SOFTWARE FOR A MOBILE PHONE

DOCUMENTATION

Written by lior israelov

Main.c

file.c

ALL HEADRS

PB.c

CL.c

ML.c

CL.h

io.h

error.h

PB.h

ML.h

list.h

list.c

error.c

io.c

func.h

The project is software for a mobile phone, it contains 3 main modules that work together to manage the contacts, calls and text messages in the program.

Explanation of the modules and the division over view

The project is divided to 8 different modules.

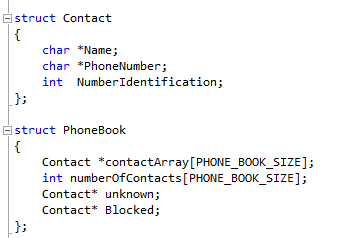
1. **Phone Book** – in this module the Phone Book structure and the Contact structure is defined and the functions that deal with those structures are implemented.
2. **Call Log** – in this module the Call Log structure and Call is defined and the functions that deal with those structures are implemented.
3. **Text Message** – in this module the Message Log structure, conversation structure and Text Message structure are defined and the functions that deal with those structures are implemented.
4. **File** – in this module all the read and write to files are performed.
5. **I/O** – this module contain the functions that print the menus and the data to the console.
6. **Error** – this module handle with errors that happens during the program – print massage, stop the operation, close file if needed and free memory if needed .
7. **Funcs (generics)** – this module contain generics functions that can be used in other modules, in addition it contain the date structure and the functions that deal with this structure.
8. **List** – in this module the Node structure is defined, the typedef to List is defined and all the functions that deal with linked lists are implemented.

* In the project headers there is a more explanation about each module, what it contains and how it built.

**THE PHONE BOOK MODULE**

The phone-book is sorted in a static array (of 26 sizes) of sorted dynamic arrays.

Each entry in the phone book keeps only one number.

The contacts in the array are sorted lexicography and by identification if there is more than one with the same name.

I decided to make two structures

1. Contact structure – contain name, number and identification
2. PhoneBook structure
3. Contact \* contactArray[PHONE\_BOOK\_SIZE]
4. Int numberOfContacts[PHONE\_BOOK\_SIZE]
5. contact \* Unknown
6. contact\* Blocked

In this implementation it easy to point to a Unknown or Blocked contact from each place in the program,

We keep only two memory blocks (one for each) instead to duplicate them.

One more reason I chose this implementation is that we keep for every array the number of contacts in the array, and we not need to count every time when we do some operation.

When I built the function I was thinking as OOP , meaning I looked at the structures as object and tried to think which functions will be needed ( like compere, print, search and more), I tried to divide the module by this functions.

And then I implemented the ask functions.

**Functions – there is explanation on functions int the project**

**// asked project function to implemented**

Contact\* AddNewContact (PhoneBook \*thePhoneBook);

Contact\* DeleteContact (PhoneBook \*thePB);

List FindContact (char\*name, PhoneBook \*ThePB, int(\*strComerator)(const char\*, const char\*));

**// needed operations in and outside the module**

Contact\* getNewContact ();

Contact\* insertContact (Contact \*oldArr, int numberOfContacts, Contact \*newContact);

Int getLexIndex (int FirstLetter);

Int isBlockedNumber (Contact \* cont);

Int isUnKnownName (Contact \* cont);

**// initializations functions**

PhoneBook \*initPB ();

Contact \* initUnKnown ();

Contact \* initBlocked ();

Contact\* setContact (char \* string, PhoneBook\* thePB);

Contact\* setNewArr (int size);

**// memory operations**

void free Contact (Contact\* cont);

void freeNumber (Contact \* toFree);

void freeName (Contact \* toFree);

Contact\* fixMemory (PhoneBook \*thePB, Contact \*toFix);

**// searching operations**

int serachFirst (PhoneBook \*thePB, Contact tempC);

Contact\* contactArrSerch (Contact \*contArr, int numOfCont, Contact \*theContact);

Contact\* searchByNumber (char \*number, PhoneBook \*thePB);

**// compere operations**

int comperIdentity (Contact\* CA, Contact\* CB);

int comperContacts (const Contact \*CA, const Contact \*CB);

int comperByName (const void\* contactA, const void\* contactB);

int comperByNumber (const void\* contactA, const void\* contactB);

int partialNameComperator (const char \*sourceName, const char \*toSerchPart);

int FullNameComperator(const char \*sourceName, const char \*toSerchPart);

**// printing operations**

void printPhoneBook(const PhoneBook \* thePhoneBook);

void printContactList(List ContactList, Contact \*prevContact, int index);

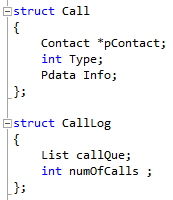
void pirntContactDetails(const Contact contact);

int listToDelete(Contact \*Arr, int numOfcontacts);

most of the functions are O(1), most of searching operation is O(NumberOfContacts) and because of that function that needs to search is O(NumberOfContacts).

**THE Call Log MODULE**

The calls are stored in dynamic FIFO queue, implemented with a linked list.

In addition to implement the Call Log we have the List module that contains the Node structure and all functions that deal with linked list.

I decided to implement the module in this way

1. Call structure – contain pointer to contact, type, time and duration.

Contact can point to – 1. Unknown (In the PB) .

2. Blocked (In the PB) .

3. contact in the phone book

4. regular number ( not recognize but not unknown also)

1. CallLog structure – CallQue list and the number of calls in the list.

* I wanted to add pointer to tail, but I request to add prev pointer to the list module which needed a big change.

I chose this implementation because in this implementation each pointer to call can be an element type in the list module.

So the call queue is built from nodes that each node contain pointer to specific call.

One more thing that we have is that we can save the number of calls, an important parameter that used a lot .

The time and duration filed is stored together in block (32 bits).

To implement this I used the data structure in the func module.

Explanation about how the block is work and how the date and duration saved is after the text message module.

* Dealing with Blocked number – I decided that if the number that given is NULL it mean the number is blocked and the contact that initialized is block contact from the phone book.
* Using characters for incoming, outgoing and missed call

Incoming - « (ascii code 174)

Outgoing - » (ascii code 175)

Missed call - ~ (ascii code 126)

**Functions – there is explanation on functions int the project**

**// asked project function to implemented**

void showCall(Call\* theCall);

void NewCall(CallLog \*theCallLog, Call\* newCall);

void DeleteCall(CallLog \* theCL);

**// needed operations**

Call \*getNewCall(char \*number, PhoneBook \*thePB, CallLog \* theCL );

**// list operation**

void insertCall(Call \*newCall, List callQue);

int isFull (int numOfCalls);

int isEmpty(int numOfCalls);

void deQue(CallLog \* theCL);

**// initializations functions**

CallLog \*initCallLog();

**// memory operations**

int freeCall(Call \* toFree, List callQue);

void reOrganizeCallsContact(CallLog \*theCL, PhoneBook \*thePB);

**// printing operations**

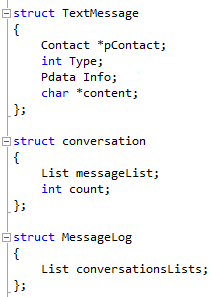
void ShowAllCalls(CallLog \* theCL);

Call \*\* CallListToDelete(List callQue, char \* nameToDEL);

**THE Message Log MODULE**

The messages are stored in linked list of messages linked lists.

In each of the messages lists we keep all messages sent or received by the same contact\number.

In addition to implement the Call Log we have the List module that contains the Node structure and all functions that deal with linked list

In my implementation I added the conversation structure.

The relation-ship between the structures :

Message Log - > list of conversations.

Conversation -> list of text Messages and the number of messages in the list .

Text Message -> pointer to contact, type, data block and the Txt itself.

The reason I use this way because I saw that in the call list holding the number of

calls are important.

here I hold as element type in Message log a conversation with single contact

and I have the number of messages in the conversation.

* Dealing with Blocked number – I decided that if the number that given is NULL it mean the number is blocked and the contact that initialized is block contact from the phone book.
* Using characters for incoming, outgoing.

Incoming - « (ascii code 174)

Outgoing - » (ascii code 175)

**Functions – there is explanation on functions int the project**

**// asked project function to implemented**

void NewMessage(MessageLog \* theML, TextMessage \* TheMessage);

void ShowRecentMessages(MessageLog \* theML);

void ShowConversation(MessageLog \* theML);

**// initializations functions**

MessageLog\* initMessageLog();

conversation\* initConversation();

**// memory operations**

int freeMessage(TextMessage \* toFree, List list);

void reOrganizeMessageContact(MessageLog \*theML, PhoneBook\* thePB);

**// printing operations**

void showAll(List list);

void showMessage(TextMessage\* message);

**// searching operations**

conversation\* findConversationList(List conversationLists, Contact \*pContact);

**// needed operations and list operations**

TextMessage \* getNewMessage(char \*number, PhoneBook \* thePB);

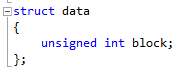
void insertMessage( List list, TextMessage \*newTxT);

int isTheList(List list, Contact \* pContact);

* Data structure – unsigned int information

Information contain as bits ( min, hour, day, month , duration).

The time and duration filed is stored together in block (32 bits) as follows:

Day – 5 least significant bits (0-4)

Month – next 4 bits (5-8)

Hour – next 6 bits (9-14)

Minutes – next 5 bits (15-19)

Duration – 12 left most bits. (20-31)

All functions that deal with bit operation are implemented in func module.

Basic operation can be modified as set date to block to his place as bit operation, set duration if we in

Call Log and get data with bit operation.

Bit operation meaning is to create mask and move the bits in the block left or right as need.

**THE FILE MODULE**

The file module operate all the reading and writing to and from files.

1. PhoneBook.txt

format - firstLetter(char)numberOfContacts(int)

nameLen(int) name(string) numberLen(int) number(string) id(int)

exp - "L1"

"13 Lior Israelov 10 0546916671 1"

2. call Log.txt

format - first number of calls(int)

calls -"numberLen(int) number(string) type(int) data(unsigend int)

exp - "1"

"10 0578182134 2 610095 "

3. Message Log.txt

format - count(int) numberLen(int) number(string)

list of messages - type(int) data(unsigend int) contentLen(int) content(string)

exp - "1 5 #3131 "

"1 30505 4 mes4 "

The other modules as Error, IO, and List not need an explanation, it modules that helps us to operate properly the program.