使用下压堆栈实现链表

/\*

使用下压堆栈实现链表

含有操作：

1. first

2. isEmpty()

3. push()

4. pop()

5. size()

\*/

|  |
| --- |
| P94\_1\_list.h  template <typename Item>  class Node  {  public:  Node(Item &a):item(a),next(nullptr){}  Node(Node &n):item(n.item),next(nullptr){}  Node():item(NULL),next(nullptr){}  Item item;  Node \*next;  };  template <typename Item>  class List  {  public:  Node<Item> \* begin();  int size()const;  bool push(Node<Item> &n);  bool pop();  bool isEmpty() const;  List();  ~List();  private:  Node<Item> \*first;  Node<Item> \*end;  int length;  }; |

|  |
| --- |
| P94\_1\_list.cpp  #include <string>  #include <iostream>  #include "P94\_1\_list.h"  //constructors  template<typename Item>  List<Item>::List():first(nullptr),end(nullptr),length(0)  {  ;  }  //desconstructor  template<typename Item>  List<Item>::~List()  {  if(isEmpty())  return;  Node<Item> \*bef;  for(int i = 0;i < length; i++)  {  bef = first;  first = bef->next;  delete bef;  }  first = end = nullptr;  }  template<typename Item>  int List<Item>::size() const  {  return length;  }  template<typename Item>  bool List<Item>::push(Node<Item> &n)  {  if(isEmpty())  {  first = new Node<Item>(n);  end = first;  end->next = nullptr;  length++;  return true;  }  end->next = new Node<Item>(n);  length++;  end = end->next;  end->next = nullptr;  return true;  }  template<typename Item>  bool List<Item>::pop()  {  if(isEmpty())  return false;  Node<Item> \*bef = first;  for(int i = 0;i<length - 2;i++)  bef = bef->next;  delete end;  end = bef;  length--;  return true;  }  template<typename Item>  bool List<Item>::isEmpty() const  {  if(length == 0)  return true;  return false;  }  int main()  {  using std::string;  using std::cout;  using std::cin;  using std::endl;  int a = 1;  Node<int> n(a);  //Node<int> n2(n);  List<int> l1;  l1.push(n);  l1.push(n);  l1.push(n);  l1.push(n);  cout << l1.size() << endl;  } |