链式存储结构

链式存储结构是存放数据的另一种重要方式。通常将链式存储的线性表称为单链表。链式存储结构相对于顺序存储结构最大的不同，一是数据的逻辑结构和物理存储相互独立，逻辑关系上相邻的元素物理位置上不一定是相邻的。二是数据元素所需的存储空间可动态生成获得。

本章的习题重点是单链表的各种建立算法及在单链表上的常见操作的程序实现。单链表的数据结构如下：

```
#define DATATYPE2 char

typedef struct node
{ DATATYPE2 data;
struct node * next;
}LINKLIST;
```

说明：单链表中元素结点的数据域为 data，类型设定为字符，结点的指针域为 next。

3.1 习题解析

【习题 1】 若线性表的元素总数基本稳定，且很少进行插入和删除，但要求快速存取表中元素，应采用哪种存储结构？为什么？

【解答】 由于顺序存储结构一旦确定了起始位置，对线性表中的任何一个元素都可以进行随机存取，且存取速度较高。已知线性表的元素总数基本稳定，且很少进行插入和删除，恰好避开了顺序存储结构的缺点。因此，应采用顺序存储结构。

【习题 2】 对线性表而言，什么情况下采用链表比顺序表好？

【解答】 如在线性表上经常进行插入和删除的操作，采用链表可避免移动大量的元素；而且经常进行插入和删除的操作，线性表需要的空间量难以预估，采用链表可以动态分配空间。

【习题 3】 分析单链表、循环链表和双向链表的相同点和各自的特点。

【解答】 相同点在于都是链式存储结构，表中元素对应的结点中除自身的信息域，还

有关联信息的指针域。

单链表的特点是链表结构较简单，表的操作也较简单，但从单链表中某一结点不能直接找到其前驱结点。

而双向链表因结点中既有指向后继结点的信息又有指向前驱结点的信息，所以从双向链表中某一结点既可直接找到其后继结点，又能直接找到其前驱结点。但表的操作比单链表要复杂些。

循环链表的特点是可从表中任一结点出发遍历表中全部元素结点。

【习题 4】 已知 L 是无表头结点的单链表，且 P 结点既不是首结点，也不是尾结点，试从下列提供的语句中选出合适的语句序列。

(1) 在 P 结点后插入 S 结点：_____。

(2) 在 P 结点前插入 S 结点：_____。

(3) 在表首插入 S 结点：_____。

(4) 在表尾插入 S 结点：_____。

① P—>next=S;

② P—>next=P—>next—>next;

③ P—>next=S—>next;

④ S—>next=P—>next;

⑤ S—>next=L;

⑥ S—>next=P;

⑦ S—>next=NULL;

⑧ Q=P;

⑨ while (P—>next !=Q) P=P—>next;

⑩ while (Q—>next !=NULL) Q=Q—>next;

⑪ P=Q;

⑫ P=L;

⑬ L=S;

⑭ L=P;

【解答】 在 P 结点后插入 S 结点：③、① 。

在 P 结点前插入 S 结点：⑧、⑫、⑨、④、① 。

在表首插入 S 结点：⑤、⑬ 。

在表尾插入 S 结点：⑦、⑪、① 。

【习题 5】 单项选择题。

(1) 在一个长度为 n 的顺序表中向第 i 个元素($0<i\leqslant n+1$)之前插入一个新元素时，需向后移动_____个元素。

A. $n-i$　　B. $n-i+1$　　C. $n-i-1$　　D. i

【解答】 在第 i 个元素($0<i\leqslant n+1$)之前插入一个新元素时，应将第 i 个元素及之后的全部元素后移，后移的元素个数为 $n-i+1$ 个，所以应选 B。注意题中 i 的范围 $0<i\leqslant n+1$ 是考虑了可在长度为 n 的顺序表中的 n 个元素之前插入新元素，并可在表中最后一

个元素之后插入新元素。

(2) 线性表采用链式存储结构时,其地址_____。

A. 必须是连续的　　B. 一定是不连续的

C. 部分地址必须是连续的　　D. 连续与否均可以

【解答】 链式存储结构是动态存储结构,不要求地址是连续的,这是与顺序存储结构最大的区别,所以选 D。

(3) 在一个单链表中,删除 * p 结点之后的一个结点的操作是_____。

A. p—>next=p;　　B. p—>next—>next=p—>next;

C. p—>next—>next=p;　　D. p—>next=p—>next—>next;

【解答】 D。

(4) 在一个双链表中,在 * p 结点之后插入结点 * s 的操作是_____。

A. s—>prior=p; p—>next=s; p—>next—>prior=s; s—>next=p—>next;

B. s—>next=p—>next; p—>next—>prior=s; p—>next=s; s—>prior=p;

C. p—>next=s; s—>prior=p; s—>next=p—>next; p—>next—>prior=s;

D. p—>next—>prior=s; s—>next=p—>next; s—>prior=p; p—>next=s;

【解答】 B。

(5) 在不带头结点 * head 的单循环链表中,尾结点 * p 的条件是_____。

A. head !=NULL　　B. head—>next !=head

C. p==NULL　　D. p—>next==head

【解答】 D。

【习题 6】 判断以下叙述的正确性。

(1) 分配给单链表的内存单元地址必须是连续的。

(2) 与顺序表相比,在链表上实现顺序访问,其算法的效率比较低。

(3) 向顺序表中插入一个元素,平均要移动约一半的元素。

(4) 在带头结点的循环单链表中,任何一个结点的指针都不可能为空。

(5) 在有 n 个元素的顺序表中,删除任意一个元素所需移动结点的平均次数为 $n-1$ 个。

【解答】

(1) 错误。分配给单链表的内存单元地址可以是不连续的。

(2) 错误。在顺序表和链表上实现顺序访问,其算法的时间复杂度都是 $O(n)$。

(3) 正确。

(4) 正确。

(5) 错误。删除第 1 个元素移动结点次数为 $n-1$ 个。删除最后一个元素没有结点移动。删除任意一个元素所需移动结点的平均次数约一半的元素为 $n/2$。

【习题 7】 设计一算法,在一个不带头结点的单链表上,用最少的辅助空间实现单链表元素的逆置。

【解答】

```
LINKLIST * invertlink(LINKLIST * head){
```

```
/*单链表元素逆置*/
  LINKLIST *p, *q, *r;

  q=NULL; p=head;
  while(p !=NULL)
      {r=q; q=p ; p=p->next; q->next=r;}
  return q;
}
```

【习题 8】 按下列要求建立单链表,编写程序实现:

(1) 按头插入法建立不带头结点的单链表。

题目分析:将输入的数据按头插入法建立一个不带头结点的单链表。输入数据时以输入一串字符的方式实现,“$”字符为输入结束字符。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int count_nohead(LINKLIST *head){
/*不带头结点的单链表:输出单链表元素值并计数*/
  int i=0;
  LINKLIST *p;

  p=head;
  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {printf("%c",p->data); i++; p=p->next;}
  printf("\n");
  return i;
}

LINKLIST *creatlink_nohead_head(LINKLIST *head) {
/*用头插入法建立不带头结点的单链表*/
  LINKLIST *t;
  char ch;

  printf("单链表元素值为单个字符,连续输入,$ 为结束字符 : ");
  while((ch=getchar())!='$ ')
    {  t=(LINKLIST *) malloc(sizeof(LINKLIST));
       t->data=ch; t->next=head; head=t;}
return(head);
}
```

```
main()
  { LINKLIST * head=NULL;
    int num;

    printf("\n 建立单链表\n\n");
    head=creatlink_nohead_head(head);
    fflush(stdin);
    num=count_nohead(head);
    printf("单链表元素个数=%d\n", num);
}
```

(2) 按头插入法建立一个带头结点的单链表。

题目分析：将输入的数据按头插入法建立一个带头结点的单链表。输入数据方式同上。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int count_head(LINKLIST * head){
/*带头结点的单链表:输出单链表元素值并计数*/
  int i=0;
  LINKLIST * p;

  p= head->next;
  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {printf(" % c",p->data); i++; p=p->next;}
  printf("\n");
  return i;
}

LINKLIST * creatlink_head_head(LINKLIST * head) {
/*用头插入法建立带头结点的单链表*/
  LINKLIST * t;
  char ch;

  t=(LINKLIST * )malloc(sizeof(LINKLIST));
  head=t; t->next=NULL;
  printf("单链表元素值为单个字符,连续输入,$ 为结束字符 : ");
  while((ch=getchar())!='$ ')
    {t=(LINKLIST * ) malloc(sizeof(LINKLIST));
    t->data=ch; t->next=head->next; head->next=t;}
  return(head);
```

```
}

main()
  { LINKLIST * head=NULL;
    int num;

    printf("\n 建立单链表\n\n");
    head=creatlink_head_head(head);
    fflush(stdin);
    num=count_head(head);
    printf("单链表元素个数=%d\n", num);
}
```

(3) 按尾插入法建立不带头结点的单链表。

题目分析：将输入的数据按尾插入法建立一个不带头结点的单链表。输入数据方式同上。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int count_nohead(LINKLIST * head){
/*不带头结点的单链表:输出单链表元素值并计数*/
  int i=0;
  LINKLIST * p;
  p=head;

  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {i++; printf(" % c",p->data); p=p->next;}
  printf("\n");
  return i;
}

LINKLIST * creatlink_nohead_rail(LINKLIST * head){
/*用尾插入法建立不带头结点的单链表*/
  LINKLIST * last, * t;
  char ch;

  last= head;
  printf("单链表元素值为单个字符,连续输入,$为结束字符 : ");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST * )malloc(sizeof(LINKLIST));
      t->data=ch; t->next=NULL;
      if (head==NULL) {head=t; last=t;}
```

```
        else { last->next=t; last=t;}
    }
    return (head);
}

main()
{
    LINKLIST * head= NULL;
    int num;

    printf("\n 建立单链表\n\n");
    head= creatlink_nohead_rail(head);
    fflush(stdin);
    num= count_nohead(head);
    printf("单链表元素个数= % d\n", num);
}
```

(4) 按尾插入法建立一个带头结点的单链表。

题目分析：将输入的数据按尾插入法建立一个带头结点的单链表。输入数据方式同上。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int count_head(LINKLIST * head){
/* 带头结点的单链表：输出单链表元素值并计数 */
    int i=0;
    LINKLIST * p;

    p=head->next;
    printf("输出单链表元素值 : ");
    while(p !=NULL)
      {i++; printf(" % c",p->data); p=p->next;}
    printf("\n");
    return i;
}

LINKLIST * creatlink_head_rail(LINKLIST * head){
/* 用尾插入法建立带头结点的单链表 */
    LINKLIST * last, * t;
    char ch;

    t= (LINKLIST * )malloc(sizeof(LINKLIST));
```

```
  head=t; last=t; t->next=NULL;
  printf("单链表元素值为单个字符, 连续输入,$为结束字符 : ");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST *)malloc(sizeof(LINKLIST));
      t->data=ch; t->next=NULL;
      last->next=t; last=t;}
  return (head);
}

main()
{ LINKLIST * head=NULL;
  int num;

printf("\n 建立单链表\n\n");
head=creatlink_head_rail(head);
fflush(stdin);
num=count_head(head);
printf("单链表元素个数=%d\n", num);
}
```

【习题 9】 设 L 为带头结点的单链表,表中元素值递增有序,编写程序删除表中值相同的多余元素。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

void delete(LINKLIST * a){
/* 在有序链表中删除重复元素,保留一个 */
LINKLIST * la;

la=a->next;
while(la !=NULL && la->next !=NULL)
  if (la->data==la->next->data) la->next=la->next->next;
  else la=la->next;
}

int count_head(LINKLIST * head){
/* 带头结点的单链表:输出单链表元素值并计数 */
  int i=0;
  LINKLIST * p;

  p=head->next;
  printf("输出单链表元素值 : ");
```

```
  while(p !=NULL)
    {i++; printf(" % c",p->data); p=p->next;}
  printf("\n\n");
  return i;
}

LINKLIST * creatlink_order_head(LINKLIST * head)
/* 建立带头结点的有序单链表 */
{ LINKLIST * t, * p, * q;
  char ch;

  t= (LINKLIST * )malloc(sizeof(LINKLIST));
  head=t; t->next=NULL;
  printf("单链表元素值为单个字符, 连续输入,$ 为结束字符 : ");
  while ((ch=getchar()) !='$ ')
    { t= (LINKLIST * )malloc(sizeof(LINKLIST));
      t->data=ch; q=head; p=head->next;
      while( p !=NULL && p->data<=ch) {q=p; p=p->next;}
      q->next=t; t->next=p;
    }
  return(head);
}

main()
{
  LINKLIST * head=NULL;
  int num;

  printf("\n 建立单链表\n\n");
  head=creatlink_order_head(head);
  fflush(stdin);
  num=count_head(head);
  printf("\n 删除重复元素后\n\n");
  delete(head);
  num=count_head(head);
}
```

【习题 10】 设 L 为带头结点的单链表,表中元素值无序,编写程序删除表中值相同的多余元素。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>
```

```
void delete(LINKLIST * a){
/* 在无序单链表中删除重复元素,保留一个 */
LINKLIST * la, * p, * q;

la=a->next;
while(la !=NULL)
  { q=la; p=la->next;
    while(p !=NULL)
      { if (p->data==la->data) { p=p->next; q->next=p;}
        else { q=p; p=p->next;}
      }
    la=la->next;
  }
}

int count_head(LINKLIST * head){
/* 带头结点的单链表:输出单链表元素值并计数 */
  int i=0;
  LINKLIST * p;

  p=head->next;
  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {i++; printf(" % c",p->data); p=p->next;}
  printf("\n");
  return i;
}

LINKLIST * creatlink_head_rail(LINKLIST * head){
/* 用尾插入法建立带头结点的单链表 */
  LINKLIST * last, * t;
  char ch;

  t=(LINKLIST * )malloc(sizeof(LINKLIST));
  head=t; last=t; t->next=NULL;
  printf("单链表元素值为单个字符,连续输入,$ 为结束字符 : ");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST * )malloc(sizeof(LINKLIST));
      t->data=ch; t->next=NULL;
      last->next=t; last=t;}
  return (head);
}

main()
```