CS 262
Project 3
Campus Food Delivery

Due April 29, 2018

A new food delivery service is being considered for the students on campus. Students will be able to call in an order from any of the campus eateries, and have the items delivered (for a fee, of course). As part of this service, the delivery personnel will use an app that can help them track the orders, so that they can first pick up the items and then deliver them to the appropriate location. You are to write a prototype program to model the functionality of the app.

The program

The program will be implemented using linked lists. It will allow the user to create a number of delivery orders and then place them in a delivery list. The program will begin by asking the user if he or she wishes to create a new delivery order. If the user answers yes, the program will ask for the location to deliver the order, and name of the eatery from which to retrieve the items. Then the program will ask for food items from that eatery. A blank response will tell the program that no further items from that eatery will be delivered. After all the food items for the order have been entered, the order will be placed in a delivery list. The user will then be prompted to see if he or she would like to enter a new order. When the user has finished entering orders, the orders in the delivery list will be displayed in the reverse order from which they were entered. Running the program may look like:

```
zeus-1$ proj3
New delivery order? (y/n) y

Delivery Address for new delivery order:  Engineering Building 5335
Restaurant from which to pick up food: Panda Express
food item: Fried Rice
food item: Orange Chicken
food item: Hunan Beef
food item:

New delivery order? (y/n) x
Invalid Input. Try again
New delivery order? (y/n) y


Delivery Address for new delivery order:  Lecture Hall 2
Restaurant from which to pick up food: Red Hot & Blue
food item: Pulled Pork Sandwich
food item: Potato Salad
food item:

New delivery order? (y/n) y


Delivery Address for new delivery order:  Merten 1200
Restaurant from which to pick up food: Dunkin' Donuts
food item: Bacon, Egg & Cheese on Plain Bagel
food item: Glazed Donut
food item: Large Unsweetened Iced Tea
food item:

new delivery order? (y/n) n

List of Deliveries:

Delivery order from Dunkin' Donuts has 3 food item(s)
    Bacon, Egg & Cheese on Plain Bagel
    Glazed Donut
    Large Unsweetened Iced Tea
Deliver to: Merten 1200

Delivery order from Red Hot & Blue has 2 food item(s)
    Potato Salad
    Pulled Pork Sandwich
Deliver to: Lecture Hall 2

Delivery order from Panda Express has 3 food item(s)
    Fried Rice
    Hunan Beef
    Orange Chicken
```

```
Deliver to: Engineering Building 5335

zeus-1$
```

The program will represent the food items in each delivery order as a linked list of nodes, one node for each food item. This linked list will be accessed by an itemList structure. When a delivery order has been entered, the itemList containing the food items will placed into another linked list -- a list of order nodes.

Structures

The list of food items will be represented by a struct:

```
typedef struct _itemList
{
   node *head;  // Pointer to first food item for the order (alphabetical)
   int count;   // Number of food items in the order
} itemList;
```

where `node` is defined as

```
typedef struct _node
{
   char *data;  // Food Item Name
   struct _node *next;
} node;
```

You will provide the following functions to operate on itemLists:

`itemList *createItem()`. Creates (using `dmalloc()` -- see below) an instance of an itemList struct, initializes its fields and returns a pointer to the newly allocated struct.

`int insert(char *str, itemList *s)`. Places a new string in the itemList by inserting a new node into its linked list and increments `count`. The food items will be inserted into the itemList in alphabetical order (use `strcmpi()` below for this). Duplicate strings (ignoring case) will not be allowed in the itemList. Returns 0 if the insertion was successful, and 1 otherwise.

`void printItems(itemList *s)`. Displays the elements of the itemList with each string on a new line.

The code for these functions will be placed in a file `itemList_<username>_<labsection>.c`. The `struct` definitions above as well as function prototypes for these functions will be put in a header file `itemList_<username>_<labsection>.h` which will be included using `#include "itemList_<username>_<labsection>.h"` (note the double quotes in place of angle brackets) in `itemList_<username>_<labsection>.c` and any other source files where they are needed.

You will also want to add some more functions to `itemList_<username>_<labsection>.c` to make programming easier. You may find the following two functions useful:

```
/* compares strings for alphabetical ordering */
int strcmpi(char *s, char *t)
{
   while (*s && tolower(*s) == tolower(*t))
   {
      s++;
      t++;
   }
   return tolower(*s) - tolower(*t);
}

/* allocates memory with a check for successful allocation */
void *dmalloc(size_t size)
{
   void *p = malloc(size);
   if (!p)
   {
      printf("memory allocation failed\n");
      exit(1);
   }
   return p;
}
```

You must always check the pointer returned by `malloc()` for successful memory allocation. You may use the `dmalloc()` function above in place of `malloc()` to ensure that this is always done. You may also find it convenient to write a function `char *stringcopy(char *s)` which creates (with `dmalloc()`) a copy of the string parameter.

Flow of control

Your program will ask the user if he or she wants to create a new delivery order and accept 'y' or 'n' for a response. If the user enters anything else the program will continue asking until 'y' or 'n' is entered. If the user answers yes the program will create an order node (see below), and prompt the user for the delivery location and restaurant name. The program will then prompt for and accept food items until the user enters an empty string. The list of food items is represented by a linked list of strings and each time the user enters a food item it is added to the delivery order's itemList. When all the food items have all been entered the itemList is added to the order node (if not previously added). This order node is then added to the head of the delivery list. This is repeated until the user gets bored.

The delivery list is another linked list, this one a list order node. It uses nodes of the type:

```
typedef struct _ordernode
{
   itemList *data;
   char *deliverTo;
   char *restaurant;
   struct _ordernode *next;
} ordernode;
```

Insert each new order as the first element in the delivery list. When the user has finished entering delivery orders, the final list of delivery orders will be displayed as in the above example run, with a count and list of each delivery's food items.

The code for the prompting and basic program flow shall be placed in a file named project3_<username>_<labsection>.c.

You will also create a Makefile to compile both source files into an executable.

Notes

Food items can use more than one word (as separated by white space) such as "Orange Chicken". This means that you must take input for food items using `fgets()` rather than with `scanf()`. `fgets()` and `scanf()` do not mix well in the same program. This means that when the user's input of a single character ('y', 'n') is read it should be read into a string using `fgets()`.

`free()` must be used as appropriate.

Testing and Submitting

Use valgrind to test your code for memory leaks and dangling pointers.

On zeus, create a directory named Project3_<username>_<labsection>. Copy your source files and Makefile to this directory. Change to this directory and create a typescript file showing that you are on zeus, listing your program to the screen, compiling it using the Makefile, and demonstrating a sample interactive run where the user orders at least three delivery orders. Change to the parent directory and create a tarfile of your project directory. Name this tarfile Project3_<username>_<labsection>.tar

Submit this tarfile to Blackboard no later than 11:59 p.m. on Sunday, April 29.