```
#include <stdio.h>

#define  MAXLINELENGTH    80

int main ()  {
    char line[MAXLINELENGTH];
    int ch, length;

    while ( (ch = getchar ()) != EOF ) {
        length = 0;                         /* Read a line */
        while ( length < MAXLINELENGTH ) {
            if (ch == '\n') break;
            line[length++] = ch;
            if ( (ch = getchar ()) == EOF ) break;
        }
        while ( length > 0 ) {    /* Print the line reversed */
            putchar ( line[--length] );
        }
        putchar ( '\n' );
    }
    return 0;
}

/*
This is a test
tset a si sihT
and so is this
siht si os dna
What about EOF
FOE tuoba tahW

Under Unix an EOF can be supplied by entering Cntrl-D
Under Unix a program can be terminated by entering Cntrl-C
Under Unix a program can be suspended by entering Cntrl-Z
*/
```

```
#include <stdio.h>
int main ()
{
    char c;
    short int p;
    int q;
    unsigned int v;
    long int r;
    float s;
    double t;
    long double u;

    printf ("Size of Variables:\n %d %d %d %d %d %d %d %d \n",
        sizeof c, sizeof p, sizeof q, sizeof r, sizeof s,
        sizeof t, sizeof u, sizeof v );
    printf ("Size of (Types):\n %d %d %d %d %d %d %d %d \n",
        sizeof (char), sizeof(short int), sizeof(int),
        sizeof(long int), sizeof(float), sizeof(double),
        sizeof(long double), sizeof(unsigned int));
    return 0;
}

/*
Size of Variables:
 1 2 4 4 4 8 8 4
Size of (Types):
 1 2 4 4 4 8 8 4
*/
/*  Advice:  Just write sizeof(something)  */

/*---------------------------------------------------------*/
```

```
# To reduce errors of repetition the make facility provides a
#    macro (or define) facility.
#    Thus in practice the make file might really look like

CREATOR   =   gcc -Wall -ansi

phone :    main.o phone.o
           $(CREATOR) -o phone main.o phone.o

phone.o :  phone.c
           $(CREATOR) -c phone.c

main.o :   main.c phone.h
           $(CREATOR) -c main.c

#    In the above the macro CREATOR is defined to be
#        "gcc -Wall -ansi"
#    and the macro is expanded whenever $(CREATOR) appears.
#    The names you choose for macros are entirely your own.
#    See the assignment 1 statement for a different makefile:
#    Namely to build and run a series of experiments,
#    recompiling only those that you modify.

#    Two useful tests on makefiles are:
#    make -vet makefile
#    checks that each command line of the makefile
#    begins with a TAB character
#
#    make -n phone
#    shows what commands will be executed when make phone
#    is requested.
```

Read King Chapters 4, 5 and 6
(expressions, simple statements and loops)
Basic C Data Types (King Chapter 7)

C has a few basic data types, they are:

- A character: char
- An integer: int
- A floating point number: float and double

There are three modifiers that are commonly applied to some basic data types

- long, which basically doubles the space allocated to int and float/double types.
- short, which handles ints up to 32768 (it is less important now that computers have more memory).
- unsigned, which is applied to integers and changes their range from [-2^31, 2^31) to [0, 2^32). A unsigned value does not view its left-most bit as a sign, it is part of the value.

The type qualifier const is important.

- const qualifies data items that must not alter during execution of the program.

I also consider a pointer to be a basic data type - a pointer is always the size of a machine address and is treated like an unsigned integer
See King Page 111, for the range of integer values for 32-bit computers.

Use the sizeof operator to determine space needs of datatypes. King P. 489 and 494 give i/o conversion specifics.

Declaration	Space	scanf	printf
`char c;`	1 byte (8 bits)	%c	%c
`short int p;`	2 bytes	%hd	%hd
`int q;`	4 bytes	%d	%d
`unsigned int r;`	4 bytes	%u	%u
`long int s;`	4 bytes, (8 ?)	%ld	%d
`float u;`	4 bytes	%f	%f
`double v;`	8 bytes	%lf	%f
`long double x;`	8 bytes, (16 ?)	%le	%f

Variables can be initialized at declaration:

```
char c = 'd';

int q = 4;

float u = 13.5;

double v = 7.9e-2;
```

Thus a declaration has the following general format:

Type VariableName = InitialValue ;

The '= InitialValue' part is optional

A pointer to a data item is declared in the following way

Type* VariableName ;

```
int* ptr;
```

The type specifies the kind of item referred to. Here *ptr* points to an integer object. As things stand this is not particularly useful. Later pointers come into their own when we access arrays and other objects indirectly with pointers.

Character Constants

Character constants are enclosed in single quotes, for example,

`'a', 'b', 'c'.`

There are a number of special characters that must be "escaped" so they can be recognized. See King P. 119.

The common ones are:

New Line	'\n'
Tab	'\t'
Backslash	'\\'

the \ is also used to provide values for octal and hexadecimal numbers. See King page 111.

String Constants (literals)

Literal strings are seen first in print statements, and are enclosed in double quotes. They may form an output message like

```
printf ("Hello World");
```

or may appear as the format string for presenting the value of an expression or a variable

```
printf ("Here we have a fifteen %d", 3*5);
```

We will deal with string variables later after we cover arrays and pointers.
See also King, Chapter 13.

Arrays

* Arrays and pointers are closely related in C
* C basically supports one dimensional arrays, the first subscript value is 0
* An array is declared in the following way:
  `Type ArrayVariable [ size ]`;
  Thus valid indices are in the range 0 to size-1.

* The type, name and size (length) of the array are all specified here.

For example

```
int a[100];
```

produces an array with room for 100 integers. The first element is
a[0] and the last element is a[99]

* Arrays can be initialized at declaration time with a list of values

For example

```
int a[10]  = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int b[10]  = {-3, -1, 0, 1, 3, 5, 7, 9, 11, 13};
int c[ ]   = {-3, -1, 0, 1, 3, 5, 7, 9};
```

* A text string is a one-dimensional array of char elements

For example, a text string can be declared and initialized in the following way:

```
char string[25] = "This uses 19 bytes";
```

* A character string constant is enclosed in double quotes ("), for example:

"this is a text string"
Note the different usage of single and double quotes

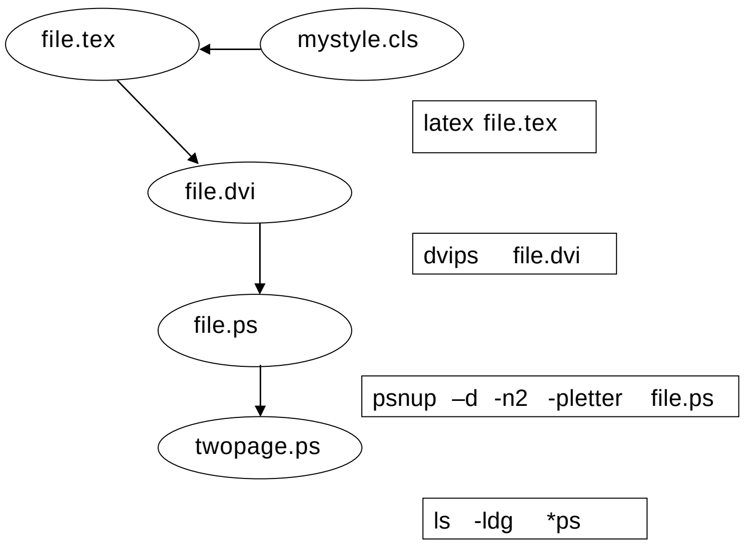
* In C and Unix a text string is terminated by a zero byte (or NULL byte), this is written as '\0'. An empty string requires one byte of storage. In general a string with n characters requires n+1 bytes of storage. Later in the course you must be careful to allocate space for that extra byte.

```
char data[5] = "WORD";
```

is the same as:

```
char  data[5];

    data[0]  = 'W';
    data[1]  = 'O';      /* this is the letter Oh */
    data[2]  = 'R';
    data[3]  = 'D';
    data[4]  = '\0';     /* this is the digit Zero */
```

```
all:    file.dvi file.ps twopage.ps
        ls -ldg *ps

twopage.ps:     file.ps
        psnup  -d -n2 -pletter  file.ps >twopage.ps

file.ps:   file.dvi
        dvips  file.dvi

file.dvi: file.tex mystyle.cls
        latex file.tex
```



```
w-rw----  1 tony    search   5782714  Jan  2 12:27  file.ps
w-rw----  1 tony    search   5945651  Jan  2 12:28  twopage.ps
```