COMP 2011 Quiz 3 - Spring 2018 - HKUST

Date: May 8, 2018
Time Allowed: 45 minutes
Instructions:

1. This is a open book, open notes examination. No electronic devices are allowed.
2. There are **5** questions on **5** pages (including this cover page and excluding the appendix).
3. Write your answers in the space provided.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, you are **NOT** allowed to define additional helper functions or structures, nor global variables unless otherwise stated. You also **cannot** use any library functions not mentioned in the questions.

Student Name	Mr. Solution
Student ID	
Email Address	
Lecture & Lab Section	L1 (Version B)

For T.A. Use Only (ref. B)

Problem	Score
1	/ 20
2	/ 20
3	/ 20
4	/ 20
5	/ 20
Total	/ 100

Problem 1 [20 points]

What is an advantage of a linked list compared to a dynamic array?

Answer:

A linked list can shrink and grow as needed.

Problem 2 [20 points]

Assume the linked list is defined as in the lecture notes, pages 55 to 64, what is the output of the following main function? (Note: the definition of the linked list is given in Appendix A for your reference.)

```
#include "ll_cnode.h"

int main() {

   ll_cnode* head = ll_create("abc");
   ll_delete(head, 'b');
   ll_insert(head, 'd', 1);
   ll_delete(head, 'a');
   ll_insert(head, 'e', 0);
   ll_insert(head, 'f', 1);
   ll_delete(head, 'c');
   ll_print(head);

   return 0;
}
```

Answer:

efd

Problem 3 [20 points]

Assume the linked list is defined as in the lecture notes, pages 55 to 64, explain what the following code does? Also, state and explain whether memory leak may occur. (Note: the definition of the linked list is given in Appendix A for your reference.)

```
#include "ll_cnode.h"

void ll_mystery(ll_cnode*& head, char c)
{
```

```
    ll_cnode* node = new ll_cnode;
    node->data = c;
    node->next = head;
    head = node;
}
```

Answer:

Insert a new node with data as c to the head(beginning) of the linked list. No, memory leak will not occurs.

Problem 4 [20 points]

Assume the linked list is defined as in the lecture notes, pages 55 to 64, implement a function `findN()` which returns a pointer to the N-th element of a given linked list pointed by head. If there are less than N elements in the linked list, returns `nullptr`. You may also assume N is a positive integer.

For example,

```
#include "ll_cnode.h"

ll_cnode* findN(ll_cnode* head, int N);

int main()
{
    ll_cnode* head = ll_create("abcdef");
    ll_print(head);
    ll_cnode* temp = findN(head, 3);
    if (temp != nullptr)
        cout << temp->data << endl;
    temp = findN(head, 5);
    if (temp != nullptr)
        cout << temp->data << endl;

    return 0;
}
```

will produce the following output:

```
abcedf
c
e
```

You should ensure that no memory leak will occur in your implementation and you cannot call any other functions. (Note: the definition of the linked list is given in Appendix A for your reference.)

```
ll_cnode* findN(ll_cnode* head, int N)
{
   // Answer here:









}
```

Answer:

```
#include "ll_cnode.h"

ll_cnode* findN(ll_cnode* head, int N)
{
    ll_cnode* p = head;
    for (int i=1; i<N; i++)
    {
        if (p == nullptr)
           return nullptr;
        p = p->next;
    }
    return p;
}
```

Problem 5 [20 points]

Assume the linked list is defined as in the lecture notes, pages 55 to 64, implement a function `arrayToLL()` to convert a dynamic array, `c`, of size `sizeArray` to a linked list. It copies all the elements in the dynamic array to a new linked list and returns the pointer to the head of the linked list. Note that the array, `c`, given is a character array, not a C String, and you may assume `sizeArray` is a non-negative number.

You should ensure that no memory leak will occur in your implementation and you cannot call any other functions. (Note: the definition of the regualr linked list is given in Appendix A for your reference.)

```
ll_cnode* arrayToLL(const char* c, int sizeArray)
{
   // Answer here:
```

}

Answer:

```
#include "ll_cnode.h"

ll_cnode* arrayToLL(const char* c, int sizeArray)
{
   ll_cnode* head, p1, p2;

   if (sizeArray <= 0)
      return nullptr;
   head = new ll_cnode;
   head->data = c[0];
   head->next = nullptr;
   p1 = head;

   for (int i=1; i<sizeArray; i++)
   {
      p2 = new ll_cnode;
      p2->data = c[i];
      p2->next = nullptr;
      p1->next = p2;
      p1 = p2;
   }
   return head;
}
```

Appendix A

```
#include <iostream> /* File: ll_cnode.h */
using namespace std;

struct ll_cnode
{
    char data;          // Contains useful information
    ll_cnode* next;     // The link to the next node
};

const char NULL_CHAR = '\0';
ll_cnode* ll_create(char);
ll_cnode* ll_create(const char []);
int ll_length(const ll_cnode*);
void ll_print(const ll_cnode*);
ll_cnode* ll_search(ll_cnode*, char c);
void ll_insert(ll_cnode*&, char, unsigned);
void ll_delete(ll_cnode*&, char);
void ll_delete_all(ll_cnode*&);

#include "ll_cnode.h"   /* File: ll_create.cpp */
// Create a ll_cnode and initialize its data
ll_cnode* ll_create(char c)
{
    ll_cnode* p = new ll_cnode; p->data = c; p->next = nullptr; return p;
}

// Create a linked list of ll_cnodes with the contents of a char array
ll_cnode* ll_create(const char s[])
{
    if (s[0] == NULL_CHAR) // Empty linked list due to empty C string
        return nullptr;

    ll_cnode* head = ll_create(s[0]); // Special case with the head

    ll_cnode* p = head; // p is the working pointer
    for (int j = 1; s[j] != NULL_CHAR; ++j)
    {
        p->next = ll_create(s[j]); // Link current cnode to the new cnode
        p = p->next; // p now points to the new ll_cnode
    }

    return head; // The WHOLE linked list can be accessed from the head
}

#include "ll_cnode.h"   /* File: ll_print.cpp */

void ll_print(const ll_cnode* head)
{
    for (const ll_cnode* p = head; p != nullptr; p = p->next)
        cout << p->data;
    cout << endl;
}
```

```cpp
#include "ll_cnode.h"   /* File: ll_search.cpp */

// The returned pointer may be used to change the content
// of the found ll_cnode. Therefore, the return type
// should not be const ll_cnode*.

ll_cnode* ll_search(ll_cnode* head, char c)
{
    for (ll_cnode* p = head; p != nullptr; p = p->next)
    {
        if (p->data == c)
            return p;
    }

    return nullptr;
}

#include "ll_cnode.h" /* File: ll_length.cpp */

int ll_length(const ll_cnode* head)
{
    int length = 0;
    for (const ll_cnode* p = head; p != nullptr; p = p->next)
        ++length;
    return length;
}

#include "ll_cnode.h"   /* File: ll_insert.cpp */

// To insert character c to the linked list so that after insertion,
// c is the n-th character (counted from zero) in the list.
// If n > current length, append to the end of the list.

void ll_insert(ll_cnode*& head, char c, unsigned n)
{
    // STEP 1: Create the new ll_cnode
    ll_cnode* new_cnode = ll_create(c);

    // Special case: insert at the beginning
    if (n == 0 || head == nullptr)
    {
        new_cnode->next = head;
        head = new_cnode;
        return;
    }


    // STEP 2: Find the node after which the new node is to be added
    ll_cnode* p = head;
    for (int position = 0;
         position < n-1  &&  p->next != nullptr;
         p = p->next, ++position)
         ;

    // STEP 3,4: Insert the new node between
    //           the found node and the next node
    new_cnode->next = p->next; // STEP 3
    p->next = new_cnode;       // STEP 4
}
```

```cpp
#include "ll_cnode.h"   /* File: ll_delete.cpp */
// To delete the character c from the linked list.
// Do nothing if the character cannot be found.
void ll_delete(ll_cnode*& head, char c)
{
    ll_cnode* prev = nullptr; // Point to previous ll_cnode
    ll_cnode* current = head; // Point to current ll_cnode

    // STEP 1: Find the item to be deleted
    while (current != nullptr && current->data != c)
    {
        prev = current;      // Advance both pointers
        current = current->next;
    }

    if (current != nullptr)  // Data is found
    {   // STEP 2: Bypass the found item
        if (current == head) // Special case: delete the first item
            head = head->next;
        else
            prev->next = current->next;

        delete current;      // STEP 3: Free up the memory of the deleted item
    }
}

#include "ll_cnode.h"   /* File: ll_delete_all.cpp */

// To delete the WHOLE linked list, given its head by recursion.
void ll_delete_all(ll_cnode*& head)
{
    if (head == nullptr) // An empty list; nothing to delete
        return;

    // STEP 1: First delete the remaining nodes
    ll_delete_all(head->next);

    // For debugging: this shows you what are deleting
    cout << "deleting " << head->data << endl;

    delete head;         // STEP 2: Then delete the current nodes
    head = nullptr;      // STEP 3: To play safe, reset head to nullptr
}
```

---------- END OF PAPER ----------