from file
	c = (char) fgetc(fptr);
	while (c != EOF) {
		// this is hacky still - switching to a hexdump mode would be better
		// but binary crap on a terminal can be v. bad news, so don't...
		if (!isprint(c) && !isspace(c) && c!='\n' && c!='\r') {
```

```
                printf ("0x%x", c); // could be better - why print binary crap?
            } else {
                printf ("%c", c);
            }
            c = fgetc(fptr);
        }

        fclose(fptr); // close the file
        return 0; // return success if that's what happened
    }
```