①.如何创建一个单链表？

链表节点的定义：  
typedef struct node  
{  
int data; //节点内容  
node \*next; //下一个节点  
}node;

单链表的创建：  
1 //创建单链表  
2 node \*create()  
3 {  
4 int i = 0; //链表中数据的个数  
5 node \*head, \*p, \*q;  
6 int x = 0;  
7 head = (node \*)malloc(sizeof(node)); //创建头节点  
8  
9 while(1)  
10 {  
11 printf("Please input the data: ");  
12 scanf("%d", &x);  
13 if (x == 0) //data为0时创建结束  
14 break;  
15 p = (node \*)malloc(sizeof(node));  
16 p->data = x;  
17 if (++i == 1)  
18 { //链表只有一个元素  
19 head->next = p; //连接到head的后面  
20 }  
21 else  
22 {  
23 q->next = p; //连接到链表尾端  
24 }  
25 q = p; //q指向末节点  
26 }  
27 q->next = NULL; //链表的最后一个指针为NULL  
28 return head;  
29 }  
上面的代码中，使用while循环每次从终端读入一个整型数据，并调用malloc动态分配链表节点内存存储这个整型数据，然后再插入到单链表的末尾。最后当数据为0时表示插入数据结束，此时把末尾节点的next指针置为NULL。

②.查找单链表中间元素？

解析：  
这里使用一个只用一遍扫描的方法。描述如下：  
假设mid指向当前已经扫描的子链表的中间元素，cur指向当前以扫描链表的尾节点，那么继续扫描即移动cur到cur->next,这时只需判断一下应不应移动mid到mid->next就行了。所以一遍扫描就能找到中间位置。代码如下：  
1 node \*search(node \*head)  
2 {  
3 int i = 0;  
4 int j = 0;  
5 node \*current = NULL;  
6 node \*middle = NULL;  
7   
8 current = middle = head->next;  
9 while(current != NULL)  
10 {  
11 if( i / 2 > j)  
12 {  
13 j++;  
14 middle = middle->next;  
15 }  
16 i++;  
17 current = current->next;  
18 }  
19   
20 return middle;  
21 }

③.打印单向链表？

解析：  
单链表的打印：  
1 //打印单链表  
2 void print(node \*head)  
3 {  
4 node \*p;  
5 int index = 0;  
6 if (head->next == NULL) //链表为空  
7 {  
8 printf("Link is empty!\n");  
9 return;  
10 }  
11 p = head->next;  
12 while(p != NULL) //遍历链表  
13 {  
14 printf("The %dth node is: %d\n", ++index, p->data); //打印元素 ----- 也可计算单链表长度 count++；  
15 p = p->next;  
16 }  
17 }  
单链表的打印与单链表的测长方法类似，使用while循环进行遍历链表所有节点并打印各个节点内容，当遇到NULL时结束循环。

④.查找单链表节点？

单链表的查找节点：  
1 //查找单链表pos位置的节点,返回节点指针  
2 //pos从0开始,0返回head节点  
3 node \*search\_node(node \*head, int pos)  
4 {  
5 node \*p = head->next;  
6 if (pos < 0) //pos位置不正确  
7 {  
8 printf("incorrect position to search node!\n");  
9 return NULL;  
10 }  
11 if (pos == 0) //在head位置，返回head  
12 {  
13 return head;  
14 }  
15 if(p == NULL)  
16 {  
17 printf("Link is empty!\n"); //链表为空  
18 return NULL;  
19 }  
20   
21 while(--pos)  
22 {   
23 if ((p = p->next) == NULL)  
24 { //超出链表返回  
25 printf("incorrect position to search node!\n");  
26 break;  
27 }  
28 }  
29 return p;  
30 }

⑤.单链表插入节点？

解析：  
向单链表中某个位置（第pos个节点）之后插入节点，这里分为插入到链表首部、插入到链表中间，以及链表尾端三种情况：  
1 //在单链表pos位置处插入节点，返回链表头指针  
2 //pos从0开始计算,0表示插入到head节点后面  
3 node \*insert\_node(node \*head, int pos, int data)  
4 {  
5 node \*item = NULL;  
6 node \*p;  
7   
8 item = (node \*)malloc(sizeof(node));  
9 item->data = data;  
10 if (pos == 0) //插入链表头后面  
11 {  
12 head->next = item; //head后面是item  
13 return head;  
14 }  
15 p = search\_node(head, pos); //获得位置pos的节点指针  
16 if (p != NULL)  
17 {  
18 item->next = p->next; //item指向原pos节点的后一个节点  
19 p->next = item; //把item插入到pos的后面  
20 }  
21 return head;  
22 }

⑥.单向链表删除节点？

解析：  
单链表删除节点：  
1 //删除单链表的pos位置的节点，返回链表头指针  
2 //pos从1开始计算，1表示删除head后的第一个节点  
3 node \*delete\_node(node \*head, int pos)  
4 {  
5 node \*item = NULL;  
6 node \*p = head->next;  
7 if (p == NULL) //链表为空  
8 {  
9 printf("link is empty!\n");  
10 return NULL;  
11 }  
12 p = search\_node(head, pos-1); //获得位置pos的节点指针  
13 if (p != NULL && p->next != NULL)  
14 {  
15 item = p->next;  
16 p->next = item->next;  
17 delete item;   
18 }  
19 return head;  
20 }

⑦.单向链表的逆序？

解析：  
这是一个经常被问到的面试题，也是一个非常基础的问题。比如一个链表是这样的： 1->2->3->4->5 通过逆置后成为5->4->3->2->1。  
最容易想到的方法是遍历一遍链表，利用一个辅助指针，存储遍历过程中当前指针指向的下一个元素，然后将当前节点元素的指针反转后，利用已经存储的指针往后面继续遍历。  
1 node \*reverse(node \*head)  
2 {  
3 node \*p, \*q, \*r;   
4   
5 if (head->next == NULL) //链表为空  
6 {  
7 return head;  
8 }  
9  
10 p = head->next;  
11 q = p->next; //保存原第二个节点  
12 p->next = NULL; //原第一个节点为末节点  
13   
14 while(q != NULL) //遍历，各个节点的next指针反转  
15 {  
16 r = q->next;  
17 q->next = p;  
18 p = q;  
19 q = r;  
20 }  
21 head->next = p; //新的第一个节点为原末节点  
22 return head;  
23 }

⑧.单链表的正向排序？

解析：  
下面试结构体定义和代码如下：  
1 typedef struct node  
2 {  
3 int data;  
4 node \*next;  
5 }node;  
6  
7 node\* InsertSort(void)  
8 {  
9 int data = 0;  
10 struct node \*head=NULL,\*New,\*Cur,\*Pre;  
11 while(1)  
12 {  
13 printf("please input the data\n");   
14 scanf("%d", &data);  
15 if (data == 0) //输入0结束  
16 {  
17 break;  
18 }  
19 New=(struct node\*)malloc(sizeof(struct node));  
20 New->data = data; //新分配一个node节点  
21 New->next = NULL;  
22 if(head == NULL)  
23 { //第一次循环时对头节点赋值  
24 head=New;  
25 continue;  
26 }  
27 if(New->data <= head->data)  
28 {//head之前插入节点  
29 New->next = head;  
30 head = New;  
31 continue;  
32 }  
33 Cur = head;  
34 while(New->data > Cur->data && //找到需要插入的位置  
35 Cur->next!=NULL)  
36 {  
37 Pre = Cur;  
38 Cur = Cur->next;  
39 }  
40 if(Cur->data >= New->data) //位置在中间  
41 { //把new节点插入到Pre和cur之间  
42 Pre->next = New;  
43 New->next = Cur;  
44 }  
45 else //位置在末尾  
46 Cur->next = New; //把new节点插入到cur之后  
47 }  
48 return head;  
49 }

⑨.有序单链表的合并？

已知两个链表head1 和head2 各自有序，请把它们合并成一个链表依然有序。使用非递归方法以及递归方法。  
解析：  
首先介绍非递归方法。因为两个链表head1 和head2都是有序的，所以我们只需要找把较短链表的各个元素有序的插入到较长的链表之中就可以了。  
源代码如下：  
1 node\* insert\_node(node \*head, node \*item) //head != NULL  
2 {  
3 node \*p = head;  
4 node \*q = NULL; //始终指向p之前的节点  
5   
6 while(p->data < item->data && p != NULL)  
7 {  
8 q = p;  
9 p = p->next;  
10 }  
11 if (p == head) //插入到原头节点之前  
12 {  
13 item->next = p;  
14 return item;  
15 }  
16 //插入到q与p之间之间  
17 q->next = item;  
18 item->next = p;  
19 return head;  
20 }  
21   
22 /\* 两个有序链表进行合并 \*/  
23 node\* merge(node\* head1, node\* head2)  
24 {   
25 node\* head; //合并后的头指针  
26 node \*p;   
27 node \*nextP; //指向p之后  
28   
29 if ( head1 == NULL ) //有一个链表为空的情况，直接返回另一个链表  
30 {  
31 return head2;  
32 }  
33 else if ( head2 == NULL )  
34 {  
35 return head1;  
36 }  
37   
38 // 两个链表都不为空  
39 if(length(head1) >= length(head2)) //选取较短的链表  
40 { //这样进行的插入次数要少些  
41 head = head1;  
42 p = head2;  
43 }  
44 else  
45 {  
46 head = head2;  
47 p = head1;  
48 }  
49   
50 while(p != NULL)  
51 {  
52 nextP = p->next; //保存p的下一个节点  
53 head = insert\_node(head, p); //把p插入到目标链表中  
54 p = nextP; //指向将要插入的下一个节点  
55 }  
56   
57 return head;  
58 }  
这里insert\_node()函数是有序的插入节点，注意与前面例题中的函数有区别，这里它传入的参数是node\*类型。然后在merge()函数中（代码52~55行）循环把短链表中的所有节点插入到长链表中。  
接下来介绍非递归方法。比如有下面两个链表：  
链表1：1->3->5  
链表2：2->4->6  
递归方法的步骤如下：  
（1）比较链表1和链表2的第一个节点数据，由于1<2，因此把结果链表头节点指向链表1中的第一个节点，即数据1所在的节点。  
（2）对剩余的链表1（3->5）和链表2再调用本过程，比较得到结果链表的第二个节点，即2与3比较得到2。此时合并后的链表节点为1->2。  
接下来的过程类似（2），如此递归知道两个链表的节点都被加到结果链表中。  
1 node \* MergeRecursive(node \*head1, node \*head2)  
2 {  
3 node \*head = NULL;  
4   
5 if (head1 == NULL)  
6 {  
7 return head2;  
8 }  
9 if (head2 == NUL)  
10 {  
11 return head1;  
12 }  
13   
14 if ( head1->data < head2->data )  
15 {  
16 head = head1 ;  
17 head->next = MergeRecursive(head1->next,head2);  
18 }  
19 else  
20 {  
21 head = head2 ;  
22 head->next = MergeRecursive(head1,head2->next);  
23 }  
24   
25 return head ;  
26 }  
下面是测试程序：  
1 int main()  
2 {  
3 node \*head1 = create(); //创建单链表1  
4 node \*head2 = create(); //创建单链表2  
5 //node \*head = merge(head1, head2);  
6 node \*head = MergeRecursive(head1, head2);  
7 print(head);  
8  
9 return 0;  
10 }  
这里使用merge()函数和MergeRecursive()函数测试，结果一致。