思考题（1）

编写一个基于动态链表的“长整数加法运算器”，来实现任意长度的两个整数的加法运算。

具体要求：

（1）必须用线性链表的形式来存储一个长整数，例如：对于整数135，可以创建一条线性链表，该链表包含三个结点，分别用来存储1、3、5这三个数字。考虑到输入整数的长度是任意的（不超过100位），因此，为了减少内存空间的浪费，在程序中必须采用动态链表的方法，即每一个链表结点都是根据需要动态创建的；

（2）只考虑两个正整数的加法，无须考虑负整数的情形；

（3）为了增强程序的可读性，应采用多函数的形式来实现，至少应包含如下的函数：创建链表(CreatList)、加法函数(AddList)、打印链表(DisplayList)等；

（4）提示：由于本题处理的整数的长度是任意的，可能会超出long的取值范围，所以应该通过字符串的方式来处理输入输出。

本题的程序实现不能调用C++标准库函数<list>来实现，否则将被扣分。

以下是一次模拟运行的结果：

样例输入：

1234567890

135

样例输出：

1234568025

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

#define MAXNUM          101

typedef struct tagNODE
{
        int value;
        struct tagNODE *next;
} Node;

Node* CreateList(char *num);  // 创建链表
Node* AddList(Node *head1, Node *head2); // 两个链表相加
void DisplayList(Node *head);  // 显示链表
void DestroyList(Node *head);  // 释放链表
int list_length(Node *head);  // 计算链表长度
```

```
int main()
{
        char num1[MAXNUM] = {0}, num2[MAXNUM] = {0};
        Node *head1, *head2, *head3;
        head1 = head2 = head3 = NULL;

        gets(num1);
        gets(num2);

        head1 = CreateList(num1);
        //DisplayList(head1);

        head2 = CreateList(num2);
        //DisplayList(head2);

        head3 = AddList(head1, head2);
        DisplayList(head3);

        DestroyList(head1);
        DestroyList(head2);
        DestroyList(head3);
        return 0;
}
```

```
1234567890

135
```

```
Node* CreateList(char *num)
{
        Node *head, *p, *q;
        int i, n;
        n = strlen(num);
        head = q = p = NULL;
        for (i = n - 1; i >= 0; i--)
        {
                p = (Node *)malloc(sizeof(Node));
                p->value = num[i] - '0';

                if(head == NULL)
                {
                        head = p;
                        q = p;
                }
                else
                {
                        q->next = p;
                        q = p;
                }
        }
        q->next = NULL;
        return head;
}
```

1234567890

135

head1: 0987654321

head2: 531

```
Node* AddList(Node *head1, Node *head2)
{
        Node *head, *p, *q;  Node *p1, *p2;        int flag, temp;
        p1 = head1;          p2 = head2;
        head = NULL;         flag = 0;
        while (p1 != NULL && p2 != NULL)
        {
                p = (Node *)malloc(sizeof(Node));
                p->value = p1->value + p2->value + flag;
                if (p->value >= 10) {
                        p->value %= 10;
                        flag = 1;
                }
                else
                        flag = 0;
                p->next = NULL;
                if (head == NULL) {
                        head = p;
                        q = p;
                }
                else {
                        q->next = p;
                        q = p;
                }
                p1 = p1->next;
                p2 = p2->next;
        }
```

1234567890

135

head1: 0987654321

head2: 531

head: 520 NULL

flag: 1

```
while (p1 != NULL)
{
        p = (Node *)malloc(sizeof(Node));
        p->value = p1->value + flag;
        if (p->value >= 10)
        {
                p->value %= 10;
                flag = 1;
        }
        else
                flag = 0;

        p->next = NULL;

        if (head == NULL)
        {
                head = p;
                q = p;
        }
        else
        {
                q->next = p;
                q = p;
        }
        p1 = p1->next;
}
```

1234567890

135

head1: 0987654321

head2: 531

head: 5208654321

```
while (p2 != NULL)
{
        p = (Node *)malloc(sizeof(Node));
        p->value = p2->value + flag;
        if (p->value >= 10)
        {
                p->value %= 10;
                flag = 1;
        }
        else
                flag = 0;

        p->next = NULL;

        if (head == NULL)
        {
                head = p;
                q = p;
        }
        else
        {
                q->next = p;
                q = p;
        }
        p2 = p2->next;
}
```

1234567890

135

head1: 0987654321

head2: 531

head: 5208654321

```
        if(flag)
        {
                p = (Node *)malloc(sizeof(Node));
                p->value = 1;
                p->next = NULL;
                q->next = p;
                q = p;
        }

        return head;
}
```

```
void DisplayList(Node *head)
{
        int num[MAXNUM] = {0};
        int i, j;

        Node *p = head;

        i = 0;
        while (p != NULL)
        {
                num[i] = p->value;
                i++;
                p = p->next;
        }

        // 数组反向打印
        for (j = i - 1; j >= 0; j--)
                printf("%d", num[j]);

        printf("\n");
}
```

```
void DestroyList(Node *head)
{
        Node *p, *q;
        q = p = head;
        while(p != NULL)
        {
                q = p;
                p = p->next;
                free(q);
        }
}

int list_length(Node *head)
{
        int len = 0;
        Node *p = head;
        if(p == NULL)
                return 0;
        while (p != NULL)
        {
                p = p->next;
                len++;
        }

        return len;
}
```

样例输入：

23456

135

样例输出：

23591

思考题（2）

编写一个基于动态链表的“长整数加法运算器”，来实现任意长度的两个整数的加法运算。

具体要求：

（1）必须用线性链表的形式来存储一个长整数，例如：对于整数135，可以创建一条线性链表，该链表包含三个结点，分别用来存储1、3、5这三个数字。考虑到输入整数的长度是任意的（不超过100位），因此，为了减少内存空间的浪费，在程序中必须采用动态链表的方法，即每一个链表结点都是根据需要动态创建的；

（2）考虑任意长度的两个整数（包括正数和负数）；

（3）为了增强程序的可读性，应采用多函数的形式来实现，至少应包含如下的函数：创建链表(CreatList)、加法函数(AddList)、打印链表(DisplayList)等；

（4）提示：由于本题处理的整数的长度是任意的，可能会超出long的取值范围，所以应该通过字符串的方式来处理输入输出。

本题的程序实现不能调用C++标准库函数<list>来实现，否则将被扣分。

以下是一次模拟运行的结果：

样例输入：

-123

23

样例输出：

-100

思考题（2）图书管理

图书的基本信息一般包含图书编号（int类型）、名称（字符串，长度小于100）和价格（int类型）三部分。编写一个程序，采用链表对图书进行管理。管理指令有两种类型，第一种指令为翻转指令，格式为“R 开始序号 结束序号”，例如：“R 2 5”表示将链表上第2至5节点的图书信息逆序翻转，即如果原来链表顺序为“1 2 3 4 5 6”那么执行指令后变为“1 5 4 3 2 6”；第二种指令为删除指令，格式为“D 序号”，例如：“D 3”表示删除链表上第3个节点的图书信息。

程序第一行输入图书本数m，接下来每一行输入一条图书信息，每一行输入信息顺序为：编号、名称和价格（整数）；然后输入一个整数n，表示n条指令，接下来每一行输入相应的指令。最后，输出执行完所有指令后的图书信息。若执行完指令后链表为空，则不输出。

说明：

a)　在创建链表时，为了减少内存空间的浪费，必须采用动态链表的方法，即每一个链表结点都是根据需要动态创建的；

b)　为了增强程序的可读性，应采用多函数的形式来实现，至少应包含如下的函数：创建链表、增加节点、翻转链表、删除节点、打印链表、释放链表等。

c)　本题的程序实现不能调用C++标准库函数<list>来实现，否则将被扣分。

样例输入：

6

1 book1 10

2 book2 20

3 book3 30

4 book4 40

5 book5 50

6 book6 60

4

R 2 5

D 1

R 1 5

D 1

样例输出：

2 book2 20

3 book3 30

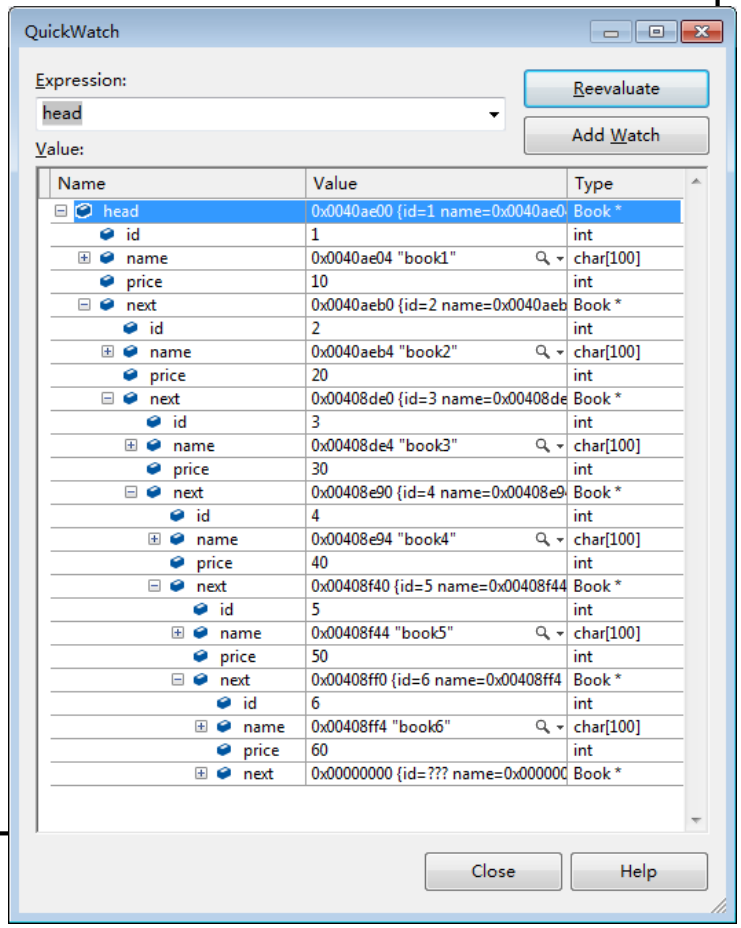
4 book4 40

5 book5 50

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include "string.h"

struct Book *CreatList(int m);
void PrintList(struct Book *head);
void DestroyList(Book *head);
struct Book *Delete(struct Book *head, int index);
struct Book *Reverse(struct Book *head, int start, int end);

struct Book
{
        int id;
        char name[100];
        int price;
        struct Book *next;
};
```

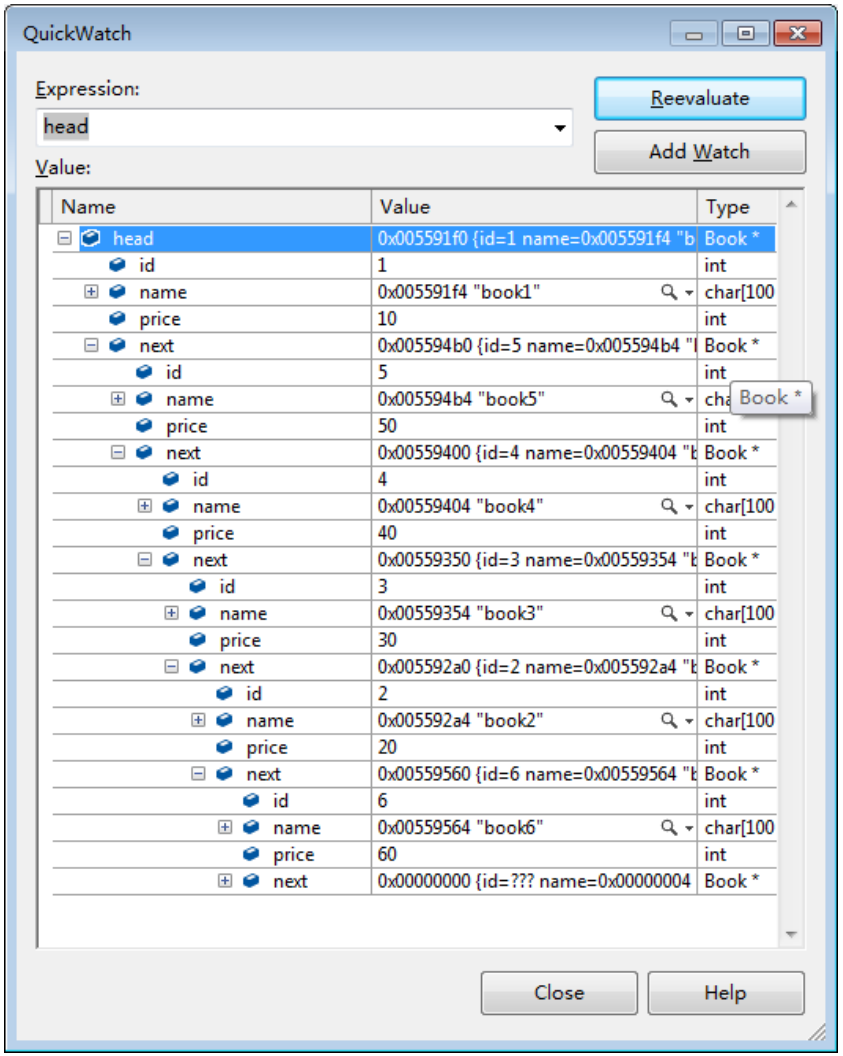
```
int main()
{
        int m, n;
        struct Book *head;
        scanf("%d", &m);
        head = CreatList(m);

        scanf("%d", &n);
        while (n > 0) {
                char command;
                //捕获换行
                getchar();
                scanf("%c", &command);
                if (command == 'R')
                {
                        int start, end;
                        scanf("%d%d", &start, &end);
                        head = Reverse(head, start, end);
                }
                else if (command == 'D')
                {
                        int index;
                        scanf("%d", &index);
                        head = Delete(head, index);
                }
                n--;
        }
        PrintList(head);
        DestroyList(head);
        return 0;
}
```

```
struct Book *CreatList(int m)
{
        struct Book *head, *p, *q;
        head = p = q = NULL;
        while (m)
        {
                q = (struct Book*)malloc(sizeof(struct Book));
                scanf("%d%s%d", &q->id, q->name, &q->price);
                if (head == NULL)
                {
                        head = q;
                        p = q;
                }
                else
                {
                        p->next = q;
                        p = q;
                }
                m--;
        }
        q->next = NULL;
        return head;
}
```



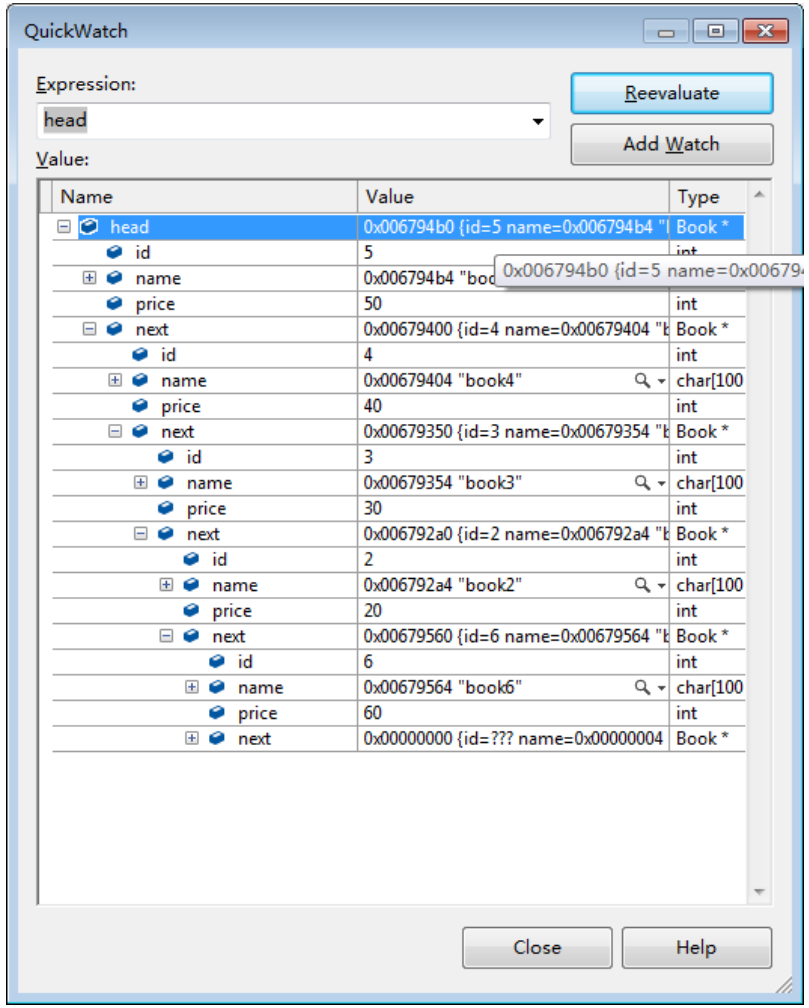
```
void PrintList(struct Book *head)
{
        while (head != NULL)
        {
                printf("%d %s %d\n", head->id, head->name, head->price);
                head = head->next;
        }
}

void DestroyList(struct Book *head)
{
        struct Book *p, *q;
        p = head;
        while (p != NULL)
        {
                q = p;
                p = p->next;
                free(q);
        }
}
```

```
struct Book *Reverse(struct Book *head, int s, int t)
{
        int si = s;
        int ti = t;
        struct Book *start, *end;
        struct Book *p, *q;
        start = end = head;
        p = q = head;
        while (si - 2 > 0)
        {
                start = start->next;
                si--;
        }
        while (ti > 0)
        {
                end = end->next;
                ti--;
        }

        if (s != 1)
        {
                p = start->next;
        }
```



```
        int count = t - s;
        while (count >= 0)
        {
                q = p;
                p = p->next;
                q->next = end;
                end = q;
                count--;
        }
        if (si != 1)
        {
                start->next = end;
        }
        else
        {
                head = q;
        }

        return head;
}
```

```
struct Book *Delete(struct Book *head, int index)
{
        struct Book *p, *q;

        p = q = head;
        if (index == 1)
        {
                head = head->next;
        }
        else
        {
                while (index - 1 > 0)
                {
                        q = p;
                        p = p->next;
                        index--;
                }
                q->next = p->next;
        }

        free(p);
        return head;
}
```



样例输入：

6

1 book1 10

2 book2 20

3 book3 30

4 book4 40

5 book5 50

6 book6 60

4

R 2 5

D 1

R 1 5

D 1

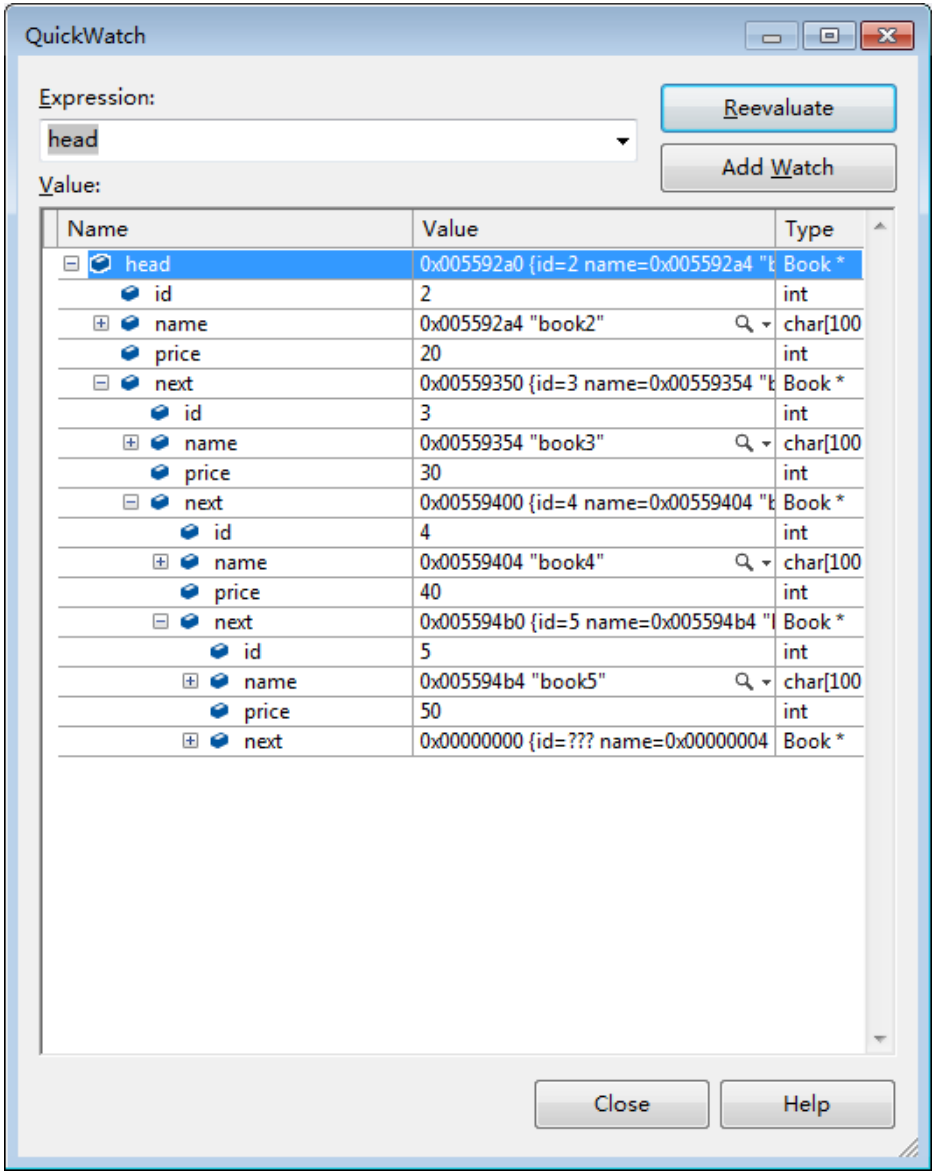
样例输出：

2 book2 20

3 book3 30

4 book4 40

5 book5 50



思考题（3）扑克牌游戏

扑克牌有4种花色：黑桃♠（Spade）、红心❤（Heart）、梅花♣（Club）、方块◆（Diamond）。每种花色有13张牌，编号从小到大为：A,2,3,4,5,6,7,8,9,10,J,Q,K。

对于一个扑克牌堆，定义以下4种操作命令：

1) 添加（Append）：添加一张扑克牌到牌堆的底部。如命令“Append Club Q”表示添加一张梅花Q到牌堆的底部。
2) 抽取（Extract）：从牌堆中抽取某种花色的所有牌，按照编号从小到大进行排序，并放到牌堆的顶部。如命令“Extract Heart”表示抽取所有红心牌，排序之后放到牌堆的顶部。
3) 反转（Revert）：使整个牌堆逆序。
4) 弹出（Pop）：如果牌堆非空，则除去牌堆顶部的第一张牌；如果牌堆为空，则不进行操作。

初始时牌堆为空。输入n个操作命令（$1 \le n \le 1000$），请问最终牌堆的顶部是什么牌？

注意：每种花色和编号的牌数量不限。

思考题（3）扑克牌游戏

输入：

第一行输入一个整数n，表示命令的数量。

接下来的n行，每一行输入一个命令。

输出：

输出共1行。如果最终牌堆为空，输出“null”；如果最终牌堆非空，输出牌堆顶部的牌的花色和编号（字母或数字），用空格分隔，并注意字母的大小写。

样例输入：

```
6
Append Club Q
Append Diamond 5
Append Club 10
Extract Club
Revert
Pop
```

样例输出：

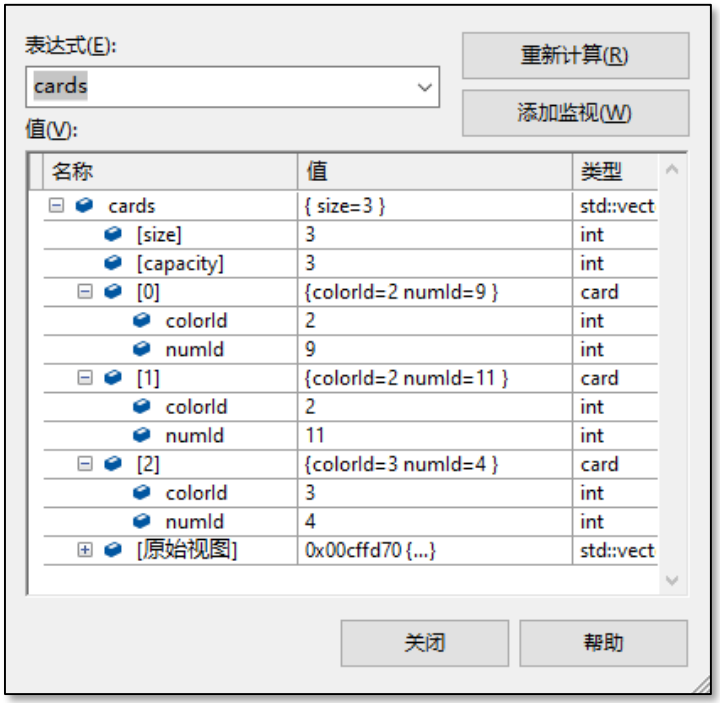
```
Club Q
```

```
#define _CRT_NONSTDC_NO_DEPRECATE
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <iostream>
#include <string>
#include <vector>

using namespace std;
struct card
{
    int colorId;
    int numId;
};



int main()
{

    vector<string> colorMap;
    vector<string> numMap;

    colorMap.push_back("Spade");
    colorMap.push_back("Heart");
    colorMap.push_back("Club");
    colorMap.push_back("Diamond");

    numMap.push_back("A");
    numMap.push_back("2");
    numMap.push_back("3");
    numMap.push_back("4");
    numMap.push_back("5");
    numMap.push_back("6");
    numMap.push_back("7");
    numMap.push_back("8");
    numMap.push_back("9");
    numMap.push_back("10");
    numMap.push_back("J");
    numMap.push_back("Q");
    numMap.push_back("K");
```

```
    numMap.push_back("Q");
    numMap.push_back("K");

    int n;
    cin>>n;

    vector<card> cards;

    for(int i=0;i<n;i++)
    {
        string opt;
        cin>>opt;

        if(opt.compare("Append")==0){
            string color, num;
            cin>>color>>num;

            card c;

            for(int j=0;j<colorMap.size();j++){
                if(color.compare(colorMap[j])==0){
                    c.colorId=j;
                    break;
                }
            }

            for(int j=0;j<numMap.size();j++){
                if(num.compare(numMap[j])==0){
                    c.numId=j;
                    break;
                }
            }

            cards.push_back(c);

        }else if(opt.compare("Extract")==0){
            string color;
            cin>>color;

            int colorId=-1;
```


表达式(E):
cards
重新计算(R)
添加监视(W)
值(V):
名称 值 类型
cards { size=3 } std::vector<card,
[size] 3 int
[capacity] 3 int
[0] {colorId=2 numId=11 } card
colorId 2 int
numId 11 int
[1] {colorId=3 numId=4 } card
colorId 3 int
numId 4 int
[2] {colorId=2 numId=9 } card
colorId 2 int
numId 9 int
[原始视图] 0x00cffd70 {...} std::vector<card,
关闭
帮助


表达式(E):
extractedCards
重新计算(R)
添加监视(W)
值(V):
名称 值 类型
extractedCards { size=2 } std::vect
[size] 2 int
[capacity] 2 int
[0] {colorId=2 numId=9 } card
colorId 2 int
numId 9 int
[1] {colorId=2 numId=11 } card
colorId 2 int
numId 11 int
[原始视图] 0x00cffc7c {...} std::vect
关闭
帮助


表达式(E):
cards
重新计算(R)
添加监视(W)
值(V):
名称 值 类型
cards { size=3 } std::vect
[size] 3 int
[capacity] 3 int
[0] {colorId=2 numId=9 } card
colorId 2 int
numId 9 int
[1] {colorId=2 numId=11 } card
colorId 2 int
numId 11 int
[2] {colorId=3 numId=4 } card
colorId 3 int
numId 4 int
[原始视图] 0x00cffd70 {...} std::vect
关闭
帮助

```
    }else if(opt.compare("Extract")==0){
        string color;
        cin>>color;

        int colorId=-1;

        for(int j=0;j<colorMap.size();j++){
            if(color.compare(colorMap[j])==0){
                colorId=j;
                break;
            }
        }


        vector<card> extractedCards;
        for(int j = cards.size()-1;j>=0;j--){
            if(cards[j].colorId==colorId){
                bool inserted = false;
                for(int k=0;k<extractedCards.size();k++){
                    if(extractedCards[k].numId>=cards[j].numId){
                        extractedCards.insert(extractedCards.begin()+k, cards[j]);
                        inserted = true;
                        break;
                    }
                }
                if(!inserted){
                    extractedCards.push_back(cards[j]);
                }
                cards.erase(cards.begin()+j);
            }
        }

        for(int k=extractedCards.size()-1;k>=0;k--){
            cards.insert(cards.begin(), extractedCards[k]);
        }

        extractedCards.clear();

    }else if(opt.compare("Revert")==0){
```

```
        }else if(opt.compare("Revert")==0){
            int size = cards.size();
            for(int j=0;j<size/2;j++){
                card temp = cards[j];
                cards[j]=cards[size-1-j];
                cards[size-1-j]=temp;
            }
        }else if(opt.compare("Pop")==0){
            if(cards.size()>0)
                cards.erase(cards.begin());
        }

    }

    if(cards.size()==0){
        cout<<"null"<<endl;
    }else{
        card topCard = cards[0];
        cout<<colorMap[topCard.colorId]<<" "<<numMap[topCard.numId];
    }

    return 0;
}
```

样例输入：

6

Append Club Q

Append Diamond 5

Append Club 10

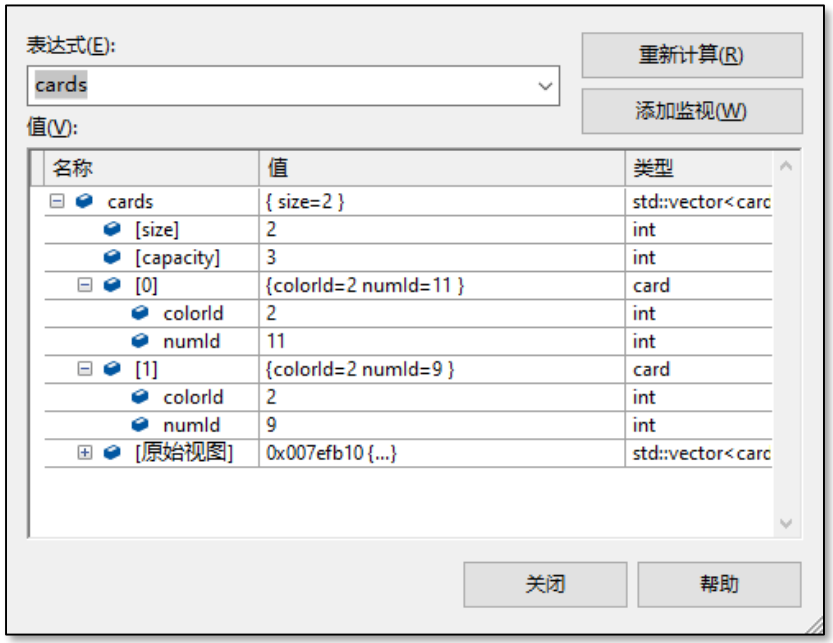
Extract Club

Revert

Pop

样例输出：

Club Q



C++ STL vector容器

vector 容器是 STL 中最常用的容器之一，它和 array 容器非常类似，都可以看做是对 C++ 普通数组的“升级版”。不同之处在于，array 实现的是静态数组（容量固定的数组），而 vector 实现的是一个动态数组，即可以进行元素的插入和删除，在此过程中，vector 会动态调整所占用的内存空间，整个过程无需人工干预。

vector 容器以类模板 vector<T>（T 表示存储元素的类型）的形式定义在 <vector> 头文件中，并位于 std 命名空间中。因此，在创建该容器之前，代码中需包含如下内容：

```
#include <vector>

using namespace std;
```

C++ STL vector容器

2.3插入元素：

```
1  /// Inserts a new element at the end.
2  void push_back(const T&);
3  /// Inserts x before pos.
4  iterator insert(iterator pos, const T& x);
5  /// Inserts the range [first, last) before pos.
6  template <class InputIterator> void insert(iterator pos, InputIterator f, InputIterator l);
7  /// Inserts n copies of x before pos.
8  void insert(iterator pos, size_type n, const T& x);
```

2.4删除元素：

```
1  /// Removes the last element.
2  void pop_back();
3  /// Erases the element at position pos.
4  iterator erase(iterator pos);
5  /// Erases the range [first, last).
6  iterator erase(iterator first, iterator last);
7  /// Erases all of the elements.
8  void clear();
9  /// Inserts or erases elements at the end such that the size becomes n.
10 void resize(n, t = T());
```

C++ STL vector容器

2.5返回元素指针或元素：

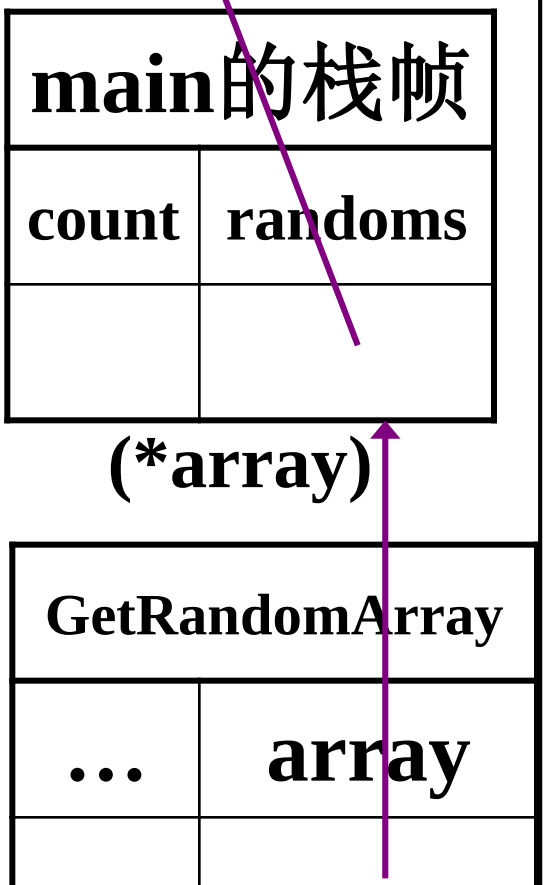
```
1  /// Returns an iterator pointing to the beginning of the vector.
2  iterator begin();
3  /// Returns an iterator pointing to the end of the vector.
4  iterator end();
5  /// Returns a reverse_iterator pointing to the beginning of the reversed vector.
6  reverse_iterator rbegin();
7  /// Returns a reverse_iterator pointing to the end of the reversed vector.
8  reverse_iterator rend();
9  /// Returns the n'th element.
10 reference operator[](size_type n);
11 /// Returns the first element.
12 reference front();
13 /// Returns the last element.
14 reference back();
```

C++ STL vector容器

表 1 删除 vector 容器元素的几种方式

函数	功能
pop_back()	删除 vector 容器中最后一个元素，该容器的大小（size）会减 1，但容量（capacity）不会发生改变。
erase(pos)	删除 vector 容器中 pos 迭代器指定位置处的元素，并返回指向被删除元素下一个位置元素的迭代器。该容器的大小（size）会减 1，但容量（capacity）不会发生改变。
swap(beg)、pop_back()	先调用 swap() 函数交换要删除的目标元素和容器最后一个元素的位置，然后使用 pop_back() 删除该目标元素。
erase(beg,end)	删除 vector 容器中位于迭代器 [beg,end)指定区域内的所有元素，并返回指向被删除区域下一个位置元素的迭代器。该容器的大小（size）会减小，但容量（capacity）不会发生改变。
remove()	删除容器中所有和指定元素值相等的元素，并返回指向最后一个元素下一个位置的迭代器。值得一提的是，调用该函数不会改变容器的大小和容量。
clear()	删除 vector 容器中所有的元素，使其变成空的 vector 容器。该函数会改变 vector 的大小（变为 0），但不是改变其容量。

动态数组




```
int  main()
{
        int  count,  *randoms;
        count  =  GetRandomArray(&randoms);
        printf("最后的那个随机数是： %d\n",
                randoms[count-1]);
}
int GetRandomArray(int **array)
{
        int  i,   count;
        printf("需要多少随机数? ");
        scanf("%d", &count);
        *array = (int *) malloc(count * sizeof(int));
        srand((unsigned)time(NULL));
        for (i = 0; i < count; i++)
              (*array)[i] = rand( );
        return count;
}
```

需要多少随机数? 5
最后的那个随机数是：10971

```
int  main()
{
        int  count,  *randoms;
        count  =  GetRandomArray(randoms);
        printf("最后的那个随机数是： %d\n",
                randoms[count-1]);
}
int GetRandomArray(int *&array)
{
        int  i,   count;
        printf("需要多少随机数? ");
        scanf("%d", &count);
        array = (int *) malloc(count * sizeof(int));
        srand((unsigned)time(NULL));
        for (i = 0; i < count; i++)
              array [i] = rand( );
        return count;
}
```

```
需要多少随机数? 5
最后的那个随机数是：10971
```

环形链表

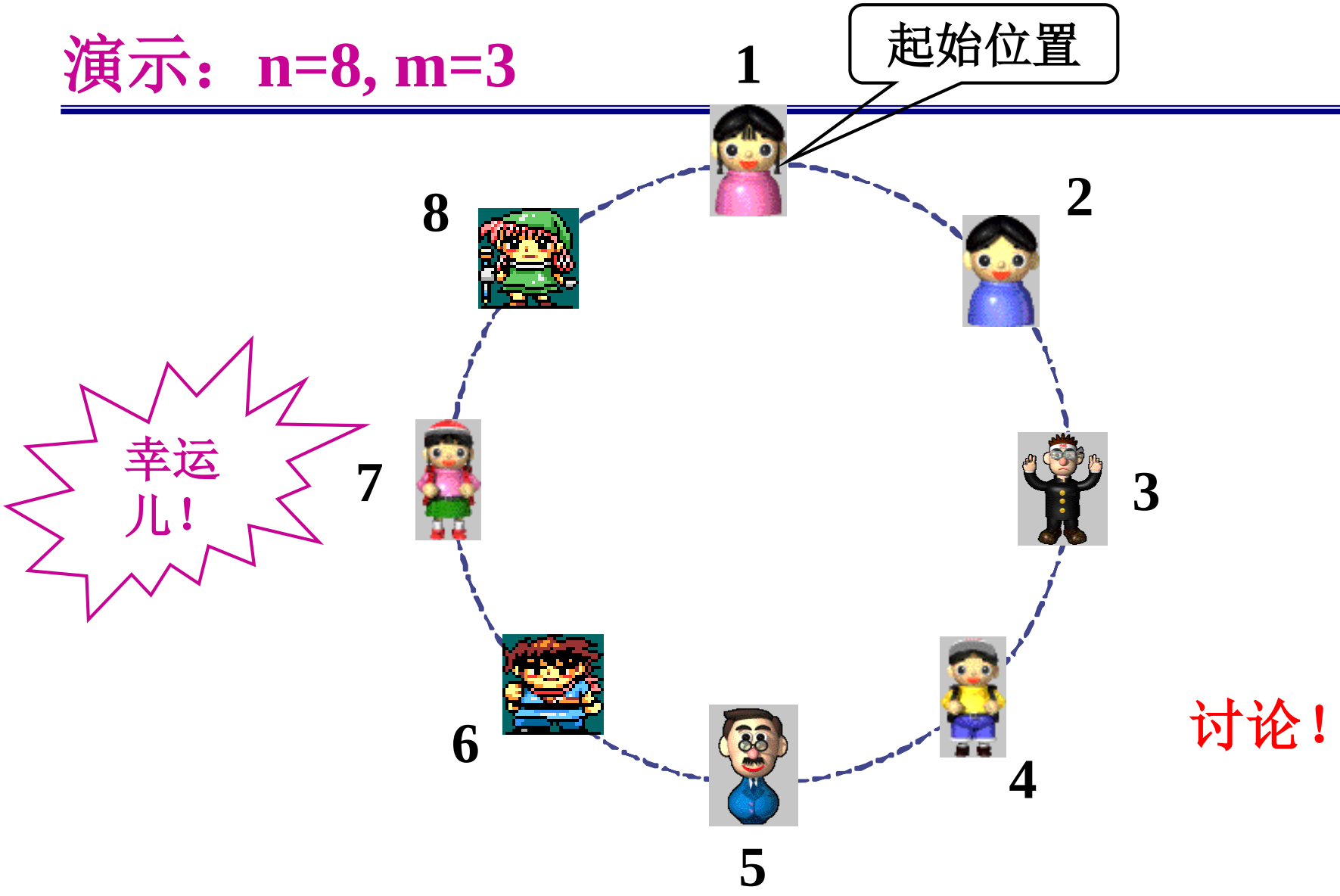
环形链表：一种特殊的链表，其尾结点的next指针，又指向了链表的首结点，从而形成了一个圆环。

从环形链表的任何一个结点出发，都可以遍历整个的链表。

问题描述：

学校给高一（三）班分配了一个名额，去参加奥运会的开幕式。每个人都争着要去，可是名额只有一个，怎么办？班长想出了一个办法，让班上的所有同学围成一圈，按照顺时针方向进行编号。然后随便选定一个数**m**，并且从**1**号同学开始按照顺时针方向依次报数，**1, 2…m**，凡报到**m**的同学，都要主动退出圈子。然后不停地按顺时针方向逐一让报出**m**者出圈，最后剩下的那个人就是去参加开幕式的人。

演示： n=8, m=3

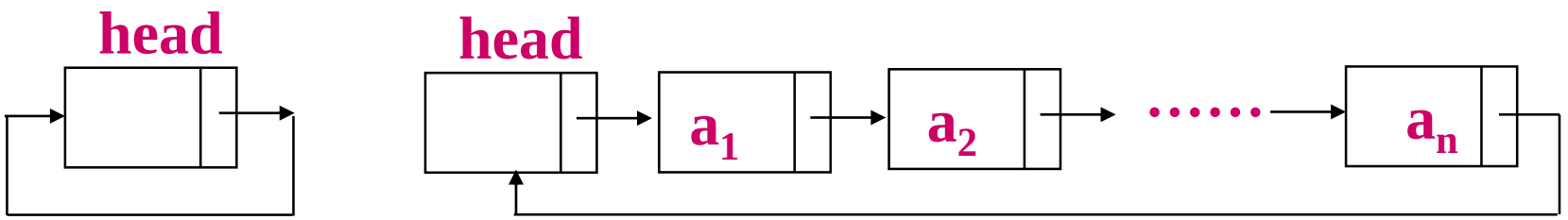


退出圈子的顺序： 3　6　1　5　2　8　4

环形链表

环形(循环)链表 (Circular linked list)：是一种头尾相接的链表。其特点是最后一个结点的指针域指向链表的头结点，整个链表指针域链接成一个环。

从循环链表的任意一个结点出发都可以找到链表中的其它结点，使得表处理更加方便灵活。



环形链表的操作

对于环形链表，除链表的合并外，其它的操作(链表创建、查找、插入、删除等)与单向链表基本上一致，仅仅需要在单向链表操作算法基础上作以下简单修改：

(1) 判断是否是空链表：**head->next == head;**

(2) 判断是否是表尾结点：**p->next == head;**

环形链表操作常常增加**tail**来记录尾指针，能使环形链表上的某些运算变得方便。

参考程序框架

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

// 定义一个名为STUDENT的结构体类型
struct STUDENT
{
    int number;                    // 表示同学的编号
    struct STUDENT  *next;         // 指向下一位同学
};

struct STUDENT *CreateList (int n, struct STUDENT *&tail);
struct STUDENT *Select (struct STUDENT *head,
                        struct STUDENT *tail, int m);
```

```
int main()
{
    struct STUDENT *head, *tail, *p;
    int n, m;
    printf("请输入总人数: ");
    scanf("%d", &n);
    printf("请输入间隔数: ");
    scanf("%d", &m);
    if (n < 1)    return 0;        // 人数小于1, 不处理

    head = CreateList (n, tail);        // 创建环形链表

    p = Select (head, tail, m);         // 找出参加的人
    printf("参加的人是: %d\n", p->number);
    if (p != NULL)     free(p);         // 释放空间
    return 0;
}
```

```
struct STUDENT *CreateList (int n, struct STUDENT *&tail)
{
    int i;
    struct STUDENT *head, *p, *q;
    head = p = q = NULL;    // 结点初始化
    for (i = 1; i <= n; i++) {
        p = (struct STUDENT *)malloc(sizeof(struct STUDENT));
        p->number = i;          // 新建结点的数据域
        p->next = NULL;         // 新建结点的指针域
        if (head == NULL)       // 新建的是首结点
            { head = p;           q = p; }
        else                    // 不是首结点
            { q->next = p;    q = p; }
    }
    q->next = head;         // 链表尾部指向链表头, 形成环形链表
    tail = q;               // 记录链表尾指针
    return head;            // 返回链表头指针
}
```

```
struct STUDENT *Select (struct STUDENT *head,
                        struct  STUDENT *tail,   int m)
{
    struct STUDENT *p, *q;
    int num = 0;                // 计数器
    q = tail;                   // 指向环形链表尾部
    do  {
        p = q->next;
        num ++;
        if (num % m == 0) {     // 是否跳过指定间隔
            q->next = p->next;  // 从环形链表移除该结点
            printf("退出的人是: %d\n", p->number);
            free(p);            // 释放空间
            p = NULL;
        }
        else
            q = p;
    } while (q != q->next);     // 剩余结点个数不为1, 则继续循环
    return q;
}
```

```
请输入总人数: 8
请输入间隔数: 3

退出的人是: 3
退出的人是: 6
退出的人是: 1
退出的人是: 5
退出的人是: 2
退出的人是: 8
退出的人是: 4

参加的人是: 7
```

```
请输入总人数: 9
请输入间隔数: 4

退出的人是: 4
退出的人是: 8
退出的人是: 3
退出的人是: 9
退出的人是: 6
退出的人是: 5
退出的人是: 7
退出的人是: 2

参加的人是: 1
```

```
请输入总人数: 1
请输入间隔数: 3

参加的人是: 1
```

扩展：双向链表

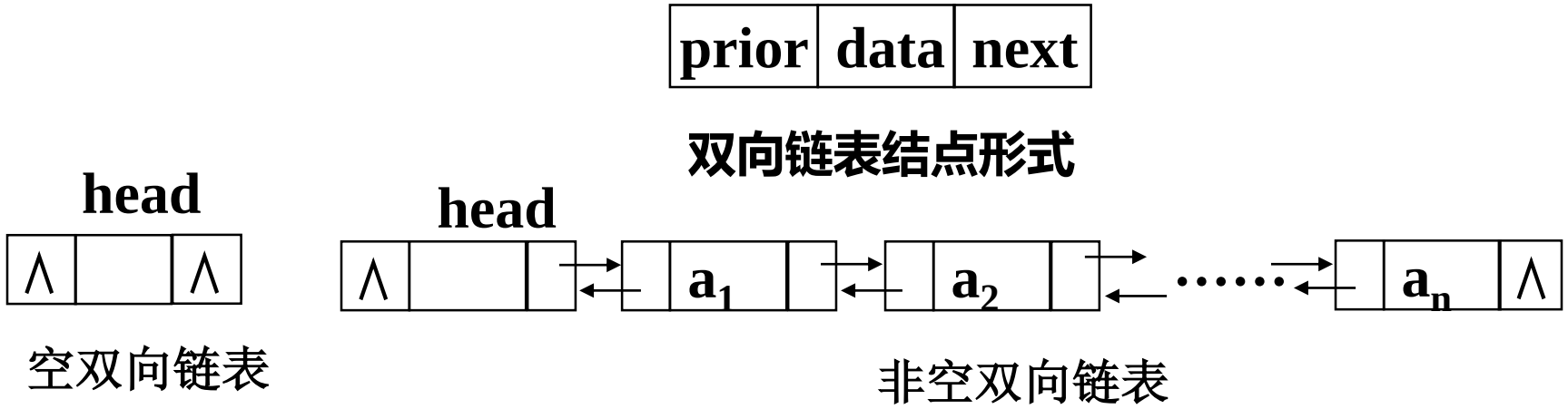
双向链表 (Double linked list)：指的是构成链表的每个结点中设立两个指针域：一个指向其直接前趋的指针域prior，一个指向其直接后继的指针域next。这样形成的链表中有两个方向不同的链，故称为双向链表。

- 和单向链表类似，双向链表一般增加头指针也能使双链表上的某些运算变得方便。
- 将头结点和尾结点链接起来也能构成循环链表，并称之为双向循环链表。
- 双向链表是为了克服单链表的单向性不足而引入的。

双向链表的数据结构

双向链表的数据结构：

```
struct DulNode
    {
        ElemType  data;
        struct DulNode  *prior, *next;
    };
```



- **基本的数据结构：**
  - 线性表：单向链表、循环链表、双向链表
  - 栈和队列
  - 串
  - 数组和广义表
  - 树和二叉树
  - 图
  - 动态存储管理,哈希表
  - ……