Adam Orrick

10/3/18

Project 3

**Source Interface.c**

#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <unistd.h>

#define BUFFLEN 2000

int main(int argc, char \*argv[], char \*envp[]){

int err; //checks for errors

int id; //process id

int i; //for while loop to convert input to lowercase

int toDB[2]; //for pipe to child

int fromDB[2]; //for pipe from child

int status;

char param0[10]; //first param for db

char param1[10]; //second param for db

char buffer[BUFFLEN]; //buffer for piping

err = pipe(toDB); //create the first pipe

if(err == -1){ //check if there is an error on first pipe

printf("Error on first pipe: %d\n", errno);

exit(1);

}

err = pipe(fromDB); //create the second pipe

if(err == -1){ //check if there is an error on the second pipe

printf("Error on second pipe: %d\n", errno);

exit(2);

}

id = fork(); //create the child

if(id == 0){ //child

close(toDB[1]); //close read pipe to DB

close(fromDB[0]); //close write pipe from DB

sprintf(param0, "%d", fromDB[1]); //send write stream as param0

sprintf(param1, "%d", toDB[0]); //send read stream as param1

err = execl("./db", param0, param1, (char \*)NULL); //execute db

if(err == -1){ //check to see if error executing db

printf("Error executing db: %d\n", errno);

exit(3);

}

}

else if(id > 0){ //parent

char input[100];

close(toDB[0]); //close read pipe to DB.

close(fromDB[1]); //close write pipe from DB

do{

//char buffer[BUFFLEN]; //buffer for piping

// convert input to lower case

scanf("%s",input);

i = 0;

while(input[i]){

input[i] = tolower(input[i]);

i++;

}

err = write(toDB[1], input, strlen(input)+1); //send input to DB

if(err == -1){

printf("Error while writing in interface: %d\n", errno);

}

err = read(fromDB[0], buffer, BUFFLEN); //recieve response from DB

if(err == -1){

printf("Error while reading in interface: %d\n", errno);

}

printf("Response: %s\n", buffer);

//if the user inputs "exit"

if(!strcmp("exit", input)){

printf("Interface: child process %d completed\n", getpid());

waitpid(-1,&status,0);

printf("Interface: child process exit status = %d\n", (status >> 8) & 0xFF);

exit(0);

}

printf("\nInput command (account,id | add,id,cn,date,amount | delete,id,cn | list | exit):\n");

}while(1);

exit(0);

}

}

**Source db.h**

#define BUFFLEN 2000

//struct to hold the records in the file

typedef struct Record{

long int id; //identity of the record

int checkNumber; //id of the check

float amount; //amount of the check

char date[9]; //date on the check

}Record;

//Get the number of records in the file

int getRecordNum(char filename[]);

//assign the records in the file to the record struct

void getRecordData(Record records[], char filename[]);

//return the list in string format to the the interface

void printList(Record records[], int recordNum, char response[]);

//determine what command was entered by the user

int readCommand(char buffer[]);

//if account is command then return the total amount for that account to the interface

void getAccountAmount(Record records[], int recordNum, char buffer[], char filename[], char response[]);

//add a record the the Record struct and append the new record to the file

void addRecord(Record records[], int \*recordNum, char buffer[], char filename[], char response[]);

//delete a record from the record struct and delete the record from the file.

void deleteRecord(Record records[], int \*recordNum, char buffer[], char filename[], char response[]);

**Source db.c**

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <stddef.h>

#include <string.h>

#include <ctype.h>

#include "db.h"

int main(int argc, char \*argv[], char \*envp[]){

if(argc == 0){ // if no arguments for db

printf("No arguments for db.");

}

else{

int recordNum; //current number records

int err; //check for error

int writeFD = atoi(argv[0]); //initialize write pipe from param0

int readFD = atoi(argv[1]); //initialize read pipe from param1

int done = 0; //to loop the do while loop

int i; //for the for loop

char filename[] = "accountData.txt"; //name of the file

char response[BUFFLEN]; //string to send to interface

char buffer[BUFFLEN]; //recieve command from interface

Record records[100]; //max number of records is 100

recordNum = getRecordNum(filename); //retrieve the number of records in file

getRecordData(records, filename); //store the records in file to the record struct

for(i = 0; i < recordNum; i++){

printf("element[%d]:",i);

printf(" id = %ld,",records[i].id);

printf(" check Number: %d,",records[i].checkNumber);

printf(" date: %s,",records[i].date);

printf(" amount: %.2f\n",records[i].amount);

}

printf("\nInput command (account,id | add,id,cn,date,amount | delete,id,cn | list | exit):\n");

do{

err = read(readFD, buffer, BUFFLEN);

if(err == -1){

printf("Error reading from pipe in db: %d\n", errno);

exit(8);

}

switch(readCommand(buffer)){

case 1: sprintf(response, "db complete");

done = 1;

break;

case 2: printList(records, recordNum, response);

break;

case 3: getAccountAmount(records, recordNum, buffer, filename, response);

break;

case 4: addRecord(records, &recordNum, buffer, filename, response);

break;

case 5: deleteRecord(records, &recordNum, buffer, filename, response);

break;

default:

sprintf(response,"Incorrect input\n");

}

err = write(writeFD, response, strlen(response)+1);

if (err == -1)

{

printf ("Error on write to pipe: %d\n", errno);

exit(9);

}

}while(!done);

}

exit(0);

}

int getRecordNum(char filename[]){

FILE \*fp; //file pointer

int count = 0;//count the number of new line characters in a file

char ch; //character to be compared to file pointer

fp = fopen(filename, "r");

if(fp == NULL){

printf("Could not open file %s", filename);

}

while((ch = fgetc(fp)) != EOF){

if(ch == '\n'){

count++;

}

}

fclose(fp);

return count;

}

void getRecordData(Record records[], char filename[]){

FILE \*fp; //file pointer

long int id; //id in file

int checkNumber, i, input; //check number in file

//i for the for loop

//input: for inputing into record structure

float amount; //amount in file

char date[9]; //date in file

fp = fopen(filename, "r"); //open for reading only

if(fp == NULL){

printf("Could not open file %s", filename);

}

for(i = 0; !feof(fp); i++){

fscanf(fp, "%ld %d %s %f", &id, &checkNumber, date, &amount);

records[i].id = id;

records[i].checkNumber = checkNumber;

records[i].amount = amount;

sprintf(records[i].date, "%s", date);

}

fclose(fp);

}

void printList(Record records[], int recordNum, char response[]){

char temp[100]; //holds one line of list

char temp2[BUFFLEN] = "\n"; //holds entire list

int i;

for(i = 0; i < recordNum; i++){

sprintf(temp,"\nelement[%d]: id = %ld, check Number: %d, date: %s, amount: %.2f", i, records[i].id, records[i].checkNumber, records[i].date, records[i].amount);

strcat(temp2, temp);

}

strcpy(response,temp2); //response = temp2

}

int readCommand(char buffer[]){

if(!strcmp(buffer,"exit")){

return 1;

}

else if(!strcmp(buffer,"list")){

return 2;

}

else if(!strncmp(buffer,"account",strlen("account"))){

return 3;

}

else if(!strncmp(buffer,"add",strlen("add"))){

return 4;

}

else if(!strncmp(buffer,"delete",strlen("delete"))){

return 5;

}

return 0;

}

void getAccountAmount(Record records[], int recordNum, char buffer[], char filename[],char response[]){

getRecordData(records, filename); //retrieve all date from acount

char \*id; //used to store char array representation of id

char token[2] = ",";

float amount = 0;

int i;

id = strtok(buffer,token);

id = strtok(NULL,token);

for(i = 0; i < recordNum; i++){

if(records[i].id == atoi(id)){

amount += records[i].amount;

}

}

sprintf(response, "Total for account %s: %.2f\n", id, amount);

}

void addRecord(Record records[], int \*recordNum, char buffer[], char filename[], char response[]){

long int id; //id to be added to file and record

int cn; //check number to be added to file and record

int rn = \*recordNum; //holds the value of the number records

char date[9]; //date to added to file and record

char token[2] = ","; //delimiter in command is ','

char \*command; //to split the command

float amount; //amount to be added to file and record

FILE \*fp; //file pointer

command = strtok(buffer,token);

command = strtok(NULL,token);

id = atol(command);

command = strtok(NULL,token);

cn = atoi(command);

command = strtok(NULL,token);

strcpy(date,command);

command = strtok(NULL,token);

amount = atof(command);

sprintf(response,"%ld, %i, %s, %.2f\n", id,cn,date,amount);

//update records structe

rn++;

records[rn-1].id = id;

records[rn-1].checkNumber = cn;

records[rn-1].amount = amount;

sprintf(records[rn-1].date, "%s", date);

\*recordNum = rn;

//update file

fp = fopen(filename, "a");

if(fp == NULL){

printf("Could not open file %s", filename);

}

fprintf(fp, "\r\n%ld %d %s %.2f", id, cn, date, amount);

fclose(fp);

sprintf(response, "added:\nelement[%d]: id = %ld, checkNumber: %d, date: %s, amount: %.2f", rn-1, id, cn, date, amount);

}

void deleteRecord(Record records[], int \*recordNum, char buffer[], char filename[], char response[]){

long int id; // store id from buffer

int cn; // store check number from buffer

int rn = \*recordNum; // total number of records

int i; // for looping

int flag = 0; // flag for updating records after delete

char token[2] = ","; // delimiter in buffer

char \*command; // to split the buffer

FILE \*fp; // file pointer

command = strtok(buffer,token);

command = strtok(NULL,token);

id = atol(command);

command = strtok(NULL,token);

cn = atoi(command);

//update record struct

rn--;

for(i = 0; i < rn; i++){

if(records[i].id == id && records[i].checkNumber == cn)

flag = 1;

if(flag)

records[i] = records[i+1]; // set the pointer of the record

// to be deleted to the next record in the record

}

\*recordNum = rn;

//update file

fp = fopen(filename, "w");

if(fp == NULL){

printf("Could not open file %s", filename);

}

for(i = 0; i < rn ; i++){

fprintf(fp, "%ld %d %s %f\r\n", records[i].id, records[i].checkNumber, records[i].date, records[i].amount);

}

fclose(fp);

sprintf(response, "Record deleted.");

}

**makefile:**

objects = interface db

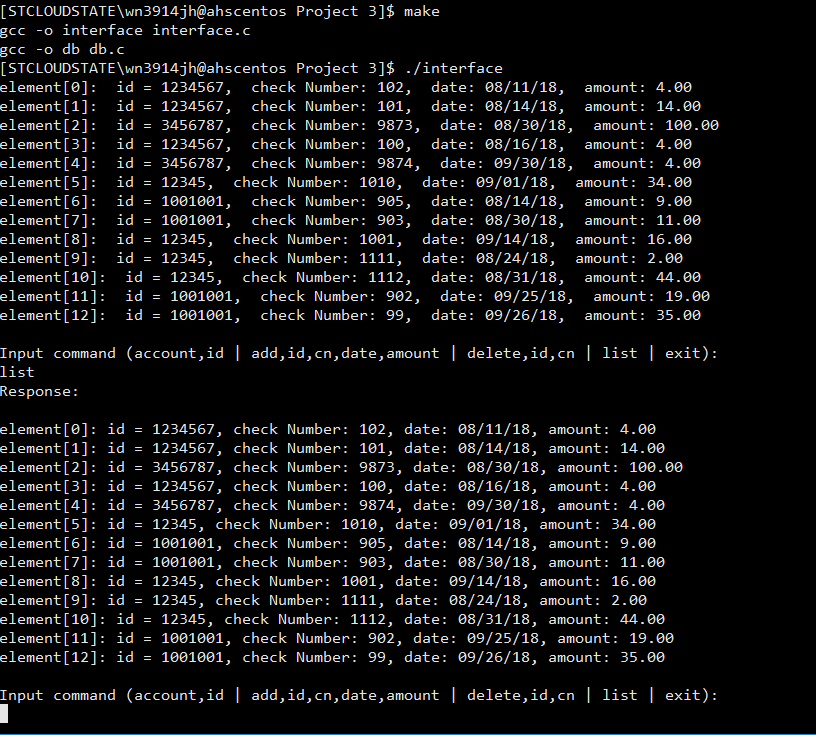
all: $(objects)

$(objects): %: %.c

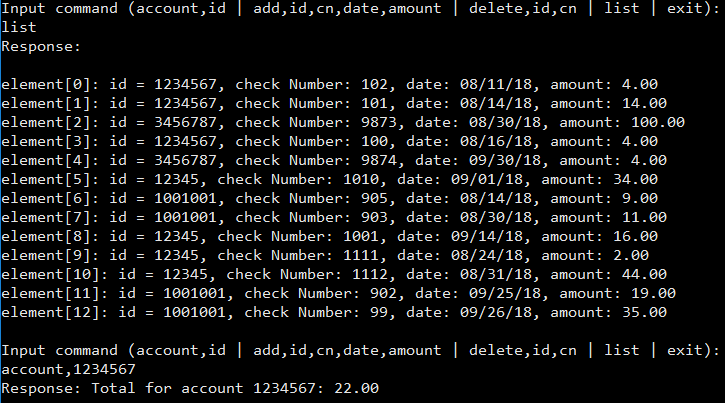
gcc -o $@ $<

**output:**

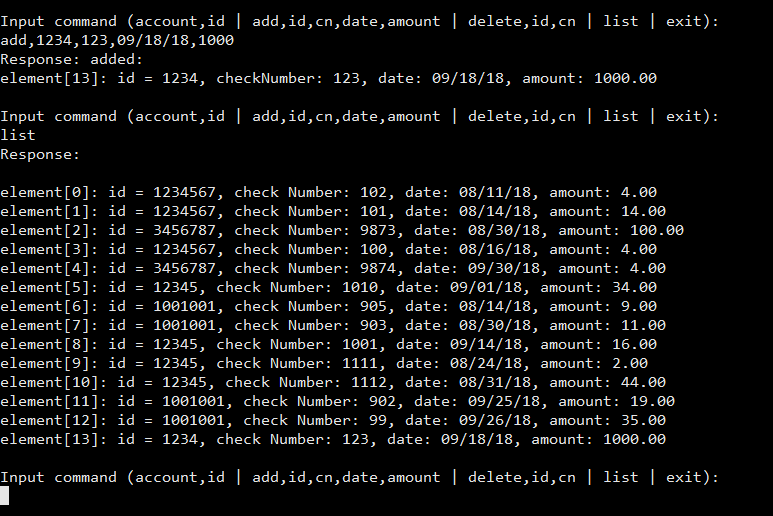
List Command:



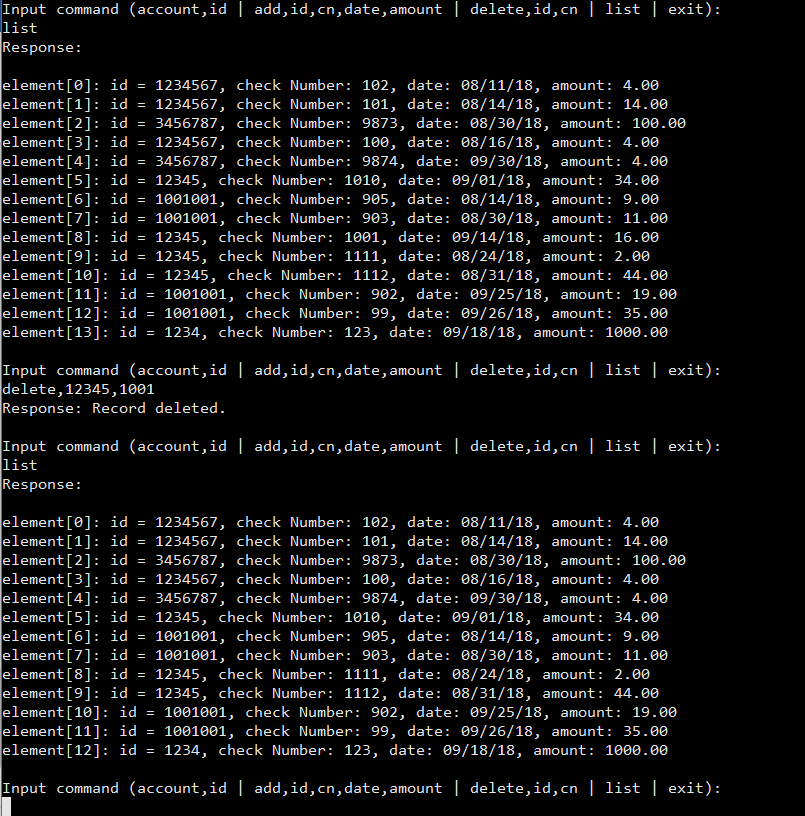
account command:



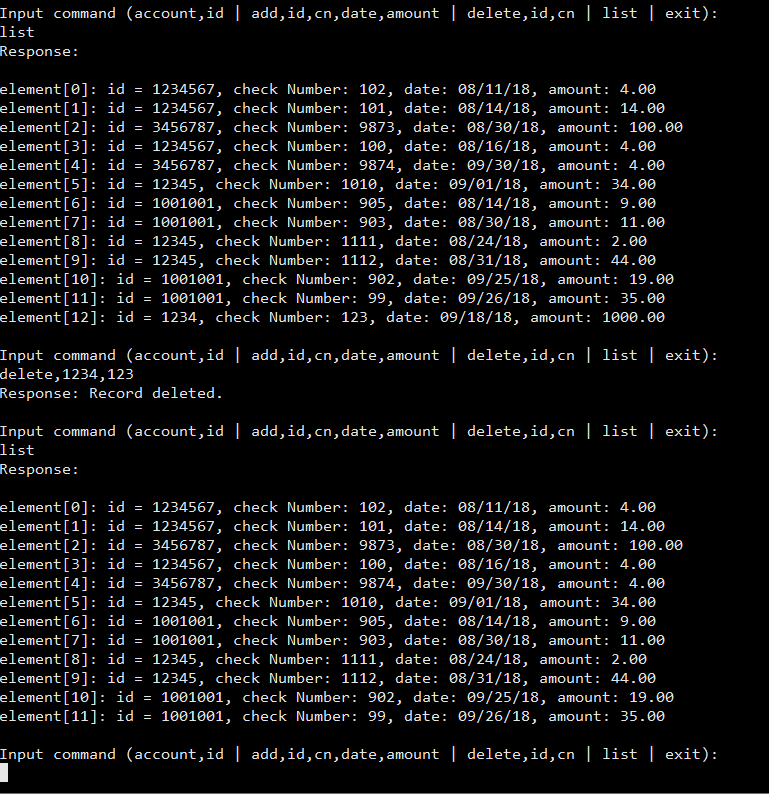
add command:



delete command:



delete last element:



exit command:

