CS13002 Programming and Data Structures, Spring 2005, Section 4/D

Laboratory Test III **Total points: 30** **April 12, 2005**

For students with odd PC numbers

Recall that the *selection sort* algorithm on an array iteratively finds out the maximum element in the initial part of the array and swaps the last element of the initial part with the maximum element. The following code snippet describes the selection sort algorithm.

```
for (i=n-1; i>=1; --i) {
    /* First find the maximum element of A[0],A[1],...,A[i] */
    /* Initialize maximum entry to be the leftmost one */
    maxidx = 0;
    max = A[0];
    /* Now search for a potentially bigger maximum */
    for (j=1; j<=i; ++j) {
        if (A[j] > max) { /* An element bigger that the current maximum is located */
            /* Adjust the maximum entry */
            maxidx = j;
            max = A[j];
        }
    }
    /* Swap A[i] with the maximum element */
    A[maxidx] = A[i]; /* Store the last element at index maxidx */
    A[i] = max; /* Store the maximum at the last index */
}
```

In this exercise, you are given an unsorted linked list (instead of an array) of integers. Your task is to sort the linked list using the selection sort algorithm.

- Write a function that creates a linked list of integers each randomly generated between 1 and 999.
- Write a function that takes as input a linked list of arbitrary size and sorts the list using the selection sort algorithm.
- Write a function that neatly prints (e.g. 20 integers per line) a linked list from beginning to end.
- Write a main function that calls the above three functions in order to do the following: first generate a linked list of 150 integers, print this list, then call the sorting function and finally print the sorted list.

For your convenience, we provide in the next page an outline of the program. Complete the details missing in this outline. Try to stick to the function prototypes suggested in the outline. Do not use *a priori* knowledge of the size of the list in the functions `printList` and `selSort`. Do not allocate/reallocate memory to nodes in the sorting function. Only adjust the pointers and/or the data of the nodes in the input list. Do not use any additional array.

Submit (along with your source code) a single run of your program.

```c
#include <stdio.h>
#include <time.h>

typedef struct _node {
   int data;
   struct _node *next;
} node;

node *createList ( int n )
/* This function creates a list of n random integers */
{
   node *head, *p;
   int i;

   /* Seed the random number generator by the system time */
   srand((unsigned int)time(NULL));

   /* Create the dummy node */
   head = (node *)malloc(sizeof(node));
   head -> data = 0;

   /* Initialize the running pointer p */
   p = head;

   /* Node creation loop */
   for (i=0; i<n; ++i) {

      /* Create the next node */
      p -> next = (node *)malloc(sizeof(node));
      p = p -> next;
      p -> data = 1 + rand() % 999;
   }

   /* Terminate the list */
   p -> next = NULL;

   return head;
}

void printList ( node *head )
/* This function should print the list elements from beginning to end. */
{
   /* Write this function */
}

void selSort ( node *head )
/* This function sorts a linked list by the selection sort algorithm. Assume that head
   points to a properly allocated linked list with a dummy node at the beginning. */
{
   /* Write this function */
}

int main ()
{
   node *head;

   head = createList(150);
   printf("The list before sorting:\n"); printList(head);
   selSort(head);
   printf("The list after sorting:\n"); printList(head);
}
```

CS13002 Programming and Data Structures, Spring 2005, Section 4/D

Laboratory Test III **Total points: 30** **April 12, 2005**

For students with even PC numbers

Recall that the *insertion sort* algorithm on an array iteratively considers array elements one-by-one starting from the beginning and inserts that element in the proper position in the part of the array before this element. The following code snippet describes the insertion sort algorithm.

```
for (i=1; i<n; ++i) { /* Consider A[i] */

   /* Search for the correct insertion location of A[i] */
   t = A[i]; /* Store A[i] in a temporary variable */
   j = 0; /* Initialize search location */

   while (t > A[j]) ++j; /* Skip smaller entries */
   /* Here j holds the desired insertion location */

   /* Shift forward the remaining entries each by one location */
   for (k=i-1; k>=j; -k) A[k+1] = A[k];

   /* Finally insert the old A[i] at the j-th location */
   A[j] = t;
}
```

In this exercise, you are given an unsorted linked list (instead of an array) of integers. Your task is to sort the linked list using the insertion sort algorithm.

- Write a function that creates a linked list of integers each randomly generated between 1 and 999.
- Write a function that takes as input a linked list of arbitrary size and sorts the list using the insertion sort algorithm.
- Write a function that neatly prints (e.g. 20 integers per line) a linked list from beginning to end.
- Write a main function that calls the above three functions in order to do the following: first generate a linked list of 150 integers, print this list, then call the sorting function and finally print the sorted list.

For your convenience, we provide in the next page an outline of the program. Complete the details missing in this outline. Try to stick to the function prototypes suggested in the outline. Do not use *a priori* knowledge of the size of the list in the functions `printList` and `insSort`. Do not allocate/reallocate memory to nodes in the sorting function. Only adjust the pointers and/or the data of the nodes in the input list. Do not use any additional array.

Submit (along with your source code) a single run of your program.

```c
#include <stdio.h>
#include <time.h>

typedef struct _node {
   int data;
   struct _node *next;
} node;

node *createList ( int n )
/* This function creates a list of n random integers */
{
   node *head, *p;
   int i;

   /* Seed the random number generator by the system time */
   srand((unsigned int)time(NULL));

   /* Create the dummy node */
   head = (node *)malloc(sizeof(node));
   head -> data = 0;

   /* Initialize the running pointer p */
   p = head;

   /* Node creation loop */
   for (i=0; i<n; ++i) {

      /* Create the next node */
      p -> next = (node *)malloc(sizeof(node));
      p = p -> next;
      p -> data = 1 + rand() % 999;
   }

   /* Terminate the list */
   p -> next = NULL;

   return head;
}

void printList ( node *head )
/* This function should print the list elements from beginning to end. */
{
   /* Write this function */
}

void insSort ( node *head )
/* This function sorts a linked list by the insertion sort algorithm. Assume that head
   points to a properly allocated linked list with a dummy node at the beginning. */
{
   /* Write this function */
}

int main ()
{
   node *head;

   head = createList(150);
   printf("The list before sorting:\n"); printList(head);
   insSort(head);
   printf("The list after sorting:\n"); printList(head);
}
```