17 高级数据表示

内容提要

- 研究数据表示
- 从数组到链表
- 链表和数组

研究数据表示

1 研究数据表示

- 在开始编写代码之前，要在程序设计方面做很多决定
- 让用户输入一年内看过的所有电影（包括DVD和蓝光光碟）。要存储每部影片的各种信息，如片名、发行年份、导演、主演、片长、影片的种类（喜剧、科幻、爱情等）、评级等。建议使用一个结构存储每部电影，一个数组存储一年内看过的电影。为简单起见，我们规定结构中只有两个成员：片名和评级（0～10）
- 程序清单17.1 films1.c
- 数据表示太不灵活。程序在编译时确定所需内存量，其实在运行时确定会更好。要解决这个问题，应该使用动态内存分配来表示数据

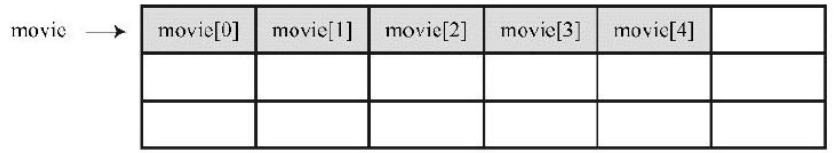
从数组到链表

2 从数组到链表

- 一种方法是调用malloc()一次，为300个film结构请求分配足够的空间；
- 另一种方法是调用malloc()300次，分别为每个file结构请求分配足够的空间
  - 需要存储300个指针，每个指针指向一个单独存储的结构
- 一种解决方法是创建一个大型的指针数组，并在分配新结构时逐个给这些指针赋值
- 链表

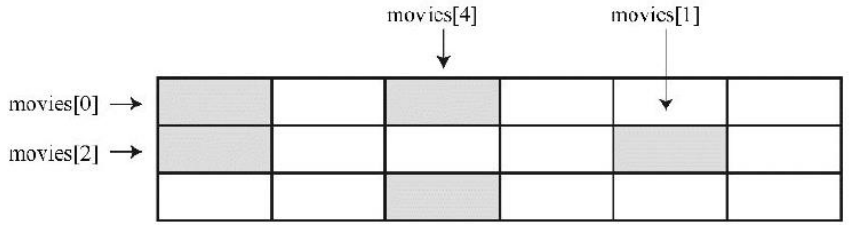
```
struct film * movie;

movie = (struct film *) malloc(5*sizeof(struct film);
```

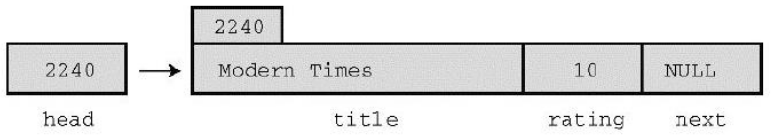



```
int i;
struct film * movies[s];

for (i = 0; i < 5; i++)
      movies[i] = (struct films *) malloc(sizeof(struct films));
```



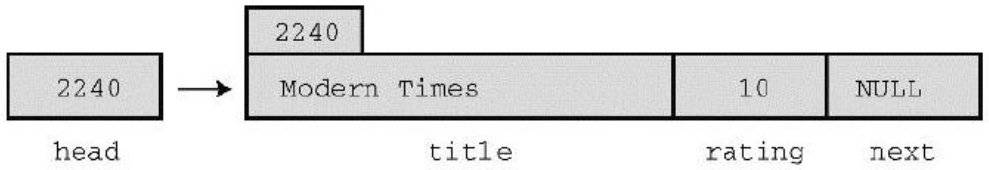

```
#define TSIZE 45
struct film {
   char title[TSIZE]
   int rating;
   struct film * next;
};
struct film * head;
```

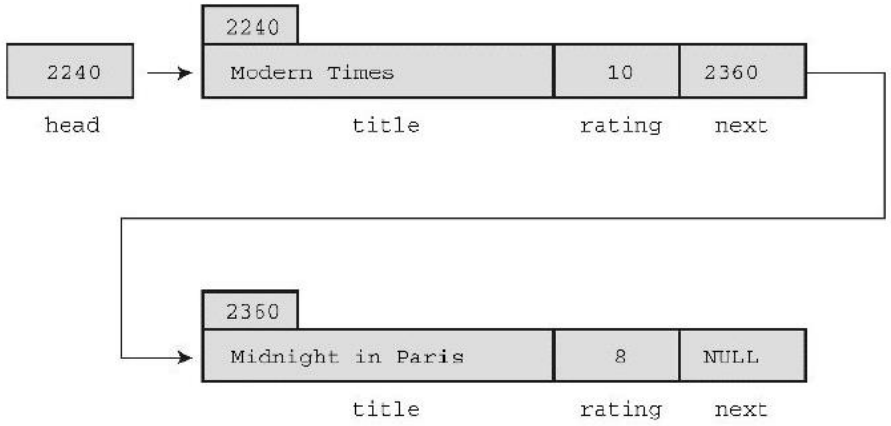


链表

- 每加入一部新电影，就以相同的方式来处理。新结构的地址将存储在上一个结构中，新信息存储在新结构中，而且新结构中的next成员设置为NULL
- 需要一个指针存储链表中第1项的地址，因为链表中没有其他项存储该项的地址
  - 头指针

```
#define TSIZE 45
struct film {
   char title[TSIZE]
   int rating;
   struct film * next;
};
struct film * head;
```





2.1 使用链表

- 程序清单17.2 films2.c
- 1. 显示链表
  - 显示链表从设置一个指向第1个结构的指针（名为current）开始
  - 然后，可以使用指针表示法访问结构的成员
  - 下一步是根据存储在该结构中next成员中的信息，重新设置current指针指向链表中的下一个结构
  - 重复整个过程。当显示到链表中最后一个项时，current将被设置为NULL
- 2. 创建链表
  - 使用malloc()为结构分配足够的空间
  - 存储结构的地址
  - 把当前信息拷贝到结构中
- 3. 释放链表
  - 程序在清理内存时为每个已分配的结构都调用了free()函数

```c
#define TSIZE    45  // size of array to hold title
struct film {
    char title[TSIZE];
    int rating;
    struct film * next;  // next struct in list
};
int main(void){
    struct film * head = NULL;
    struct film * prev, * current;
    char input[TSIZE];
    /* Gather  and store information */
    while (s_gets(input, TSIZE) != NULL && input[0] != '\0'){
        current = (struct film *) malloc(sizeof(struct film));
        if (head == NULL) head = current;
        else  prev->next = current;
        current->next = NULL;
        strcpy(current->title, input);
        scanf("%d", &current->rating);
        while(getchar() != '\n')   continue;
        puts("Next movie title (empty line to stop):");
        prev = current;
    }
    /* Show list of movies   */
    if (head == NULL) printf("No data entered. ");
    else printf ("Here is the movie list:\n");
    current = head;
    while (current != NULL){
        printf("M: %s R: %d\n", current->title, current->rating);
        current = current->next;
    }
    /* Program done, so free allocated memory */
    current = head;
    while (current != NULL){
        free(current);
        current = current->next;
    }
    return 0;
}
```

链表和数组

6 链表和数组

- 许多编程问题，如创建一个简单链表或队列，都可以用链表（指的是动态分配结构的序列链）或数组来处理
  - 每种形式都有其优缺点，要根据具体问题的要求来决定选择哪一种形式
- 何访问元素
  - 数组，可以使用数组下标直接访问该数组中的任意元素，这叫作随机访问（random access）
  - 链表，必须从链表首节点开始，逐个节点移动到要访问的节点，这叫作顺序访问（sequential access）
- 假设要查找链表中的特定项
  - 一种算法是从列表的开头开始按顺序查找，这叫作顺序查找（sequential search）
  - 对于一个排序的列表，用二分查找（binary search）比顺序查找好得多

表17.1　比较数组和链表

数据形式	优点	缺点
数组	C直接支持 提供随机访问	在编译时确定大小 插入和删除元素很费时
链表	运行时确定大小 快速插入和删除元素	不能随机访问 用户必须提供编程支持