Codes and/or tips:

scanf returns -1 for EOF or Ctrl+D

File:

Header files:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/fcntl.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <errno.h>
```

Functions:

```
int fd;
fd = open("filename", O_RDONLY |
O_WRONLY | O_CREAT, S_IRUSR, S_IWUSR);

//fd returns -1 if it fails to open
file.
close(fd);

//FOR EVERY OPEN THERE MUST BE A
CLOSE.
//FOR EVERY MALLOC THERE MUST BE A
FREE.

read(fd, &i, sizeof(int)); //if int I;
This reads from the descriptor of fd,
into I, sizeof(int) number of bytes.

//RETURNS 0 if nothing is read.

write(fd, &i, sizeof(int)); //if int
I;
This write into the descriptor of fd,
from i, sizeof(int) number of bytes.

FILE *fp;
fp = fopen("filename", "filemode");
Filemode :
      r: Read, file must exist
      w: creates empty file for
writing, if files exists, its content
is erased
      a: append
      r+: for r&w, file must exist
      w+: creates file for r&w
      a+: for a&r

fclose(fp);
to close a file you opened using fopen
```

```
FILE *fp;
char c[]="hi", buffer[100];
```

- fwrite(c, strlen(c), 1, fp)

writes from c, strlen(c) bytes once into fp.
Return value: number of bytes successfully written.

- fread(buffer, strlen(c), 1, fp)

reads into buffer, strlen(c) bytes once from fp
Return value: number of bytes successfully read.

- fpritnf(fp, "%s %s %s %d", "we", "are", "in", 2017);

basically like printf, but into fp. Converts even ints to character array and writes them

- fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);

basically like scanf, but from fp.

- sprintf(str, "Value of a = %d", x);

basically like printf, but prints everything into str and makes it a string.

- c = fgetc(fp)

gets one character from fp and stores in char c.

- fputc(ch, fp)

writes the character stored in ch into fp.

- fgets(str, 60, fp)

reads upto 60 characters from fp(including \0) and stores in str.
If fp is stdin, this is scanf.

- fputs(str, fp)

puts str into fp, if fp is stdout, this is printf

- c=getchar()

gets one character from stdin and stores in c.

- putchar(c)

prints c

- fseek(fp, int val, SEEK_XXX)

```
val = number of bytes to offset
XXX:
      SET: Beginning of file
      CUR: From current pos
      END: From EOF
```

- feof(fp)

```
returns 1 if we have reached end of
file.
Usage:
while ( 1 )
   {
     c = fgetc ( fp ) ;
     if( feof(fp) )
           break ;
     printf ( "%c", c ) ;
   }
```

TO READ CSV DATA AND USE OF STRTOK:

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
      FILE *fp;
      char line[1024], *token,
copyline[1024];
      fp = fopen(argv[2], "r");
      if(fp == NULL) {
            perror("fopen failed");
            return errno;
      }
      while(fgets(line, 1024, fp)) {
            strcpy(copyline, line);
            token = strtok(line, ",");
            token = strtok(NULL, ",");
            token = strtok(NULL, ",");
            if(strcmp(token, argv[1])
== 0)
                  printf("%s",
copyline);
      }
      return 0;
}
```

CAT PROGRAM (PRINTS A CHARACTER FILE)

```
int main(int argc, char *argv[]) {
      int fd = open(argv[1],
O_RDONLY);
      char ch;
      if(argc != 2) {
            printf("usage: ./mycat
<filename>\n");
            return EINVAL;
      }
      if(fd == -1) {
            perror("mycat: open
failed");
            return errno;
      }

      /* OPTION 1 */
      while(read(fd, &ch, 1))
            putchar(ch);

      /* OPTION 2, open using fopen()
      And then you fread like this

      while(x= fread(ch, 1, 3, fp)) {
            for(i = 0; i < x; i++)
                  putchar(ch[i]);
      }
      */

      close(fd);
      return 0;
}
```

CP PROGRAM (PRINTS A CHARACTER FILE)

```
int main(int argc, char *argv[]) {
      int fd, fdw;
      char ch;
      //check for argc !=3
      fd = open(argv[1], O_RDONLY);
      //check if fd == -1
      fdw = open(argv[2], O_WRONLY |
O_CREAT, S_IRUSR);
      //check if fdw == -1
      while(read(fd, &ch, 1))
            write(fdw, &ch, 1);
      close(fd);
      close(fdw);
      return 0;
}
```

Strings:

Important stuff from string.h

```
char *a, c, str[100], dest[100],
str1[100];
int n;
```

- a = strchr(str1, c)

Searches for the first occurrence of the character c in str
Basically strstr but with a character and upto n bytes in a string.

- a = strrchar(str1, c)

Searches for the first occurrence of the character c in str.

- strcat(dest, src)

Appends src at the end of dest and then adds a '\0' to the end

- strncat(dest, src, n)

Appends upto n bytes of src to dest.

- strcmp(str1, str2)

Compares str1 and str2. Returns 0 if it's the same.

- strncmp(str1, str2, n)

Compares upto n bytes str1 and str2. Returns 0 if it's the same.

- strcpy(str1, str2)

Copies str2 into str1

- strncpy(str1, str2, n)

Copies upto n bytes from str2 to str1.

- a = strstr(haystack, needle)

Returns char* (here a) to the first occurrence of needle in haystack

strtok(str, delim) example:

```
int main() {
   char str[80] = "Yash .Shah is .the
best;
   const char s[2] = ".";
   char *token;

   /* get the first token */
   token = strtok(str, s);

   /* walk through other tokens */
   while( token != NULL )
   {
      printf( " %s\n", token );

      token = strtok(NULL, s);
   }

   return(0);
}
```

```
Output is:
Yash_ // '_' represents a ' '
Shah is_
the best
```

strstr() implementation #1

```
/*This implementation is a pseudo
implementation as it doesn't return
the character pointer of the needle,
but the index at where it is found and
-1 if it isn't found */

int find(char *hs, char *ne) {
      int i = 0, j = 0;
      while(hs[j] && ne [i]) {
            if(hs[i] == ne[j]) {
                  i++;
                  j++;
            } else {
                  j = j - (i - 1);
                  i = 0; // Note: i = 0
should be the second line here
            }
      }
      if(ne[i] == '\0')
            return j - (i - 1) - 1 ;
      return -1;
}
```

strstr() implementation #1

```
/* Actual implementation*/

char* StrStr(char *str, char *substr)
{
      while(*str) {
            char *Begin = str;
            char *pattern = substr;
            while(*str && *pattern){
                  if(*str == *pattern){
                        str++;
                        pattern++;
                  }
            }
            if(!*pattern) {
                  return Begin;
            }
            str = Begin + 1;
      }
      return NULL;
}
```

How to input a line using getchar()

```
/*Program adds character to a string
till it reads a newline and then makes
it a string and stores it in arr, n is
the max size of arr*/

void readline(char *arr, int n) {
      char ch;
      int i = 0;
      while((ch = getchar()) != '\n'}
            if(i == n) {
                  break;
            }
            arr[i++] = ch;
      arr[i] = '\0';
}
```

IMP STUFF:

1) Always take care that you add a \0 at the end of the string, or always make sure that it's there.

2) Passing a string is like passing it's pointer to the function, no difference.
a[i] = *(a+i)

String Replace

```
/* © Yash Shah's assignment
Replaces orig by new in text and
returns number of replacements */

int stringreplace(char *text, char
*orig, char *new) {
      char test[1024];
      char *x = text;
      int count = 0;
      while((x = strstr(x, orig)) !=
NULL) {
         ++count;
         strncpy(test, text, x-text);
         test[x-text] = 0;
         strcat(test, new);
         strcat(test,
x+mystrlen(orig));
         strcpy(text, test);
         x += strlen(new);
      }
      return count;
}
```

OTHER MISC STUFF:

Remove Dup from double

```
int removeduplicate(double a[], int
n){
   int j = 0, k, l;
   while (j < n) {
      k = j + 1;
      while (k < n){
         if (a[k] == a[j]) {
            l = k;
            while(l < n) {
               a[l] = a[l+1];
               l++;
            }
            n--;
         }
         else
            k++;
      }
      j++;
   }
      return n;
}
```

FASTEST WAY TO DO x^y

```
double power(int x, int y) {
      long long ans = 1;
      long long term = x;
      int sign = 0;
      if(y < 0) {
            sign = 1;
            y = -y;
      }
      while(y > 0) {
            if(y % 2 == 1)
                  ans *= term;
            term = term * term;
            y = y/2;
      }
      if(sign)
            return 1.0 / ans;
      return ans;
}
```

sin(x)

```
double sine(double x) {
      double sum, term, xsq;
      int i = 2, d;
      sum = x;
      term = x;
      while(isgreater(fabs(term), 1e-
6)) {
            d = (2*i-1)*(2*i-2);
            xsq = double(d);
            term = ((term*x* x)/xsq;
            if(i % 2 == 1)
                  sum += term;
            else
                  sum -= term;
            i = i + 1;
      }
      return sum;
}
```

IMPORANT STUFF FROM math.h

```
//While compiling use -lm

double x, a;
```

- a = acos(x)
- a = asin(x)
- a = atan(x)

a will be cos/sin/tan inverse of x, in radinans

- a = cos(x)
- a = cosh(x)
- a = sin(x)
- a = sinh(x)
- a = tanh(x)

x is in radians

- a = exp(x) //a = e^x
- log(x) // natural log
- log10(x) //common lok a.k.a base 10
- pow(x, y)
- sqrt(x)
- fabs(x) // |x|

- ceil(x)

Smallest integer greater or equal to x

- floor(x)

Largest integer lesser or equal to x

STACK:

```
//To use stack as adt, compile stack.c
as cc –c stack.c, this will give you a
stack.o file.

//Final compile using:
cc stack.o mainpg.o -omainpg
```

stack.c

```
#include "stack.h"
int isempty(stack *s) {
      return s->i == 0;
}
int isfull(stack *s) {
      return s->i == MAX;
}
void init(stack *s) {
      s->i = 0;
}
void push(stack *s, int x) {
      s->a[s->i++] = x;
}
/* the caller should check isempty()
before calling pop()..
 */
int pop(stack *s) {
      int temp;
      temp = s->a[s->i - 1];
      s->i--;
      return temp;
}
```

stack.h

```
#define MAX 32
typedef struct stack{
      int a[MAX];
      int i;
}stack;
int isempty(stack *s);
int isfull(stack *s);
void init(stack *s);
void push(stack *s, int x);
int pop(stack *s);
```

stack.c (with structure and pointer)

```
void init(stack *s) {
      *s = NULL;
}
void push(stack *s, int n) {
      node *tmp= malloc(sizeof(node));
      tmp->val  = num;
      tmp->next = *s;
      *s = tmp;

}
int pop(stack *s) {
      int temp;
      node *tmp;
      temp = (*s)->val;
      tmp = *s;
      (*s) = (*s)->next;
      free(tmp);
      return temp;

}
int isempty(stack *s) {
      return *s == NULL;
}
int isfull(stack *s) {
      return 0;
}
```

stack.h (with structure and pointer)

```
#define MAX 32
typedef struct node{
      int val;
      struct node *next
}node;
typedef node *stack;
int isempty(stack *s);
int isfull(stack *s);
void init(stack *s);
void push(stack *s, int x);
int pop(stack *s);
```

postfix.c

```
#include <limits.h> //for INT_MIN value
#include "stack.h"
/* Reads a line of input from the user, till \n
 * and stores it in the array arr and makes arr
 * a string, and returns no. of characters read
 */
int readline(char *arr, int n) {
      char ch;
      int i = 0;
      while((ch = getchar()) != '\n' && i < n)
            arr[i++] = ch;
      arr[i] = '\0';
      return i;
}

#define OPERATOR 100
#define    OPERAND          200
#define    END              300
#define    ERROR      400
typedef struct token{
      int type;
      union data {
            int num;
            char op;
      }data;
}token;

/* input: a postfix string, possibly with errors
 * output: the 'next' token from string, separated on
 *         space or operator
 *    type: OPERAND, OPERATOR, END, ERROR
 */

enum states {START, DIG, OP, STOP, ERR, SPC};

token getnext(char *str) {
      static int currstate = START;
      int nextstate;
      static int i = 0;
      token t;
      int sum = 0;
      char currchar, currop;

      while(1) {
            currchar = str[i];
            switch(currstate) {
                  case START:
                        switch(currchar) {
                              case '0': case '1': case '2':
                              case '3': case '4': case '5':
                              case '7': case '8': case '9':
```

```
                case '6':
                        nextstate = DIG;
                        sum = currchar - '0';
                        break;
                case '+': case '-': case '*':
                case '/': case '%':
                        nextstate = OP;
                        currop = currchar;
                        break;
                case ' ': case '\t':
                        nextstate = SPC;
                        break;
                case '\0':
                        nextstate = STOP;
                        break;
                default:
                        break;
        }
        break;
case DIG:
        switch(currchar) {
                case '0': case '1': case '2':
                case '3': case '4': case '5':
                case '7': case '8': case '9':
                case '6':
                        nextstate = DIG;
                        sum = sum * 10 + currchar - '0';
                        break;
                case '+': case '-': case '*':
                case '/': case '%':
                        nextstate = OP;
                        t.type = OPERAND;
                        t.data.num = sum;
                        i++;
                        currstate = nextstate;
                        return t;
                        break;
                case ' ': case '\t':
                        nextstate = SPC;
                        t.type = OPERAND;
                        t.data.num = sum;
                        i++;
                        currstate = nextstate;
                        return t;
                        break;
                case '\0':
                        nextstate = END;
                        t.type = OPERAND;
                        t.data.num = sum;
                        i++;
                        currstate = nextstate;
                        return t;
                        break;
```

```
            default:
                  nextstate = ERR;
                  t.type = OPERAND;
                  t.data.num = sum;
                  i++;
                  currstate = nextstate;
                  return t;
                  break;
      }

      break;
case OP:
      switch(currchar) {
            case '0': case '1': case '2':
            case '3': case '4': case '5':
            case '7': case '8': case '9':
            case '6':
                  nextstate = DIG;
                  sum = currchar - '0';
                  t.type = OPERATOR;
                  t.data.op = currop;
                  currop = currchar;
                  i++;
                  currstate = nextstate;
                  return t;
                  break;
            case '+': case '-': case '*':
            case '/': case '%':
                  nextstate = OP;
                  t.type = OPERATOR;
                  t.data.op = currop;
                  currop = currchar;
                  i++;
                  currstate = nextstate;
                  return t;
                  break;
            case ' ': case '\t':
                  nextstate = SPC;
                  t.type = OPERATOR;
                  t.data.op = currop;
                  i++;
                  currstate = nextstate;
                  return t;
                  break;
            case '\0':
                  nextstate = STOP;
                  t.type = OPERATOR;
                  t.data.op = currop;
                  i++;
                  currstate = nextstate;
                  return t;
                  break;
            default:
```

```
                        nextstate = ERR;
                        t.type = OPERATOR;
                        t.data.op = currop;
                        i++;
                        currstate = nextstate;
                        return t;
                        break;
                }
                break;
            case SPC:
                switch(currchar) {
                    case '0': case '1': case '2':
                    case '3': case '4': case '5':
                    case '7': case '8': case '9':
                    case '6':
                        nextstate = DIG;
                        sum = currchar - '0';
                        break;
                    case '+': case '-': case '*':
                    case '/': case '%':
                        nextstate = OP;
                        currop = currchar;
                        break;
                    case ' ': case '\t':
                        nextstate = SPC;
                        break;
                    case '\0':
                        nextstate = STOP;
                        break;
                    default:
                        nextstate = ERR;
                        break;
                }

                break;
            case STOP:
                t.type = END;
                return t;
                break;
            case ERR:
                t.type = ERROR;
                return t;
                break;
        }
        currstate = nextstate;
        i++;
    }
}
```

```
/* Evaluates the postfix expression in str
 * and returns the result as int
 */
int postfix(char *str) {
      int x, y, res;
      token t;
      stack s, yy;
      init(&s);
      while(1) {
            t = getnext(str);
            printf("t.type = %d t.num = %d t.op = %c\n",
                        t.type, t.data.num, t.data.op);
            if(t.type == OPERAND) {
                  if(!isfull(&s))
                        push(&s, t.data.num);
                  else
                        return INT_MIN;
            } else if(t.type == OPERATOR) {
                  if(!isempty(&s))
                        x = pop(&s);
                  else
                        return INT_MIN;
                  //a[i - 1]; i--;
                  if(!isempty(&s))
                        y = pop(&s);
                  else
                        return INT_MIN;
                  //a[i - 1]; i--;
                  switch(t.data.op) {
                        case '+':
                              res = y + x;
                              break;
                        case '-':
                              res = y - x;
                              break;
                        case '*':
                              res = y * x;
                              break;
                        case '/':
                              res = y / x;
                              break;
                        case '%':
                              res = y % x;
                              break;
                  }
                  if(!isfull(&s))
                        push(&s, res);
                  else
                        return INT_MIN;
                  //i--;
            } else if(t.type == END) {
                  break;
            } else if(t.type == ERROR) {
```

```
                return INT_MIN;
          }
     }

     if(!isempty(&s))
          res = pop(&s);
     else
          return INT_MIN;
     if(isempty(&s))
          return res;
     else
          return INT_MIN;
}

/* Postfix evaluator:
 * Reads an input string, and evalutes the postfix
 * expression stored in the strinng.
 * The string contains operators and operands sepearted
 * by spaces. operands are separated by 1 or more spaces.
 * operators and operands are seperated by zero or more spaces.
 * Result: number
 * E.g.  11  22 +
 * Ans: 33
 * E.g.   11 22 33+ -
 * Ans:  -44
 *
 */
int main(int argc, char *argv[]) {
     char line[128];
     int x, y;
     while(x = readline(line, 128)) {
          //printf("%s\n", line);
          y = postfix(line);
          if(y != INT_MIN)
               printf("%d\n", y);
          else
               fprintf(stderr, "Error in expression\n");
     }
     //printf("%d\n", y);
     return 0;
}
```