CptS 122 – Data Structures
February 1, 2021

Your Name: _____ TA's Name: _____
ID#: _____ Section #: _____

SOLUTION: Take-Home: Quiz 2 (15 pts) – Dynamic Singly Linked Lists

Using Blackboard Learn https://learn.wsu.edu/webapps/login/ submit your quiz. You will submit your assignment in the ***lab*** Blackboard space. Under the "Content" link navigate to the "Quiz Submissions" folder and upload your solution to the appropriate "Quiz" space. You must upload your solution, through an attachment, as <your last name>_quiz2.pdf by the due date and time. The quiz is due before your Lab 2 starts next week February 1.

1. Write a function `insertPosN()` for a *singly-linked* list that has the following declaration and precondition:

```
int insertPosN (struct node **pHead, int n,
                char *newData);

Precondition: n > 0.
```

The function should allocate space for a node, initialize it with a copy of the data found in parameter `newData`, and attempt to insert it at position `n` in the list. The first node starts at position 1. If the node is successfully inserted into the list at position `n`, then the function must return 1. If memory could not be allocated for the node, then the function must return 0. If there are fewer nodes in the list than specified by `n-1`, then the function must insert the node at the end of the list and return -1. A node, `struct node`, is defined as follows:

```
struct node
{
     char data[100];
     struct node *pNext;
};
```

See next page for solution!

CptS 122 – Data Structures
February 1, 2021

Your Name: _____ TA's Name: _____
ID#: _____ Section #: _____

```
int insertPosN(struct node **pHead, int n, char *newData)
{
	int count = 0, success = 0; // success = 0 ==> couldn't allocate space
	struct node *pMem = NULL, *pCur = NULL, *pPrev = NULL;
	// try to allocate memory for a node
	pMem = (struct node *) malloc(sizeof(struct node));

	if (pMem != NULL) // were we able to allocate memory for a node?
	{
		// let's initialize the node; we are not given a value to copy
		// into the node,
		// so we'll initialize the next pointer to NULL and set the data to
		// the attempted position in the list to insert
		strcpy (pMem->data, newData);
		pMem->pNext = NULL;

		pCur = *pHead; // start at the front of the list
		// let's try to find position n in the list
		for (count = 1; (pCur != NULL) && (count < n); ++count)
		{
			pPrev = pCur;
			// find the nth position in the list - pCur will
			// point to current nth node
			pCur = pCur->pNext;
		}

		// insert before pCur, because pCur points at current nth node –
		// insert node between the node pointed
		// to be pPrev and node pointed to by pCur
		if (pPrev != NULL) // not inserting at the front
		{
			pPrev->pNext = pMem;

		}
		else // insert at front - the first node in the list
		{
			*pHead = pMem;
		}

		// are we inserting at position n? or at the end?
		if (count < n) // inserted at end, number of nodes < n
		{
			success = -1;
		}
		else // inserted at position n
		{
			success = 1;
		}
		pMem->pNext = pCur; // connect the rest of the list
	}
	return success;
}
```

Instructor: Andrew S. O'Fallon