Requirements: Linked list template class –

A C++ template class that will take an arbitrary class or type and an integer as template parameters:

```
template <class ITEM_TYPE, int MAX_NODES = 100000>
class LinkedListNode;
```

ITEM_TYPE -- The type of data to be contained by each node.

MAX_NODES -- The maximum number of nodes of "ITEM_TYPE" data that can exist at any one time.

Methods:

Constructor – takes one parameter of type ITEM_TYPE and initializes a local member from that item.

insert – Takes a reference to a pointer to a node and inserts the current node before the node the parameter points to.

append -- Takes a reference to a pointer to a node and inserts the current node after the node pointed to.

remove -- Takes a reference to a pointer to a node and deletes that node from the list, connecting its tail instead.

data – returns a reference to the node's instance of ITEM_TYPE

get_count – returns an unsigned integer count of nodes of this type.

~destructor – destroys the node and all nodes that follow it in the list.

linked_list.h

```
#if !defined(LINKED_LIST_H)
#define LINKED_LIST_H

#include <stdlib.h> // For definition of NULL
extern short g_node_counter;

template <class ITEM_TYPE, int MAX_NODES = 100000>
class LinkedListNode
{
public:
   LinkedListNode()
   {
      if (g_node_counter < MAX_NODES) {
         m_next = NULL;
         ++g_node_counter;
      }
   }

   ~LinkedListNode()
   {
      delete m_next;
      m_next = NULL.

      --g_node_counter;
   }

   void insert(LinkedListNode & * pNode)
   {
      m_next = pNode;
      pNode = m_next;
   }

   void append(LinkedListNode & * pNode)
   {
      m_next = pNode->m_next;
      pNode->m_next = this;
   }

   void remove(LinkedListNode & * pNode)
   {

      pNode = pNode->m_next;
   }

   unsigned int get_count(void)
   {
      return g_node_counter;
   }

   ITEM_TYPE & data(void) { return m_data; }

   ITEM_TYPE  m_data;
   LinkedListNode * m_next;
}

#endif // LINKED_LIST_H

//////////////////////////////////////////////////
```

linked_list.cpp

```
#include "linked_list.h"
short g_node_counter;
```

```
//////////////////////////////////////////////////
                            main.cpp
#include <iostream>
#include <map>

#include <linked_list.h>

#define MAX_NODES 75000

typedef LinkedListNode<int, MAX_NODES> int_list_t;

typedef LinkedListNode<double, MAX_NODES> double_list_t;

//////////////////////////////////////
class UberNode : public int_list
{
   UberNode() { }

   ~UberNode() {}

   void add_value(std::string key, int value)
   {
      m_map[key] = value;
   }

   int value(std::string key) { return m_map[key]; }

private:
   std::map<std::string, int> m_map;
};
```

```
////////////////////////////////////////
int main(int, char **)
{

   UberNode *pRoot = new UberNode;
   UberNode *pLast = pRoot;
   int count = pRoot->get_count();

   while (count < MAX_NODES)
   {
      UberNode * pNew = new UberNode();
      pNew->add_value("COUNT", count);
      pLast->append(pNew);
      pLast = pNew;
   }

   double_list_t pDoubleRoot =
      new double_list_t;
   double_list_t *pDoubleLast = pDoubleRoot;
   int count = pDoubleRoot->get_count();

   while (count < MAX_NODES)
   {
      double_list_t * pNew =
         new double_list_t;
      pNew->add_value("COUNT", count);
      pDoubleLast->append(pNew);
      pDoubleLast = pNew;
   }

   delete pRoot;
   delete pDoubleRoot;

   return 0;
}
```