第 3 章 链表及链式栈、链式队列

3.1 指针（Pointer）

1、什么是指针？

指针（Pointer）：变量的地址，通过它能找到以它为地址的内存单元。

例子：理解指针的概念，区分什么是地址（指针），什么是地址指向的值！

理解如何通过指针修改变量的值！

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int x = 10;
    //定义指针，指向 x 的地址
    int *p = &x;
    //p 是 int*类型的指针（整数的地址）
    cout<<p<<endl;
    //*p 不是地址，是 p 这个地址指向的值
    cout<<*p<<endl;

    *p = *p + 2;//修改指针指向的变量的值
    cout<<p<<" "<<*p<<endl;
    cout<<x<<endl;

    //数组的本质是数组中下标为 0 的元素的地址
    int arr[5] = {0};
    cout<<&arr<<" "<<&arr[0]<<" "<<arr<<endl;

    //采用 scanf 读入
    int a,b;
    scanf("%d%d",&a,&b);
    printf("a=%d\n",a);
    printf("b=%d\n",b);
    printf("%d+%d=%d\n",a,b,a+b);


   return 0;
}
```

问题：对比如下代码输出的结果，理解指针和普通变量的不同；

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int x = 10;
    int y = x;
    y = y + 2;
    cout<<y<<endl<<x<<endl;

    int *p = &x;
    *p = *p + 2;
    cout<<*p<<endl<<x<<endl;

    int a = 10;
    int *p2 = &a;
    cout<<p2<<" "<<a<<endl;
    (*p2)++;//注意++的优先级高于*
    cout<<p2<<" "<<a<<endl;

    return 0;
}
```

例子：*p++和(*p)++的区别

```
int x = 10;
int *p = &x;
cout<<p<<endl<<x<<endl;
*p++;
cout<<p<<endl<<x<<endl;
```

注意：上述代码并不能通过指针修改 x 的值，*p++相当于 p++是在对指针的值自增（相当于地址值自增）。

2、结构体指针示例

例子：理解结构体指针的各种定义方法。

```
#include <bits/stdc++.h>
using namespace std;

struct stu{
    char name[100];
    double score;
};

int main(){
    //定义结构体变量
    struct stu s;
    strcpy(s.name,"ZhangSan");
    s.score = 99.99;
    //输出结构体
    cout<<s.name<<" "<<s.score<<endl;

    //定义指针，指向结构体 struct stu *stu = &s;//struct 可以省略
    stu *stu1 = &s;
    //在结构体指针中，不能用.来指向结构体成员变量(stu 是地址)
    cout<<stu1->name<<" "<<stu1->score<<endl;
    //(*stu)是结构体
    cout<<(*stu1).name<<" "<<(*stu1).score<<endl;

    //new 结构体对象，直接赋值给一个指针
    stu *stu2 = new stu;
    strcpy(stu2->name,"ZhangSan");
    stu2->score = 99.8;
    cout<<stu2->name<<" "<<stu2->score<<endl;
    delete stu2;//释放内存

    //malloc 结构体对象，直接赋值给一个指针
    stu *stu3 = NULL;
    stu3 = (stu*)malloc(sizeof(stu));
    strcpy(stu3->name,"WangWu");
    stu3->score = 98;
    cout<<stu3->name<<" "<<stu3->score<<endl;
    free(stu3);//释放内存
   return 0;
}
```

注意：理解 malloc 和 free 函数的作用！

（1）malloc：动态内存分配，用于申请一块连续的指定大小的内存块区域并返回分配的内存区域地址。

（2）free 函数能释放某个动态分配的地址，表明不再使用这块动态分配的内存了，实现把之前动态申请的内存返还给系统。

（3）如果结构体中有 string 类型的成员，那么不能使用 malloc 动态分配内存（可以使用 new），这是因为 malloc/free 和 new/delete 的重要区别之一是：

new 会先调用 operator new 函数，申请足够的内存（通常底层使用 malloc 实现）。然后调用类型的构造函数，初始化成员变量，最后返回自定义类型指针。delete 先调用析构函数，然后调用 operator delete 函数释放内存（通常底层使用 free 实现）。

malloc/free 是库函数，只能动态的申请和释放内存，无法强制要求其做自定义类型对象构造和析构工作。

所以 string 的构造函数并没有被调用，name 没有被初始化，改为 new 能正常运行。

例子：通过函数输出结构体的 2 种方法示例

```
#include <bits/stdc++.h>
using namespace std;

struct stu {
    string name;
    double score;
};

void show(struct stu s) {
    cout<<s.name<<" "<<s.score<<endl;
}

void show2(struct stu *s) {
    cout<<s->name<<" "<<s->score<<endl;
}

int main() {
    struct stu s;
    s.name = "ZhangSan";
    s.score = 99;
    show(s);

    stu *t = &s;
    show2(t);
    return 0;
}
```

3.2 链表

1、链表介绍

（1）线性表的优点

A、无需为表示结点间的逻辑关系而增加额外的存储空间；

B、可方便地随机存取表中的任一元素。

（2）线性表的缺点

A、插入或删除平均需要移动一半的结点；

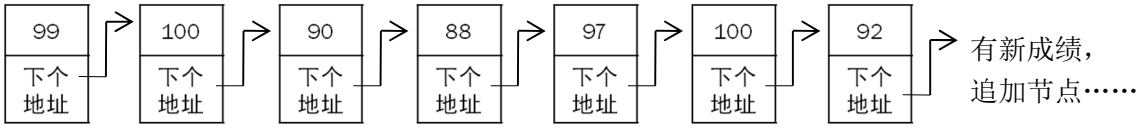
B、顺序表要求占用连续的存储空间；

问题：存储一个班同学的成绩（人数不确定），如何存储？

解决方法 1：定义尽可能长的数组来存储！

99	100	98	90	88	97	100	92	…	98

解决方法 2：利用链表来存储！



注意：链表结构并不一定占用连续的内存空间，其占用的内存空间可以不连续！

（3）链表的基本概念

A.结点(Node)组成

B.数据域：存储数据元素本身

C.指针域：存储邻接元素的存储地址(位置)

D.链接方式：

单链表：每个结点只有一个指针域。

双向链表：每个结点有两个指针域。

循环链表：是一个首尾相接的链表。

E.头指针 ：指向链表头结点的指针。

2、链表的基本操作

（1）要实现的基本操作：

add(int x)：在链表末尾追加元素（注意头结点）

insert(int n,int x)：在链表中插入一个元素（注意头结点）

deldata(int data)：删除链表中第一个出现的 data 的值（注意：如果 data 是头节点）

delpos(int n)：删除链表的第 n 个节点（注意如果 n==1，也就是删除头结点的情况）

display()：显示链表

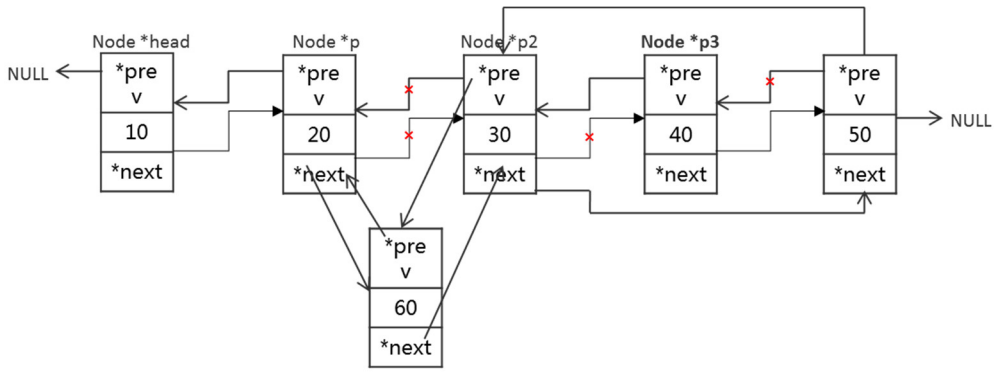
（2）实现代码：

```
#include <bits/stdc++.h>
using namespace std;

//结点定义
struct Node{
  int data;
  struct Node *next;
};

//头结点
Node *head = NULL;

//在链表末尾追加
void add(int x){
   //如果有 head 的值
   if(head != NULL){
      //要追加的节点
      Node *a = new Node;
      a->data = x;
      a->next = NULL;
```

```
        Node *p = head;
        //移动到最后一个节点
        while(p->next != NULL){
            p = p->next;
        }
        p->next = a;
    }else{
        head = new Node;
        head->data = x;
        head->next = NULL;
    }
}

//在中间追加元素（在第 n 个元素的位置）
void insert(int n,int x){
  //新节点
  Node *d = new Node;
  d->data = x;
  d->next = NULL;
  //如果是头结点
  if(n == 1){
        d->next = head;
        head = d;
  }else{
        int i;
        Node *p = head;
        for(i = 1;i <= n - 2;i++){
            p = p->next;
            if(p == NULL){
                break;
            }
        }

        if(p == NULL){
            cout<<"n 有误"<<endl;
        }else{
            d->next = p->next;
            p->next = d;
        }
  }
}

//删除链表中第一个出现的 data
void deldata(int data){
  Node *p = head,*pre = NULL;
  while(p != NULL){
        if(data == p->data){
            if(p == head){
                head = p->next;
            }else{
                pre->next = p->next;
            }

            delete p;
            break;
        }

        pre = p;
        p = p->next;
  }
}

//删除某个位置的元素
void delpos(int n){
  Node *p = head,*t;

  //如果要删除头节点
  if(n == 1){
```

```
            if(head != NULL){
                head = head->next;
                delete p;
            }else{
                cout<<"链表空"<<endl;
            }
        }else{
            int i;
            //移动到要删除位置之前的节点
            for(i = 1;i <= n - 2;i++){
                p = p->next;
                if(p == NULL) break;
            }

            if(p == NULL || p->next == NULL){
                cout<<"n 的值有误!"<<endl;
            }else{
                t = p->next;
                p->next = t->next;
                delete t;
            }
        }

    }

    //输出链表
    void display(){
        Node *p = head;
        while(p != NULL){
            cout<<p->data<<" ";
            p = p->next;
        }

        cout<<endl;
    }

    int main(){
        int order,x,p;
        cout<<"输入指令:";
        while(1 == 1){
            cout<<"1:追加,2:插入,3:删除值,4:删除位置,5:显示!"<<endl;
            cin>>order;
            if(order == 1){
                cin>>x;
                add(x);
                display();
            }else if(order == 2){
                cin>>p>>x;
                insert(p,x);
                display();
            }else if(order == 3){
                cin>>x;
                deldata(x);
                display();
            }else if(order == 4){
                cin>>x;
                delpos(x);
                display();
            }else if(order == 5){
                display();
            }
        }
    }
```

注意：双向链表的基本操作；



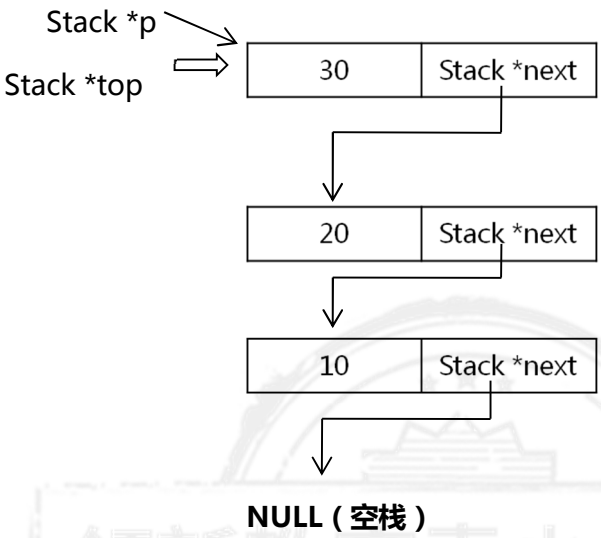
插入的实现：

- e->prev = p;
- e->next = p->next;
- p->next->prev=e;
- p->next = e;

删除的实现：

- p2->next = p2->next->next;
- p3->next->prev=p2;
- delete p3;

3、链式栈



NULL（空栈）

入栈的实现：

- p->next = top;
- top = p;//修改栈顶位置

出栈的实现：

- p = top;
- p = p->next;
- delete top;
- top = p;

（1）栈的基本操作

push(int x)：入栈

pop()：出栈

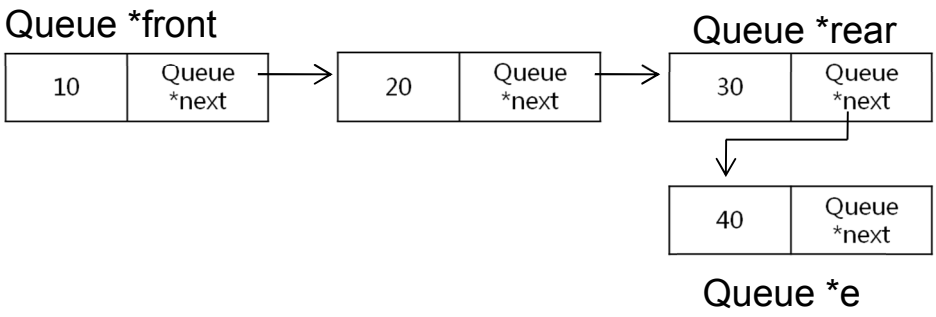
display()：显示栈

（2）实现代码

```
#include <bits/stdc++.h>
using namespace std;
//栈元素
struct Stack{
    int data;
    struct Stack *next;
};

Stack *top = NULL;//栈顶指针

//入栈
void push(int x){
    Stack *p = new Stack;
    p->data = x;
    p->next = top;
    top = p;//修改栈顶位置
}

//出栈
void pop(){
    Stack *p = top;
    if(p != NULL){
        cout<<p->data<<"出栈"<<endl;
        p = p->next;
        delete top;
        top = p;//修改指针位置
    }else{
        cout<<"栈空"<<endl;
    }

}

//获得栈长度
int getlen(){
    int len = 0;
    Stack *p = top;
    while(p != NULL){
        len++;
        p = p->next;
    }
    return len;
}

//显示栈元素
void display(){
    Stack *p = top;
    while(p != NULL){
        cout<<p->data<<" ";
        p = p->next;
    }
    cout<<endl;
}

int main(){
    int order,x;
    cout<<"输入指令:"<<endl;
    while(1 == 1){
        cout<<"1:入栈,2:出栈,3:显示,4:求栈长!"<<endl;
        cin>>order;
        if(order == 1){
```

```
            cin>>x;
            push(x);
            display();
        }else if(order == 2){
            pop();
            display();
        }else if(order == 3){
            display();
        }else if(order == 4){
            cout<<getlen()<<endl;
        }
    }
    return 0;
}
```

4、链式队列



入队的实现：

- rear->next = e;
- rear = e;//修改尾指针位置

出队的实现：

- t = front;
- front=front->next;
- if(front == NULL) rear = NULL;
- delete t;

（1）队列的基本操作

add(int x)：入队

del()：出队

display()：显示队

（2）实现代码

```
#include <bits/stdc++.h>
using namespace std;

//队列节点
struct Queue{
    int data;
    struct Queue *next;
};

Queue *front = NULL;//队头
Queue *rear = NULL;//队尾

//入队
void add(int value){
```

```
    Queue *e = new Queue;
    e->data = value;
    e->next = NULL;
    //队列第一个元素
    if(front == NULL){
        front = e;
    }else{
        rear->next = e;
    }

    rear = e;
}
//出队
void del(){
    Queue *t;
    //如果对列有元素
    if(front != NULL){
        cout<<front->data<<"出队"<<endl;
        t = front;
        front = front->next;
        if(front == NULL) rear = NULL;
        delete t;
    }else{
        cout<<"队列空"<<endl;
    }
}

//显示队
void display(){
    Queue *p = front;
    while(p != NULL){
        cout<<p->data<<" ";
        p = p->next;
    }
    cout<<endl;
}

int main(){
    int order,x;
    cout<<"输入指令:"<<endl;
    while(1 == 1){
        cout<<"1:入队,2:出队,3:显示队!"<<endl;
        cin>>order;
        if(order == 1){
            cin>>x;
            add(x);
            display();
        }else if(order == 2){
            del();
            display();
        }else if(order == 3){
            display();
        }
    }
   return 0;
}
```

5、链表练习题

【noip2019 普及组】 6.链表不具有的特点是(　　)

A.插入删除不需要移动元素　　B.不必事先估计存储空间

C.所需空间与线性表长度成正比　　D.可随机访问任一元素

答案：D

解析:链表是通过记录每个元素的后继位置来实现数据存储,所需空间与元素个数成正比,优点是不必事先估计存储空间、插入或删除指定位置元素的时间复杂度为 O(1)；但缺点是由于其元素的内存地址不连续,无法进行 O(1)的随机访问。

【noip2017 普及组】13.向一个栈顶指针为 hs 的链式栈中插入一个指针 s 指向的结点时，应执行（　）。

A.hs->next = s;

B.s->next = hs; hs = s;

C.s->next = hs->next; hs->next = s;

D.s->next = hs; hs = hs->next;

答案：B

【noip2015 普及组】14. 线性表若采用链表存储结构，要求内存中可用存储单元地址(　)

A.必须连续　B.部分地址必须连续　C. 一定不连续　D.连续不连续均可

答案：D

【noip2015 提高组】13.双向链表中有两个指针域，llink 和 rlink，分别指回前驱及后继，设 p 指向链表中的一个结点，q 指向一待插入结点，现要求在 p 前插入 q，则正确的插入为（　）。

A.　p->llink = q;　q->rlink = p;

　　p->llink->rlink = q;　q->llink = p->llink;

B.　q->llink = p->llink;　p->llink->rlink = q;

　　q->rlink = p;　p->llink = q->rlink;

C.　q->rlink = p;　p->rlink = q;

　　p->llink->rlink = q;　q->rlink = p;

D.　p->llink->rlink = q;　q->rlink = p

　　q->llink = p->llink;　p->llink = q;

答案：D

3.3 链表习题

1、用单链表表示的链式队列的队头在链表的（　　）位置。

A. 链头　　B. 链尾　　C. 链中　　D. 以上都不是

2、在双向循环链表中，在 p 所指的结点之后插入 s 指针所指的结点，其操作是(　　)。

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

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

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

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

3、向一个栈顶指针为 hs 的链栈中插入一个 s 结点时，应执行(　　)。

A. hs->next=s;

B. s->next=hs; hs=s;

C. s->next=hs->next; hs->next=s;

D. s->next=hs; hs=hs->next;

4、在一个链队列中，假定 front 和 rear 分别为队首和队尾指针，则删除一个结点的操作为(　　)。

A. front=front->next　　B. rear=rear->next

C. rear=front->next　　D. front=rear->next

5、用链接方式存储的队列，在进行删除运算时（　　）。

A. 仅修改头指针　　B. 仅修改尾指针

C. 头、尾指针都要修改　　D. 头、尾指针可能都要修改

6、设链式栈中结点的结构为(data，link)，且 top 是指向栈顶的指针。若想在链式栈的栈顶插入一个由指针 s 所指的结点，则应执行的操作是(　　)。

A. top->link=s;

B. s->link=top->link; top->link=s;

C. s->link=top; top=s;

D. s->link=top; top=top->link;

7、设链式栈中结点的结构为(data，link)，且 top 是指向栈顶的指针。若想摘除链式栈的栈顶结点，并将被摘除结点的值保存到 x 中，则应执行的操作是（　　）。

A. x=top->data; top=top->link;

B. top=top->link; x=top->data;

C. x=top; top=top->link;

D. x=top->data;

8、判断题

（1）链式栈通常会选用链表的表尾一段作为栈顶。（　　）

（2）链式栈与顺序栈相比，一个明显的优点是通常不会出现栈满的情况。（　　）

参考答案：

1.A　2.D　3.B　4.A　5.D

6.C　7.A　8.(1)错　(2)对