Please tidy up your emotions.
We will be landing soon.

C 程 序 设 计

C Programming

17

高级数据表示

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

（国家示范性软件学院） 博士，副教授

School of Informatics Dr. Wei Huang

知识框架

- 高级数据表示
- 链表
  - 链表的建立
  - 链表的清理
- 抽象数据类型

内容纲要

1	高级数据表示
2	链表及其基本操作
3	抽象数据类型

研究数据表示

- 程序=数据结构+算法
  - 数据结构：程序以何种形式来存储信息

例：位图，一般采用二维数组存储其灰度（亮度）信息。为什么？

- 示例：
  - 题：某影评机构请你做一个程序，输入客户看过的电影列表，记录片名、（中间省略若干字）、您的评价等。
  - 拟采用数据结构
    - 结构体：意义不同，数据类型不同，范畴相近
    - 数组：意义相同，数据类型相同，仅有次序区别

```c
/* films1.c -- using an array of structures */
#include <stdio.h>
#include <string.h>
#define TSIZE        45       /* size of array to hold title   */
#define FMAX         5        /* maximum number of film titles */
struct film {
    char title[TSIZE];
    int rating;
};
char * s_gets(char str[], int lim);

int main(void)
{
    struct film movies[FMAX];
    int i = 0;
    int j;
    puts("Enter first movie title:");
    while (i < FMAX && s_gets(movies[i].title, TSIZE) != NULL &&
           movies[i].title[0] != '\0')
```

```
    {
        puts("Enter your rating <0-10>:");
        scanf("%d", &movies[i++].rating);
        while(getchar() != '\n')
            continue;
        puts("Enter next movie title (empty line to stop):");
    }
    if (i == 0)
        printf("No data entered. ");
    else
        printf ("Here is the movie list:\n");
    for (j = 0; j < i; j++)
        printf("Movie: %s  Rating: %d\n", movies[j].title,
                movies[j].rating);
    printf("Bye!\n");
    return 0;
}
```

```c
char * s_gets(char * st, int n)
{
    char * ret_val;
    char * find;

    ret_val = fgets(st, n, stdin);
    if (ret_val)
    {
        find = strchr(st, '\n');   // look for newline
        if (find)                  // if the address is not NULL,
            *find = '\0';          // place a null character there
        else
            while (getchar() != '\n')
                continue;          // dispose of rest of line
    }
    return ret_val;
}
```

数组的局限性

- 数组的总长度应在声明时固定，不可扩充
  - 电影名总长度可调研，预先设定
  - 电影数量不可预设，用户也没法算一辈子会看几部电影
    - 万一我看了FMAX+1部电影，会引发一个运行错误

```
int n, i;
struct film * movies; /* pointer to a structure */
...
printf("Enter the maximum number of movies you'll enter:\n");
scanf("%d", &n);
movies = (struct film *) malloc(n * sizeof(struct film));
```

内容纲要

1	高级数据表示
2	链表及其基本操作
3	抽象数据类型

从数组到链表

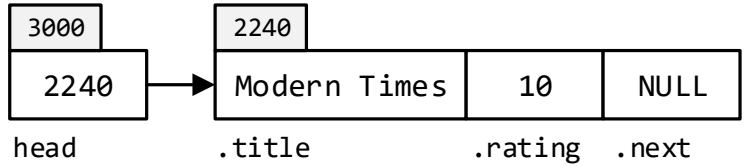
- 理想情况下，你希望不确定地添加数据
  - 依次输入*N*部电影，输入结束前*N*未知
- 方案
  - 每次添加数据时调用malloc()函数
    - malloc()多次开辟的空间是分散的
  - 结构体需要指示下一个元素的地址
    - 结构体不能自身，但可以包含指向自身类型的指针
  - 需要有一个指向头部的指针

```
struct film {
    char title[TSIZE];
    int rating;
    struct film *next;
};
```

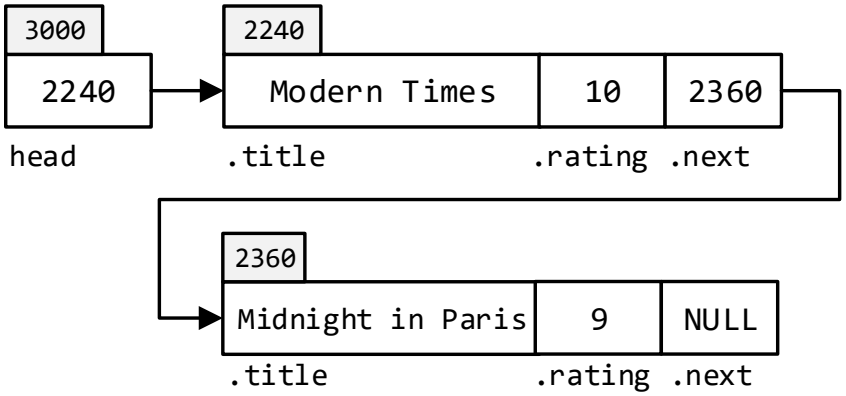
```
struct film *head
```

从数组到链表

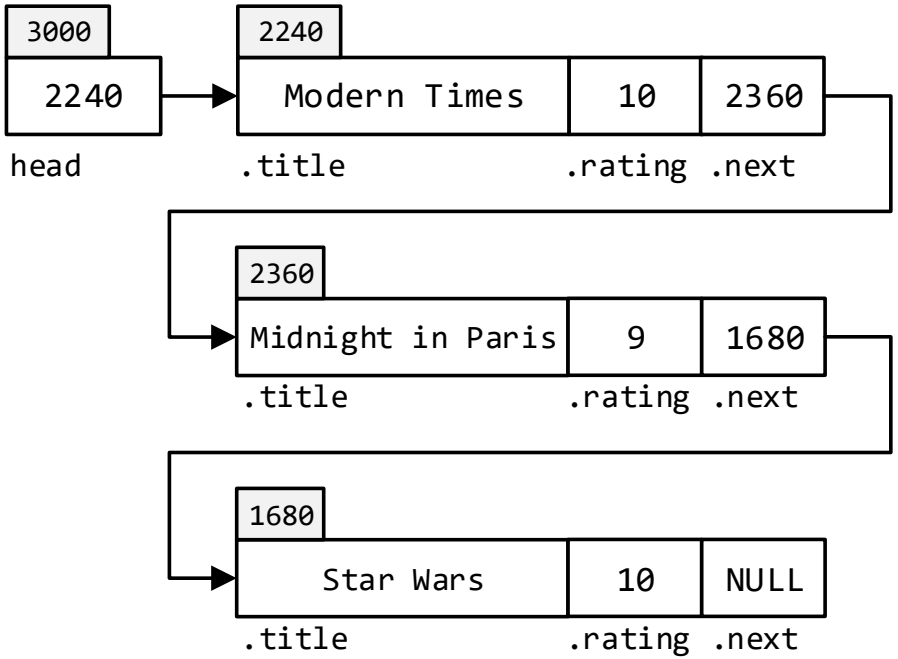
- 仅含有一项的列表



- 含有两个项目的列表



- 含有多项的列表



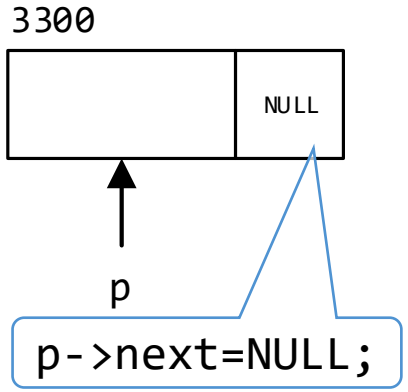
培养“读+写”的能力

- 读：先将代码转成图示
- 记：通过理解记住图示
- 写：再将图示转成代码

创建结点并链接到另一个结点

- 新建结点

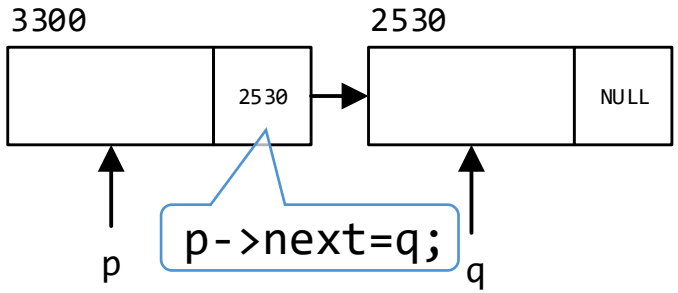
```
p = (struct node *)malloc(sizeof(struct node));
```



- 赋初始值（结点的属性赋值+下一结点）
  - next属性赋值为NULL，表示最后一个结点
- next属性赋值为另一个结点的地址，形成前后顺序

```
p->name = "test";
p->price = 5.80;
p->next = NULL;
```

```
p->name = "test";
p->price = 5.80;
p->next = q;
```

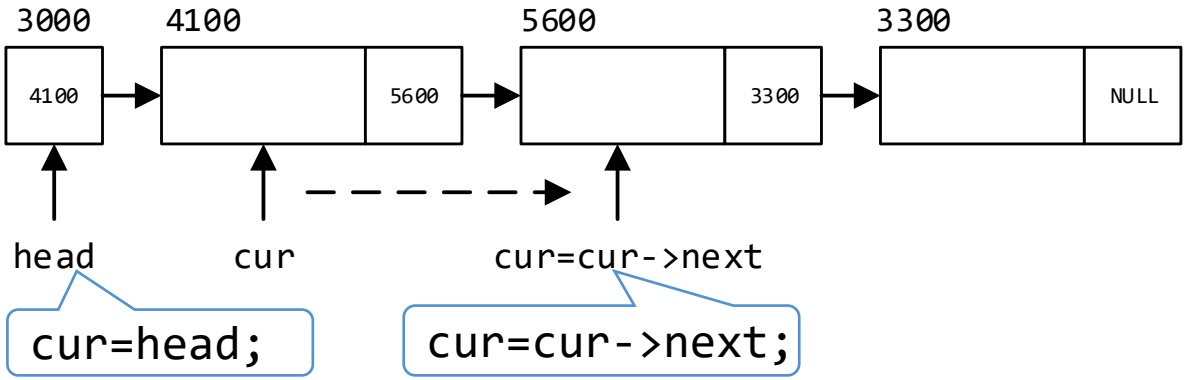


从数组到链表

- 按顺序遍历数组和链表

功能	数组	链表
定位到第一个元素	i=0	cur=head
判断是否最后一个元素	i<N	cur!=NULL
移动到下一个元素	i++	cur=cur->next

```
i = 0;
while (i < N)
{
    ...;
    i++;
}
```




```
cur = head;
while (cur != NULL)
{
    ...;
    cur = cur->next;
}
```

使用链表：显示链表

- 链表操作

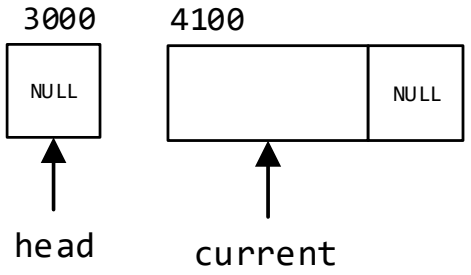
操作	指针操作	相当于数组操作
初始化	将当前指针置于起始位置	数组的下标为0
步进	通过当前指针寻找下一指针	数组的下标自增
循环条件	当前指针非空	数组到最后一项

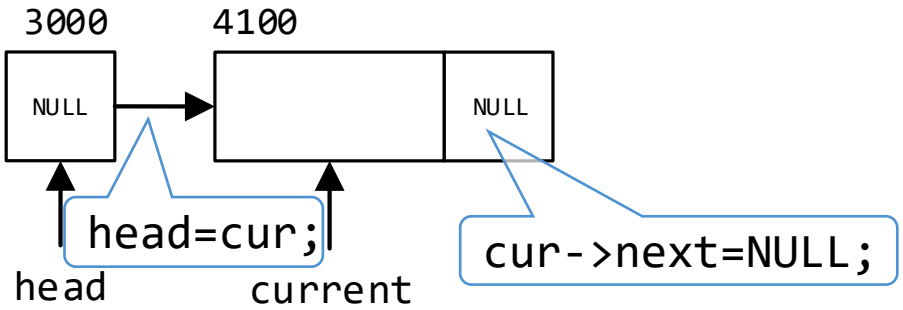
声明指针类型后，应赋初始值NULL

```
current = head;
while (current != NULL)
{
    printf("Movie: %s  Rating: %d\n", current->title, current->rating);
    current = current->next;
}
```

创建列表：尾插法

- 从空表开始
  - 初始化头指针 head = NULL
- 循环
  - 分配空间、赋初值
  - 头指向

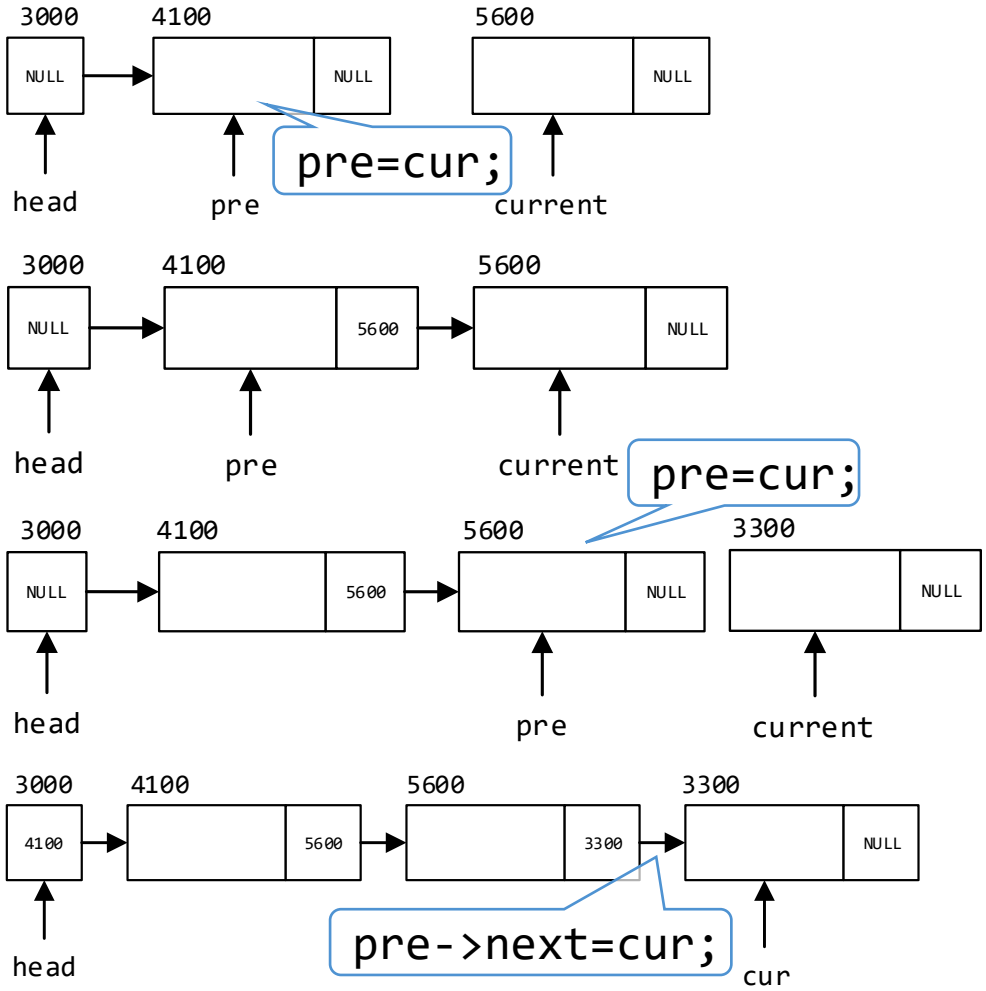





```
while (s_gets(input, TSIZE) != NULL && input[0] != '\0')
{
    current = (struct film *) malloc(sizeof(struct film));
    current->price = 5;
    current->next = NULL;
    head = current;
}
```

创建列表：尾插法

- 循环
  - 分配空间
  - 赋初值
  - 头指向（头为空）
  - 上一结点的next指向当前结点（头不为空）
  - 记录当前结点作为下次循环的上一结点



创建列表：尾插法

- 示例代码

```
while (s_gets(input, TSIZE) != NULL && input[0] != '\0')
{
    current = (struct film *) malloc(sizeof(struct film));

    strcpy(current->title, input);
    puts("Enter your rating <0-10>:");
    scanf("%d", &current->rating);
    current->next = NULL;

    if (head == NULL)
        head = current;
    else
        prev->next = current;

    prev = current;
}
```

分配空间

赋初值

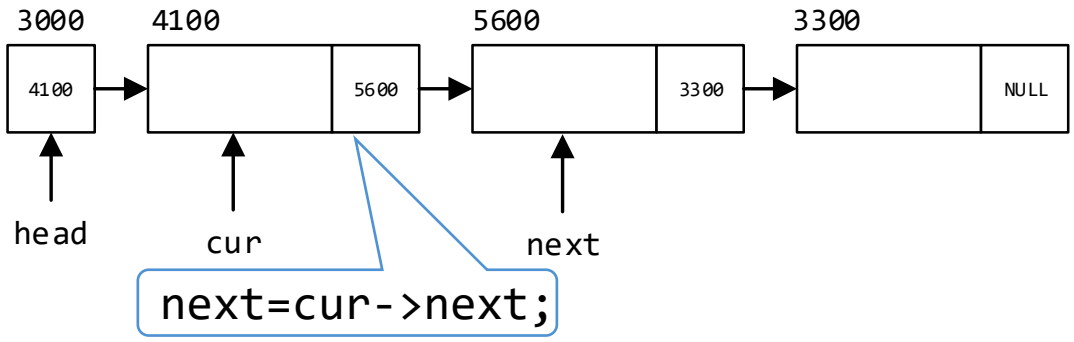
头指向（头为空）

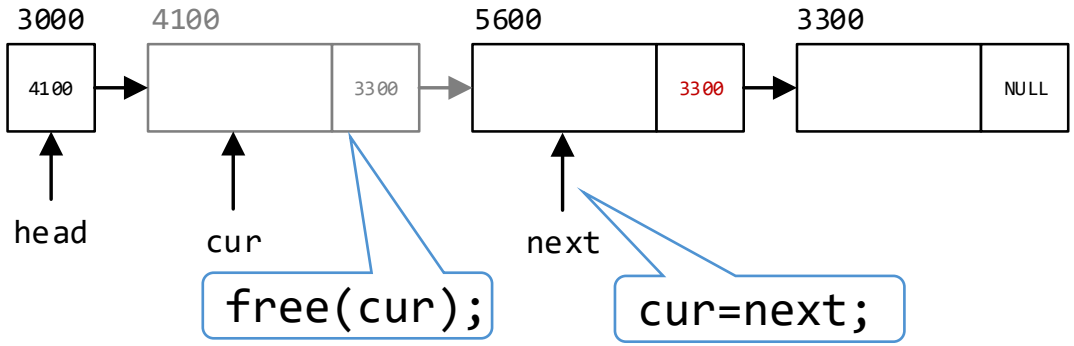
上一结点的next指向当前结点（头不为空）

记录当前结点作为下一次循环的上一结点

释放列表

- 由malloc()开辟的空间在程序终止时释放
  - 但程序员不能放任这种情况，应调用free()释放






```
current = head;
while (current != NULL)
{
    free(current);
    current = current->next;
}
```

```
current = head;
while (current != NULL)
{
    next = current->next;
    free(current);
    current = next;
}
```

```c
/* films2.c -- using a linked list of structures */
#include <stdio.h>
#include <stdlib.h>      /* has the malloc prototype      */
#include <string.h>      /* has the strcpy prototype      */
#define TSIZE    45      /* size of array to hold title   */

struct film {
    char title[TSIZE];
    int rating;
    struct film * next;  /* points to next struct in list */
};
char * s_gets(char * st, int n);

int main(void)
{
    struct film * head = NULL;
    struct film * prev, * current;
    char input[TSIZE];
```

```
/* Gather  and store information          */
puts("Enter first movie title:");
while (s_gets(input, TSIZE) != NULL && input[0] != '\0')
{
    current = (struct film *) malloc(sizeof(struct film));
    if (head == NULL)       /* first structure       */
        head = current;
    else                    /* subsequent structures */
        prev->next = current;
    current->next = NULL;
    strcpy(current->title, input);
    puts("Enter your rating <0-10>:");
    scanf("%d", &current->rating);
    while(getchar() != '\n')
        continue;
    puts("Enter 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("Movie: %s  Rating: %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;
}
```

```
    printf("Bye!\n");
    return 0;
}
char * s_gets(char * st, int n)
{
    char * ret_val;
    char * find;
    ret_val = fgets(st, n, stdin);
    if (ret_val)
    {
        find = strchr(st, '\n');   // look for newline
        if (find)                  // if the address is not NULL,
            *find = '\0';          // place a null character there
        else
            while (getchar() != '\n')
                continue;          // dispose of rest of line
    }
    return ret_val;
}
```

反思

扩展内容

- 开辟的内存就一定能成功吗？不成功怎么办？
- 这样的方法可以解决特定问题，但难以添加功能
  - 一个好的程序往往是由小程序不断添加功能长大的
  - 上述程序将“编码细节”与“概念模型”混合在一起

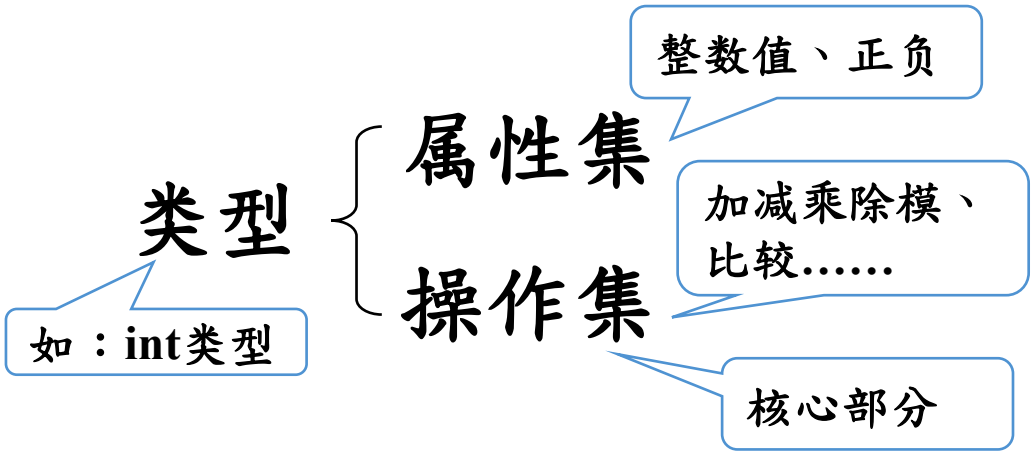
内容纲要

1	高级数据表示
2	链表及其基本操作
3	抽象数据类型

抽象数据类型（ADT）

扩展内容

- 影评系统的具体做法
  - C中没有和合适影评的数据类型，于是我们设计了一个结构体，再变成链表，来表示影评。
- 系统性的做法：定义类型（Type）



抽象数据类型（ADT）

扩展内容

- 类型（type）：由属性集和操作集组成
  - 操作是核心部分，没有操作的数据类型是没有什么用的
    - 设想：一个不能加减乘除的int类型
  - 数学提供了整数的抽象概念，C提供了概念的实现
  - 但是int类型并没有很好地实现整数
    - 长度4B的int型最多为$2^{31}-1$；而整数是无穷的

抽象数据类型（ADT）

扩展内容

- 定义一个新的类型的成功方法
  - 为类型的属性和可对类型执行的操作提供一个抽象的描述
  - 开发一个实现该ADT的编程接口
  - 编写代码来实现这个接口

```c
/* list.h -- header file for a simple list type */
#ifndef LIST_H_
#define LIST_H_
#include <stdbool.h>     /* C99 feature          */
/* program-specific declarations */
#define TSIZE      45    /* size of array to hold title  */
struct film
{
    char title[TSIZE];
    int rating;
};
/* general type definitions */
typedef struct film Item;
typedef struct node
{
    Item item;
    struct node * next;
} Node;
```

```
typedef Node * List;
/* function prototypes */

/* operation:        initialize a list                    */
/* preconditions:    plist points to a list               */
/* postconditions:   the list is initialized to empty     */
void InitializeList(List * plist);

/* operation:        determine if list is empty           */
/*                   plist points to an initialized list  */
/* postconditions:   function returns True if list is empty */
/*                   and returns False otherwise          */
bool ListIsEmpty(const List *plist);

/* operation:        determine if list is full            */
/*                   plist points to an initialized list  */
/* postconditions:   function returns True if list is full  */
/*                   and returns False otherwise          */
bool ListIsFull(const List *plist);
```

```
/* operation:        determine number of items in list     */
/*                   plist points to an initialized list   */
/* postconditions:   function returns number of items in list  */
unsigned int ListItemCount(const List *plist);
/* operation:        add item to end of list               */
/* preconditions:    item is an item to be added to list   */
/*                   plist points to an initialized list   */
/* postconditions:   if possible, function adds item to end    */
/*                   of list and returns True; otherwise the   */
/*                   function returns False                */
bool AddItem(Item item, List * plist);

/* operation:        apply a function to each item in list */
/*                   plist points to an initialized list   */
/*                   pfun points to a function that takes an   */
/*                   Item argument and has no return value */
/* postcondition:    the function pointed to by pfun is    */
/*                   executed once for each item in the list   */
void Traverse (const List *plist, void (* pfun)(Item item) );
```

```
/* operation:          free allocated memory, if any          */
/*                     plist points to an initialized list    */
/* postconditions:     any memory allocated for the list is freed*/
/*                     and the list is set to empty           */
void EmptyTheList(List * plist);

#endif
```

使用接口

扩展内容

- 先写一些伪代码方案
- 然后不必关心（拘泥于）细节，注重类型和操作即可
- 如何使用接口

```c
/* films3.c -- using an ADT-style linked list */
/* compile with list.c                        */
#include <stdio.h>
#include <stdlib.h>    /* prototype for exit() */
#include "list.h"      /* defines List, Item   */
void showmovies(Item item);
char * s_gets(char * st, int n);
int main(void)
{
    List movies;
    Item temp;

    /* initialize       */
    InitializeList(&movies);
    if (ListIsFull(&movies))
    {
        fprintf(stderr,"No memory available! Bye!\n");
        exit(1);
    }
```

```
    /* gather and store */
    puts("Enter first movie title:");
    while (s_gets(temp.title, TSIZE) != NULL &&
temp.title[0] != '\0')
    {
        puts("Enter your rating <0-10>:");
        scanf("%d", &temp.rating);
        while(getchar() != '\n')
            continue;
        if (AddItem(temp, &movies)==false)
        {
            fprintf(stderr,"Problem allocating memory\n");
            break;
        }
        if (ListIsFull(&movies))
        {
            puts("The list is now full.");
            break;
        }
```

```
        puts("Enter next movie title (empty line to stop):");
    }
    /* display           */
    if (ListIsEmpty(&movies))
        printf("No data entered. ");
    else
    {
        printf ("Here is the movie list:\n");
        Traverse(&movies, showmovies);
    }
    printf("You entered %d movies.\n", ListItemCount(&movies));

    /* clean up          */
    EmptyTheList(&movies);
    printf("Bye!\n");

    return 0;
}
```

```c
void showmovies(Item item) {
    printf("Movie: %s  Rating: %d\n", item.title, item.rating);
}

char * s_gets(char * st, int n) {
    char * ret_val;
    char * find;
    ret_val = fgets(st, n, stdin);
    if (ret_val) {
        find = strchr(st, '\n');   // look for newline
        if (find)                  // if the address is not NULL,
            *find = '\0';          // place a null character there
        else
            while (getchar() != '\n')
                continue;          // dispose of rest of line
    }
    return ret_val;
}
```

实现接口

扩展内容

- 思考你的工作
  - 比较films2.c和list.c，前者暴露了太多的编程细节（高耦合）
  - 如果需要另一个简单的列表，比如电话簿，仍可以通过较少的改动，使用这些文件list.h、list.c
- 这就是面向对象的思想雏形！
  - C语言在实现面向对象时，代码不够简洁
  - 简洁：C++，Java

队列ADT

扩展内容

- 好了，本课程的理论学习就到这里了
- 剩下的部分，请自行学习
- 至此，基本已经衔接上面向对象编程等后续课程了
- 课程结束了，但学习没有结束
  - 继续练习，继续OJ
  - 班级同学多交流，不要怕交流
  - 在这个时代，自己拼搏的结果往往容易被人模仿、超越；团队合作出来的成果才高大上。

```c
/* list.c -- functions supporting list operations */
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

/* local function prototype */
static void CopyToNode(Item item, Node * pnode);

/* interface functions   */
/* set the list to empty */
void InitializeList(List * plist)
{
    * plist = NULL;
}

/* returns true if list is empty */
bool ListIsEmpty(const List * plist)
{
```

```
    if (*plist == NULL)
        return true;
    else
        return false;
}

/* returns true if list is full */
bool ListIsFull(const List * plist)
{
    Node * pt;
    bool full;

    pt = (Node *) malloc(sizeof(Node));
    if (pt == NULL)
        full = true;
    else
        full = false;
    free(pt);
```

```
    return full;
}

/* returns number of nodes */
unsigned int ListItemCount(const List * plist)
{
    unsigned int count = 0;
    Node * pnode = *plist;    /* set to start of list */
    while (pnode != NULL)
    {
        ++count;
        pnode = pnode->next;  /* set to next node     */
    }

    return count;
}
```

```
/* creates node to hold item and adds it to the end of */
/* the list pointed to by plist (slow implementation)  */
bool AddItem(Item item, List * plist)
{
    Node * pnew;
    Node * scan = *plist;

    pnew = (Node *) malloc(sizeof(Node));
    if (pnew == NULL)
        return false;     /* quit function on failure  */

    CopyToNode(item, pnew);
    pnew->next = NULL;
    if (scan == NULL)           /* empty list, so place */
        *plist = pnew;          /* pnew at head of list */
    else
    {
```

```
        while (scan->next != NULL)
            scan = scan->next;  /* find end of list    */
        scan->next = pnew;      /* add pnew to end     */
    }
    return true;
}

/* visit each node and execute function pointed to by pfun */
void Traverse(const List * plist, void (* pfun)(Item item) )
{
    Node * pnode = *plist;    /* set to start of list   */

    while (pnode != NULL)
    {
        (*pfun)(pnode->item); /* apply function to item */
        pnode = pnode->next;  /* advance to next item   */
    }
}
```

```
/* free memory allocated by malloc() */
/* set list pointer to NULL          */
void EmptyTheList(List * plist)
{
    Node * psave;
    while (*plist != NULL)
    {
        psave = (*plist)->next; /* save address of next node */
        free(*plist);           /* free current node         */
        *plist = psave;         /* advance to next node      */
    }
}
/* local function definition  */
/* copies an item into a node */
static void CopyToNode(Item item, Node * pnode)
{
    pnode->item = item;  /* structure copy */
}
```

```c
/* queue.h -- interface for a queue */
#ifndef _QUEUE_H_
#define _QUEUE_H_
#include <stdbool.h>

// INSERT ITEM TYPE HERE
// FOR EXAMPLE,
// typedef int Item;  // for use_q.c
// OR typedef struct item {int gumption; int charisma;} Item;
// OR  (for mall.c)
/**/
typedef struct item
{
    long arrive;       // the time when a customer joins the queue
    int processtime;   // the number of consultation minutes desired
} Item;
/**/
```

```c
#define MAXQUEUE 10
typedef struct node
{
    Item item;
    struct node * next;
} Node;

typedef struct queue
{
    Node * front;  /* pointer to front of queue  */
    Node * rear;   /* pointer to rear of queue   */
    int items;     /* number of items in queue   */
} Queue;

/* operation:        initialize the queue                */
/* precondition:     pq points to a queue                */
/* postcondition:    queue is initialized to being empty */
void InitializeQueue(Queue * pq);
```

```
/* operation:       check if queue is full                    */
/* precondition:    pq points to previously initialized queue  */
/* postcondition:   returns True if queue is full, else False  */
bool QueueIsFull(const Queue * pq);
/* operation:       check if queue is empty                   */
/* precondition:    pq points to previously initialized queue  */
/* postcondition:   returns True if queue is empty, else False */
bool QueueIsEmpty(const Queue *pq);

/* operation:       determine number of items in queue        */
/* precondition:    pq points to previously initialized queue  */
/* postcondition:   returns number of items in queue          */
int QueueItemCount(const Queue * pq);
```

```
/* operation:        add item to rear of queue                    */
/* precondition:     pq points to previously initialized queue     */
/*                   item is to be placed at rear of queue        */
/* postcondition:    if queue is not empty, item is placed at     */
/*                   rear of queue and function returns           */
/*                   True; otherwise, queue is unchanged and      */
/*                   function returns False                       */
bool EnQueue(Item item, Queue * pq);
```

```
/* operation:        remove item from front of queue          */
/* precondition:     pq points to previously initialized queue */
/* postcondition:    if queue is not empty, item at head of */
/*                   queue is copied to *pitem and deleted from */
/*                   queue, and function returns True; if the   */
/*                   operation empties the queue, the queue is  */
/*                   reset to empty. If the queue is empty to   */
/*                   begin with, queue is unchanged and the */
/*                   function returns False                     */
bool DeQueue(Item *pitem, Queue * pq);

/* operation:        empty the queue                          */
/* precondition:     pq points to previously initialized queue */
/* postconditions:   the queue is empty                       */
void EmptyTheQueue(Queue * pq);

#endif
```

```c
/* queue.c -- the Queue type implementation*/
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

/* local functions */
static void CopyToNode(Item item, Node * pn);
static void CopyToItem(Node * pn, Item * pi);

void InitializeQueue(Queue * pq) {
    pq->front = pq->rear = NULL;
    pq->items = 0;
}

bool QueueIsFull(const Queue * pq) {
    return pq->items == MAXQUEUE;
}
```

```
bool QueueIsEmpty(const Queue * pq) {
    return pq->items == 0;
}

int QueueItemCount(const Queue * pq) {
    return pq->items;
}

bool EnQueue(Item item, Queue * pq) {
    Node * pnew;
    if (QueueIsFull(pq))
        return false;
    pnew = (Node *) malloc( sizeof(Node));
    if (pnew == NULL) {
        fprintf(stderr,"Unable to allocate memory!\n");
        exit(1);
    }
    CopyToNode(item, pnew);
```

```
    pnew->next = NULL;
    if (QueueIsEmpty(pq))
        pq->front = pnew;          /* item goes to front     */
    else
        pq->rear->next = pnew;     /* link at end of queue   */
    pq->rear = pnew;               /* record location of end */
    pq->items++;                   /* one more item in queue */
    return true;
}

bool DeQueue(Item * pitem, Queue * pq) {
    Node * pt;
    if (QueueIsEmpty(pq))
        return false;
    CopyToItem(pq->front, pitem);
    pt = pq->front;
    pq->front = pq->front->next;
    free(pt);
```

```
    pq->items--;
    if (pq->items == 0)
        pq->rear = NULL;
    return true;
}
/* empty the queue                    */
void EmptyTheQueue(Queue * pq) {
    Item dummy;
    while (!QueueIsEmpty(pq))
        DeQueue(&dummy, pq);
}
/* Local functions                     */
static void CopyToNode(Item item, Node * pn) {
    pn->item = item;
}
static void CopyToItem(Node * pn, Item * pi) {
    *pi = pn->item;
}
```

```c
/* use_q.c -- driver testing the Queue interface */
/* compile with queue.c                          */
#include <stdio.h>
#include "queue.h"  /* defines Queue, Item       */

int main(void)
{
    Queue line;
    Item temp;
    char ch;

    InitializeQueue(&line);
    puts("Testing the Queue interface. Type a to add a value,");
    puts("type d to delete a value, and type q to quit.");
    while ((ch = getchar()) != 'q')
    {
        if (ch != 'a' && ch != 'd')  /* ignore other input */
            continue;
        if ( ch == 'a')
```

```
    {
        printf("Integer to add: ");
        scanf("%d", &temp);
        if (!QueueIsFull(&line))
        {
            printf("Putting %d into queue\n", temp);
            EnQueue(temp,&line);
        }
        else
            puts("Queue is full!");
    }
    else
    {
        if (QueueIsEmpty(&line))
            puts("Nothing to delete!");
        else
        {
            DeQueue(&temp,&line);
            printf("Removing %d from queue\n", temp);
```

```
            }
        }
        printf("%d items in queue\n", QueueItemCount(&line));
        puts("Type a to add, d to delete, q to quit:");
    }
    EmptyTheQueue(&line);
    puts("Bye!");

    return 0;
}
```

```c
// mall.c -- use the Queue interface
// compile with queue.c
#include <stdio.h>
#include <stdlib.h>    // for rand() and srand()
#include <time.h>      // for time()
#include "queue.h"     // change Item typedef
#define MIN_PER_HR 60.0

bool newcustomer(double x);   // is there a new customer?
Item customertime(long when); // set customer parameters

int main(void)
{
    Queue line;
    Item temp;                // new customer data
    int hours;                // hours of simulation
    int perhour;              // average # of arrivals per hour
    long cycle, cyclelimit;   // loop counter, limit
```

```
long turnaways = 0;          // turned away by full queue
long customers = 0;          // joined the queue
long served = 0;             // served during the simulation
long sum_line = 0;           // cumulative line length
int wait_time = 0;           // time until Sigmund is free
double min_per_cust;         // average time between arrivals
long line_wait = 0;          // cumulative time in line

InitializeQueue(&line);
srand((unsigned int) time(0)); // random initializing of rand()
puts("Case Study: Sigmund Lander's Advice Booth");
puts("Enter the number of simulation hours:");
scanf("%d", &hours);
cyclelimit = MIN_PER_HR * hours;
puts("Enter the average number of customers per hour:");
scanf("%d", &perhour);
```

```
min_per_cust = MIN_PER_HR / perhour;

for (cycle = 0; cycle < cyclelimit; cycle++)
{
    if (newcustomer(min_per_cust))
    {
        if (QueueIsFull(&line))
            turnaways++;
        else
        {
            customers++;
            temp = customertime(cycle);
            EnQueue(temp, &line);
        }
    }
    if (wait_time <= 0 && !QueueIsEmpty(&line))
```

```
        {
            DeQueue (&temp, &line);
            wait_time = temp.processtime;
            line_wait += cycle - temp.arrive;
            served++;
        }
        if (wait_time > 0)
            wait_time--;
        sum_line += QueueItemCount(&line);
    }

    if (customers > 0)
    {
        printf("customers accepted: %ld\n", customers);
        printf("  customers served: %ld\n", served);
        printf("       turnaways: %ld\n", turnaways);
        printf("average queue size: %.2f\n",
               (double) sum_line / cyclelimit);
```

```
        printf(" average wait time: %.2f minutes\n",
               (double) line_wait / served);
    }
    else
        puts("No customers!");
    EmptyTheQueue(&line);
    puts("Bye!");

    return 0;
}

// x = average time, in minutes, between customers
// return value is true if customer shows up this minute
bool newcustomer(double x)
{
    if (rand() * x / RAND_MAX < 1)
        return true;
```

```
    else
        return false;
}

// when is the time at which the customer arrives
// function returns an Item structure with the arrival time
// set to when and the processing time set to a random value
// in the range 1 - 3
Item customertime(long when)
{
    Item cust;

    cust.processtime = rand() % 3 + 1;
    cust.arrive = when;

    return cust;
}
```

```c
/* tree.h -- binary search tree                                    */
/*           no duplicate items are allowed in this tree */
#ifndef _TREE_H_
#define _TREE_H_
#include <stdbool.h>

/* redefine Item as appropriate */
#define SLEN 20
typedef struct item
{
    char petname[SLEN];
    char petkind[SLEN];
} Item;

#define MAXITEMS 10
```

```
typedef struct trnode
{
    Item item;
    struct trnode * left;  /* pointer to right branch  */
    struct trnode * right; /* pointer to left branch   */
} Trnode;

typedef struct tree
{
    Trnode * root;         /* pointer to root of tree  */
    int size;              /* number of items in tree  */
} Tree;

/* function prototypes */
/* operation:      initialize a tree to empty           */
/* preconditions:  ptree points to a tree               */
/* postconditions: the tree is initialized to empty     */
void InitializeTree(Tree * ptree);
```

```
/* operation:      determine if tree is empty           */
/* preconditions:  ptree points to a tree               */
/* postconditions: function returns true if tree is     */
/*                 empty and returns false otherwise    */
bool TreeIsEmpty(const Tree * ptree);

/* operation:      determine if tree is full            */
/* preconditions:  ptree points to a tree               */
/* postconditions: function returns true if tree is     */
/*                 full and returns false otherwise     */
bool TreeIsFull(const Tree * ptree);

/* operation:      determine number of items in tree    */
/* preconditions:  ptree points to a tree               */
/* postconditions: function returns number of items in  */
/*                 tree                                 */
int TreeItemCount(const Tree * ptree);
```

```
/* operation:      add an item to a tree                    */
/* preconditions:  pi is address of item to be added        */
/*                 ptree points to an initialized tree      */
/* postconditions: if possible, function adds item to       */
/*                 tree and returns true; otherwise,        */
/*                 the function returns false               */
bool AddItem(const Item * pi, Tree * ptree);

/* operation: find an item in a tree                        */
/* preconditions:  pi points to an item                     */
/*                 ptree points to an initialized tree      */
/* postconditions: function returns true if item is in      */
/*                 tree and returns false otherwise         */
bool InTree(const Item * pi, const Tree * ptree);
```

```
/* operation:      delete an item from a tree          */
/* preconditions:  pi is address of item to be deleted */
/*                 ptree points to an initialized tree */
/* postconditions: if possible, function deletes item  */
/*                 from tree and returns true;         */
/*                 otherwise the function returns false*/
bool DeleteItem(const Item * pi, Tree * ptree);

/* operation:      apply a function to each item in    */
/*                 the tree                            */
/* preconditions:  ptree points to a tree              */
/*                 pfun points to a function that takes*/
/*                 an Item argument and has no return  */
/*                 value                               */
/* postcondition:  the function pointed to by pfun is  */
/*                 executed once for each item in tree */
void Traverse (const Tree * ptree, void (* pfun)(Item item));
```

```
/* operation:      delete everything from a tree        */
/* preconditions:  ptree points to an initialized tree */
/* postconditions: tree is empty                        */
void DeleteAll(Tree * ptree);

#endif
```

```c
/* tree.c -- tree support functions */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "tree.h"

/* local data type */
typedef struct pair {
    Trnode * parent;
    Trnode * child;
} Pair;

/* prototypes for local functions */
static Trnode * MakeNode(const Item * pi);
static bool ToLeft(const Item * i1, const Item * i2);
static bool ToRight(const Item * i1, const Item * i2);
static void AddNode (Trnode * new_node, Trnode * root);
```

```
static void InOrder(const Trnode * root, void (* pfun)(Item
item));
static Pair SeekItem(const Item * pi, const Tree * ptree);
static void DeleteNode(Trnode **ptr);
static void DeleteAllNodes(Trnode * ptr);

/* function definitions */
void InitializeTree(Tree * ptree)
{
    ptree->root = NULL;
    ptree->size = 0;
}

bool TreeIsEmpty(const Tree * ptree)
{
    if (ptree->root == NULL)
        return true;
    else
```

```
        return false;
}

bool TreeIsFull(const Tree * ptree)
{
    if (ptree->size == MAXITEMS)
        return true;
    else
        return false;
}

int TreeItemCount(const Tree * ptree)
{
    return ptree->size;
}
```

```
bool AddItem(const Item * pi, Tree * ptree)
{
    Trnode * new_node;

    if  (TreeIsFull(ptree))
    {
        fprintf(stderr,"Tree is full\n");
        return false;              /* early return          */
    }
    if (SeekItem(pi, ptree).child != NULL)
    {
        fprintf(stderr, "Attempted to add duplicate item\n");
        return false;              /* early return          */
    }
    new_node = MakeNode(pi);       /* points to new node    */
    if (new_node == NULL)
    {
        fprintf(stderr, "Couldn't create node\n");
```

```
        return false;                /* early return                */
    }
    /* succeeded in creating a new node */
    ptree->size++;

    if (ptree->root == NULL)      /* case 1: tree is empty  */
        ptree->root = new_node;   /* new node is tree root  */
    else                          /* case 2: not empty      */
        AddNode(new_node,ptree->root); /* add node to tree */

    return true;                  /* successful return      */
}

bool InTree(const Item * pi, const Tree * ptree)
{
    return (SeekItem(pi, ptree).child == NULL) ? false : true;
}
```

```
bool DeleteItem(const Item * pi, Tree * ptree)
{
    Pair look;

    look = SeekItem(pi, ptree);
    if (look.child == NULL)
        return false;

    if (look.parent == NULL)      /* delete root item        */
        DeleteNode(&ptree->root);
    else if (look.parent->left == look.child)
        DeleteNode(&look.parent->left);
    else
        DeleteNode(&look.parent->right);
    ptree->size--;

    return true;
}
```

```
void Traverse (const Tree * ptree, void (* pfun)(Item item))
{
    if (ptree != NULL)
        InOrder(ptree->root, pfun);
}

void DeleteAll(Tree * ptree)
{
    if (ptree != NULL)
        DeleteAllNodes(ptree->root);
    ptree->root = NULL;
    ptree->size = 0;
}

/* local functions */
static void InOrder(const Trnode * root, void (* pfun)(Item
item))
{
```

```
    if (root != NULL)
    {
        InOrder(root->left, pfun);
        (*pfun)(root->item);
        InOrder(root->right, pfun);
    }
}

static void DeleteAllNodes(Trnode * root)
{
    Trnode * pright;

    if (root != NULL)
    {
        pright = root->right;
        DeleteAllNodes(root->left);
        free(root);
        DeleteAllNodes(pright);
```

```
    }
}

static void AddNode (Trnode * new_node, Trnode * root)
{
    if (ToLeft(&new_node->item, &root->item))
    {
        if (root->left == NULL)      /* empty subtree        */
            root->left = new_node;   /* so add node here     */
        else
            AddNode(new_node, root->left);
                                     /* else process subtree*/
    }
    else if (ToRight(&new_node->item, &root->item))
    {
        if (root->right == NULL)
            root->right = new_node;
        else
```

```
            AddNode(new_node, root->right);
    }
    else                              /* should be no duplicates */
    {
        fprintf(stderr,"location error in AddNode()\n");
        exit(1);
    }
}

static bool ToLeft(const Item * i1, const Item * i2)
{
    int comp1;
    if ((comp1 = strcmp(i1->petname, i2->petname)) < 0)
        return true;
    else if (comp1 == 0 &&
             strcmp(i1->petkind, i2->petkind) < 0 )
        return true;
    else
```

```
        return false;
}

static bool ToRight(const Item * i1, const Item * i2)
{
    int comp1;

    if ((comp1 = strcmp(i1->petname, i2->petname)) > 0)
        return true;
    else if (comp1 == 0 &&
             strcmp(i1->petkind, i2->petkind) > 0 )
        return true;
    else
        return false;
}
```

```
static Trnode * MakeNode(const Item * pi)
{
    Trnode * new_node;

    new_node = (Trnode *) malloc(sizeof(Trnode));
    if (new_node != NULL)
    {
        new_node->item = *pi;
        new_node->left = NULL;
        new_node->right = NULL;
    }

    return new_node;
}

static Pair SeekItem(const Item * pi, const Tree * ptree)
{
    Pair look;
```

```
look.parent = NULL;
look.child = ptree->root;

if (look.child == NULL)
    return look;                        /* early return   */

while (look.child != NULL)
{
    if (ToLeft(pi, &(look.child->item)))
    {
        look.parent = look.child;
        look.child = look.child->left;
    }
    else if (ToRight(pi, &(look.child->item)))
    {
        look.parent = look.child;
        look.child = look.child->right;
    }
```

```
        else    /* must be same if not to left or right    */
            break; /* look.child is address of node with item  */
    }

    return look;    /* successful return   */
}

static void DeleteNode(Trnode **ptr)
/* ptr is address of parent member pointing to target node  */
{
    Trnode * temp;

    if ( (*ptr)->left == NULL)
    {
        temp = *ptr;
        *ptr = (*ptr)->right;
        free(temp);
    }
```

```
    else if ( (*ptr)->right == NULL)
    {
        temp = *ptr;
        *ptr = (*ptr)->left;
        free(temp);
    }
    else    /* deleted node has two children */
    {
        /* find where to reattach right subtree */
        for (temp = (*ptr)->left; temp->right != NULL;
             temp = temp->right)
            continue;
        temp->right = (*ptr)->right;
        temp = *ptr;
        *ptr =(*ptr)->left;
        free(temp);
    }
}
```

```c
/* petclub.c -- use a binary search tree */
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "tree.h"

char menu(void);
void addpet(Tree * pt);
void droppet(Tree * pt);
void showpets(const Tree * pt);
void findpet(const Tree * pt);
void printitem(Item item);
void uppercase(char * str);
char * s_gets(char * st, int n);

int main(void)
{
    Tree pets;
```

```
char choice;

InitializeTree(&pets);
while ((choice = menu()) != 'q')
{
    switch (choice)
    {
        case 'a' :  addpet(&pets);
            break;
        case 'l' :  showpets(&pets);
            break;
        case 'f' :  findpet(&pets);
            break;
        case 'n' :  printf("%d pets in club\n",
                           TreeItemCount(&pets));
            break;
        case 'd' :  droppet(&pets);
            break;
```

```
            default   :  puts("Switching error");
        }
    }
    DeleteAll(&pets);
    puts("Bye.");
    return 0;
}

char menu(void)
{
    int ch;
    puts("Nerfville Pet Club Membership Program");
    puts("Enter the letter corresponding to your choice:");
    puts("a) add a pet          l) show list of pets");
    puts("n) number of pets     f) find pets");
    puts("d) delete a pet       q) quit");
    while ((ch = getchar()) != EOF)
    {
```

```
        while (getchar() != '\n')  /* discard rest of line */
            continue;
        ch = tolower(ch);
        if (strchr("alrfndq",ch) == NULL)
            puts("Please enter an a, l, f, n, d, or q:");
        else
            break;
    }
    if (ch == EOF)          /* make EOF cause program to quit */
        ch = 'q';
    return ch;
}

void addpet(Tree * pt)
{
    Item temp;
```

```
    if (TreeIsFull(pt))
        puts("No room in the club!");
    else
    {
        puts("Please enter name of pet:");
        s_gets(temp.petname,SLEN);
        puts("Please enter pet kind:");
        s_gets(temp.petkind,SLEN);
        uppercase(temp.petname);
        uppercase(temp.petkind);
        AddItem(&temp, pt);
    }
}

void showpets(const Tree * pt)
{
    if (TreeIsEmpty(pt))
        puts("No entries!");
```

```
    else
        Traverse(pt, printitem);
}

void printitem(Item item)
{
    printf("Pet: %-19s  Kind: %-19s\n", item.petname,
           item.petkind);
}

void findpet(const Tree * pt)
{
    Item temp;

    if (TreeIsEmpty(pt))
    {
        puts("No entries!");
        return;     /* quit function if tree is empty */
```

```
    }

    puts("Please enter name of pet you wish to find:");
    s_gets(temp.petname, SLEN);
    puts("Please enter pet kind:");
    s_gets(temp.petkind, SLEN);
    uppercase(temp.petname);
    uppercase(temp.petkind);
    printf("%s the %s ", temp.petname, temp.petkind);
    if (InTree(&temp, pt))
        printf("is a member.\n");
    else
        printf("is not a member.\n");
}

void droppet(Tree * pt)
{
    Item temp;
```

```
    if (TreeIsEmpty(pt))
    {
        puts("No entries!");
        return;     /* quit function if tree is empty */
    }

    puts("Please enter name of pet you wish to delete:");
    s_gets(temp.petname, SLEN);
    puts("Please enter pet kind:");
    s_gets(temp.petkind, SLEN);
    uppercase(temp.petname);
    uppercase(temp.petkind);
    printf("%s the %s ", temp.petname, temp.petkind);
    if (DeleteItem(&temp, pt))
        printf("is dropped from the club.\n");
    else
        printf("is not a member.\n");
}
```

```c
void uppercase(char * str)
{
    while (*str)
    {
        *str = toupper(*str);
        str++;
    }
}

char * s_gets(char * st, int n)
{
    char * ret_val;
    char * find;

    ret_val = fgets(st, n, stdin);
    if (ret_val)
    {
        find = strchr(st, '\n');   // look for newline
```

```
    if (find)                   // if the address is not NULL,
            *find = '\0';       // place a null character there
        else
            while (getchar() != '\n')
                continue;               // dispose of rest of line
    }
    return ret_val;
}
```

C 程 序 设 计

C Programming

17

谢谢观看

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

（国家示范性软件学院） 博士，副教授

School of Informatics Dr. Wei Huang