实验二报告

1. 代码

llgen.h 通用双向链表头文件

```
#ifndef LLGEN_H  // 确保只包含一次头文件
#define LLGEN_H 1

// 定义双链表数据节点 LLNode
typedef struct Node
{
  struct Node * prev;
  struct Node * next;
  void        * pdata; // 通用数据
}LLNode, * Link;

// 定义双链表数据结构 List
typedef struct List
{
  Link LHead;
  Link LTail;
  unsigned int LCount;
  // 定义四个通用函数指针，操作链表数
  void * ( * LCreateData)    (void *); // void * 保证数据的通用性
  int    ( * LDeleteData)    (void *);
  int    ( * LDuplicateNode) (Link, Link);
  int    ( * LNodeDataCmp)   (void *, void *);
}LList, * PList;

// 通用函数原型
int AddNodeAscend(PList, void *);
int AddNodeAtHead(PList, void *);
PList CreateLList(
  void * (*) (void *),         // 创建
  int    (*) (void *),         // 删除
  int    (*) (Link, Link),     // 重复
  int    (*) (void *, void *)  // 比较
);
Link CreateNode     (PList, void *);
int  DeleteNode     (PList, Link);
Link FindNode       (PList, void *);
Link FindNodeAscend (PList, void *);
Link GoToNext       (PList, Link);
PList GoToPrev      (PList, Link);
```

```
#endif
```

Llgen.c 双向链表原型函数

```
// 双向链表原型函数
#include <stdlib.h>
#include <string.h>

#define IN_LL_LIB 1 // 标识在原函数库中
#include "llgen.h"

// 定义链表成员变量别名
#define LLHead    L->LHead
#define LLTail    L->LTail
#define NodeCount L->LCount

// 定义链表成员函数指针别名
#define CreateData  L->LCreateData
#define DeleteData  L->LDeleteData
#define DuplicateNode   L->LDuplicateNode
#define NodeDataCmp L->LNodeDataCmp

// 在头部添加节点
int AddNodeAtHead(PList L, void *nd)
{
  Link pn;
  // 为数据创建节点，返回节点指针
  pn = CreateNode(L, nd);
  if (pn == NULL) return 0;
  if (LLHead == NULL)  // 是否为第一个节点
  {
    LLHead = pn;
    LLTail = pn;
  } else { // 插入节点
    LLHead->prev = pn;
    pn->next = LLHead;
    LLHead = pn;
  }
  NodeCount += 1;
  return 1;
}

// 升序添加节点
int AddNodeAscend(PList L, void *nd)
{
  Link    pn;           // 指向创建的新节点
  Link    prev, curr;   // 搜索节点
  LLNode  dummy;        // 哑结点
```

```c
int     compare;

pn = CreateNode(L, nd);
if (pn == NULL) return 0;
// 在首部添加哑结点
dummy.next  = LLHead;
dummy.prev = NULL;
if (dummy.next != NULL) // 头结点不为空
  dummy.next->prev = &dummy;

prev = &dummy;
curr = dummy.next;
// 查找第一个 <= 的位置
while (curr != NULL)
{
  compare = NodeDataCmp(pn->pdata, curr->pdata);
  if (compare <= 0) break;
  prev = curr;
  curr = curr->next;
}

// 处理尾节点前面的重复节点
if (curr != NULL && compare == 0)
{
  compare = DuplicateNode(pn, curr);
  if (compare == 2) ; // 不处理，下面插入该节点
  else {
    // 恢复链表
    LLHead       = dummy.next;
    LLHead->prev = NULL;
    // 删除重复节点，先删除节点中的数据，避免孤儿指针，再释放节点指针
    if (compare == 1)
    {
      DeleteData(pn->pdata);
      free(pn);
    }
    return 1;
  }
}

// 没有重复节点，直接插入
prev->next = pn;
pn->prev   = prev;
pn->next   = curr;
if (curr != NULL) // 不是尾节点
  curr->prev = pn;
else
  LLTail = pn; // pn 为新的尾节点
  NodeCount += 1;
  // 恢复链表
  LLHead = dummy.next;
  LLHead->prev = NULL;
  return 1;
}

// 创建双链表，用四个函数指针初始化该双链表，返回指针
PList CreateLList(
  void * ( * fCreateData) (void *),
  int    ( * fDeleteData) (void *),
  int    ( * fDuplicateNode) (Link, Link),
  int    ( * fNodeDataCmp) (void *, void *))
{
  PList pL;
  pL = (PList)malloc(sizeof(LList));
  if (pL == NULL) return NULL;
  // 初始化链表成员数据
  pL->LHead  = NULL;
  pL->LTail  = NULL;
  pL->LCount = 0;

  pL->LCreateData    = fCreateData;
  pL->LDeleteData    = fDeleteData;
  pL->LDuplicateNode = fDuplicateNode;
  pL->LNodeDataCmp   = fNodeDataCmp;
  return pL;
}

// 创建数据节点，节点中数据由具体函数分配
Link CreateNode(PList L, void *data)
{
  Link new_node;
  new_node = (Link)malloc(sizeof(LLNode));
  if (new_node == NULL) return NULL;

  new_node->prev = NULL;
  new_node->next = NULL;

  // 调用具体的数据分配函数
  new_node->pdata = CreateData(data);
  if (new_node->pdata == NULL)
  {
    free(new_node);
    return NULL;
  }else
    return new_node;
}
```

```
// 删除节点，节点中的数据由具体的函数负责删除
int DeleteNode(PList L, Link to_delete)
{
  Link pn;
  // 合法性检查
  if (to_delete == NULL) return 0;
  if (to_delete->prev == NULL) // 头结点
  {
    LLHead = to_delete->next;
    LLHead->prev = NULL;
  } else if (to_delete->next == NULL) // 尾节点
  {
    pn = to_delete->prev;
    pn->next = NULL;
    LLTail = pn;
  } else { // 删除节点，修改两个链
    pn = to_delete->prev;
    pn->next = to_delete->next;
    to_delete->next->prev = pn;
  }
  // 具体函数删除
  DeleteData(to_delete->pdata);
  free(to_delete);
  NodeCount -= 1;
  return 1;
}

// 从头查找节点
Link FindNode(PList L, void * nd)
{
  Link pcurr = LLHead;
  if (LLHead == NULL) // 空链表
    return NULL;
  while (pcurr != NULL)
  {
    if (NodeDataCmp(nd, pcurr->pdata) == 0)
      return pcurr;
    pcurr = pcurr->next;
  }
  return NULL;
}

Link FindNodeAscend(PList L, void * nd)
{
  Link pcurr = LLHead;
  int cmp_result;
  if (LLHead == NULL)
    return NULL;
  while (pcurr != NULL)
  {
    cmp_result = NodeDataCmp(nd, pcurr->pdata);
    if (cmp_result < 0) // 小于则没有改节点
      return NULL;
    if (cmp_result == 0)
      return pcurr;
    pcurr = pcurr->next;
  }

  return NULL;
}


Link GotoNext ( struct List *L, Link pcurr )
{
    if ( pcurr->next == NULL || pcurr == LLTail )
        return ( NULL );
    else
        return ( pcurr->next );
}

Link GotoPrev ( struct List *L, Link pcurr )
{
    if ( pcurr->prev == NULL || pcurr == LLHead )
        return ( NULL );
    else
        return ( pcurr->prev );
}
```

llapp.h 应用特定头文件

```
// 应用特定头文件
#ifndef LLAPP_H
#define LLAPP_H 1

// 节点只包含一个字符
typedef struct NodeData1
{
  char* word;
}ND1, * pND1;

extern void * CreateData1(void *);
extern int    DeleteData1(void *);
extern int    DuplicatedNode1(Link, Link);
extern int    NodeDataCmp1(void *, void *);
```

```
#endif
```

llapp.c 应用 cpp 文件

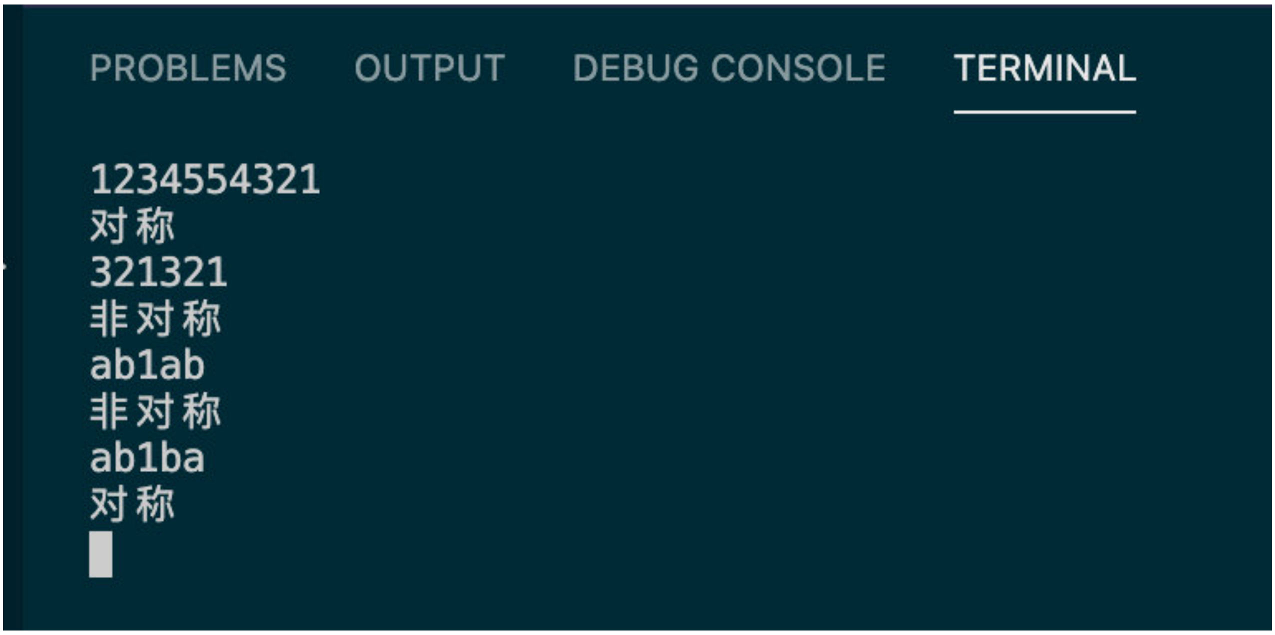
```
#include<stdlib.h>
#include<string.h>

#include "llgen.h"
#include "llapp.h"

void * CreateData1(void * data)
{

  pND1 new_data;
  if ((new_data = (pND1)malloc(sizeof(ND1))) == NULL)
    return NULL;

  new_data->word = strdup((char*)data);
  if (new_data->word == NULL)
  {
    free(new_data);
    return NULL;
  }

  return new_data;
}

int DeleteData1(void * data)
{
  free ( ((pND1) data)->word );
  return 0;
}

int DuplicatedNode1(Link new_node, Link list_node)
{
  return 2;
}

int NodeDataCmp1 ( void *first, void *second )
{
    return ( strcmp ( ((pND1) first)->word,
                      ((pND1) second)->word ));
}
```

Lab2.cpp 主函数文件

```
#include<stdio.h>
```

```
#include<stdlib.h>
#include<string.h>

#include "llgen.h"
#include "llapp.h"

int main()
{
  char ch;
  // 创建并初始化双链表
  PList L1 = CreateLList(CreateData1, DeleteData1, DuplicatedNode1,
 NodeDataCmp1);
  if (L1 == NULL)
  {
    fprintf(stderr, "双链表创建失败\n");
    exit(EXIT_FAILURE);
  }
  while (1)
  {
    // 读入一行字符串
    ch = getchar();
    while (ch != '\n')
    {
      // 读入一个字符创建一个节点，加入到链表头部
      if (AddNodeAtHead(L1, &ch) == 0)
        fprintf(stderr, "add error\n");
      ch = getchar();
    }

    Link head = L1->LHead, tail = L1->LTail;
    int cmp = 0;
    while (head != tail)
    {
      cmp = L1->LNodeDataCmp(head->pdata, tail->pdata);
      if (cmp != 0)
        break;
      head = head->next;
       // 偶数长度退出条件
      if (head == tail)
        break;
      tail = tail->prev;
    }

    if (cmp == 0)
      printf("对称\n");
    else
      printf("非对称\n");

    Link p = L1->LHead, nex = p->next;
    while(L1->LCount != 1)
    {
      DeleteNode(L1, p);
      p = nex;
      nex = nex->next;
    }
    L1->LHead = NULL;
    L1->LTail = NULL;
    L1->LCount = 0;
  }

  return 0;
}
```

2. 运行截图



```
1234554321
对称
321321
非对称
ab1ab
非对称
ab1ba
对称
█
```