```
/* Thom Hemenway
 * 360 lab 2
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct node{
  struct node *childPtr, *siblingPtr, *parentPtr;
  char name[64];
  char type;
}NODE;

// helper struct for saving file function
typedef struct stack{
  struct stack *prev;
  char name[64];
}Stack;

// global variables
NODE *root, *cwd;
char line[128], command[16], pathname[64],
     dirname[64], basename[64];
char *cmd[] = {"mkdir", "rmdir", "ls", "cd", "pwd", "creat", "rm", "quit", "help", "menu", "reload",
"save", "?", 0};
FILE* file;

// helper function to convert string to int
int findCommand(char *command)
{
  int i = 0;
  while(cmd[i])
  {
    if(!strcmp(command, cmd[i])) return i;
    i++;
  }
  return -1;
}

int mkdir(char path[])
{
  if(strcmp(path, "") == 0)
  {
    printf("   ERROR: no path name\n");
    return 0;
  }
  // don't need to add root when loading from file
  else if(strcmp(path, "/") == 0) return 0;

  NODE *child, *parent;
  if(path[0] == '/') parent = root;
  else parent = cwd;

  splitPathName(path);

//Go to dirname
  char *mover = strtok(dirname, "/");

  while(mover != NULL)
  {
    child = parent->childPtr;
    while(1)
    {
      if(child == NULL)
      {
        printf("   ERROR: no pathname exists.\n");
        return 0;
      }
      if(strcmp(child->name, mover) == 0 && 'D' == child->type)
      {
        parent = child;
        break;
      }

      child = child->siblingPtr;
    }
    mover = strtok(NULL, "/");
  }

  NODE *sibling = child = parent->childPtr;
  NODE *prevSibling;
  while(sibling != NULL)
  {
    if(strcmp(sibling->name, basename) == 0)
    {
      printf("   ERROR: path already exists.\n");
      return 0;
    }
    prevSibling = sibling;
    sibling = sibling->siblingPtr;
  }

  if(child == NULL)
  {
    child = malloc(sizeof(NODE));
    child->parentPtr = parent;
    strcpy(child->name, basename);
    child->type = 'D';
    child->siblingPtr = NULL;
    child->childPtr = NULL;

    parent->childPtr = child;
  }
  else
  {
    sibling = malloc(sizeof(NODE));
    sibling->parentPtr = parent;
    strcpy(sibling->name, basename);
    sibling->type = 'D';
    sibling->childPtr = NULL;
    sibling->siblingPtr = NULL;

    prevSibling->siblingPtr = sibling;
  }
}

void rmdir(char *path)
{
  if(strcmp(path, "") == 0)
  {
    printf("   ERROR: no path name\n");
    return 0;
  }

  //get dirname & basename
  splitPathName(path);

  NODE *child, *parent;
  if(dirname[0] == '/') parent = root;
  else parent = cwd;

//Go to dirname
  char *mover = strtok(dirname, "/");
  while(mover != NULL)
  {
    child = parent->childPtr;
    while(1)
    {
      if(child == NULL)
      {
        printf("   ERROR: no pathname exists.\n");
        return 0;
      }
      // Go into directory
      if(strcmp(child->name, mover) == 0 && 'D' == child->type)
      {
        parent = child;
        break;
      }

      // go to each sibling
      child = child->siblingPtr;
    }
    mover = strtok(NULL, "/");
  }

  NODE *sibling = child = parent->childPtr;
  if(child != NULL)
  {
    if(strcmp(child->name, basename) == 0)
    {
      // File node
      if('F' == child->type)
      {
        printf("   ERROR: can't delete file. Use rm to delete file.\n");
        return 0;
      }
      // is directory empty?
      if(child->childPtr == NULL)
      {
        // parent's child becomes child's sibling
        parent->childPtr = child->siblingPtr;
      }
      else printf("   ERROR: directory not empty.\n");
      return 0;
    }
    else
    {
      while(sibling != NULL)
      {
        if(strcmp(sibling->name, basename) == 0 && 'D' == sibling->type)
        {
          if(sibling->childPtr == NULL)
          {
            child->siblingPtr = sibling->siblingPtr;
          }
          else
          {
            printf("   ERROR: directory not empty.\n");
          }
          return 0;
        }
        child = sibling;
        sibling = sibling->siblingPtr;
      }
    }
  }
  printf("   ERROR: directory not found.\n");
}

int cd()
```

```
{
  if(strcmp(pathname, "") == 0 || strcmp(pathname, "/") == 0)
  {
    cwd = root;
    return 0;
  }

  //Go to path
  NODE *parent, *child;
  if(pathname[0] == '/') parent = root;
  else parent = cwd;
  child = parent->childPtr;

  char *mover = strtok(pathname, "/");
  while(mover != NULL)
  {
    while(1)
    {
      if(child == NULL)
      {
        printf("   ERROR: no pathname exists.\n");
        return 0;
      }
      if(strcmp(child->name, mover) == 0 && 'D' == child->type)
      {
        parent = child;
        break;
      }
      child = child->siblingPtr;
    }
    child = parent->childPtr;
    mover = strtok(NULL, "/");
  }

  cwd = parent;
}

int ls()
{
  NODE *child, *parent;
  if((strcmp(pathname, "") == 0) || pathname[0] != '/') parent = cwd;
  else parent = root;

//Go to path
  char *mover = strtok(pathname, "/");
  child = parent->childPtr;
  while(mover != NULL)
  {
    while(1)
    {
      if(child == NULL)
      {
        printf("   ERROR: no pathname exists.\n");
        return 0;
      }
      if(strcmp(child->name, mover) == 0)
      {
        parent = child;
        break;
      }
      child = child->siblingPtr;
    }
    child = parent->childPtr;
    mover = strtok(NULL, "/");
  }

  child = parent->childPtr;
  while(child != NULL)
  {
    printf("   %c\t%s\n", child->type, child->name);
    child = child->siblingPtr;
  }
}


int save(char filename[])
{
  if(strcmp(filename, "") == 0)
  {
    printf("   ERROR: no file name specified.\n");
    return 0;
  }

  printf("saving %s...\n", filename);
  file = fopen(filename, "w+");


  if(file == NULL)
  {
    printf("   ERROR: couldn't open file.\n");
    return 0;
  }

  // don't need to include root
  preOrderWrite(root->childPtr);

  fclose(file);                                 // close FILE stream when done
}

void preOrderWrite(NODE* node)
{
  // for safety measures
  if(node == NULL)
  {
    return 0;
  }

  fprintf(file, "%c\t", node->type);

  // Get absolute path
  NODE *temp = node;
  Stack *top = malloc(sizeof(Stack)), *pusher;
  strcpy(top->name, temp->name);
  top->prev = NULL;

  // push parents on stack
  while(temp->parentPtr != NULL)
  {
    // create stack
    temp = temp->parentPtr;

    if(temp != NULL)
    {
      pusher = malloc(sizeof(Stack));
      strcpy(pusher->name, temp->name);

      // push
      pusher->prev = top;
      top = pusher;
    }
  }

  // pop root
  top = top->prev;
  // pop stack
  while(top != NULL)
  {
    // print absolute path
    fprintf(file, "/%s", top->name);
    // pop
    top = top->prev;
  }
  // end of line
  fprintf(file, "\n");

  // recursively go pre-order
  if(node->childPtr != NULL)
  {
    preOrderWrite(node->childPtr);
  }
  if(node->siblingPtr != NULL)
  {
    preOrderWrite(node->siblingPtr);
  }
}

int pwd()
{
  printf("   ");
  rpwd(cwd);
  printf("\n");
}

int rpwd(NODE *node)
{
  if(strcmp(node->name, "/"))
  {
    rpwd(node->parentPtr);
    printf("%s/", node->name);
  }
  else printf("%s", node->name);
}

int creat(char path[])
{
  if(strcmp(path, "") == 0)
  {
    printf("   ERROR: no file name\n");
    return 0;
  }

  NODE *child, *parent;
  if(path[0] == '/') parent = root;
  else parent = cwd;

  // split pathname
  splitPathName(path);

//Go to dirname
  char *mover = strtok(dirname, "/");

  while(mover != NULL)
  {
    child = parent->childPtr;
    while(1)
    {
      if(child == NULL)
      {
        printf("   ERROR: no pathname exists.\n");
        return 0;
      }
      if(strcmp(child->name, mover) == 0 && 'D' == child->type)
      {
        parent = child;
        break;
      }

      child = child->siblingPtr;
    }
    mover = strtok(NULL, "/");
  }

  child = parent->childPtr;
  if(child == NULL)
  {
    // Create file node
    child = malloc(sizeof(NODE));
    strcpy(child->name, basename);
    child->type = 'F';
    child->parentPtr = parent;
    child->childPtr = NULL;
    child->siblingPtr = NULL;
    parent->childPtr = child;
    return 0;
  }
  // Go through all siblings
  NODE *sibling;
  while(child != NULL)
  {
    if(strcmp(child->name, basename) == 0)
    {
      printf("   ERROR: file already exists.\n");
      return 0;
    }
    sibling = child;
    child = child->siblingPtr;
  }
  // Create file node
  child = malloc(sizeof(NODE));
  strcpy(child->name, basename);
  child->type = 'F';
  child->parentPtr = parent;
  child->siblingPtr = NULL;
  child->childPtr = NULL;
  sibling->siblingPtr = child;
}

int splitPathName(char *path)
{
  char temp[64];
  strcpy(temp, path);
  char *mover = strtok(temp, "/");
  // get basename
  while(mover != NULL)
  {
    strcpy(basename, mover);
    mover = strtok(NULL, "/");
  }

  // get dirname
  // dirname = path[0 : (strlen(path) - strlen(basename) - 1)];
  if(strlen(path) - strlen(basename) == 0) strncpy(dirname, path, strlen(path) - strlen(basename));
  else strncpy(dirname, path, strlen(path) - strlen(basename) - 1);
}

int rm(char *path)
{
  if(strcmp(path, "") == 0)
  {
    printf("   ERROR: no pathname.\n");
    return 0;
  }

  NODE *child, *parent;
  if(path[0] == '/') parent = root;
  else parent = cwd;

  // split pathname
  splitPathName(path);

//Go to dirname
  char *mover = strtok(dirname, "/");
  while(mover != NULL)
  {
    child = parent->childPtr;
    while(1)
    {
      if(child == NULL)
      {
        printf("   ERROR: no pathname exists.\n");
        return 0;
      }
      if(strcmp(child->name, mover) == 0 && 'D' == child->type)
      {
        parent = child;
        break;
      }

      child = child->siblingPtr;
    }
    mover = strtok(NULL, "/");
  }

  // Seek out file and destroy
  child = parent->childPtr;
  NODE *sibling = child->siblingPtr;
  if(child != NULL)
  {
    if(strcmp(child->name, basename) == 0)
    {
      // is it a directory node?
      if('D' == child->type)
      {
        printf("   ERROR: can't delete directory. Use rmdir to delete directory.\n");
        return 0;
      }
      // Delete file
      parent->childPtr = child->siblingPtr;
      return 0;
    }
    else
    {
      while(sibling != NULL)
      {
        // Did we find the file and does it have the correct type?
        if(strcmp(sibling->name, basename) == 0 && 'F' == sibling->type)
        {
          child->siblingPtr = sibling->siblingPtr;
          return 0;
        }
        child = sibling;
        sibling = sibling->siblingPtr;
      }
    }
  }
  printf("   ERROR: directory not found.\n");
}

int reload()
{
  if(strcmp(pathname, "") == 0)
  {
    printf("   ERROR: no file name specified.\n");
    return 0;
  }
  printf("reloading %s...\n", pathname);
  file = fopen(pathname, "r");           // open a FILE stream for READ

  if(file == NULL)
  {
    printf("   ERROR: couldn't open file.\n");
    return 0;
  }

  char type, path[100], line[100];
  // at end of file
  while(1)
  {
    // get line of file
    fgets(line, 100, file);
    // Did we read the end of the file?
    if(feof(file)) break;

    line[strlen(line)] = 0; line[strlen(line)-1] = 0;
    if(line != NULL || strcmp(line, "") != 0)
    {

      // split line
      char *splitter = strtok(line, "\t");
      type = *splitter;
      splitter = strtok(NULL, "\t");
      strcpy(path, splitter);
      switch(type)
      {
        case 'F':
          creat(path);
        break;
        case 'D':
          mkdir(path);
        break;
      }
    }
    clearGlobals();
  }
  fclose(file);
}

int menu()
{
}

int quit()
{
  char path[100];
  // save file
  printf("Enter file name: ");
  gets(path);

  save(path);

  // Terminate
  printf("Exiting...\n");
  exit(0);
}

int help()
{
  printf(":::::::::: Commands ::::::::::\n");
  printf("mkdir [pathname]: creates new directory if it doesn't exist.\n");
  printf("rmdir [pathname]: removes directory if it is empty.\n");
  printf("cd [pathname]: change current directory.\n");
  printf("ls [pathname]: list the directory contents.\n");
  printf("pwd: print the (absolute) pathname of the current directory.\n");
  printf("creat [pathname]: create a file.\n");
  printf("rm [pathname]: remove a file.\n");
  printf("save [filename]: save the current file system tree in.\n");
  printf("reload [filename]: re-initalize the file system tree from a.\n");
  printf("quit: save the file system tree, then terminate the program.\n");

}

int (*fptr[])(char*) = {(int(*)())mkdir, rmdir, ls, cd, pwd, creat, rm, quit, help, menu, reload, save};

int initialize()
{
  // Init root
  root = malloc(sizeof(NODE));
  strcpy(root->name, "/");
  root->type = 'D';
  root->siblingPtr = NULL;
  root->parentPtr = NULL;
  root->childPtr = NULL;

  // Currently at '/'
  cwd = root;
}

int clearGlobals()
{
  // Reset for each command
  memset(dirname, 0, sizeof(dirname));
  memset(pathname, 0, sizeof(pathname));
  memset(basename, 0, sizeof(basename));
}
int main()
{
  int r, com;
  initialize();

  printf("enter help to get a list of commands.\n");
  while(1)
  {
    printf(" > ");
    gets(line);
    sscanf(line, "%s %s", command, pathname);

    // for using a file as stdin to make things pretty
    printf("%s\n", line);

    com = findCommand(command);
    if(com != -1) r = fptr[com](pathname);
    else printf("   ERROR: %s, isn't a valid command!\n", command);

    clearGlobals();
  }

  return 0;
}
```