
```
lab10: main.c list.c files.c
      gcc -pthread -o lab10 main.c list.c files.c -I.
      ./lab10 contactData.txt contactBinData.bin autosaveToBin.bin\
```

```
#ifndef lab10
#define lab10

/* Base File - contains all global variables, function declarations, and
structure definitions */

/* CONTACT_NODE - a node containing the contact information */
struct CONTACT_NODE {
    char name[50];
    char phoneNumber[50];
    struct CONTACT_NODE *PREV;
    struct CONTACT_NODE *NEXT;
};

/* LETTER_HEAD_CONTACT_NODES - an array in which each head index represents an
alphabetical letter*/
struct CONTACT_NODE *LETTER_HEAD_CONTACT_NODES[26];

/* Mutex for locking and unlocking threads */
pthread_mutex_t autosave_mutex;

/* Functions list */
void insert(char newName[50], char newPhoneNumber[50]);
void delete(char oldName[50]);
void show();
void show_letter(char letter);
void save_textfile(char filename[50]);
void read_textfile(char filename[50]);
void write_binaryfile(char filename[50]);
void print_binaryfile(char filename[50]);
void reverse(struct CONTACT_NODE *nodeP1, struct CONTACT_NODE *nodeP2, int
index);
void * autosaveToBin(void * filename);


#endif
```

```
/* Mods */
#include <stdio.h>
```

```
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <pthread.h>
#include "lab10.h"

/* main.c - contains the function main() and is the centric file of the
project */

/*

Phone Book 7.1:
A directory program in which the computer allows the user to add, delete, or
display contacts.

Commands:
1 - Adds a new contact to the directory
2 - Delete an old contact from the directory
3 - Show a table of all contacts
4 - Show a table of all contacts with names of a specified starting letter
5 - Record directory contents into a binary file
6 - Show a table of the binary file contents in readable text
7 - Reverses the order of contents
8 - Quit and save directory to file

Requirements (to add on top of PHONEBOOK 7.0):
> Create a thread that would save the directory contents to a separate binary
file every five seconds
    - The thread and the program must run separately yet concurrently
    - Use to sleep function to initiate the five second pauses
    - Locks should be used when changing the list, handling the auto-save
file, and killing the thread

*/

/* MAIN */
int main(int argc, char* argv[]) {

    /* Create Directory and Recall FILEs, if nonexistent */
    FILE *DIRECTORY = fopen(argv[1], "a");
    FILE *BINARYRCL = fopen(argv[2], "ab");
    FILE *FAUTOSAVE = fopen(argv[3], "ab");

    /* Introduces the user to the program and lists out the possible commands
*/
    char openingMessage[1000] = "\n"
    "#================================#\n"
    "| PHONE BOOK 7.1 (AUTOSAVE ADDED) |\n"
    "#================================#\n"
```

```
    "1 -> Create a new contact for the directory\n"
    "2 -> Delete a contact off of the directory\n"
    "3 -> Show all known contacts in the directory in alphabetical order\n"
    "4 -> Show all known contacts that start with a specified letter\n"
    "5 -> Stores current directory in a separate file\n"
    "6 -> Recalls the stored directory from the separate file\n"
    "7 -> Show all known contacts in the directory in reverse alphabetical
order\n"
    "8 -> Quits the program and saves to file\n"
    "9 -> Shows the contents of the autosaved file\n"
    "NOTE: Please start names with letter and keep all inputs under 50
words\n\n";
    printf("%s", openingMessage);

    /* Set up the titles used to introduce tables */
    char contactListTitle[100] =
    "#==================#\n"
    "| LIST OF CONTACTS |\n"
    "#==================#\n";
    char contactListTitle2[100] =
    "#====================#\n"
    "| LIST OF CONTACTS 0 |\n"
    "#====================#\n";
    char contactListTitle3[150] =
    "#======================#\n"
    "| LIST OF CONTACTS RCL |\n"
    "#======================#\n";
    char contactListTitle4[150] =
    "#======================#\n"
    "| LIST OF CONTACTS REV |\n"
    "#======================#\n";
    char contactListTitle5[150] =
    "#======================#\n"
    "| LIST OF CONTACTS ASF |\n"
    "#======================#\n";


      /* Initialize LETTER_HEAD_CONTACT_NODES */
      int i; for (i=0;i<26;i++) LETTER_HEAD_CONTACT_NODES[i] = NULL;
      read_textfile(argv[1]);

      /* Define parsing variables*/
      char callNumber, callNumberMSG[50], phoneNumber[50], name[50],
letterMSG[50], letter;

      /* Define, initialize, and start the thread */
      pthread_t THREAD_FOR_AUTOSAVE; void *autosave_ret;
      pthread_create(&THREAD_FOR_AUTOSAVE, NULL, autosaveToBin,(void *)
argv[3]);
```

```
      /* Start Game Loop */
      for (;;) {

          /* Get user input */
          // NOTE: fgets also copies returns so the string length of a single
inputedcharacter = 2
          printf(">>> "); fgets(callNumberMSG ,sizeof(callNumberMSG), stdin);
          if (strlen(callNumberMSG)==2) callNumber = callNumberMSG[0];
          else {printf("***ERROR: Please enter a single character.\n");
continue;}

          /* Parse user input */
          switch(callNumber) {

              /* 1 - adds a new contact to the directory*/
              case '1':

                  /* Get new contact information from user */
                  printf("Name: "); fgets(name, sizeof(name), stdin);
name[strlen(name)-1] = '\0';
                  printf("Phone Number: "); fgets(phoneNumber, sizeof(name),
stdin);  phoneNumber[strlen(phoneNumber)-1] = '\0';

                  /* Creates new contact and inserts said contact into
directory*/
                  insert(name, phoneNumber); break;

              /* 2 - subtract a contact from the directory */
              case '2':

                  /* Get name for deletion */
                  printf("Name (for deletion): "); fgets(name, sizeof(name),
stdin); name[strlen(name)-1] = '\0';

                  /* Delete contact associated with name off the directory */
                  delete(name); break;

              /* 3 - show all contacts */
              case '3':

                  /* Print introduction message and tabuluated contacts */
                  printf("%s", contactListTitle);
                  show(); break;

              /* 4 - show all contacts starting with a specific letter */
              case '4':

                  /* Get letter from the user */
```

```
                printf("Enter a Letter: "); fgets(letterMSG, sizeof(name),
stdin); letter = letterMSG[0];

                /* Parse letter here and change introduction message */
                if (letter >= 97 && letter <= 122) letter -= 32;
                else if (letter < 65 || letter > 90)  {printf("***ERROR:
Please type a alphabetical letter\n"); break;}
                contactListTitle2[42] = letter;

                /* print introduction message and tabuluated contacts */
                if (strlen(letterMSG) == 2) {
                    printf("%s", contactListTitle2);
                    show_letter(letter);
                } else printf("***ERROR: Please type a alphabetical
letter\n");
                break;

            /* 5 - Save current directory to binary file */
            case '5':

                /* Print a statement and save directory to binary file*/
                printf("Saving to RCL (binary file)...\n");
                write_binaryfile(argv[2]);
                break;

            /* 6 - Show binary file contents */
            case '6':

                /* Print introductory message and tabulate
contentspthread_cancel(thread1) */
                printf("%s", contactListTitle3);
                print_binaryfile(argv[2]);
                break;

            /* 7 - Show all contacts in reverse order and retrieve the
original list after you're done*/
            case '7':

                /* Print introduction message and tabuluate contacts */
              printf("%s", contactListTitle4);
              reverse(NULL, NULL, 0);
              show();
              reverse(NULL, NULL, 0);
              break;

          /* 8 - save the directory to file and quits */
          case '8':
              printf("Thank you for using PHONE BOOK 7.1.\n");
```

```
            save_textfile(argv[1]); write_binaryfile(argv[2]);
            fclose(DIRECTORY); fclose(BINARYRCL); fclose(FAUTOSAVE);
            pthread_cancel(THREAD_FOR_AUTOSAVE);
            return 0;

          /* 9 - print the file from the auto-save thread */
          case '9':
              printf("%s",contactListTitle5);
          print_binaryfile(argv[3]);
            break;

        /* End of switch statement */
        default:
            printf("***ERROR: Please type a valid command.\n");

        }
    }
}
```

```
/* Mods */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <pthread.h>
#include "lab10.h"

/* list.c - file containing functions that directly interact with the data
pertaining to the contacts */

/* insert - creates a new contact, and, through iteration, ensures that the
name is not a duplicate and places it into the appropriate *
*           alphabetical position in the linked-lists under
LETTER_HEAD_CONTACT_NODES.*/
void insert(char newName[50], char newPhoneNumber[50]) {

    /* Find the index of the appropriate linked-list using the first letter of
the given name */
    char letter = newName[0]; int letterIndex;
    if (letter >= 65 && letter <= 90) letterIndex = (int) letter - 65;
    else if (letter >= 97 && letter <= 122) letterIndex = (int) letter - 97;
    else {printf("***ERROR: Please start names with a letter\n"); return;}
    struct CONTACT_NODE *contact_node_i =
LETTER_HEAD_CONTACT_NODES[letterIndex];

    /* Creates new contact and initialize data variables */
```

```
    struct CONTACT_NODE *newContact = (struct CONTACT_NODE *)
malloc(sizeof(struct CONTACT_NODE));
    strcpy(newContact->name,newName);
    strcpy(newContact->phoneNumber,newPhoneNumber);

    /* After newName has been implemented into the newContact, set newName to
uppercase (for case insentitivity) */
    int i; for (i=0;i<strlen(newName); i++) if (newName[i] >= 97 && newName[i]
<= 122) newName[i] -= 32;

    /* Case 1: Empty: Make the new contact the head */
    if (contact_node_i == NULL) {
        newContact->NEXT = NULL;
        newContact->PREV = NULL;
        LETTER_HEAD_CONTACT_NODES[letterIndex] = newContact;
        return;
    }

    /* Starts iterating over LETTER_HEAD_CONTACT_NODES */
    char savedName[50], savedName2[50];
    while (contact_node_i != NULL) {

      /* Cast all character to uppercase for comparison  */
        strcpy(savedName,contact_node_i->name);
        for (i=0;i<strlen(savedName); i++) if (savedName[i] >= 97 &&
savedName[i] <= 122) savedName[i] -= 32;
        if (contact_node_i->NEXT != NULL) {
             strcpy(savedName2,contact_node_i->NEXT->name);
             for (i=0;i<strlen(savedName2); i++) if (savedName2[i] >= 97 &&
savedName2[i] <= 122) savedName2[i] -= 32;
      }

        /* Check for duplicate */
        if (strcmp(savedName, newName) == 0) {
            printf("***ERROR: A contact with this name already exist in the
directory; please choose another name\n");
            return;
        }

        /* Case 2: Before Head: Make the new contact the head and place it
before the former */
        if (contact_node_i->PREV == NULL && strcmp(newName, savedName)<0) {
            LETTER_HEAD_CONTACT_NODES[letterIndex] = newContact;
            newContact->PREV = NULL;
            newContact->NEXT = contact_node_i;
            contact_node_i->PREV = newContact;
            return;
        }
```

```
        /* Case 3: After Tail: Assign the new contact after the tail */
        else if (contact_node_i->NEXT == NULL && strcmp(newName,
savedName)>0){
            newContact->PREV = contact_node_i;
            newContact->NEXT = NULL;
            contact_node_i->NEXT = newContact;
            return;
        }

        /* Case 4: Middle: Change adjacent nodes and insert new node between
them */
        else if (strcmp(newName,savedName2)<0 && strcmp(newName, savedName)>0)
{
            newContact->PREV = contact_node_i;
            newContact->NEXT = contact_node_i->NEXT;
            contact_node_i->NEXT->PREV = newContact;
            contact_node_i->NEXT = newContact;
            return;
        }

        /* Next iteration */
        contact_node_i = contact_node_i->NEXT;

    }
}

/* delete - indentifies the contact with the given name through iteration, use
"free" to delete it, and conjoin adjacent lists */
void delete(char* oldName) {

    /* Find the index of the appropriate linked-list using the first letter of
the given name */
    char letter = oldName[0]; int letterIndex;
    if (letter >= 65 && letter <= 90) letterIndex = (int) letter - 65;
    else if (letter >= 97 && letter <= 122) letterIndex = (int) letter - 97;
    else {printf("***ERROR: Please start names with a letter\n"); return;}
    struct CONTACT_NODE *contact_node_i =
LETTER_HEAD_CONTACT_NODES[letterIndex];

    /* Cast oldName to uppercase */
    int i; for (i=0;i<strlen(oldName); i++) if (oldName[i] >= 97 && oldName[i]
<= 122) oldName[i] -= 32;

    /* Starts iterating over LETTER_HEAD_CONTACT_NODES */
    char savedName[50];
    while (contact_node_i != NULL) {

          /* Case current node name to uppercase */
        strcpy(savedName, contact_node_i->name);
```

```
        for (i=0;i<strlen(savedName); i++) if (savedName[i] >= 97 &&
savedName[i] <= 122) savedName[i] -= 32;


        /* Check if the old name is equal to the one in the current node */
        if (strcmp(oldName, savedName)==0) {

            /* Case 1: Dead Single: Delete the head */
            if (contact_node_i->PREV == NULL && contact_node_i->NEXT == NULL)
{
                free(LETTER_HEAD_CONTACT_NODES[letterIndex]);
                LETTER_HEAD_CONTACT_NODES[letterIndex] = NULL;
                return;
            }

            /* Case 2: Dead Head: Delete the head, and make the next node the
new head */
            else if (contact_node_i->PREV == NULL) {
                free(LETTER_HEAD_CONTACT_NODES[letterIndex]);
                LETTER_HEAD_CONTACT_NODES[letterIndex] = contact_node_i->NEXT;
                return;
            }

            /* Case 3: Dead Tail: Delete the tail, and adjust the second-last
node */
            else if (contact_node_i->NEXT == NULL) {
                free(contact_node_i);
                contact_node_i->PREV->NEXT = NULL;
                return;
            }

            /* Case 4: Conjoined Middle: Delete the current node and link
adjacent nodes */
            else {
                free(contact_node_i);
                contact_node_i->PREV->NEXT = contact_node_i->NEXT;
                contact_node_i->NEXT->PREV = contact_node_i->PREV;
                return;
            }

        }

        /* Next iteration */
        contact_node_i = contact_node_i->NEXT;

    } printf("***ERROR: Contact associated with given name is not found\n");

}
```

```
/* reverse() - Reverse the order of all linked lists (DEFAULT: index should
equal 0) */
void reverse(struct CONTACT_NODE *nodeP1, struct CONTACT_NODE *nodeP2, int
index) {

    /* Base case: index = 26 */
    if (index == 26) return;

    /* Recursive Case 1: Head does not exist */
    if (LETTER_HEAD_CONTACT_NODES[index] == NULL) {return reverse(NULL, NULL,
index+1);}

    /* Recursive Case 2: Node arguments are both NULL */
    if (nodeP1 == NULL) {
        nodeP1 = LETTER_HEAD_CONTACT_NODES[index];
        nodeP2 = LETTER_HEAD_CONTACT_NODES[index];
        while (nodeP2->NEXT != NULL) nodeP2 = nodeP2->NEXT;
        return reverse(nodeP1, nodeP2, index);
    }

    /* Recursive Case 3: Both are center */
    if (nodeP1 == nodeP2 || nodeP1->PREV == nodeP2) {return reverse(NULL,
NULL, index+1);}

    /* Recursive Case 4: Both are outliers */
    if (nodeP1->PREV == NULL) {
        char tn[50], tpn[50];
        strcpy(tn, nodeP1->name);
        strcpy(tpn, nodeP1->phoneNumber);
        strcpy(LETTER_HEAD_CONTACT_NODES[index]->name, nodeP2->name);
        strcpy(LETTER_HEAD_CONTACT_NODES[index]->phoneNumber,
nodeP2->phoneNumber);
        strcpy(nodeP2->name, tn);
        strcpy(nodeP2->phoneNumber, tpn);
        return reverse(LETTER_HEAD_CONTACT_NODES[index]->NEXT, nodeP2->PREV,
index);
    }

    /* Recursive Case 5: Switch nodes and move towards center */
    else {
        char tn[50], tpn[50];
        strcpy(tn, nodeP2->name);
        strcpy(tpn, nodeP2->phoneNumber);
        strcpy(nodeP2->name, nodeP1->name);
        strcpy(nodeP2->phoneNumber, nodeP1->phoneNumber);
        strcpy(nodeP1->name, tn);
        strcpy(nodeP1->phoneNumber, tpn);
        return reverse(nodeP1->NEXT, nodeP2->PREV, index);
    }
```

```
}
```

```
/* Mods */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <pthread.h>
#include "lab10.h"

/* files.c - files for functions that writes and recalls data from .txt and
.bin files */

/* read_textfile() - creates and writes nodes of the linked-lists based on the
names and phone numbers written on the text file */
void read_textfile(char filename[50]) {

    /* Refer to contactData.txt */
    FILE *DFP = fopen(filename,"r");

    /* Find the number of characters in the file; if it doesn't have more than
201 chars, escape this function */
    /* NOTE: 202 chars is the string length of the intro */
    fseek(DFP, 0, SEEK_END); int lengthOfFile = (int) ftell(DFP);
    if (lengthOfFile<201+1) return;

    /* Create variables for iteration and move in-front of introduction */
    fseek(DFP, 202, SEEK_SET);
    char name[50], phoneNumber[50], test[3];

    /* Start parsing text from contactData.txt */
    while (getc(DFP) != EOF) {

        /* Get strings of 50 chars */
        fseek(DFP, -1, SEEK_CUR);
        fgets(name, 50+1, DFP);
        fgets(phoneNumber, 50, DFP);

        /* Trim the ending spaces because they are annoying */
        int i;
        for (i=strlen(name)-1; i>=0; i--){
            if (name[i] == ' ') name[i] = '\0';
            else if (name[i] == '\0') continue;
            else break;
        }
        for (i=strlen(phoneNumber)-1; i>=0; i--){
            if (phoneNumber[i] == ' ') phoneNumber[i] = '\0';
```

```
            else if (phoneNumber[i] == '\0') continue;
            else break;
        }

        /* Adjust DFP position over newline and insert reminiscent contact*/
        fseek(DFP, 2, SEEK_CUR);
        insert(name, phoneNumber);

    }

    /* Close FILE */
    fclose(DFP); return;

}

/* save_textfile() - prints the names and phone numbers from the linked-list
nodes onto the text file */
void save_textfile(char filename[50]) {

    /* Refer to contactData.txt */
    FILE *DFP = fopen(filename,"w");

    /* Write the beginning of the file */
    char *fileIntro = "Names
Number                                                \n"

"-------------------------------------------------------------------------
-----------------------\n";
    fprintf(DFP, fileIntro, "%s");

    /* Start iterating over lists */
    struct CONTACT_NODE *contact_node_i; int i, j;
    for (i=0; i<26; i++) {
        contact_node_i = LETTER_HEAD_CONTACT_NODES[i];
        while (contact_node_i != NULL) {

            /* Explanation:
               1) print name
               2) print (50 - length of String name) spaces
               3) repeat step 1-2 with phoneNumber
               4) print newline */
            fprintf(DFP, "%s", contact_node_i->name);
            for (j=0; j<(50-strlen(contact_node_i->name)); j++) fprintf(DFP, "
");
            fprintf(DFP, "%s", contact_node_i->phoneNumber);
            for (j=0; j<(50-strlen(contact_node_i->phoneNumber)); j++)
fprintf(DFP, " ");
            fprintf(DFP, "\n");
```

```
            /* Next Iteration */
            contact_node_i = contact_node_i->NEXT;

        }
    }

    /* Close FILE */
    fclose(DFP); return;

}

/* write_binaryfile - writes the current linked-list nodes into a binary file
*/
void write_binaryfile(char filename[50]) {

    /* Refer to contactBinData.txt */
    FILE *DFP = fopen(filename,"wb");

    /* Start iterating over lists */
    struct CONTACT_NODE *contact_node_i; int i;
    for (i=0;i<26;i++) {
          contact_node_i = LETTER_HEAD_CONTACT_NODES[i];
        while (contact_node_i != NULL) {
           fwrite(contact_node_i, sizeof(struct CONTACT_NODE), 1, DFP);
             contact_node_i = contact_node_i->NEXT;
          }
    }

    /* Close FILE */
    fclose(DFP); return;

}

/* print_binaryfile - retrieves the data from the binary file and displays it
into a table */
void print_binaryfile(char filename[50]) {

    /* Refer to contactBinData.txt */
    FILE *DFP = fopen(filename,"rb");

    /* Start iterating over file contents */
    struct CONTACT_NODE *contact_node_i= (struct CONTACT_NODE*)
malloc(sizeof(struct CONTACT_NODE));
    while (fread(contact_node_i, sizeof(struct CONTACT_NODE), 1, DFP) == 1) {
         printf("%s : %s\n", contact_node_i->name,
contact_node_i->phoneNumber);
    } printf("\n");

    /* Close FILE */
```

```
    fclose(DFP); return;

}

/* show - Iterates over each contact of the linked-lists in
LETTER_HEAD_CONTACT_NODES to print their information */
void show() {

    /* Start iterating over LETTER_HEAD_CONTACT_NODES */
    struct CONTACT_NODE *contact_node_i; int i;
    for (i=0;i<26;i++) {

        /* Loop over linked-lists */
        contact_node_i = LETTER_HEAD_CONTACT_NODES[i];
        while (contact_node_i != NULL) {
            printf("%s : %s\n", contact_node_i->name,
contact_node_i->phoneNumber);
            contact_node_i = contact_node_i->NEXT;
        }

    /* Exit function */
    } printf("\n"); return;

}

/* show_letter - Identifies the index of LETTER_HEAD_CONTACT_NODES associated
with the alphabet letter and only iterates over contacts
*                associated with that index/letter */
void show_letter(char letter) {

    /* Assign the iterative contact to the appropriate index of
LETTER_HEAD_CONTACT_NODES, based on the letter */
    struct CONTACT_NODE *contact_node_i =
LETTER_HEAD_CONTACT_NODES[letter-65];

    /* Start iterating over lists */
    while (contact_node_i != NULL) {
        printf("%s : %s\n", contact_node_i->name,
contact_node_i->phoneNumber);
        contact_node_i = contact_node_i->NEXT;

    /* Exit Function */
    } printf("\n"); return;

}


void * autosaveToBin(void * filename){
    struct CONTACT_NODE *contact_node_i; int i;
```

```
    for (;;) {
        pthread_mutex_lock(&autosave_mutex);
        FILE * DFP = fopen( (char *) filename, "wb");
        for (i=0;i<26;i++) {
           contact_node_i = LETTER_HEAD_CONTACT_NODES[i];
             while (contact_node_i != NULL) {
                fwrite(contact_node_i, sizeof(struct CONTACT_NODE), 1, DFP);
                  contact_node_i = contact_node_i->NEXT;
           }
        }
        fclose(DFP);
        pthread_mutex_unlock(&autosave_mutex);
        sleep(5);
    }
}
```
