C++ TUTORIAL QUIZ - LINKED LIST - 2017

G+1 1

K Hong
google.com/+KHongSanF Francisc...
Ikuti
3.858 pengikut

Ph.D. / Golden Gate Ave, San Francisco / Seoul National Univ / Carnegie Mellon / UC Berkeley / DevOps / Deep Learning / Visualization

(http://www.addthis.com/bookmark.php?v=250&username=khhong7)

Sponsor Open Source development activities and free contents for everyone.

Donate with PayPal

Thank you.

- K Hong (http://bogotobogo.com/about_us.php)

bogotobogo.com site search:
Custom Search Search

Introduction - Linked List

Most of the linked list operations are straight forward. However, special care is neeeded when we deal with head element.

The head element of a linked list should always be tracked, otherwise we will lose the list. This means that the pointer to the head of the list must be updated when a new element is inserted in from of the head element or when the head element is removed from the list.

The following code illustrates two operations:

1. deleting head element
2. inserting an element ahead of the first element of a list

```
#include <iostream>
using namespace std;

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

Node *createNode(int n)
{
        Node *ptr = new Node();
        ptr->data = n;
        ptr->next = NULL;
        return ptr;
}

Node *appendNode(Node *node, int n)
{
        Node *cur = node;
        while(cur) {
                if(cur->next == NULL) {
                        cur->next = createNode(n);
                        return cur->next;
                }
                cur = cur->next;
        }
        return cur;
}

void printNodes(Node *head)
{
        Node *cur = head;
        while(cur) {
                cout << cur->data << " ";
                cur = cur->next;
        }
        cout << endl;
}

Node *deleteNode(Node **head)
{
        Node *temp = *head;   // NOT Node *temp = head;
        *head = temp->next;   // NOT head = temp->next;
        delete temp;
        return *head;         // NOT return head;
}

Node *insertFront(Node **head, int n)
{
        Node *newNode = createNode(n);
        newNode->next = *head; // NOT newNode->next = head;
        *head = newNode;       // NOT head = newNode;
        return *head;          // NOT return head;
}

int main()
{
        Node *head = createNode(100);
        appendNode(head, 200);
        appendNode(head, 300);
        printNodes(head);
        head = deleteNode(&head;);
        printNodes(head);
        head = insertFront(&head;, 100);
        printNodes(head);
```

Sponsor Open Source development activities and free contents for everyone.

Donate with PayPal

Thank you.

- K Hong (http://bogotobogo.com/about_us.php)

C++ Tutorials

C++ Home (/cplusplus/cpptut.php)

Algorithms & Data Structures in C++ ... (/Algorithms/algorithms.php)

Application (UI) - using Windows Forms (Visual Studio 2013/2012) (/cplusplus/application_visual_st

```
    return 0;
}
```

Problems and Solutions - Linked List

Removing Duplicates

A code to remove duplicates from an unsorted list when a temporary buffer is not allowed.

```
/* Removing duplicate from an unsorted list */
#include <iostream>

using namespace std;

typedef struct Node
{
        int data;
        Node* next;
} Node;

bool createList(Node **head)
{
        *head = NULL;
        return true;
}

void addNode(Node **head, int n)
{
        Node *node = new Node();
        node->data = n;
        node->next = NULL;
        if(*head == NULL) {
                *head = node;
                return;
        }
        Node *cur = *head;
        while(cur) {
                if(cur->next == NULL) {
                        cur->next = node;
                        return;
                }
                cur = cur -> next;
        }
        return;
}

void deleteNode(Node **head, Node *node)
{
        Node *cur = *head;
        Node *prev = *head;
        if(node == *head) {
                if((*head)->next != NULL) {
                        *head = (*head)->next;
                }
                return;
        }
        while(cur) {
                if(cur == node) {
                        prev->next = cur->next;
                        return;
                }
                prev = cur;
                cur = cur->next;
        }
}

/* No buffer allowed - neeed two pointers */
void removeDuplicateNode(Node **head)
{
        if((*head)->next == NULL) return;
        Node *cur = *head;
        Node *iter;
        while(cur) {
                iter = cur->next;
                while(iter) {
                        if(cur->data == iter->data) {
                                deleteNode(head, iter);
```

(/cplusplus/embeddedSystemsF

Embedded Systems Programming III - Eclipse CDT Plugin for gcc ARM Toolchain (/cplusplus/embeddedSystemsF

Exceptions (/cplusplus/exceptions.php)

Friend Functions and Friend Classes (/cplusplus/friendclass.php)

fstream: input & output (/cplusplus/fstream_input_outpu

Function Overloading (/cplusplus/function_overloadin

Functors (Function Objects) I - Introduction (/cplusplus/functor_function_ob

Functors (Function Objects) II - Converting function to functor (/cplusplus/functor_function_ob

Functors (Function Objects) - General (/cplusplus/functors.php)

Git and GitHub Express... (/cplusplus/Git/Git_GitHub_Expr

GTest (Google Unit Test) with Visual Studio 2012 (/cplusplus/google_unit_test_gte

Inheritance & Virtual Inheritance (multiple inheritance) (/cplusplus/multipleinheritance.

Libraries - Static, Shared (Dynamic) (/cplusplus/libraries.php)

Linked List Basics (/cplusplus/linked_list_basics.ph

Linked List Examples (/cplusplus/linkedlist.php)

make & CMake (/cplusplus/make.php)

```
                                }
                                iter = iter->next;
                        }
                        cur = cur->next;
                }
                return;
        }

        void print(Node *head)
        {
                Node *cur = head;
                while(cur) {
                        cout << cur->data << " ";
                        cur = cur->next;
                }
                cout << endl;
        }

        int main()
        {
                Node *head;
                createList(&head;);
                addNode(&head;,10);
                addNode(&head;,30);
                addNode(&head;,40);
                addNode(&head;,50);
                addNode(&head;,20);
                addNode(&head;,30);
                addNode(&head;,70);
                addNode(&head;,20);
                addNode(&head;,30);
                addNode(&head;,90);

                print(head);
                removeDuplicateNode(&head;);
                print(head);
                return 0;
        }
```

Output from the run:

```
10 30 40 50 20 30 70 20 30 90
10 30 40 50 20 70 90
```

Beginning of a Circular List

Returning a node of the beginning of the loop in circular linked list.
From a corrupt (circular) linked list, the code sample below shows how to find the beginning of the loop.
Two pointers are used: the first pointer iterates one node per step and the other pointer iterated two nodes per step. It first find the node where the two pointers meet. Then, each pointer iterates one node per step. It will meet at the beginning of the loop.

```
/* Locate the beginning of circular list */
#include <iostream>

using namespace std;

typedef struct Node
{
        char data;
        Node* next;
} Node;

bool createList(Node **head)
{
        *head = NULL;
        return true;
}

void addNode(Node **head, char c)
{
        Node *node = new Node();
        node->data = c;
        node->next = NULL;
        if(*head == NULL) {
                *head = node;
                return;
        }
        Node *cur = *head;
        while(cur) {
                if(cur->next == NULL) {
                        cur->next = node;
                        return;
                }
                cur = cur -> next;
        }
        return;
}

/* Make it circulat at node 'D' */
void makeCircular(Node **head, char c)
{
        if(*head == NULL) return;
        Node *cur = *head;
        Node *tmp;
        while(cur) {
                if(cur->data == c) {
                        tmp = cur;
                }
                if(cur->next == NULL) break;
                cur = cur->next;
        }
        cur->next = tmp;
        return;
}

/* cur : iterates one node at a time */
/* cur2 : iterates two nodes at a time */
char findCircularAt(Node *head)
{
        Node *cur = head;
        Node *cur2 = head;
        if(head == NULL || head->next == NULL) return '\0';
        while(cur) {
                cur = cur->next;
                if(cur2 ->next == NULL ) return '\0';
                cur2 = cur2->next;
                if(cur2 ->next == NULL ) return '\0';
                cur2 = cur2->next;
                if(cur == cur2 ) {
                        cur->data;
                        break;
                }
        }
        cur2 = cur;
        cur = head;
        while(cur) {
                cur = cur->next;
                cur2 = cur2->next;
                if(cur == cur2) {
                        return cur->data;
                }
        }
}

void print(Node *head)
{
        Node *cur = head;
        while(cur) {
                cout << cur->data << " ";
                cur = cur->next;
        }
        cout << endl;
}

/* A -> B -> C -> D -> E -> .... -> M -> N -> D -> E -> .... */
int main()
{
        Node *head;
        createList(&head;);
        addNode(&head;, 'A');
        addNode(&head;, 'B');
        addNode(&head;, 'C');
        addNode(&head;, 'D');
        addNode(&head;, 'E');
        addNode(&head;, 'F');
        addNode(&head;, 'G');
        addNode(&head;, 'H');
        addNode(&head;, 'I');
        addNode(&head;, 'J');
        addNode(&head;, 'K');
        addNode(&head;, 'L');
        addNode(&head;, 'M');
        addNode(&head;, 'N');

        print(head);

        /* Make it circular at node 'D' */
        makeCircular(&head;,'D');

        cout << "List is circular at " <<
                findCircularAt(head) << endl;

        return 0;
}
```

Output is

```
A B C D E F G H I J K L M N
List is circular at D
```

A List Representing the Sum of the Two Lists

(STL) II - Maps
(/cplusplus/stl2_map.php)

Standard Template Library
(STL) II - unordered_map
(/cplusplus/stl2_unorderd_map_

Standard Template Library
(STL) II - Sets
(/cplusplus/stl2B_set.php)

Standard Template Library
(STL) III - Iterators
(/cplusplus/stl3_iterators.php)

Standard Template Library
(STL) IV - Algorithms
(/cplusplus/stl4_algorithms.php)

Standard Template Library
(STL) V - Function Objects
(/cplusplus/stl5_function_object

Static Variables and Static
Class Members
(/cplusplus/statics.php)

String (/cplusplus/string.php)

String II - sstream etc.
(/cplusplus/string2.php)

Taste of Assembly
(/cplusplus/assembly.php)

Templates
(/cplusplus/templates.php)

Template Specialization
(/cplusplus/template_specializat

Template Specialization -
Traits
(/cplusplus/template_specializat

Template Implementation &
Compiler (.h or .cpp?)
(/cplusplus/template_declaratio

The this Pointer
(/cplusplus/this_pointer.php)

Type Cast Operators
(/cplusplus/typecast.php)

Upcasting and Downcasting
(/cplusplus/upcasting_downcast

Two linked lists are representing two different numbers. Each node contains a single digit. For example, 234 and 5678, respectively. This code adds the two numbers and returns the sum as a linked list.

Virtual Destructor & boost::shared_ptr (/cplusplus/virtual_destructors_s

Virtual Functions (/cplusplus/virtualfunctions.php

Programming Questions and Solutions ↓

Strings and Arrays (/cplusplus/quiz_strings_arrays.p

Linked List (/cplusplus/quiz_linkedlist.php)

Recursion (/cplusplus/quiz_recursion.php)

Bit Manipulation (/cplusplus/quiz_bit_manipulatio

Small Programs (string, memory functions etc.) (/cplusplus/smallprograms.php)

Math & Probability (/cplusplus/quiz_math_probabili

Multithreading (/cplusplus/quiz_multithreading

140 Questions by Google (/cplusplus/google_interview_qu

Qt 5 EXPRESS... (/Qt/Qt5_Creating_QtQuick2_QM

Win32 DLL ... (/Win32API/Win32API_DLL.php)

Articles On C++ (/cplusplus/cppNews.php)

What's new in C++11... (/cplusplus/C11/C11_initializer_l

C++11 Threads EXPRESS... (/cplusplus/C11/1_C11_creating_

OpenCV... (/OpenCV/opencv_3_tutorial_im

```
/* A new list representing the sum of the two lists
 * The three lists stored the one digit per node of the list */
#include <iostream>

using namespace std;

typedef struct Node
{
        int data;
        Node* next;
} Node;

bool createList(Node **head)
{
        *head = NULL;
        return true;
}

void addNode(Node **head, int n)
{
        Node *node = new Node();
        node->data = n;
        node->next = NULL;
        if(*head == NULL) {
                *head = node;
                return;
        }
        Node *cur = *head;
        while(cur) {
                if(cur->next == NULL) {
                        cur->next = node;
                        return;
                }
                cur = cur -> next;
        }
        return;
}

void insertFront(Node **head, int n)
{
        Node *cur = new Node();
        if(*head == NULL) {
                cur->data = n;
                *head = cur;
                (*head)->next = NULL;
                return;
        }
        cur->next = *head;
        cur->data = n;
        *head = cur;
        return;

}

/* Get the number from each digit stored in a node of a list */
int getNumber(Node *head)
{
        int sum = 0;
        if(head == NULL) return sum;
        Node *cur = head;
        while(cur) {
                sum *= 10;
                sum += cur->data;
                cur = cur->next;
        }
        return sum;
}
```

```
/* just convert unsigned int to string - itoa()*/
void i2a(int n, char s[])
{
        int i = 0;
        while(1) {
                s[i++] = (n % 10) + '0';
                if((n /= 10) == 0) break;
        }
        s[i]='\0';
}

/* Get the sum of numbers from the two list and
 * put each digit of the sum to a new list
 * 5912 = 234 + 5678
 * '5' into the head, then '9' next in the list, and so on. */
void putNumber(Node **head, Node *headA, Node *headB)
{
        int sum = getNumber(headA)+getNumber(headB);
        char s[10];

        /* s has reversed digits */
        i2a(sum,s);

        /* get each character one by one
        and insert into the head of the list
        because the list has reversed digit*/
        for(int i = 0; i < strlen(s); i++) {
                insertFront(head, s[i]-'0');
        }
        return;
}

void print(Node *head)
{
        Node *cur = head;
        while(cur) {
                cout << cur->data << " ";
                cur = cur->next;
        }
        cout << endl;
}

int main()
{
        Node *headA, *headB;
        Node *headSum;
        createList(&headA;);
        createList(&headB;);
        createList(&headSum;);

        // 234
        addNode(&headA;,2);
        addNode(&headA;,3);
        addNode(&headA;,4);
        cout << "Sum A = " << getNumber(headA) << endl;

        // 5678
        addNode(&headB;,5);
        addNode(&headB;,6);
        addNode(&headB;,7);
        addNode(&headB;,8);
        cout << "Sum B = " << getNumber(headB) << endl;

        // 234 + 5678
        putNumber(&headSum;,headA,headB);

        print(headA);
        print(headB);
        print(headSum);
        return 0;
}
```

Output from the run is:

```
Sum A = 234
Sum B = 5678
2 3 4
5 6 7 8
5 9 1 2
```

Making a Sorted List from an Unsorted List

This example code makes sorted list out of unsorted list. Because we are looping through both sorted and unsorted lists to compare the data, it's *$O(n^2)$*. So, it's not going to be efficient for a huge list.

```
#include <iostream>

using namespace std;

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

// append at the end of the list
void append(struct node **head, int n)
{
        struct node *cur;
        struct node *new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = n;
        new_node->next = NULL;
        if(*head == NULL) {
                *head = new_node;
                return;
        }

        cur = *head;
        while(cur) {
                if(cur->next == NULL) {
                        cur->next = new_node;
                        return;
                }
                cur = cur->next;
        }
}

void insert(struct node **head, int newData)
{
        struct node *current = *head;
        struct node *prev = *head;
        struct node *newNode = NULL;

        newNode = (struct node *)malloc(sizeof(struct node));
        newNode->data = newData;
        newNode->next = NULL;

        if(*head == NULL) {
                *head = newNode;
                return;
        }

        current = *head;
        // insert if new data is smaller than existing data
        while(current) {
                if(newData <= current->data ) {
                        if(current == *head) {
                                newNode->next = *head;
                                *head = newNode;
                                return;
                        }
                        newNode->next = current;
                        prev->next = newNode;
                        return;
                }
                prev = current;
                current = current->next;
        }
        // append if new data is the biggest
        append(head,newData);
        return;
}
```

```
struct node *getSorted(struct node *unsorted)
{
        struct node *in = unsorted;
        struct node *res = NULL;
        struct node *current = NULL;

        if(unsorted == NULL) return res;

        current = unsorted;

        // loop through unsorted list one by one
        while(current) {
                insert(&res;, current->data);
                current = current->next;
        }
        return res;
}

void display(struct node *head)
{
        if(head == NULL) return;
        struct node *cur = head;
        while(cur) {
                cout << cur->data << ' ';
                cur = cur->next;
        }
        cout << endl;
}

int main()
{
        struct node *unsortedList = NULL;

        // populate list by random number: unsorted
        for(int i = 0; i < 10; i++)
                append(&unsortedList;, rand() % 1000);

        cout << "Unsorted: ";
        display(unsortedList);

        cout << "Sorted: ";
        // display after converting unsorted list to sorted list
        display(getSorted(unsortedList));

        return 0;
}
```

Here is the output:

```
Unsorted: 41 467 334 500 169 724 478 358 962 464
Sorted: 41 169 334 358 464 467 478 500 724 962
```

Mth-to-Last Element of a Linked List

Given a singly-linked list, devise a time- and space-efficient algorithm to find the mth-to-last element of the list. Define mth to last such that when m=0, the last element of the list is returned.

Algorithm: while we are traversing the list, we mark the head element as mth behind when we arrives at the mth element of the list which is marked as current. As we're looping through, we update both of the marks. So, when the current element reaches the end of the list, we can just return the mth behind element. In this way, we can have O(n) in time.

```
#include <iostream>
using namespace std;

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

Node *createNode(int n)
{
        Node *ptr = new Node();
        ptr->data = n;
        ptr->next = NULL;
        return ptr;
}

Node *appendNode(Node *node, int n)
{
        Node *cur = node;
        while(cur) {
                if(cur->next == NULL) {
                        cur->next = createNode(n);
                        return cur->next;
                }
                cur = cur->next;
        }
        return cur;
}

void printNodes(Node *head)
{
        Node *cur = head;
        while(cur) {
                cout << cur->data << " ";
                cur = cur->next;
        }
        cout << endl;
}

// find the mth-to-last element of a list
Node *findMthToLastNode(Node *head, int mth)
{
        Node *cur = head;

        for(int i = 0; i < mth; i++) {
                if(cur)
                        cur = cur->next;
                else
                        return NULL;
        }

        Node *mBehind = head;
        while(cur) {
                cur = cur->next;
                if(cur) mBehind = mBehind->next;
        }
        return mBehind;
}

int main()
{
        Node *head = createNode(1);
        for(int i = 2; i <= 10; i++)
                appendNode(head, i);
        printNodes(head);
        for (int i = 0; i < 10; i++) {
                Node *found = findMthToLastNode(head, i);
                if(found)
                        cout << i << "-th to last = " <<
                              findMthToLastNode(head, i)->data << endl;
        }
        return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10
0-th to last = 10
1-th to last = 9
2-th to last = 8
3-th to last = 7
4-th to last = 6
5-th to last = 5
6-th to last = 4
7-th to last = 3
8-th to last = 2
9-th to last = 1
```

Problems and Solutions

1. Removing Duplicates
2. Beginning of a Circular List
3. A List Representing the Sum of the Two Lists
4. Making a Sorted List from an Unsorted List
5. Mth-to-Last Element of a Linked List

Additional codes related to linked list:

1. Linked List (linkedlist.html)

1. **Example 1 (linkedlist.php#linkedlistexample1)**
   When the head of the list is not a global pointer.
2. **Example 2 (linkedlist.php#linkedlistexample2)** and **Example 3 (linkedlist.html#linkedlistexample3)**
   When the head of the list is a global pointer.
   There are some implementation differences between these two examples.
3. **Example 4 (linkedlist.php#linkedlistexample4)**
   Used class & structure in that class.
4. **Example 5A (linkedlist.php#linkedlistexample5)**
   Detecting circular (loop) linked list.
5. **Example 5B (linkedlist.php#linkedlistexample5B)**
   Detecting circular (loop) linked list (Generic Class with Template).
6. **Example 6 (linkedlist.php#linkedlistexample6)**
   Stack with linked list data structure.
7. **Example 7 (linkedlist.php#linkedlistexample7)**
   Class Stack with linked list data structure.

8. **Example 8 (linkedlist.php#linkedlistexample8)**
   Queue with linked list data structure.
9. **Example 9 (linkedlist.php#linkedlistexample9)**
   Finding intersection and union of two linked lists.
10. **Example 10 (linkedlist.php#linkedlistexample10)**
    Generic linked lists.

CONTACT

BogoToBogo
contactus@bogotobogo.com (mailto:#)

FOLLOW BOGOTOBOGO

(https://www.facebook.com/KHongSanFrancisco) (https://twitter.com/KHongTwit) (https://plus.google.com/u/0/+KHongSanFrancisco/posts)

ABOUT US (/ABOUT_US.PHP)

contactus@bogotobogo.com (mailto:#)

Golden Gate Ave, San Francisco, CA 94115

Golden Gate Ave, San Francisco, CA 94115

Copyright © 2016, bogotobogo
Design: Web Master (http://www.bogotobogo.com)