数据结构

struct 结构
void 不返回
del delete 删除
deldata 删除数据
add 增加
null 零值
insert 插入
delpos 删除数据
value 数值

pointer 指针
stack 栈
init 建栈
push 入栈
display 显示
pop 出栈
gettop 取栈顶
empty 测试栈
setnull 释放栈
queue 队列
front 队头
rear 队尾
getlen 获得长度
get 获得
len 长度
data 数据

栈

```
#include <iostream>// 栈的代码实现 （选自东方博宜）
// 常量：栈的长度
#define MAXN 5
using namespace std;
int stack[MAXN];// 数组模拟栈
int top = -1;// 初始化栈指针
// 出栈
int pop()
{
	int r;
	if(top < 0)
	{
		r = -1;
		cout<<" 栈空！ "<<endl;
	}
	else
	{
		r = stack[top];
		top--;
	}
	return r;
}
// 入栈
```

```
// 入栈
void push(int value)//value 价值
{
	if(top >= MAXN - 1)
	{
		cout<<" 栈满 "<<endl;
	}
	else
	{
		top++;
		stack[top] = value;
	}
}
// 显示栈
void display()//display 显示
{
	int i;
	for(i = top;i >= 0;i--)
	{
		cout<<stack[i]<<" ";
	}
	cout<<endl;
}
```

```
int main()
{
	int order;//指令
	int x,t;
	cout<<"输入指令：  ";
	while(1 == 1)
	{
		cout<<"1:入栈,2:出栈,3:显示栈!"<<endl;
		cin>>order;
		if(order == 1)
		{
			cin>>x;
			push(x);
			display();
		}
		else if(order == 2)
		{
			t = pop();
			if(t != -1)
			{
				cout<<t<<"出栈!"<<endl;
				display();
			}
		}
		else if(order == 3)
		{
			display();
		}
	}
	return 0;
}
```

队列

```
#include <iostream>// 队列代码实现（选自东方博宜）
#define MAXN 5
using namespace std;
int queue[MAXN] = {0}; // 队列
int front = 0;// 头指针
int rear = 0; // 尾指针

void addqueue(int value)// 入队
{
	if(rear >= MAXN)
	{
		cout<<" 队满！"<<endl;
	}
	else
	{
		queue[rear] = value;
		rear++;
	}
}

// 出队
```

```
int delqueue()//出队
{
	int r;
	if(front == rear)
	{
		cout<<"队空"<<endl;
		r = -1;
	}
	else
	{
		r = queue[front];
		front++;
	}
	return r;
}
//显示队
void display()
{
	int i;
	for(i = front;i < rear;i++)
	{
		cout<<queue[i]<<" ";
	}
	cout<<endl;
}
```

```
int main()
{
	int order;// 口令
	int value,t;
	cout<<" 输入指令 "<<endl;
	while(1 == 1)
	{
		cout<<"1 入队,2 出队,3 显示! "<<endl;
		cin>>order;
		if(order == 1)
		{
			cin>>value;
			addqueue(value);
			display();
		}
		else if(order == 2)
		{
			t = delqueue();
			if(t != -1)
			{
				cout<<t<<" 已经出队 "<<endl;
				display();
			}
			else
			{
				cout<<" 队列已空 "<<endl;
			}
		}
		else if(order == 3)
		{
			display();
		}
		else  {	cout<<" 口令错误! "<<endl;	}
	}
}
```

```cpp
#include <iostream>// 循环队列代码实现 （选自东方博宜）
#define MAXN 5
using namespace std;
int queue[MAXN] = {0}; // 队列
int front = 0;// 头指针
int rear = 0; // 尾指针

void addqueue(int value)// 入队
{

	if(front == 0 && (rear + 1) % MAXN == front)// 元素从未出队的情况下，队满
	{
		cout<<" 队满 "<<endl;

	}
	else if((rear + 1) % MAXN == front)// 有元素出队，队满
	{
		cout<<" 队满 "<<endl;
	}
	else
	{
		queue[rear] = value;
		rear = (rear + 1) % MAXN;
	}
}
```

```
int delqueue()// 出队
{
	int r;
	if(front == rear)
	{
		cout<<" 队空 "<<endl;
		r = -1;
	}
	else
	{
		r = queue[front];
		queue[front] = -1;// 出队标记
		front = (front + 1) % MAXN;
	}
	return r;
}

void display()// 显示队
{
	if(front == rear)
	{
		cout<<" 队空 "<<endl;
	}
	else
	{
		int i = front;
		do
		{
			cout<<queue[i]<<" ";
			i = (i + 1) % MAXN;
		}
		while(i != rear);
	}
	cout<<endl;
}
```

```
int main()
{
	int order;// 口令
	int value,t;
	cout<<" 输入指令 "<<endl;
	while(1 == 1)
	{
		cout<<"1 入队,2 出队,3 显示! "<<endl;
		cin>>order;
		if(order == 1)
		{
			cin>>value;
			addqueue(value);
			display();
		}
		else if(order == 2)
		{
			t = delqueue();
			if(t != -1)
			{
				cout<<t<<" 已经出队 "<<endl;
				display();
			}
			else
			{
				cout<<" 队列已空 "<<endl;
			}
		}
		else if(order == 3)
		{
			display();
		}
		else  {	cout<<" 口令错误! "<<endl;	}
	}
}
```

指针

```
#include<iostream>// 指针输入输出
using namespace std;
int a,b;
int *p,*q;
int main()
{
	cin>>a>>b;
	p=&a;
	q=&b;
	cout<<*p<<" "<<*q<<endl;   // 数值
	cout<<p<<" "<<q;    // 地址
	return 0;
}
```

```
35 67
35 67
0x4a3020 0x4a3024
```

8.2-2

```
#include <bits/stdc++.h>// 指针 （选自东方博宜）
using namespace std;
int main(){
	int x = 10;

	int *p = &x;		// 定义指针，指向 x 的地址

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

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

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

	return 0;
}
```

```
0x6cfed8
10
0x6cfed8 12
12
0x6cfec4 0x6cfec4
0x6cfec4
35 56
a=35
b=56
35+56=91
```

8.2-3

```cpp
#include <bits/stdc++.h>// 指针与普通变量区别（选自东方博宜）
using namespace std;
int main()
{
	int x = 10;
	int y = x;
	y = y + 2;
	cout<<y<<" "<<x<<endl;

	int *p = &x;
	*p = *p + 2;
	cout<<*p<<" "<<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;
}
```

```
12 10
12 12
0x6cfedc 10
11 11
```

8.2-4

```
#include <bits/stdc++.h>// 结构体指针（选自东方博宜）
using namespace std;
struct stu
{
	char name[100];
	double score;
};
int main()
{
	struct stu s;	// 定义结构体变量
	strcpy(s.name,"ZhangSan");//strcpy 字符串复制
	s.score = 99.99;
	cout<<s.name<<" "<<s.score<<endl;	// 输出结构体
	stu *stu1 = &s;	// 定义指针，指向结构体
	// 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;

	cout<<(*stu1).name<<" "<<(*stu1).score<<endl;	//(*stu) 是结构体
	//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;
}
```

```
ZhangSan 99.99
ZhangSan 99.99
ZhangSan 99.99
ZhangSan 99.8
WangWu 98
```

8.2-5

```
#include <bits/stdc++.h>// 通过函数输出结构体（选自东方博宜）
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;
}
```

```
ZhangSan 99
ZhangSan 99
```

链表

本节选自东方博宜

```
#include <bits/stdc++.h>// 单向链表代码实现（选自东方博宜）
using namespace std;

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

Node *head = NULL; // 头结点     null 零值

void add(int x)// 在链表末尾追加
{
	if(head != NULL)// 如果链表中有数据
	{
		Node *a = new Node;             // 要追加的节点
		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// 如果链表中没有数据
	{
		head = new Node;// 创建新节点
		head->data = x;
		head->next = NULL;
	}
}
```

```
void insert(int n,int x)// 在中间追加元素（在第 n 个元素的位置）
{
	Node *d = new Node;// 新节点
	d->data = x;
	d->next = NULL;

	if(n == 1)// 如果是头结点
	{
		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;
		}
	}
}
```

```
void deldata(int data)// 删除某个数值
{
	Node *p = head,*pre = NULL;
	while(p != NULL)// 如果链表中有数据
	{
		if(data == p->data)
		{
			if(p == head)
			{
				head = p->next;
			}
			else
			{
				pre->next = p->next;
			}
			delete p;//delete 删除
			break;
		}
		pre = p;
		p = p->next;// 遍历链表，查找需要删除的数值
	}
}
```

```
void delpos(int n)// 删除某个位置的元素
{
	Node *p = head,*t;
	if(n == 1)  // 如果要删除头节点
	{
		if(head != NULL)
		{
			head = head->next;
			delete p;
		}
		else
		{
			cout<<" 链表空 "<<endl;
		}
	}
	else
	{
		int i;
		for(i = 1;i <= n - 2;i++)		// 移动到要删除位置之前的节点
		{
			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;
		}
	}
}
```

```
void display()//输出链表
{
	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();	}
	}
}
```

```
#include <bits/stdc++.h>// 链式栈代码实现（选自东方博宜）
using namespace std;

struct Stack{// 栈元素
	int data;
	struct Stack *next;
};
Stack *top = NULL;    // 栈顶指针

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

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

void display()    //显示栈元素
{
	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;
}
```

```
#include <bits/stdc++.h>// 链式队列代码实现（选自东方博宜）
using namespace std;

struct Queue{   // 队列节点
	int data;
	struct Queue *next;
};
Queue *front = NULL;    // 队头
Queue *rear = NULL;    // 队尾

void add(int value)    // 入队
{
	Queue *e = new Queue;// 创建新节点
	e->data = value;
	e->next = NULL;

	if(front == NULL) // 如果队列是空的
	{
		front = e;
	}
	else
	{
		rear->next = e;
	}
	rear = e;
}

  // 出队
```

```
void del()    //出队
{
	Queue *t;

	if(front != NULL) //如果队列有元素
	{
		cout<<front->data<<"出队"<<endl;
		t = front;
		front = front->next;
		if(front == NULL) rear = NULL;
		delete t;
	}
	else
	{
		cout<<"队列空"<<endl;
	}
}

void display()    //显示队
{
	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;
}
```