链表要求掌握链表的基本操作， 包括创建链表（首插法， 尾插法）， 遍历链表， 查找， 插入一个结点， 删除一个结点或多个结点

结点结构为：

typedef struct node

{ int num;

struct node \*next; } NODE;

1. 下面的函数是将p所指的结点插入head（形参中）所指的单向链表中。设链表已按编号号(num)从小到大顺序排列，要求将p(形参)所指的结点插入后链表保持有序，函数返回插入后的链表的头指针。(带头结点的单链表)

函数为：

NODE \*insert(NODE \*head ,NODE \* p)

{ NODE \*front,\*rear;

front=head->next;

rear=head;

while (front!=NULL && front->num < p->num) {

rear=front;

front=front->next ;

}

//插入p指向的结点

p->next=front;

rear->next=p;

return head ;

}

2.下面的函数create是建立带头结点单向链表的函数，它返回已建立的链表的头指针。（尾插法：先输入的结点值在头，后输入的结点在尾）

结点结构为：

typedef struct node { int num; struct node \*next; } NODE;

NODE \*create()

{

NODE \*head,\*tail,\*p;

int num;

head==(NODE \*)malloc(sizeof(NODE)); /\* 建立头结点 \*/

head->next=NULL;

tail=head;

scanf("%d",&num);

while (num!=0) {

p=(NODE \*)malloc(sizeof(NODE));

//如果malloc返回值是NULL，申请空间不成功， 退出

if ( p==NULL ) {

printf("No enugh memory!\n");exit(1);

}

p->num=num;

//插入结点

tail->next=p ;

tail=p; /\* 给表尾指针tail赋值 \*/

scanf("%d",&num);

}

tail->next=NULL; /\* 最后一个结点的next域为空 \*/

return head ;

}

3.下面的函数create是建立带头结点单向链表的函数，它返回已建立的链表的头指针。（首插法：先输入的结点值在后，后输入的结点在前）

结点结构为：

typedef struct node { int num; struct node \*next; } NODE;

函数为：

NODE \*create()

{

NODE \*head, \*p;

int num;

head==(NODE \*)malloc(sizeof(NODE)); /\* 建立头结点 \*/

head->next=NULL;

scanf("%d",&num);

while (num!=0) {

p=(NODE \*)malloc(sizeof(NODE));

//如果malloc返回值是NULL，申请空间不成功， 退出

if ( !p ) {

printf("No enugh memory!\n");exit(1);

}

p->num=num;

p->next=head->next; ; //插入结点

head->next=p;

scanf("%d",&num);

}

return head ;

}

4.函数display() , 输出带头结点单向链表的所有数据结点的值。

结点结构为：

typedef struct node { int num; struct node \*next; } NODE;

函数为：

void display(Node \*head)

{

NODE \*p=head->next;

while (p!=NULL ) {

printf("%d \n", p->num);

p =p->next;

}

}

5-1. 查找, 在带头结点单向链表head中查找值为no的结点。 如果找到， 返回指向结点的指针，否则返回NULL。

typedef struct node { int num; struct node \*next; } NODE;

NODE \* find(Node \*head, int no)

{

NODE \*p;

p= head->next ;

while (p!=NULL ) {

if( p->num == no)

break;

else

p=p->next;

}

return p;

}

5-2. 查找, 在带头结点单向链表head中查找名字为name的结点。 如果找到， 输出该结点的值。（可能有多个结点name域的值相同）

typedef struct node { int num; char name[20]; struct node \*next; } NODE;

void find(Node \*head, char \*name)

{

NODE \*p;

p= head->next ;

while (p!=NULL ) {

if( strcmp(p->name,name) == 0) //注意字符串比较用函数strcmp

printf(“%d %s\n”, p->num, p->name);

p =p->next;

}

return p;

}

6-1. 删除. 在带头结点单向链表head中删除值为no的结点。

typedef struct node { int num; struct node \*next; } NODE;

voide del(Node \*head, int no)

{

NODE \*p, \*pre;

p= head->next;

pre=head;

while (p!=NULL ) {

if( p->num == no)

{ pre->next=p->next;

free(p);

p=NULL;

break; }

else

{pre=p;

p=p->next; }

}

}

6-2. 删除. 在带头结点单向链表head中删除名字为name的结点。

typedef struct node { int num; char name[20]; struct node \*next; } NODE;

voide del(Node \*head, char \*name)

{

NODE \*p, \*pre;

p= head->next;

pre=head;

while (p!=NULL ) {

if( strcmp(p->name, name) == 0)

{ pre->next=p->next;

free(p); }

else pre=p;

p =pre->next;

}

}