CPSC 223C: C Programming - Spring 2019

Project One, **grep,** due Wednesday, 20 Mar 2019

Based on: http://www.cs.princeton.edu/courses/archive/spr08/cos333/ed_to_grep.html

As you know, Ken Thompson and Dennis Ritchie co-created Unix. You may not know that Ken Thompson created the original grep command in one evening, beginning with the source code for Unix's ed editor.
(Source code for ed.c is included in this assignment.)

Anything Ken Thompson did, surely we at CSUF can also do! After all, we have the following advantages:
Advantages:
(1) We have two weeks (not one evening) in which to complete it
(2) We have have many examples of what grep does and how it is used
(3) ed is now written in C. At the time Thompson wrote grep, it was written in PDP-11 assembler.

Disadvantage?:
We are not Ken Thompson, co-creator of Unix, who now works at Google. (but one day, who knows?)

The source code for `ed` (`ed.c`) is approximately 1700 lines long, and comes from the 1989 version of `ed`. (It is written in a style that is typical for mid-1970's Unix code: concise, efficient, and basically uncommented. In other words, much in the same style as employers expect developers to write their code (well, maybe not the uncommented part).

Your `grep` program should use the code for regular expression processing. You will have to throw away quite a bit of code, since your version of `grep` should not need more than about 400 lines. Will you need to add your own code? Yes, but not more than about 30 lines or so.

Your `grep` code should be able to read its input from stdin or from one or more named files, like so:

```
  grep regexpr [files...]
```

If there is more than one file to search, each matching line should be prefixed by the filename it came from (e.g., file1: I found this line   file2: and this line   file2: and this line too)

You don't have to provide any of the `grep` options (`-i -n -v ...`), but must return status values, such as:
0: One or more matches were found.
1: No matches were found.
2: Syntax errors or inaccessible files (even if matches were found).

Note: `ed.c` contains goto statements! (Needless to say, your code should remove them, but don't get crazy here and cut them all out at once. Proceed cautiously, and remove them one by one AFTER you have made the code much simpler.)

Any unneeded variables, functions, and so on should be removed. Not doing so will cost you points.

Use a header file to prototype all functions used within `grep`. Your implementation (.c) code should be inside a single file called `grep.c`. You cannot use system functions.

Recompile frequently when doing this project. Save your work in a series of intermediate files, so you can roll back your work when everything suddenly stops working (e.g., grep00.c, grep01.c, ...). Try your `grep` program using stdin (or using `./grep < testfile.txt`) before you graduate to searching multiple files. Make sure `grep` works with its goto statements intact before beginning to remove them. Divide your code sensibly into functions, especially the regexp code, so it could be used again in a later program.

This kind of project is typical of what new developers are asked to do: make small changes to a big program. Of course, you have to understand the scope of the program you're changing before changing it, and make sure you're not breaking it. (Hint: of course you will break it, and some of the fun is in fixing it again, and trying a different approach until you get the whole thing working.) Code that does not relate to regular expressions almost surely has to go.

For those unfamiliar with `ed` or `grep`, check out the manual pages for them (`man ed(1)` and `man grep(1)`).\

Submission

Turn in the code for this project by uploading all of the source files you created to a single public repository on GitHub.
While you may discuss this assignment with other students, work you submit must have been completed on your own.
To complete your submission, print the sheet at the back of this file, fill out its spaces, and submit it to the instructor in class by the deadline. Failure to follow the instructions exactly will incur a 10% penalty on the grade for this assignment.

The basic form of using `grep` is as follows: `grep search_string (options) search_files`

Here are some examples (from https://alvinalexander.com/unix/edu/examples/grep.shtml)

search for a string in one or more files

```
grep 'fred' /etc/passwd     # search for lines containing 'fred' in /etc/passwd
grep fred /etc/passwd       # quotes usually not when you don't use regex patterns
grep null *.scala           # search multiple files
```

regular expressions

```
grep '^fred' /etc/passwd               # find 'fred', but only at the start of a line
grep '[FG]oo' *                        # find Foo or Goo in all files in the current dir
grep '[0-9][0-9][0-9]' *     # find all lines in all files in the current dir with three numbers in a row
```

You do NOT have to implement the following functions (although they are useful)...

case-insensitive

```
grep -i joe users.txt                  # find joe, Joe, JOe, JOE, etc.
```

display matching filenames, not lines

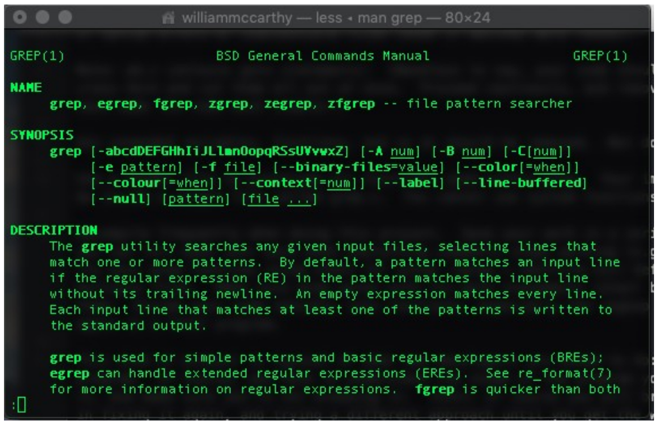
```
grep -l StartInterval *.plist          # show all filenames containing the string 'StartInterval'
grep -il StartInterval *.plist         # same thing, case-insensitive
```

show matching line numbers

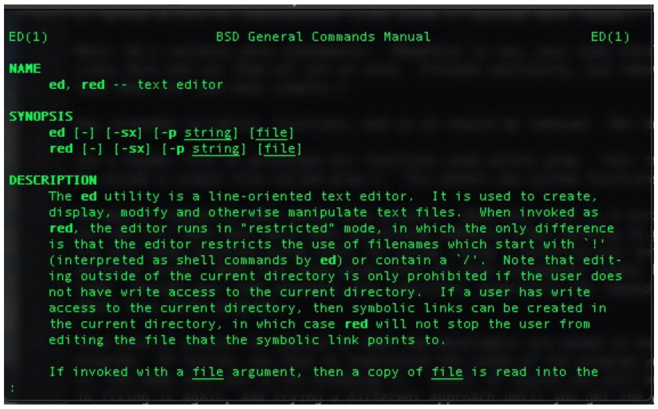
```
grep -n we gettysburg-address.txt  # show line numbers as well as the matching lines
```

reverse the meaning

```
grep -v fred /etc/passwd          # find any line *not* containing 'fred'
grep -vi fred /etc/passwd        # same thing, case-insensitive
```



```
williammccarthy — less ◂ man grep — 80×24

GREP(1)                   BSD General Commands Manual                  GREP(1)

NAME
     grep, egrep, fgrep, zgrep, zegrep, zfgrep -- file pattern searcher

SYNOPSIS
     grep [-abcdDEFGHhIiJLlmnOopqRSsUVvwxZ] [-A num] [-B num] [-C[num]]
          [-e pattern] [-f file] [--binary-files=value] [--color[=when]]
          [--colour[=when]] [--context[=num]] [--label] [--line-buffered]
          [--null] [pattern] [file ...]

DESCRIPTION
     The grep utility searches any given input files, selecting lines that
     match one or more patterns.  By default, a pattern matches an input line
     if the regular expression (RE) in the pattern matches the input line
     without its trailing newline.  An empty expression matches every line.
     Each input line that matches at least one of the patterns is written to
     the standard output.

     grep is used for simple patterns and basic regular expressions (BREs);
     egrep can handle extended regular expressions (EREs).  See re_format(7)
     for more information on regular expressions.  fgrep is quicker than both
:
```



```
ED(1)                     BSD General Commands Manual                    ED(1)

NAME
     ed, red -- text editor

SYNOPSIS
     ed [-] [-sx] [-p string] [file]
     red [-] [-sx] [-p string] [file]

DESCRIPTION
     The ed utility is a line-oriented text editor.  It is used to create,
     display, modify and otherwise manipulate text files.  When invoked as
     red, the editor runs in "restricted" mode, in which the only difference
     is that the editor restricts the use of filenames which start with `!'
     (interpreted as shell commands by ed) or contain a `/'.  Note that edit-
     ing outside of the current directory is only prohibited if the user does
     not have write access to the current directory.  If a user has write
     access to the current directory, then symbolic links can be created in
     the current directory, in which case red will not stop the user from
     editing the file that the symbolic link points to.

     If invoked with a file argument, then a copy of file is read into the
:
```

CPSC 223C Project 1 – grep (from ed), due Wednesday, 20 Mar 2019

Your name and company name: ____

Repository `https://github.com/`____________________/____________________

Verify each of the following items with a corresponding checkmark. Incorrect items will incur a 5% penalty on the grade.

Complete	Incomplete	grep (from ed)
❑	❑	Researched `grep` on the Unix man page for `grep` `(type man grep for grep(1))`
❑	❑	Researched ed on the Unix man page for ed `(type man ed for ed(1))`
❑	❑	Researched regex (regular expressions), and have experimented with using them in ed.
❑	❑	Read through in detail the source code for ed (ed.c).
❑	❑	Made a duplicate of ed.c's 1700+ lines of code, and have compiled it to confirm it works like Unix's ed editor.
❑	❑	Changed main so the program's user interface acts like `grep`, not like ed.
❑	❑	Identified the code unlikely to be associated with the `grep` functionality, commented it out, and confirmed the code still compiles.
❑	❑	Used a header file to prototype all functions in `grep`.h
❑	❑	Removed all unnecessary variables and functions from `grep`.c
❑	❑	Removed all goto statements from `grep`.c
❑	❑	`grep` can search for a string in one or more files
❑	❑	`grep` prints all lines (in all search files) matching the regexp string
❑	❑	`grep` prints a leading filename and colon on each line if multiple files are searched
❑	❑	`grep` supports regular expressions searches (like `'^Fred'` to search for Fred only at the beginning of a line, `'Fred.$'` to search for Fred only at the end of a line, `'[FG]oo' *` to search for either Foo or Goo, and `'[0-9][0-9][0-9]'` to search for three digits in a row, or '[0-8][A-Z]{3}[0-9]{3} to search for a California license plate number
❑	❑	Project directory pushed to new GitHub repository listed above using GitHub client.

Your comments

```
/*
 * Editor   ed.c source code
 */

#include <signal.h>
#include <setjmp.h>

/* make BLKSIZE and LBSIZE 512 for smaller machines */
#define  BLKSIZE  4096
#define  NBLK  2047

#define  NULL  0
#define  FNSIZE  128
#define  LBSIZE  4096
#define  ESIZE  256
#define  GBSIZE  256
#define  NBRA  5
#define  EOF  -1
#define  KSIZE  9

#define  CBRA  1
#define  CCHR  2
#define  CDOT  4
#define  CCL  6
#define  NCCL  8
#define  CDOL  10
#define  CEOF  11
#define  CKET  12
#define  CBACK  14
#define  CCIRC  15

#define  STAR  01

char  Q[]  = "";
char  T[]  = "TMP";
#define  READ  0
#define  WRITE  1

int  peekc;
int  lastc;
char  savedfile[FNSIZE];
char  file[FNSIZE];
char  linebuf[LBSIZE];
char  rhsbuf[LBSIZE/2];
char  expbuf[ESIZE+4];
int  given;
unsigned int  *addr1, *addr2;
unsigned int  *dot, *dol, *zero;
char  genbuf[LBSIZE];
long  count;
char  *nextip;
char  *linebp;
int  ninbuf;
int  io;
int  pflag;
long  lseek(int, long, int);
int  open(char *, int);
int  creat(char *, int);
int  read(int, char*, int);
int  write(int, char*, int);
int  close(int);
int  fork(void);
int  execl(char *, ...);
int  exit(int);
int  wait(int *);
int  unlink(char *);


int  vflag  = 1;
int  oflag;
int  listf;
int  listn;
int  col;
char  *globp;
int  tfile  = -1;
int  tline;
char  *tfname;
char  *loc1;
char  *loc2;
char  ibuff[BLKSIZE];
int  iblock  = -1;
char  obuff[BLKSIZE];
```

```
int  oblock  = -1;
int  ichanged;
int  nleft;
char  WRERR[]  = "WRITE ERROR";
int  names[26];
int  anymarks;
char  *braslist[NBRA];
char  *braelist[NBRA];
int  nbra;
int  subnewa;
int  subolda;
int  fchange;
int  wrapp;
int  bpagesize = 20;
unsigned nlall = 128;

char  *mktemp(char *);
char  tmpXXXXX[50] = "/tmp/eXXXXX";
char  *malloc(int);
char  *realloc(char *, int);

char *getblock(unsigned int atl, int iof);
char *getline(unsigned int tl);
char *place(char *sp, char *l1, char *l2);
void add(int i);
int advance(char *lp, char *ep);
int append(int (*f)(void), unsigned int *a);
int backref(int i, char *lp);
void blkio(int b, char *buf, int (*iofcn)(int, char*, int));
void callunix(void);
int cclass(char *set, int c, int af);
void commands(void);
void compile(int eof);
int compsub(void);
void dosub(void);
void error(char *s);
int execute(unsigned int *addr);
void exfile(void);
void filename(int comm);
void gdelete(void);
int getchr(void);
int getcopy(void);
int getfile(void);
int getnum(void);
int getsub(void);
int gettty(void);
int gety(void);
void global(int k);
void init(void);
unsigned int *address(void);
void join(void);
void move(int cflag);
void newline(void);
void nonzero(void);
void onhup(int n);
void onintr(int n);
void print(void);
void putchr(int ac);
void putd(void);
void putfile(void);
int putline(void);
void puts(char *sp);
void quit(int n);
void rdelete(unsigned int *ad1, unsigned int *ad2);
void reverse(unsigned int *a1, unsigned int *a2);
void setwide(void);
void setnoaddr(void);
void squeeze(int i);
void substitute(int inglob);

jmp_buf  savej;

typedef void  (*SIG_TYP)(int);
SIG_TYP  oldhup;
SIG_TYP  oldquit;
/* these two are not in ansi, but we need them */
#define  SIGHUP  1  /* hangup */
#define  SIGQUIT  3  /* quit (ASCII FS) */

int main(int argc, char *argv[]) {
  char *p1, *p2;
  SIG_TYP oldintr;
  oldquit = signal(SIGQUIT, SIG_IGN);
  oldhup = signal(SIGHUP, SIG_IGN);
  oldintr = signal(SIGINT, SIG_IGN);
  if (signal(SIGTERM, SIG_IGN) == SIG_DFL)
    signal(SIGTERM, quit);
  argv++;
  while (argc > 1 && **argv=='-') {
    switch((*argv)[1]) {

    case '\0':
      vflag = 0;
      break;

    case 'q':
      signal(SIGQUIT, SIG_DFL);
      vflag = 1;
      break;

    case 'o':
      oflag = 1;
      break;
    }
    argv++;
    argc--;
  }
  if (oflag) {
    p1 = "/dev/stdout";
    p2 = savedfile;
    while (*p2++ = *p1++)
      ;
  }
  if (argc>1) {
    p1 = *argv;
    p2 = savedfile;
    while (*p2++ = *p1++)
      if (p2 >= &savedfile[sizeof(savedfile)])
        p2--;
    globp = "r";
  }
  zero = (unsigned *)malloc(nlall*sizeof(unsigned));
  tfname = mktemp(tmpXXXXX);
  init();
  if (oldintr!=SIG_IGN)
    signal(SIGINT, onintr);
  if (oldhup!=SIG_IGN)
    signal(SIGHUP, onhup);
  setjmp(savej);
  commands();
  quit(0);
  return 0;
}

void commands(void) {
  unsigned int *a1;
  int c;
  int temp;
  char lastsep;

  for (;;) {
  if (pflag) {
    pflag = 0;
    addr1 = addr2 = dot;
    print();
  }
  c = '\n';
  for (addr1 = 0;;) {
    lastsep = c;
    a1 = address();
    c = getchr();
    if (c!=',' && c!=';')
      break;
    if (lastsep==',')
      error(Q);
    if (a1==0) {
      a1 = zero+1;
      if (a1>dol)
        a1--;
    }
    addr1 = a1;
    if (c==';')
      dot = a1;
  }
  if (lastsep!='\n' && a1==0)
    a1 = dol;
  if ((addr2=a1)==0) {
    given = 0;
    addr2 = dot;
  }
  else
    given = 1;
  if (addr1==0)
    addr1 = addr2;
  switch(c) {

  case 'a':
    add(0);
    continue;

  case 'c':
    nonzero();
    newline();
    rdelete(addr1, addr2);
    append(gettty, addr1-1);
    continue;

  case 'd':
    nonzero();
    newline();
    rdelete(addr1, addr2);
    continue;

  case 'E':
    fchange = 0;
    c = 'e';
  case 'e':
    setnoaddr();
    if (vflag && fchange) {
      fchange = 0;
      error(Q);
    }
    filename(c);
    init();
    addr2 = zero;
    goto caseread;

  case 'f':
    setnoaddr();
    filename(c);
    puts(savedfile);
    continue;

  case 'g':
    global(1);
    continue;

  case 'i':
    add(-1);
    continue;


  case 'j':
    if (!given)
      addr2++;
    newline();
    join();
    continue;

  case 'k':
    nonzero();
    if ((c = getchr()) < 'a' || c > 'z')
      error(Q);
    newline();
    names[c-'a'] = *addr2 & ~01;
    anymarks |= 01;
    continue;

  case 'm':
    move(0);
    continue;

  case 'n':
    listn++;
    newline();
    print();
    continue;

  case '\n':
    if (a1==0) {
      a1 = dot+1;
      addr2 = a1;
      addr1 = a1;
    }
    if (lastsep==';')
      addr1 = a1;
    print();
    continue;

  case 'l':
    listf++;
  case 'p':
  case 'P':
    newline();
    print();
    continue;

  case 'Q':
    fchange = 0;
  case 'q':
    setnoaddr();
    newline();
    quit(0);

  case 'r':
    filename(c);
  caseread:
    if ((io = open(file, 0)) < 0) {
      lastc = '\n';
      error(file);
    }
    setwide();
    squeeze(0);
    ninbuf = 0;
    c = zero != dol;
    append(getfile, addr2);
    exfile();
    fchange = c;
    continue;

  case 's':
    nonzero();
    substitute(globp!=0);
    continue;

  case 't':
    move(1);
    continue;

  case 'u':
    nonzero();
    newline();
    if ((*addr2&~01) != subnewa)
      error(Q);
    *addr2 = subolda;
    dot = addr2;
    continue;

  case 'v':
    global(0);
    continue;

  case 'W':
    wrapp++;
  case 'w':
    setwide();
    squeeze(dol>zero);
    if ((temp = getchr()) != 'q' && temp != 'Q') {
      peekc = temp;
      temp = 0;
    }
    filename(c);
    if(!wrapp ||
      ((io = open(file,1)) == -1) ||
      ((lseek(io, 0L, 2)) == -1))
      if ((io = creat(file, 0666)) < 0)
        error(file);
    wrapp = 0;
    if (dol > zero)
      putfile();
    exfile();
    if (addr1<=zero+1 && addr2==dol)
      fchange = 0;
    if (temp == 'Q')
      fchange = 0;
    if (temp)
      quit(0);
    continue;

  case '=':
    setwide();
    squeeze(0);
    newline();
    count = addr2 - zero;
    putd();
    putchr('\n');
    continue;

  case '!':
    callunix();
    continue;

  case EOF:
    return;

  }
  error(Q);
  }
}

void print(void) {
  unsigned int *a1;

  nonzero();
  a1 = addr1;
  do {
    if (listn) {
      count = a1-zero;
      putd();
      putchr('\t');
    }
    puts(getline(*a1++));
  } while (a1 <= addr2);
  dot = addr2;
  listf = 0;
  listn = 0;
  pflag = 0;
}

unsigned int *
address(void) {
  int sign;
  unsigned int *a, *b;
  int opcnt, nextopand;
  int c;

  nextopand = -1;
  sign = 1;
  opcnt = 0;
  a = dot;
  do {
    do c = getchr(); while (c==' ' || c=='\t');
    if ('0'<=c && c<='9') {
      peekc = c;
      if (!opcnt)
        a = zero;
      a += sign*getnum();
    } else switch (c) {
    case '$':
      a = dol;
      /* fall through */
    case '.':
      if (opcnt)
        error(Q);
      break;
    case '\'':
      c = getchr();
      if (opcnt || c<'a' || 'z'<c)
        error(Q);
      a = zero;
      do a++; while (a<=dol && names[c-'a']!=(*a&~01));
      break;
    case '?':
      sign = -sign;
      /* fall through */
    case '/':
      compile(c);
      b = a;
      for (;;) {
        a += sign;
        if (a<=zero)
          a = dol;
        if (a>dol)
          a = zero;
        if (execute(a))
          break;
        if (a==b)
          error(Q);
      }
      break;
    default:
      if (nextopand == opcnt) {
        a += sign;
        if (a<zero || dol<a)
          continue;       /* error(Q); */
      }
      if (c!='+' && c!='-' && c!='^') {
        peekc = c;
        if (opcnt==0)
          a = 0;
        return (a);
      }
      sign = 1;
      if (c!='+')
        sign = -sign;
      nextopand = ++opcnt;
      continue;
    }
    sign = 1;
    opcnt++;
  } while (zero<=a && a<=dol);
  error(Q);
  /*NOTREACHED*/
  return 0;
}

int getnum(void) {
  int r, c;

  r = 0;
  while ((c=getchr())>='0' && c<='9')
    r = r*10 + c - '0';
  peekc = c;
  return (r);
}

void setwide(void) {
  if (!given) {
    addr1 = zero + (dol>zero);
    addr2 = dol;
  }
}

void setnoaddr(void) {
  if (given)
    error(Q);
}

void nonzero(void) {
  squeeze(1);
}

void squeeze(int i) {
  if (addr1<zero+i || addr2>dol || addr1>addr2)
    error(Q);
}

void newline(void) {
  int c;

  if ((c = getchr()) == '\n' || c == EOF)
```

```
    return;
  if (c=='p' || c=='l' || c=='n') {
    pflag++;
    if (c=='l')
      listf++;
    else if (c=='n')
      listn++;
    if ((c=getchr())=='\n')
      return;
  }
  error(Q);
}

void filename(int comm) {
  char *p1, *p2;
  int c;

  count = 0;
  c = getchr();
  if (c=='\n' || c==EOF) {
    p1 = savedfile;
    if (*p1==0 && comm!='f')
      error(Q);
    p2 = file;
    while (*p2++ = *p1++)
      ;
    return;
  }
  if (c!=' ')
    error(Q);
  while ((c = getchr()) == ' ')
    ;
  if (c=='\n')
    error(Q);
  p1 = file;
  do {
    if (p1 >= &file[sizeof(file)-1] || c==' ' || c==EOF)
      error(Q);
    *p1++ = c;
  } while ((c = getchr()) != '\n');
  *p1++ = 0;
  if (savedfile[0]==0 || comm=='e' || comm=='f') {
    p1 = savedfile;
    p2 = file;
    while (*p1++ = *p2++)
      ;
  }
}

void exfile(void) {
  close(io);
  io = -1;
  if (vflag) {
    putd();
    putchr('\n');
  }
}

void onintr(int n) {
  signal(SIGINT, onintr);
  putchr('\n');
  lastc = '\n';
  error(Q);
}

void onhup(int n) {
  signal(SIGINT, SIG_IGN);
  signal(SIGHUP, SIG_IGN);
  if (dol > zero) {
    addr1 = zero+1;
    addr2 = dol;
    io = creat("ed.hup", 0600);
    if (io > 0)
      putfile();
  }
  fchange = 0;
  quit(0);
}

void error(char *s) {
  int c;
  wrapp = 0;
  listf = 0;
  listn = 0;
  putchr('?');
  puts(s);
  count = 0;
  lseek(0, (long)0, 2);
  pflag = 0;
  if (globp)
    lastc = '\n';
  globp = 0;
  peekc = lastc;
  if(lastc)
    while ((c = getchr()) != '\n' && c != EOF)
      ;
  if (io > 0) {
    close(io);
    io = -1;
  }
  longjmp(savej, 1);
}

int getchr(void) {
  char c;
  if (lastc=peekc) {
    peekc = 0;
    return(lastc);
  }
  if (globp) {
    if ((lastc = *globp++) != 0)
      return(lastc);
    globp = 0;
    return(EOF);
  }
  if (read(0, &c, 1) <= 0)
    return(lastc = EOF);
  lastc = c&0177;
  return(lastc);
}

int gettty(void) {
  int rc;

  if (rc = gety())
    return(rc);
  if (linebuf[0]=='.' && linebuf[1]==0)
    return(EOF);
  return(0);
}

int gety(void) {
  int c;
  char *gf;
  char *p;

  p = linebuf;
  gf = globp;
  while ((c = getchr()) != '\n') {
    if (c==EOF) {
      if (gf)
        peekc = c;
      return(c);
    }
    if ((c &= 0177) == 0)
      continue;
    *p++ = c;
    if (p >= &linebuf[LBSIZE-2])
      error(Q);
  }

  *p++ = 0;
  return(0);
}

int getfile(void) {
  int c;
  char *lp, *fp;

  lp = linebuf;
  fp = nextip;
  do {
    if (--ninbuf < 0) {
      if ((ninbuf = read(io, genbuf, LBSIZE)-1) < 0)
        if (lp>linebuf) {
          puts("'\\n' appended");
          *genbuf = '\n';
        }
        else return(EOF);
      fp = genbuf;
      while(fp < &genbuf[ninbuf]) {
        if (*fp++ & 0200)
          break;
      }
      fp = genbuf;
    }
    c = *fp++;
    if (c=='\0')
      continue;
    if (c&0200 || lp >= &linebuf[LBSIZE]) {
      lastc = '\n';
      error(Q);
    }
    *lp++ = c;
    count++;
  } while (c != '\n');
  *--lp = 0;
  nextip = fp;
  return(0);
}

void putfile(void) {
  unsigned int *a1;
  int n;
  char *fp, *lp;
  int nib;

  nib = BLKSIZE;
  fp = genbuf;
  a1 = addr1;
  do {
    lp = getline(*a1++);
    for (;;) {
      if (--nib < 0) {
        n = fp-genbuf;
        if(write(io, genbuf, n) != n) {
          puts(WRERR);
          error(Q);
        }
        nib = BLKSIZE-1;
        fp = genbuf;
      }
      count++;
      if ((*fp++ = *lp++) == 0) {
        fp[-1] = '\n';
        break;
      }
    }
  } while (a1 <= addr2);
  n = fp-genbuf;
  if(write(io, genbuf, n) != n) {
    puts(WRERR);
    error(Q);
  }
}

int append(int (*f)(void), unsigned int *a) {
  unsigned int *a1, *a2, *rdot;
  int nline, tl;

  nline = 0;
  dot = a;
  while ((*f)() == 0) {
    if ((dol-zero)+1 >= nlall) {
      unsigned *ozero = zero;

      nlall += 1024;
      if ((zero = (unsigned *)realloc((char *)zero, nlall*sizeof(unsigned)))==NULL) {
        error("MEM?");
        onhup(0);
      }
      dot += zero - ozero;
      dol += zero - ozero;
    }
    tl = putline();
    nline++;
    a1 = ++dol;
    a2 = a1+1;
    rdot = ++dot;
    while (a1 > rdot)
      *--a2 = *--a1;
    *rdot = tl;
  }
  return(nline);
}

void add(int i) {
  if (i && (given || dol>zero)) {
    addr1--;
    addr2--;
  }
  squeeze(0);
  newline();
  append(gettty, addr2);
}

void callunix(void) {
  SIG_TYP savint;
  int pid, rpid;
  int retcode;

  setnoaddr();
  if ((pid = fork()) == 0) {
    signal(SIGHUP, oldhup);
    signal(SIGQUIT, oldquit);
    execl("/bin/sh", "sh", "-t", 0);
    exit(0100);
  }
  savint = signal(SIGINT, SIG_IGN);
  while ((rpid = wait(&retcode)) != pid && rpid != -1)
    ;
  signal(SIGINT, savint);
  if (vflag) {
    puts("!");
  }
}

void quit(int n) {
  if (vflag && fchange && dol!=zero) {
    fchange = 0;
    error(Q);
  }
  unlink(tfname);
  exit(0);
}

void rdelete(unsigned int *ad1, unsigned int *ad2) {
  unsigned int *a1, *a2, *a3;

  a1 = ad1;
  a2 = ad2+1;
  a3 = dol;
  dol -= a2 - a1;
  do {
    *a1++ = *a2++;
  } while (a2 <= a3);
  a1 = ad1;
  if (a1 > dol)
    a1 = dol;
  dot = a1;
  fchange = 1;
}

void gdelete(void) {
  unsigned int *a1, *a2, *a3;

  a3 = dol;
  for (a1=zero; (*a1&01)==0; a1++)
    if (a1>=a3)
      return;
  for (a2=a1+1; a2<=a3;) {
    if (*a2&01) {
      a2++;
      dot = a1;
    } else
      *a1++ = *a2++;
  }
  dol = a1-1;
  if (dot>dol)
    dot = dol;
  fchange = 1;
}

char *
getline(unsigned int tl) {
  char *bp, *lp;
  int nl;

  lp = linebuf;
  bp = getblock(tl, READ);
  nl = nleft;
  tl &= ~((BLKSIZE/2)-1);
  while (*lp++ = *bp++)
    if (--nl == 0) {
      bp = getblock(tl+=(BLKSIZE/2), READ);
      nl = nleft;
    }
  return(linebuf);
}

int putline(void) {
  char *bp, *lp;
  int nl;
  unsigned int tl;

  fchange = 1;
  lp = linebuf;
  tl = tline;
  bp = getblock(tl, WRITE);
  nl = nleft;
  tl &= ~((BLKSIZE/2)-1);
  while (*bp = *lp++) {
    if (*bp++ == '\n') {
      *--bp = 0;
      linebp = lp;
      break;
    }
    if (--nl == 0) {
      bp = getblock(tl+=(BLKSIZE/2), WRITE);
      nl = nleft;
    }
  }
  nl = tline;
  tline += (((lp-linebuf)+03)>>1)&077776;
  return(nl);
}

char *
getblock(unsigned int atl, int iof) {
  int bno, off;

  bno = (atl/(BLKSIZE/2));
  off = (atl<<1) & (BLKSIZE-1) & ~03;
  if (bno >= NBLK) {
    lastc = '\n';
    error(T);
  }
  nleft = BLKSIZE - off;
  if (bno==iblock) {
    ichanged |= iof;
    return(ibuff+off);
  }
  if (bno==oblock)
    return(obuff+off);
  if (iof==READ) {
    if (ichanged)
      blkio(iblock, ibuff, write);
    ichanged = 0;
    iblock = bno;
    blkio(bno, ibuff, read);
    return(ibuff+off);
  }
  if (oblock>=0)
    blkio(oblock, obuff, write);
  oblock = bno;
  return(obuff+off);
}

void blkio(int b, char *buf, int (*iofcn)(int, char*, int)) {
  lseek(tfile, (long)b*BLKSIZE, 0);
  if ((*iofcn)(tfile, buf, BLKSIZE) != BLKSIZE) {
    error(T);
  }
}

void init(void) {
  int *markp;

  close(tfile);
  tline = 2;
  for (markp = names; markp < &names[26]; )
    *markp++ = 0;
  subnewa = 0;
  anymarks = 0;
  iblock = -1;
  oblock = -1;
  ichanged = 0;
  close(creat(tfname, 0600));
  tfile = open(tfname, 2);
  dot = dol = zero;
}

void global(int k) {
  char *gp;
  int c;
  unsigned int *a1;
  char globuf[GBSIZE];

  if (globp)
    error(Q);
  setwide();
  squeeze(dol>zero);
  if ((c=getchr())=='\n')
    error(Q);
  compile(c);
  gp = globuf;
  while ((c = getchr()) != '\n') {
    if (c==EOF)
      error(Q);
    if (c=='\\') {
      c = getchr();
      if (c!='\n')
        *gp++ = '\\';
    }
    *gp++ = c;
    if (gp >= &globuf[GBSIZE-2])
      error(Q);
  }
  if (gp == globuf)
    *gp++ = 'p';
  *gp++ = '\n';
  *gp++ = 0;
  for (a1=zero; a1<=dol; a1++) {
    *a1 &= ~01;
    if (a1>=addr1 && a1<=addr2 && execute(a1)==k)
      *a1 |= 01;
  }
  /*
   * Special case: g/.../d (avoid n^2 algorithm)
   */
  if (globuf[0]=='d' && globuf[1]=='\n' && globuf[2]=='\0') {
    gdelete();
    return;
  }
  for (a1=zero; a1<=dol; a1++) {
    if (*a1 & 01) {
      *a1 &= ~01;
      dot = a1;
      globp = globuf;
      commands();
      a1 = zero;
    }
  }
}

void join(void) {
  char *gp, *lp;
  unsigned int *a1;

  nonzero();
  gp = genbuf;
  for (a1=addr1; a1<=addr2; a1++) {
    lp = getline(*a1);
    while (*gp = *lp++)
      if (gp++ >= &genbuf[LBSIZE-2])
        error(Q);
  }
  lp = linebuf;
  gp = genbuf;
  while (*lp++ = *gp++)
    ;
  *addr1 = putline();
  if (addr1<addr2)
    rdelete(addr1+1, addr2);
  dot = addr1;
}

void substitute(int inglob) {
  int *mp, nl;
  unsigned int *a1;
  int gsubf;
  int n;

  n = getnum();  /* OK even if n==0 */
  gsubf = compsub();
  for (a1 = addr1; a1 <= addr2; a1++) {
    if (execute(a1)){
      unsigned *ozero;
      int m = n;
      do {
        int span = loc2-loc1;
        if (--m <= 0) {
          dosub();
          if (!gsubf)
            break;
          if (span==0) {  /* null RE match */
            if (*loc2=='\0')
              break;
            loc2++;
          }
        }
      } while (execute((unsigned *)0));
      if (m <= 0) {
        inglob |= 01;
        subnewa = putline();
        *a1 &= ~01;
        if (anymarks) {
          for (mp = names; mp < &names[26]; mp++)
            if (*mp == *a1)
              *mp = subnewa;
        }
        subolda = *a1;
        *a1 = subnewa;
        ozero = zero;
        nl = append(getsub, a1);
        nl += zero-ozero;
        a1 += nl;
        addr2 += nl;
      }
    }
  }
  if (inglob==0)
    error(Q);
}

int compsub(void) {
  int seof, c;
  char *p;

  if ((seof = getchr()) == '\n' || seof == ' ')
    error(Q);
  compile(seof);
  p = rhsbuf;
  for (;;) {
    c = getchr();
    if (c=='\\')
      c = getchr() | 0200;
    if (c=='\n') {
      if (globp && globp[0])  /* last '\n' does not count */
        c |= 0200;
      else {
        peekc = c;
        pflag++;
        break;
      }
    }
    if (c==seof)
      break;
    *p++ = c;
    if (p >= &rhsbuf[LBSIZE/2])
      error(Q);
  }
  *p++ = 0;
  if ((peekc = getchr()) == 'g') {
    peekc = 0;
    newline();
    return(1);
  }
  newline();
  return(0);
}

int getsub(void) {
  char *p1, *p2;

  p1 = linebuf;
  if ((p2 = linebp) == 0)
    return(EOF);
  while (*p1++ = *p2++)
    ;
  linebp = 0;
  return(0);
}

void dosub(void) {
  char *lp, *sp, *rp;
  int c;

  lp = linebuf;
  sp = genbuf;
  rp = rhsbuf;
  while (lp < loc1)
    *sp++ = *lp++;
  while (c = *rp++&0377) {
    if (c=='&') {
      sp = place(sp, loc1, loc2);
      continue;
    } else if (c&0200 && (c &= 0177) >='1' && c < nbra+'1') {
      sp = place(sp, braslist[c-'1'], braelist[c-'1']);
      continue;
    }
    *sp++ = c&0177;
    if (sp >= &genbuf[LBSIZE])
      error(Q);
  }
  lp = loc2;
  loc2 = sp - genbuf + linebuf;
  while (*sp++ = *lp++)
    if (sp >= &genbuf[LBSIZE])
      error(Q);
  lp = linebuf;
  sp = genbuf;
  while (*lp++ = *sp++)
    ;
}

char *
place(char *sp, char *l1, char *l2) {
  while (l1 < l2) {
    *sp++ = *l1++;
    if (sp >= &genbuf[LBSIZE])
      error(Q);
  }
  return(sp);
}

void move(int cflag) {
  unsigned int *adt, *ad1, *ad2;

  nonzero();
  if ((adt = address())==0)  /* address() guarantees addr is in range */
    error(Q);
  newline();
  if (cflag) {
    unsigned int *ozero;
    int delta;

    ad1 = dol;
    ozero = zero;
    append(getcopy, ad1++);
    ad2 = dol;
    delta = zero - ozero;
    ad1 += delta;
    adt += delta;
  } else {
    ad2 = addr2;
    for (ad1 = addr1; ad1 <= ad2;)
      *ad1++ &= ~01;
    ad1 = addr1;
  }
  ad2++;
  if (adt<ad1) {
    dot = adt + (ad2-ad1);
    if ((++adt)==ad1)
      return;
    reverse(adt, ad1);
    reverse(ad1, ad2);
    reverse(adt, ad2);
  } else if (adt >= ad2) {
    dot = adt++;
    reverse(ad1, ad2);
    reverse(ad2, adt);
    reverse(ad1, adt);
  } else
    error(Q);
  fchange = 1;
}

void reverse(unsigned int *a1, unsigned int *a2) {
  int t;

  for (;;) {
    t = *--a2;
    if (a2 <= a1)
      return;
    *a2 = *a1;
    *a1++ = t;
  }
}

int getcopy(void) {
  if (addr1 > addr2)
    return(EOF);
  getline(*addr1++);
  return(0);
}

void compile(int eof) {
  int c;
  char *ep;
  char *lastep;
  char bracket[NBRA], *bracketp;
  int cclcnt;

  ep = expbuf;
  bracketp = bracket;
  if ((c = getchr()) == '\n') {
    peekc = c;
    c = eof;
  }
  if (c == eof) {
    if (*ep==0)
      error(Q);
    return;
  }
  nbra = 0;
  if (c=='^') {
    c = getchr();
    *ep++ = CCIRC;
  }
  peekc = c;
  lastep = 0;
  for (;;) {
    if (ep >= &expbuf[ESIZE])
      goto cerror;
    c = getchr();
    if (c == '\n') {
```

```
    peekc = c;
    c = eof;
  }
  if (c==eof) {
    if (bracketp != bracket)
      goto cerror;
    *ep++ = CEOF;
    return;
  }
  if (c!='*')
    lastep = ep;
  switch (c) {

  case '\\':
    if ((c = getchr())=='(') {
      if (nbra >= NBRA)
        goto cerror;
      *bracketp++ = nbra;
      *ep++ = CBRA;
      *ep++ = nbra++;
      continue;
    }
    if (c == ')') {
      if (bracketp <= bracket)
        goto cerror;
      *ep++ = CKET;
      *ep++ = *--bracketp;
      continue;
    }
    if (c>='1' && c<'1'+NBRA) {
      *ep++ = CBACK;
      *ep++ = c-'1';
      continue;
    }
    *ep++ = CCHR;
    if (c=='\n')
      goto cerror;
    *ep++ = c;
    continue;

  case '.':
    *ep++ = CDOT;
    continue;

  case '\n':
    goto cerror;

  case '*':
    if (lastep==0 || *lastep==CBRA || *lastep==CKET)
      goto defchar;
    *lastep |= STAR;
    continue;

  case '$':
    if ((peekc=getchr()) != eof && peekc!='\n')
      goto defchar;
    *ep++ = CDOL;
    continue;

  case '[':
    *ep++ = CCL;
    *ep++ = 0;
    cclcnt = 1;
    if ((c=getchr()) == '^') {
      c = getchr();
      ep[-2] = NCCL;
    }
    do {
      if (c=='\n')
        goto cerror;
      if (c=='-' && ep[-1]!=0) {
        if ((c=getchr())==']') {
          *ep++ = '-';
          cclcnt++;
          break;
        }
        while (ep[-1]<c) {
          *ep = ep[-1]+1;
          ep++;
          cclcnt++;
          if (ep>=&expbuf[ESIZE])
            goto cerror;
          }
        }
        *ep++ = c;
        cclcnt++;
        if (ep >= &expbuf[ESIZE])
          goto cerror;
      } while ((c = getchr()) != ']');
      lastep[1] = cclcnt;
      continue;

    defchar:
    default:
      *ep++ = CCHR;
      *ep++ = c;
    }
  }
   cerror:
  expbuf[0] = 0;
  nbra = 0;
  error(Q);
}

int execute(unsigned int *addr) {
  char *p1, *p2;
  int c;

  for (c=0; c<NBRA; c++) {
    braslist[c] = 0;
    braelist[c] = 0;
  }
  p2 = expbuf;
  if (addr == (unsigned *)0) {
    if (*p2==CCIRC)
      return(0);
    p1 = loc2;
  } else if (addr==zero)
    return(0);
  else
    p1 = getline(*addr);
  if (*p2==CCIRC) {
    loc1 = p1;
    return(advance(p1, p2+1));
  }
  /* fast check for first character */
  if (*p2==CCHR) {
    c = p2[1];
    do {
      if (*p1!=c)
        continue;
      if (advance(p1, p2)) {
        loc1 = p1;
        return(1);
      }
    } while (*p1++);
    return(0);
  }
  /* regular algorithm */
  do {
    if (advance(p1, p2)) {
      loc1 = p1;
      return(1);
    }
  } while (*p1++);
  return(0);
}

int advance(char *lp, char *ep) {
  char *curlp;
  int i;

  for (;;) switch (*ep++) {

  case CCHR:
    if (*ep++ == *lp++)
      continue;
    return(0);

  case CDOT:
    if (*lp++)
      continue;
    return(0);
```

```
case CDOL:
  if (*lp==0)
    continue;
  return(0);

case CEOF:
  loc2 = lp;
  return(1);

case CCL:
  if (cclass(ep, *lp++, 1)) {
    ep += *ep;
    continue;
  }
  return(0);

case NCCL:
  if (cclass(ep, *lp++, 0)) {
    ep += *ep;
    continue;
  }
  return(0);

case CBRA:
  braslist[*ep++] = lp;
  continue;

case CKET:
  braelist[*ep++] = lp;
  continue;

case CBACK:
  if (braelist[i = *ep++]==0)
    error(Q);
  if (backref(i, lp)) {
    lp += braelist[i] - braslist[i];
    continue;
  }
  return(0);

case CBACK|STAR:
  if (braelist[i = *ep++] == 0)
    error(Q);
  curlp = lp;
  while (backref(i, lp))
    lp += braelist[i] - braslist[i];
  while (lp >= curlp) {
    if (advance(lp, ep))
      return(1);
    lp -= braelist[i] - braslist[i];
  }
  continue;

case CDOT|STAR:
  curlp = lp;
  while (*lp++)
    ;
  goto star;

case CCHR|STAR:
  curlp = lp;
  while (*lp++ == *ep)
    ;
  ep++;
  goto star;

case CCL|STAR:
case NCCL|STAR:
  curlp = lp;
  while (cclass(ep, *lp++, ep[-1]==(CCL|STAR)))
    ;
  ep += *ep;
  goto star;

star:
  do {
    lp--;
    if (advance(lp, ep))
      return(1);
  } while (lp > curlp);
  return(0);
  default:
    error(Q);
  }
}

int backref(int i, char *lp) {
  char *bp;

  bp = braslist[i];
  while (*bp++ == *lp++)
    if (bp >= braelist[i])
      return(1);
  return(0);
}

int cclass(char *set, int c, int af) {
  int n;

  if (c==0)
    return(0);
  n = *set++;
  while (--n)
    if (*set++ == c)
      return(af);
  return(!af);
}

void putd(void) {
  int r;

  r = count%10;
  count /= 10;
  if (count)
    putd();
  putchr(r + '0');
}

void puts(char *sp) {
  col = 0;
  while (*sp)
    putchr(*sp++);
  putchr('\n');
}

char  line[70];
char  *linp  = line;

void putchr(int ac) {
  char *lp;
  int c;

  lp = linp;
  c = ac;
  if (listf) {
    if (c=='\n') {
      if (linp!=line && linp[-1]==' ') {
        *lp++ = '\\';
        *lp++ = 'n';
      }
    } else {
      if (col > (72-4-2)) {
        col = 8;
        *lp++ = '\\';
        *lp++ = '\n';
        *lp++ = '\t';
      }
      col++;
      if (c=='\b' || c=='\t' || c=='\\') {
        *lp++ = '\\';
        if (c=='\b')
          c = 'b';
        else if (c=='\t')
          c = 't';
        col++;
      } else if (c<' ' || c=='\177') {
        *lp++ = '\\';
        *lp++ =  (c>>6)    +'0';
        *lp++ = ((c>>3)&07)+'0';
        c     = ( c    &07)+'0';
        col += 3;
      }
    }
  }
  *lp++ = c;
  if(c == '\n' || lp >= &line[64]) {
    linp = line;
    write(oflag?2:1, line, lp-line);
    return;
  }
  linp = lp;
}
```