```c
/*
Kendall Ronzano
Long Assignment#2
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define STRSZ 50
#define FILESZ 50

struct agent {
    long ID;
    char lastname[STRSZ];
    char first[STRSZ];
    int year;
    char term;
    int rank1;
    int rank2;
    struct agent *next;
    char key;

};

typedef struct list{
    struct agent *head;
    struct agent *tail;
    int size;
}list;

//Prototypes
struct agent *make_agent(long ID, char last[STRSZ], char first[STRSZ], int year,
 char term, int rank1, int rank2);
void printDatabase(list *database, FILE *fptr);
void printDatabaseSave(list *database, FILE *fptr);
void sortByIDA(list *database);
void sortByIDD(list *database);
void sortByLastA(list *database);
void sortByLastD(list *database);
void sortByFirstA(list *database);
void sortByFirstD(list *database);
void sortByTermA(list *database);
void sortByTermD(list *database);
void sortByRank1A(list *database);
void sortByRank1D(list *database);
void sortByRank2A(list *database);
void sortByRank2D(list *database);
struct agent * SearchLastname(char last[STRSZ], list *database);
struct agent * SearchFirst(char first[STRSZ], list *database);
void printAgent(struct agent * agent);
void deleteAgent(list *database);
void addAgent(list *database);
void clear();
int compareTerm(int year1, int year2, char term1, char term2);
void encryptWord(char *word, char key);
void decryptWord(char *word, char key);
void readCodes(list *database);

//Def main function
int main(void)
{
    char input;

    input = 'i';
    list database;
    database.head = NULL;
    database.tail = NULL;
    database.size = 0;
```

```
while (input != 'q' && input != 'Q'){

    printf("Please enter task: \nF=> file\nP=> print\nN=> new\n?=> search\nD=>
     delete\nS=> save\nQ=> quit\n");
    scanf("%c", &input);

//switch prompting task selection
switch(input){
    case 'F':
    case 'f':
    //ask the user for the filename(s) containing the information
        printf("Please enter the file name:");
        char filename[FILESZ];
        scanf("%s", filename);

        FILE *fptr;
        fptr = fopen(filename, "r");

        if (fptr == NULL){
            printf("Error, could not read file\n");
            return(0); }

        long ID;
        char lastname[STRSZ];
        char first[STRSZ];
        int year;
        char term;
        int rank1;
        int rank2;

        struct agent *last = database.head = NULL;
        struct agent *current;

        //get info from file and put into list
        while(fscanf(fptr, "%09lu %s %s %d%c %d %d", &ID, lastname, first,
         &year, &term, &rank1, &rank2) == 7){
            current = make_agent(ID, lastname, first, year, term, rank1, rank2);
            if(database.head == NULL){
                last = database.head = current;
            } else {
                last = last->next = current;
            }
            database.size++;
        }

        database.tail = last;
        database.tail->next = NULL;

        fclose(fptr);
        //decrypt file
        readCodes(&database);
        current = database.head;
        while(current != NULL){

            decryptWord(current->first, current->key);
            decryptWord(current->lastname, current->key);

            current= current->next;
        }
        break;

    case 'P':
    case 'p':
    //print the contents of the database, sorted by any of the items in the list
        if (database.size == 0) {
        printf("Please make a database before trying to store it\n");
        break;}
        clear();
```

```
char p_input;
printf("Please enter sort by:\nI=> ID\nL=> Last Name\nF=> First
 Name\nT=> Term\nR=> Ranking1\nK=> Ranking2\n");
scanf("%c", &p_input);

// prompt for which sort by
switch(p_input){
    case 'I':
    case 'i':
    //SORT BY ID
        ;
        char s_input;
        clear();
        printf("Please select sort order:\nA=> Ascending\nD=>
         Descending\n");
        scanf("%c", &s_input);
        //ascending vs descending
        switch(s_input){
            case 'A':
            case 'a':
                ;
                sortByIDA(&database);
                printDatabase(&database, stdout);
                break;
            case 'D':
            case 'd':
                sortByIDD(&database);
                printDatabase(&database, stdout);
                break;
            default:
                printf("Invalid input.\n");
                break;
            }
        break;
    case 'L':
    case 'l':
    //SORT BY LAST NAME
        ;
        char l_input;
        clear();
        printf("Please select sort order:\nA=> Ascending\nD=>
         Descending\n");
        scanf("%c", &l_input);
        switch(l_input){
            case 'A':
            case 'a':
                ;
                sortByLastA(&database);
                printDatabase(&database, stdout);
                break;
            case 'D':
            case 'd':
                sortByLastD(&database);
                printDatabase(&database, stdout);
                break;
            default:
                printf("Invalid input.\n");
                break;
            }
        break;
    case 'F':
    case 'f':
    //SORT BY FIRST NAME
        ;
        char f_input;
        clear();
        printf("Please select sort order:\nA=> Ascending\nD=>
         Descending\n");
        scanf("%c", &f_input);
```

```
        switch(f_input){
            case 'A':
            case 'a':
                ;
                sortByFirstA(&database);
                printDatabase(&database, stdout);
                break;
            case 'D':
            case 'd':
                sortByFirstD(&database);
                printDatabase(&database, stdout);
                break;
            default:
                printf("Invalid input.\n");
                break;
            }
        break;
    case 'T':
    case 't':
    //SORT BY TERM => year then char
        ;
        char t_input;
        clear();
        printf("Please select sort order:\nA=> Ascending\nD=>
         Descending\n");
        scanf("%c", &t_input);
        switch(t_input){
            case 'A':
            case 'a':
            ;
                sortByTermA(&database);
                printDatabase(&database, stdout);
                break;
            case 'D':
            case 'd':
                sortByTermD(&database);
                printDatabase(&database, stdout);
                break;
            default:
                printf("Invalid input.\n");
                break;
            }
        break;
    case 'R':
    case 'r':
    //SORT BY RANKING1
        ;
        char r_input;
        clear();
        printf("Please select sort order:\nA=> Ascending\nD=>
         Descending\n");
        scanf("%c", &r_input);
        switch(r_input){
            case 'A':
            case 'a':
            ;
                sortByRank1A(&database);
                printDatabase(&database, stdout);
                break;
            case 'D':
            case 'd':
                sortByRank1D(&database);
                printDatabase(&database, stdout);
                break;
            default:
                printf("Invalid input.\n");
                break;
            }
        break;
```

```
            case 'K':
            case 'k':
            //SORT BY RANKING2
                ;
                char k_input;
                clear();
                printf("Please select sort order:\nA=> Ascending\nD=>
                 Descending\n");
                scanf("%c", &k_input);
                switch(k_input){
                    case 'A':
                    case 'a':
                    ;
                        sortByRank2A(&database);
                        printDatabase(&database, stdout);
                        break;
                    case 'D':
                    case 'd':
                        sortByRank2D(&database);
                        printDatabase(&database, stdout);
                        break;
                    default:
                        printf("Invalid input.\n");
                        break;
                    }
                break;
            }
        break;
    case 'N':
    case 'n':
    //add a new candidate to the list
        ;
        if (database.size == 0) {
        printf("Please make a database before trying to store it\n");
        break;}
        addAgent(&database);
        break;
    case '?':
    //search for a particular candidate's name and display their information
        ;
        if (database.size == 0) {
        printf("Please make a database before trying to store it\n");
        break;}

        int x = -1;
        struct agent * found;
        while (x != 0 && x != 1){
            printf("Please enter 0 for first name/ 1 for last:\n");
            scanf("%d", &x);
            }
        if (x == 0){
            printf("Please enter name search:\n");
            scanf("%s", first);
            found = SearchFirst(first, &database);
            if (found != NULL)
                printAgent(found);
            else
                printf("Did not find agent with last name %s.\n", first);
            }
        else{
            printf("Please enter name search:\n");
            scanf("%s", lastname);
            found = SearchLastname(lastname, &database);
            if (found != NULL)
                printAgent(found);
            else
                printf("Did not find agent with last name %s.\n", lastname);
            }
        break;
```

```
        case 'D':
        case 'd':
        //remove a candidate from the database (requires searching)
            if (database.size == 0) {
            printf("Please make a database before trying to store it\n");
            break;}
            deleteAgent(&database);
            break;
        case 'S':
        case 's':
        //save the new database to file(s) that can be later used in the program
            ;
            if (database.size == 0) {
            printf("Please make a database before trying to store it\n");
            break;}
            printf("Please enter a new file name:");
            char newfile[FILESZ];
            scanf("%s", newfile);

            FILE *newfptr;
            newfptr = fopen(newfile, "w");

            if (newfptr == NULL){
                printf("Error, could not write file, will print to screen\n");
                newfptr = stdout;
                }
            else {
                printf("Save Successful.\n");
                }
            current = database.head;
            //encrypt new saved file
            while(current != NULL){

                    printf("%s %s\n", current->first, current->lastname);
                encryptWord(current->first, current->key);
                encryptWord(current->lastname, current->key);
                printf("%s %s\n", current->first, current->lastname);

                current= current->next;
            }
            printDatabase(&database, newfptr);
            fclose(newfptr);
            break;

        case 'Q':
        case 'q':
            //Quit / Exit program
            printf("Peace Out Homies\n");
            break;

        default:
            printf("Invalid input.\n");
            break;

    }
    clear();
    }
    return(0);
}

//make a new agent in list
struct agent *make_agent(long ID, char last[STRSZ], char first[STRSZ], int year,
 char term, int rank1, int rank2){
    struct agent *new = malloc(sizeof(struct agent));
    new->ID = ID;
    strcpy(new->lastname, last);
    strcpy(new->first, first);
    new->year = year;
    new->term = term;
```

```
    new->rank1 = rank1;
    new->rank2 = rank2;
    new->next = NULL;
    new->key = 13;
    return new;
}

//functions to print database
void printDatabase(list *database, FILE *fptr) {
    struct agent *curr;
    printf("    ID    Last Name     1st    Term  R1   R2\n");
    for(curr = database->head; curr != NULL; curr = curr->next) {
        fprintf(fptr, "%09lu %s %s %02d%c %d %d\n", curr->ID,curr->lastname,curr-
         >first, curr->year, curr->term, curr->rank1, curr->rank2);
    }
}
void printDatabaseSave(list *database, FILE *fptr) {
    struct agent *curr;
    for(curr = database->head; curr != NULL; curr = curr->next) {
        fprintf(fptr, "%09lu %s %s %02d%c %d %d\n", curr->ID,curr->lastname,curr-
         >first, curr->year, curr->term, curr->rank1, curr->rank2);
    }
}

//sort functions A for ascending D for descending
void sortByIDA(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (current->ID > next->ID) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByIDD(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (current->ID < next->ID) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }
```

```
                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByLastA(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (strcmp(current->lastname, next->lastname) > 0) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByLastD(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (strcmp(current->lastname, next->lastname) < 0) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;
```

```
                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByFirstA(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (strcmp(current->first, next->first) > 0) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByFirstD(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (strcmp(current->first, next->first) < 0) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
```

```
            }
        }
    }
}
void sortByTermA(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (compareTerm(current->year, next->year, current->term, next->term) ==
             1) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }

            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByTermD(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (compareTerm(current->year, next->year, current->term, next->term) ==
             0) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }

            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
```

```
}
void sortByRank1A(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (current->rank1 > next->rank1) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByRank1D(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (current->rank1 < next->rank1) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByRank2A(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
```

```
        while(next != NULL) {
            if (current->rank2 > next->rank2) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}
void sortByRank2D(list *database) {
    int i;
    for(i = 0; i < database->size; i++) {
        struct agent* current = database->head;
        struct agent* next = current->next;
        struct agent* previous = NULL;
        while(next != NULL) {
            if (current->rank2 < next->rank2) {
                if (current == database->head){
                    database->head = next;
                } else  {
                    previous->next = next;
                }

                if(next == database->tail){
                    database->tail = current;
                }
                current->next = next->next;
                next->next = current;

                previous = next;
                next = current->next;
            }
            else {
                previous = current;
                current = next;
                next = next->next;
            }
        }
    }
}

//search functions
struct agent * SearchLastname(char lastname[STRSZ], list *database){
    struct agent* current = database->head;
    while(current != NULL){
        if( strcmp(current -> lastname, lastname) == 0)
            return current;
        current = current -> next;
        }
        return NULL;
    }
struct agent * SearchFirst(char first[STRSZ], list *database){
    struct agent* current = database->head;
```

```
    while(current != NULL){
        if( strcmp(current -> first, first) == 1)
            return current;
        current = current -> next;
        }

        return NULL;
    }

void printAgent(struct agent * agent)
{
    printf("%09lu %s %s %02d%c %d %d\n", agent->ID,agent->lastname,agent->first,
     agent->year, agent->term, agent->rank1, agent->rank2);
}
void deleteAgent(list *database)
{
    int y = -1;
    while (y != 0 && y != 1){
            printf("Find Agent to Remove\nPlease enter 0 to search by first name/ 1
             to search by last name:\n");
            scanf("%d", &y);
    }
    struct agent * prev = NULL;
    struct agent * curr = database->head;
    if (y == 0){
        char firstname[STRSZ];
        printf("Please enter name search:\n");
        scanf("%s", firstname);
        int flag = 0;
        while(curr != NULL){
            if(strcmp(curr->first, firstname) == 0){
                if(curr == database->head){
                    database->head = curr->next;
                }
                if (curr == database->tail){
                    database->tail = prev;

                }
                if (prev != NULL)
                    prev->next = curr->next;
                printf("Deleted agent: ");
                printAgent(curr);
                free(curr);
                flag = 1;
                database->size--;
                curr = NULL;
                break;
            }
            prev = curr;
            curr = curr->next;
        }
        if(flag == 0)
            printf("Did not find agent with first name %s.\n", firstname);

        }
    else{
        char lastname[STRSZ];
        printf("Please enter name search:\n");
        scanf("%s", lastname);
        int flag = 0;
        while(curr != NULL){
            if(strcmp(curr->lastname, lastname) == 0){
                if(curr == database->head){
                    database->head = curr->next;

                }
                if (curr == database->tail){
                    database->tail = prev;
```

```
                }
                if (prev != NULL)
                    prev->next = curr->next;
                database->size--;
                printf("Deleted agent: ");
                printAgent(curr);
                free(curr);
                curr = NULL;
                flag = 1;

                    break;
            }

            prev = curr;
            curr = curr->next;
        }
        if(flag == 0)
            printf("Did not find agent with first name %s.\n", lastname);
        }
    }
void addAgent(list *database)
{
    long ID;
    char lastname[STRSZ];
    char first[STRSZ];
    int year;
    char term;
    int rank1;
    int rank2;
    char key = -1;
    printf("Please enter Agent ID:\n");
    scanf("%lu", &ID);
    clear();
    printf("Please enter Agent last name:\n");
    scanf("%s", lastname);
    printf("Please enter Agent first name:\n");
    scanf("%s", first);
    printf("Please enter Agent term:\n");
    scanf("%02d%c", &year, &term);
    printf("Please enter Agent ranking1:\n");
    scanf("%d", &rank1);
    printf("Please enter Agent ranking2:\n");
    scanf("%d", &rank2);


    database->tail->next = make_agent(ID, lastname, first, year, term, rank1,
     rank2);
    database->tail = database->tail->next;
    while(key < 97 || key >122){
        printf("Please enter an Agent key between a-z inclusive:\n");
        scanf("%c", &key);
    }

    database->tail->key = key - 96;
}

void clear() {
    char junk;
    do {
        scanf("%c", &junk);
    } while (junk != '\n');
}

//if 1 > 2 then return 1, else return 0
int compareTerm(int year1, int year2, char term1, char term2){
    if (year1 > year2)
        return 1;
    if (year1 < year2)
        return 0;
```

```
    //compare the chars
    int term_num_1, term_num_2;

    if (term1 == 'W' || term1 == 'w')
        term_num_1 = 1;
    else if (term1 == 'S' || term1 == 's')
        term_num_1 = 2;
    else if (term1 == 'X' || term1 == 'x')
        term_num_1 = 3;
    else
        term_num_1 =4;

    if (term2 == 'W' || term2 == 'w')
        term_num_2 = 1;
    else if (term2 == 'S' || term2 == 's')
        term_num_2 = 2;
    else if (term2 == 'X' || term2 == 'x')
        term_num_2 = 3;
    else
        term_num_2 =4;

    if (term1 > term2)
        return 1;
    else
        return 0;

}

void decryptWord(char *word, char key) {
    int i;
    for(i = 0; i < strlen(word); i++){
        char curr = word[i]-(int)key;
        if(curr < 'a')
            word[i] = curr+26;
        else word[i] = curr;
    }

}

void readCodes(list *database){
    FILE *codesptr;
    long ID;
    char key;
    codesptr = fopen("codes.txt", "r");

    if (codesptr == NULL){
            printf("Error, could not read file\n");
            return;
        }

    while(fscanf(codesptr, "%lu %c", &ID, &key) == 2){
        struct agent* current = database->head;
        while(current != NULL){
            if( current->ID == ID){
                current->key = (current->lastname[0] - key);
                if (current->key < 0)
                    current->key += 26;
            }
            current = current -> next;
        }

    }

    fclose(codesptr);

}

void encryptWord(char *word, char key) {
```

```
        int i;
    for(i = 0; i < strlen(word); i++){
        word[i]-= 26-(int)key;
        if(word[i] > 'z')
            word[i] -= 26;
        else if (word[i] < 'a')
            word[i] += 26;
    }
}
```