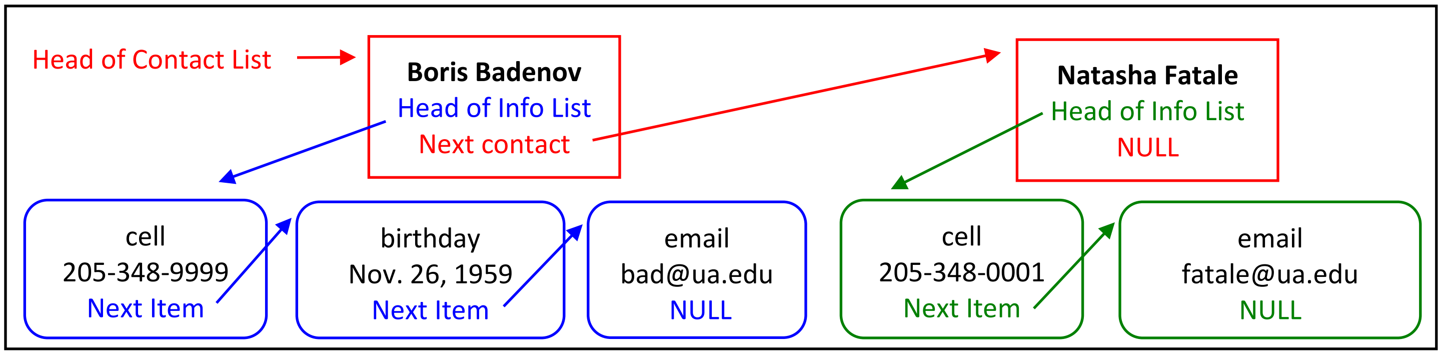
CS 100 Project Six – Fall 2017

Project Overview: The world is full of lists. You realized this as you were helping a friend add new contacts to their phone. You realize that your phone's contact list is nothing more than a set of linked lists. For this project, we will write our own version of "contacts" that uses the two data structures shown below:

```
typedef struct contact {
        char *name;
        Info *information;
        struct contact *next;
} Contact;
```

```
typedef struct info {
        char *name;
        char *value;
        struct info *next;
} Info;
```

The basic idea is that you have a list of contacts. Each contact has a name and a list of information regarding that contact (person). Each piece of information contains the information's name and value and a pointer to the next informational item for that contact. The example below shows two contacts, the first with three pieces of information and the second with two pieces of information. You have one linked list of Contacts (containing two contacts – Boris Badenov and Natasha Fatale). This linked list is shown in red. Each "contact" contains a linked list of information items, so you have two linked lists of Info (one for Boris Badenov and one for Natasha Fatale). These two linked lists are shown in blue (Boris) and green (Natasha).



In building this project, we establish the user interface (where the user interacts to create and modify and print and remove contacts) and then define a set of back-end functions that actually manage/manipulate the linked lists. Given this, we separate our program into three files as shown below:

- `main.c` –the user interface code. This prompts the user for an action and performs that action. All input is done using **fgets** so that you can capture spaces in contact names, informational item names, and the value associated with an informational item.
- `contact.h` –definitions of our structures and function prototypes for all the linked list functions.
- `contact.c` – implementation of the functions listed in `contact.h` that manipulate the linked lists. These functions are called by the `main.c` routine as needed to complete the various operations.

For this project, we will give you a working version of `main.c` and `contact.h` and a skeleton (empty) `contact.c` file. You only have to complete the nine functions that manipulate linked lists in `contact.c`.

What You Need To Do

- Create a directory `project6` on your machine. In that directory, you will have three files, `main.c` and `contact.h` and `contact.c`.
- The files `main.c` and `contact.h` have already been written, you must complete `contact.c`.
- The file `contact.c` contains the actual code for the nine functions that are used by the main routine. Right now there is no code (just comments) in each function. You must code these nine functions.
  - **addContact** – add a new contact, use "**add-at-front**" for this
  - **addInformation** – add a new piece of information for a contact, use "**add-at-end**" for this

- **printContact** – displays the contents of a contact (the contact name and all pieces of information that you have for the contact)
- **count** – print the count of contacts that you have
- **print** – prints all contacts. For each contact, print the name of the contact and all the information that you have for that contact
- **addContactOrdered** – add a new contact in alphabetical order (ordered add of a contact)
- **addInformationOrdered** – add a new piece of information in alphabetical order (ordered add for an information item)
- **removeContact** – remove the specified contact and all information for that contact
- **removeInformation** – remove the specified piece of information for that contact

- You can get the three files mentioned above from your course Blackboard web site. Obviously, the file **`contact.c`** is only a skeleton; you need to insert the actual code into each of the functions.
- Things you must remember when implementing these functions
  - When you try to add a contact and a contact with that name exists, generate an error message.
  - When you try to add information to a contact that does not exist, generate an error message
  - When you try to add information to a contact, if the contact already has that piece of information then update that information for the contact
  - When you try to remove information or a contact that does not exist, generate an error message.
- To compile this program, you use **`gcc -Wall -std=c99 main.c contact.c -o contact`**
- You can compile and run this program without doing any coding. It does nothing, but you can see how it works. The skeleton provided gives you a running program (that does nothing initially).
- We recommend building this project one function at a time. We will grade each of them individually, so you can get partial credit even if you don't complete them all. We recommend the following order when completing the functions. However, you can do them in any order that you want.
  - **addContact** and **addInformation** and **printContact**
  - **count** and **print**
  - **addContactOrdered** and **addInformationOrdered**
  - **removeContact** and **removeInformation**
- When testing the program, we will never combine **addContact** and **addContactOrdered** (or **addInformation** and **addInformationOrdered**) in the same test case. We will test both ordered lists and unordered lists, but never together
- When adding in order, use the standard **strcmp** to determine ordering of contacts or information items.
- Input for this program is entered from standard input. The user types in commands that call the various functions mentioned above. For example, to add a contact, the user types **addContact** at the prompt and then gives a name for the contact.
- You can also redirect input from a file (that the program believes it is reading standard input) using **`./a.out < testData`** You must submit one sample data file (named **testData**) that contains a set of user input for the program. A sample data file is shown at the right.

```
addContact
Boris Badenov
addInformation
Boris Badenov
cell
205-348-9999
addInformation
Boris Badenov
email
bad@ua.edu
printContact
Boris Badenov
addContact
Natasha Fatale
addInformation
Natasha Fatale
cell
205-348-0001
count
print
quit
```

__**When you are finished and ready to submit your project:**__

- Make sure your **`project6`** directory has your **contact.c** file and a **testData** file in it. When testing, we will use the original **contact.h** file and a modified **main.c** file that does not have as many print statements associated with it.
- Bundle your **`project6`** directory into a single (compressed) zip file.
- Once you have a compressed zip file that contains your **`project6`** code, submit that file to Blackboard.

Project 6 is due at 5:00pm on __Friday, December 1__. Late projects are not accepted.