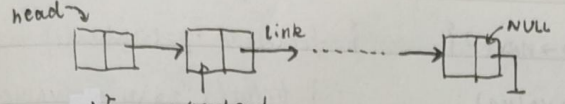
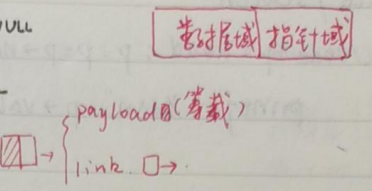
No: Date:

2020.3.9 Linked-List





- data structure.

定义

```
typedef struct _node{
    int value;
    struct _node * next
}Node;
```

→不能写成Node * next，此时Node未定义.

- insert head. (关于头结点、首结点、头指针见补充)

```
int main(){
    Node * head = NULL;
    int x = 0;
    while(1){
        scanf("%d", &x);
        if(x==-1){break;}
        Node *n = (Node *)malloc(sizeof(Node));
        n->value = x;
        n->next = head;
        head = n;
    }
    .....
}
```

Steps:
① malloc
② p->value = ?
③ p->next = head
④ 令head=p.

```
typedef struct _list{
    Node * head;
}List;
```

相当于 **head

```
void list_insert_head(List *list, ElementType value)
{
    Node *n = (Node *)malloc(sizeof(Node));
    n->value = i;
    n->next = list->head;
    list->head = n;
}
```

被改函数.

No: Date :

• iterate / search.

```
for (Node *p = head ; p ; p=p->next ) {
        printf ( "%d\n", p->value) ;
}
```

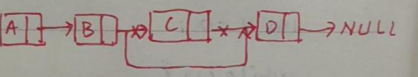
```
void list_print ( List *list )
{ for(Node *p=list->head; p; p=p->next)
  { printf ("%d\n", p->value) ;
  }
}
```

```
ret =0 ;
for (p=head ; p ; p=p->next ){
    if ( p-> value = i ){
      ret =p ;
      break;
    }
}
```

```
int list_search ( List *list , int x){
  int ret =0;
  for(Node *p =list->head; p ; p=p->next){
    if (p->value ==x){
        ret =1;
        break;
    }
  }
  return ret ;
}.
```

• remove.

```
for (Node *p = head ; p ; p=p->next ) {
    if ( p-> value ==x) {
        if (q) {
            q-> next = p-> next ;
        } else {
            head = p->next;
        }
        free (p) ;
    }
}
```




```
void list_remove_node (List *list , int x){
  for(Node *p=list->head, *q=NULL;
      p; q=p, p=p->next) {
    if (p->value ==x) {
      if (q) q->next =p->next;
      else list->head =p->next;
      free(p); break;
    }
  }
```

No: Date:

- append · tail

```
Node *n = malloc(sizeof(Node));
n->value = x;
n->next = NULL;
~~for(Node *p = head; p; p=p->next){~~
Node *p = head;
if(p){
    for( ; p->next; p=p->next){
    }
    p->next = n;
} else{
    ~~head~~ head = n;
}
```

steps: ① 建点 { n->value = x; n->next = NULL;
② find the end
③ attach

```
void List_append_tail(List *list, int x){
    Node *n = malloc(sizeof(Node));
    n->value = x;
    n->next = NULL;
    Node *p = list->head;
    if(p){
        for( ; p->next; p=p->next){}
        p->next = n;
    }else{
        list->head = n;
    }
}
```

- clear all

```
for(Node *q=NULL, *p=head; p; p=q){
    q=p->next;
    free(p);
}
```

```
void clear_list(List *list){
    for(Node *q=NULL, *p=list->head; p; p=q)
    { q=p->next; free(p); }.
}
```

operation: insert head, iterate/search, append tail, remove, clear all.

注：上述几种代码都用了 Node *p = head.
为了降低这几种间的耦合度，加一个指向head所指对 list->head.

No: Date:

加入 list→head 后，将上述各段代码改为函数.

ex: void remove-node (List * list , int x) {

for (Node *q=NULL, *p=list→head ; p ; q=p , p=p→next){

.....

}

}

需要在代码的最前面加入一段:

```
typedef struct{
    Node *head;
}List;
```

此时只需将各段中的 head 改为 list→head.

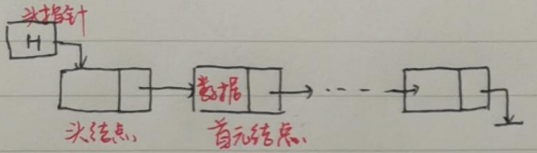
No: Date:

补充：

头结点：有时，在链表的第一个结点之前会额外增设一个结点，结点的数据域一般不存放数据（有些情况下也可以存放链表的长度等信息），此结点称为头结点。

首元结点：链表中第一个元素所存储的结点，它是头结点后边的第一个结点。

头指针：永远指向链表中第一个结点的位置。（如果有头结点，指向头结点；否则指向首元结点）。



单链表可以没有头结点，但不能没有头指针。

线性链表的链式存储相比顺式存储，有两大优势：

1、链式存储的数据元素在物理结构没有限制，当内存空间中没有足够大的连续的内存空间供顺序表使用时，可能可使用链表解决问题。

2、链表中结点之间采用指针进行链接，当对链表中的数据元素实行插入或者删除操作时，只需要改变指针的指向，无需像顺序表那样移动插入或删除位置后面元素，简单快捷。