Example of Linked List

Posted by Alejandro G. Carlstein Ramos Mejia on October 15, 2010 October 15, 2010 About Programming / C++

Example of Linked List

This code was compiled and tested on a Linux machine

NOTIFICATION: These examples are provided for educational purposes. Using this code is under your own responsibility and risk. The code is given 'as is'. I do not take responsibilities of how they are used.

List.h:

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

#ifndef LIST
#define LIST

typedef int ElementType;

class Node {
   public:

   // Create a linked list node by setting its data to the input parameter
   // and the next node in the linked list to NULL (0)
   Node(ElementType data_input);

   //The default constructor doesn't initialize the data members
   Node();

   // Contains the data of the linked list node
   ElementType data;

   // Contains a pointer to the next item in the linked list
   Node *next;
};

class List
{
   private:

      // List class's private data member:
      Node *list_head; // Contains a pointer to the first node in the
                       // linked list (the head of the list)

      int num_elements;
```

```
      // takes out the node with the smallest data member from this list,
      // and returns a pointer to it.
      Node *extractLargest();

   public:
      // Create a Linked list by initializing it's head to NULL (0)
      List();

      // Delete all the elements in the linked list when the list is deleted
      ~List();

      // Add a single element to the head of the linked list
      void insert(ElementType data_input);

      // Remove a single element from the head of the linked list
      void remove();

      // Print all the elements in the linked list
      void show();

      // This function splits the list in half, adding the those half of
      // the elements with the largest values into a new linked list, and
      // keeping only those half of the elements with the smallest values
      // in the list on which this function is called. The function
      // returns a pointer to the new (heap allocated) list with the
      // larger elements in it. If the original list contains an odd
      // number of elements, then the returned list contains one
      // fewer element than the list on which this function is called.
      // If the original list is empty, the function returns a pointer
      // to a new empty List (not a NULL pointer).
      List *splitBigSmall();

      // This function splits the list in two, adding the the elements
      // with odd integer values into a new linked list, and keeping only
      // those elements with even integer values in the list on which
      // this function is called. The function returns a pointer to
      // the new (heap allocated) list with the odd elements in it. If the
      // original list is empty, or if there are no odd elements, the
      // function returns a pointer to a new empty List (not a NULL
      // pointer).
      List *splitOddEven();
};

#endif
```

List.cpp:

```
#include 'List.h'

#include <string>

using namespace std;
```

```
// -------------------------------------
// ------------Node class---------------
// -------------------------------------

//
// Create a linked list node by setting its data to the input parameter
// and the next node in the linked list to NULL (0)
Node::Node(ElementType data_input) {

    data = data_input;

    next = 0;

}

// Default constructor leaves data items unspecified
Node::Node() {
}

// -------------------------------------
// ------------List class---------------
// -------------------------------------

//
// Default constructor creates an empty list
//
List::List() {

    list_head = NULL;

    num_elements = 0;

}

//
// Delete all the elements in the linked list when the list is deleted
//
List::~List() {

    Node *next_node;

    Node *current_node = list_head;

    while (current_node != NULL) {

        next_node = current_node->next;

        delete(current_node);

        current_node = next_node;

    }
```

```
}

//
// Add a single element to the head of the linked list
//
void List::insert(ElementType data_input) {

    Node *new_node = new Node(data_input);

 // If list_head == 0 then assign new_node as the list_head
 // else point new_node->next to list_head. After make that
 // new node the list_head
    if (list_head == 0) {

        list_head = new_node;

    } else {

        new_node->next = list_head;

        list_head = new_node;

    }

    num_elements++;

}

//
// Remove a single element from the head of the linked list
//
void List::remove() {

    if (list_head == NULL)
        return;

 // Make new_head the node that is pointed by list_head->next
    Node *new_head = list_head->next;

 // Delete the node list_head
    delete(list_head);

    //Make the new_head, the new list_head
    list_head = new_head;

    num_elements--;

}

//
// Print all the elements in the linked list
//
void List::show()
```

```
{

    cout << 'List of ' << num_elements << ' elements: ' ;

    Node *current_node = list_head;

    while (current_node != NULL) {

        cout << current_node->data << ' ';

        current_node = current_node->next;

    }

    cout << endl;
}

//
// splitOddEven()
//
List *List::splitOddEven()
{

 List *rlist = new List();

 if (num_elements > 0){

  Node *current_node = list_head;

  while (current_node != NULL){

   if (current_node->data % 2 == 1){

    rlist->insert(current_node->data);

    if (current_node == list_head){

     current_node = list_head->next;

     remove();

     current_node = current_node->next;

    }else{

     Node* previous_node = list_head;

     while (previous_node->next != current_node){
      previous_node = previous_node->next;
     }

     previous_node->next = current_node->next;
```

```
     // Delete the node list_head
     Node *temp = current_node;
     current_node = current_node->next;
        delete(temp);

     num_elements--;

    }

   }else{

    current_node = current_node->next;
   }
  }

 }
    return(rlist);

}

//
// extractLargest()
//
Node *List::extractLargest()
{

 Node *current_node = list_head;

 Node *largest_node = list_head;

 if (list_head != NULL){

  // Search Largest
  while(current_node != NULL ){

   if (largest_node->data < current_node->data){
    largest_node = current_node;
   }

   current_node = current_node->next;
  }

  current_node = largest_node;

  if (current_node == list_head){

   current_node = list_head->next;

   current_node = list_head;

   list_head = list_head->next;

  }else{
```

```
   Node* previous_node = list_head;

   while (previous_node->next != current_node){
    previous_node = previous_node->next;
   }

   previous_node->next = current_node->next;

  }

  //delete(current_node);

  num_elements--;
 }

 return (largest_node);
}

//
// splitBigSmall()
//
List *List::splitBigSmall()
{
 List *rlist = new List();

 int numExtract;

 numExtract = num_elements / 2;

 if (num_elements > 0){

  Node *current_node = list_head;

  // 2a. If original list have odd number of elements
  //  then return list with larger elements in it,
  //  should contain  one fewer element than
  //  the list obj which this function is called.
  //   This is already done by the division

  for (int i = 0; i < numExtract; i++){

   current_node = extractLargest();

   if (current_node != NULL)
    rlist->insert(current_node->data);

   //delete (current_node);
  }
 }

   return(rlist);
```

```
}
```

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks.