```
    *v2 = temp;
}
void bad_charPtrSwap(char *p1, char *p2)
{
    char *temp;
    temp = p1;
    p1 = p2;
    p2 = temp;
}
void good_charPtrSwap(char **p1, char **p2)
{
    char *temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```

Basic Linked List Operations

Consider the source file shown below (delimited by the label "linked_list.cpp"), which contains the type definitions and function prototypes. Complete the code as follows:

- Implement the body of each function.
- After implementing a function, debug it and test it by making calls from the main function, as shown by the examples in the main given further below. Make other testing examples.
- Compare your implementation with the code given in Sec.3, make any necessary fixes. Debug and test again with examples.
- Optional: when the functions have been tested, replace the hard-coded calls in the main function by writing code for a command-line interface. The interface would take a command symbol from the user as follows: i(InsertNewLast), d(DeleteLastNode), s(ListSearch), p(PrintList) or q(quit). In the case of i and s, the user should enter an integer data value. Debug and test the interface code separately. When it is working use it to make corresponding function calls and test the program.

```
/* ------------------linked_list.cpp-----------------*/
/* Basic Linked List Operations */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

typedef int DataItem;

struct Node {
    DataItem data;
    Node *next;
};

// Functions Prototypes
Node* ListSearch(DataItem value, Node *head);
void InsertNewLast(DataItem value, Node **L);
void DeleteLastNode(Node **L);
void PrintList(Node *head);

int main()
{
}

// This function searches for a node in the list whose data field
// equals to value. It returns a pointer to such node if it finds it.
// Otherwise, it returns NULL.
Node* ListSearch(DataItem value, Node *head)
{
}

// This function inserts a new node with data field set to
// value at the end of the list. L points to the pointer of the first
// node, or NULL if the list is empty.
void InsertNewLast(DataItem value, Node **L)
{
}

// This function deletes a node from the back end (tail) of the list.
// L points to the pointer of the first node, or NULL if the list is empty.
void DeleteLastNode(Node **L)
{
```

```
}

// This function displays the current elements of the list.
void PrintList(Node *head)
{
}

/* ------------------linked_list.cpp-----------------*/
```

The following are call examples that can be used in the main function.

```
int main ()
{
    Node *head;  // Declare list header as a pointer to Node
    Node *nodePtr;  // used as return value for ListSearch
    DataItem searchValue;

    head = NULL;  // set list to empty.

    // Some examples of function calls:
    // Printing and Inserting...
    PrintList(head);
    InsertNewLast(10, &head);
    PrintList(head);
    InsertNewLast(20, &head);
    PrintList(head);
    InsertNewLast(30, &head);
    PrintList(head);
    InsertNewLast(40, &head);
    PrintList(head);
    InsertNewLast(50, &head);
    PrintList(head);

    // Searching...
    searchValue = 20;
    nodePtr = ListSearch(searchValue, head);
    if (nodePtr != NULL)
    {
        cout << "Search value " << searchValue << " was FOUND" << endl;
    }
```

```
    else
    {
        cout << "Search value " << searchValue << " was NOT FOUND" << endl;
    }

    sarchValue = 5;
    nodePtr = ListSearch(searchValue, head);
    if (nodePtr != NULL)
    {
        cout << "Search value " << searchValue << " was FOUND\n";
    }
    else
    {
        cout << "Search value " << searchValue << " was NOT FOUND\n";
    }

    searchValue = 40;
    nodePtr = ListSearch ( searchValue, head );
    if (nodePtr != NULL)
    {
        cout << "Search value " << searchValue << " was FOUND\n";
    }
    else
    {
        cout << "Search value " << searchValue << " was NOT FOUND\n";
    }

    // Deleting and Printing...
    DeleteLastNode(&head);
    PrintList(head);
    DeleteLastNode(&head);
    PrintList(head);
    DeleteLastNode(&head);
    PrintList(head);
    DeleteLastNode(&head);
    PrintList(head);
    DeleteLastNode(&head);
    PrintList(head);
    system("pause");
}
```