



课程表程序的设计与实现

Design and Implementation of The course Table Program

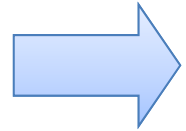
——结合软件工程的漫谈

liuwei@hust.edu.cn

2015.04.23



软件项目的开发流程



- 需求分析 Requirement Analysis
- 概要设计
- 详细设计
- 技术实现
- 测试



需求分析的原则

- 需求分析
 - 调查用户对软件的各方面的需求
 - 功能：大致功能、操作流程、.....
 - 性能：处理速度、并发能力、.....
 - 存储：存放要求、备份要求、.....
 - 扩展性：未来对功能、性能等方面的升级预期
 - ...
 - 需求分析的结果
 - 即需求分析报告，用用户的语言表达，经过用户确认，是开发方与用户方的主要交互文档



需求分析：功能、性能

- 课程表程序的功能需求

- 1. 逐条记录每次上课的记录

例如：

ID	Date	Seq.	Course
1	02-25	3	C++
2	04-27	3	C++
3	05-03	1	Digital-circuit
4	05-06	2	Math
5	05-07	1	C++

- 2. 对上课记录进行增加、删除和修改的操作

- 3. 有简单的用户界面

- 课程表程序的性能需求

- 暂无




需求分析：存储、扩展、...

- 课程表程序的存储需求
 - 1. 数据可以保存在文本文件中，程序结束后可查阅该文件
- 课程表程序的扩展性需求
 - 1. 实现基于课程的批量管理
 - 例如，输入课程名称，查阅该课程的所有授课记录
 - 例如，输入上课的周次范围、星期几、节次、课程名称，可以批量插入一个学期内该课程的多条授课记录
 - 例如，输入待修改的课程名称，可以将该课程的授课记录中的课程名都修改过来
 - 2. 实现基于日期的批量管理
 - 例如，输入某月某日，查阅该日期的所有授课记录
 - 例如，输入调课日期，将该日期的所有课程调整到另外一天
 - 3. 实现查询统计
 - 例如，该学期的所有课时数量
 - 例如，该学期最繁忙的一周



软件项目的开发流程

- 
- 需求分析
 - 概要设计 High Level Design
 - 详细设计
 - 技术实现
 - 测试



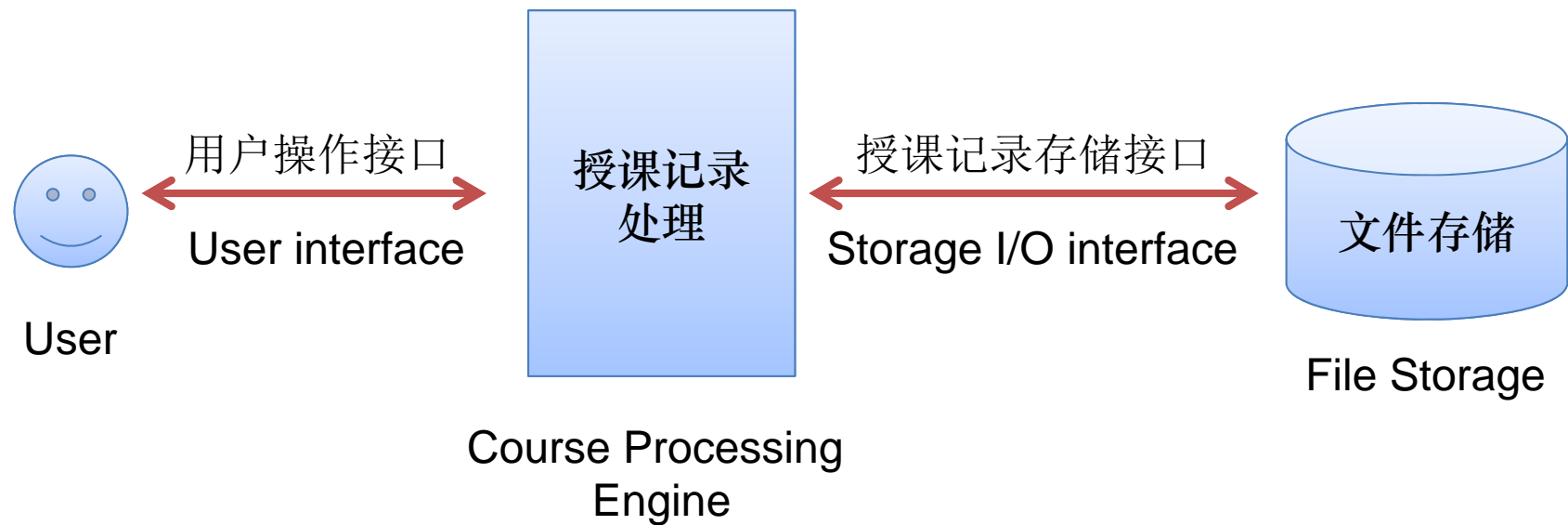
概要设计的原则

- 系统设计、概要设计、模块设计
 - 分析该系统的主要业务流程、数据流程，进行模块划分，定义不同的模块功能
 - 确定该系统的数据存储方案，影响模块实现的主要数据结构方案
 - 确定技术实现的基本路线，软件开发环境，系统运行环境等



概要设计：模块划分

- 系统模块划分





概要设计：存储方案

- 存储方案设计

- 文本文件



- ASCII码文本存储，便于调试
- 在大量数据存储时，效率比较低，适用于小程序的存储需求
- 相关C语言函数是 `fprintf()`, `fscanf()`

- 二进制文件

- 非直接可见的文本存储，调试不便
- 节省存储空间，大量数据的存储效率比较高，适用于专用程序的保存，例如microsoft office、photoshop等
- 相关C语言函数是 `fread()`, `fwrite()`

- 数据库

- 有专用的DBMS系统对数据进行管理，便于检索和统计
- 对数据库的访问需要专用的接口（例如ODBC）和专用的语言（SQL语言）



概要设计：数据结构

- 数据结构的初步设计

- 授课记录

- 可结构化数据，即每条记录的各字段类型可知
 - 数据容量上限可估计，所有课时*所有可上课天数

- 基于数组的授课记录存储

- 基本方案：C语言结构体的数组，结构体内用一个成员变量表示有课还是没课，例如 `int haveCourse = 1;`
 - 优势：程序编制容易，便于统计，通过循环遍历
 - 劣势：程序运行占用内存较多，即使无课也需要耗费存储；查询效率较低，结构体数组的每个结构体变量都需要判断是否有课

- 基于链表的授课记录存储

- 基本方案：C语言结构体的链表，结构体内用一个指针成员指向下一条授课记录
 - 优势：消耗内存少，查询时每条记录都是授课记录
 - 劣势：程序编制复杂，需要用链表的首指针遍历整个链表，统计不便





概要设计：课程表

● 小结

- 模块划分：用户交互接口、授课记录处理模块、授课记录存储接口
- 存储方案：基于文本的文件存储
- 数据结构：基于链表的授课记录

● 启发

- 程序 = 数据结构 + 算法
- 本程序中，数据结构（是数组还是链表）决定了相关操作函数的编制方式，因此数据结构的设计是软件设计的核心

Programming = Data Structure + Algorithm



软件项目的开发流程

- 需求分析
- 概要设计
- ➡ ● 详细设计 Low Level Design
- 技术实现
- 测试



详细设计的原则

- 数据结构的详细设计
 - 主要数据结构的设计，例如结构体的成员，可共享的全局变量，可共享的全局常量等
- 界面设计、模块流程设计、异常处理
 - 流程设计：对各模块的主要功能流程进行设计，数据在哪里创建/获得，下一步传给哪个函数
 - 函数设计：包括函数名、函数功能、访问限制等；
 - 异常处理：考虑特殊情况，比如用户输入错误、文件访问错误、内存不足时，程序的响应，需要设计相关的提示信息以及流程
 - 测试用例：设计主要流程的测试用例，例如当用户输入某组测试数据时，如果输入错误的预期响应是什么，如果输入正确的预期结果是什么



详细设计：数据结构 step 1

- 授课记录的数据结构设计

```
typedef struct {  
    int month;  
    int day;  
    int hourSeq;  
    char name[MAXCOURSENAME];  
} Course;
```

- 关于日期的存储
 - 分开记录month和day导致日期比较时需要判断两个值，为简化起见，改为只存储一个值，即daySeq，全年第几天的序号
`int month; int day;` 改为 `int daySeq;`
- 关于授课记录的列表
 - 因为大多数时候课程表都是按照时间排序的，因此约定链表中的节点按照授课日期和课时排序后存储，即每次插入链表时需要找到合适的插入点



详细设计：数据结构 step 2

- 授课记录的数据结构设计

```
typedef struct {  
    int daySeq;  
    int hourSeq;  
    char name[MAXCOURSENAME];  
} Course;
```

- 关于Course.id 的讨论

- 方案一：做为每条授课记录的唯一标识。每次插入新记录时自动追加编号，此后对授课记录的修改不会导致ID的修改。其好处是可以识别每条授课记录（事实上，数据库记录中常常约定每条记录都有ID）。在我们目前的项目需求中这种设计的优势不明显。



- 方案二：做为当前列表中授课记录的序号。每次有增加或者删除时，需要对授课记录的ID进行更新。该序号做为用户选择某条记录进行修改或者删除的标识。
- 本程序中采用第二种方案，因此需要考虑维护ID序号的操作。



详细设计：数据结构 step 3

- 授课记录的数据结构的设计结果

```
typedef struct {  
    int id;  
    int daySeq;  
    int hourSeq;  
    char name[MAXCOURSENAME];  
} Course;
```

- 该数据结构的使用要求小结

- 取值范围校验：日期序号[1,365]、课序号[1,8]、课程名称长度、ID序号不能超过当前列表的范围
- 顺序存储操作：ID序号由程序维护，修改授课记录时不允许用户修改，在新增、删除授课记录后需要对链表内的各记录ID进行重新排序
- 记录总数：为便于检查用户输入的ID序号的合理范围，需要设计一个函数返回当前链表中的长度Length，即记录的总数量。Length的使用是为了范围校验，而非链表的遍历，Length的值应该是即取即用的

Data-oriented programming



详细设计：数据结构小结

- 授课记录的数据结构的设计结果

```
typedef struct {  
    int id;  
    int daySeq;  
    int hourSeq;  
    char name[MAXCOURSENAME];  
} Course;
```

- 辅助函数的设计结果

// 链表的创建，与用户交互并检查各值是否在区间内，返回首节点的指针

```
Course * CreatCourseList( void );
```

// 链表的销毁，与用户交互并检查各值是否在区间内，返回是否成功

```
int DestroyCourseList( Course *list );
```

// 链表的长度

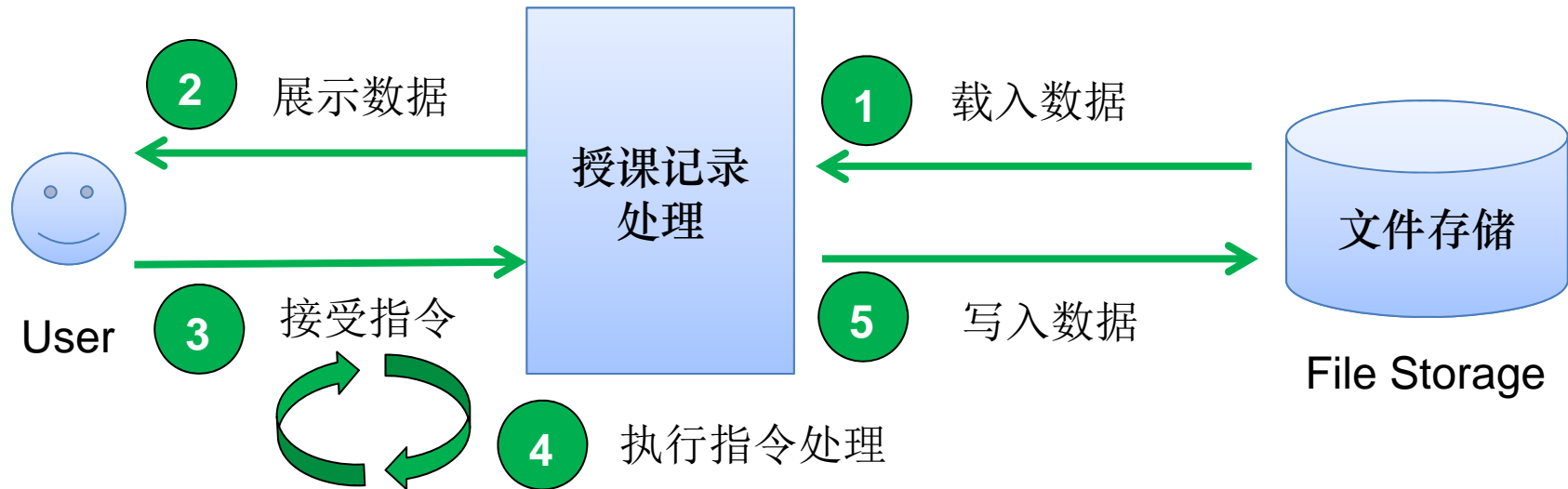
```
int GetCourseListLength ( const Course * const list );
```

// 链表的ID重排，返回重排后的链表首节点的指针

```
Course * ResortCourseListNodeID ( Course * const list );
```

详细设计：数据流向与函数划分

- 根据数据的流向依次对各模块的函数进行设计



- 模块设计的顺序依次为：存储接口、用户交互接口、授课记录处理模块
- 函数设计的常用原则
 - 可重用的代码：例如授课记录的打印
 - 需要隔离权限的代码：例如授课记录的删除
 - 对某层次数据结构进行操作的代码：例如授课记录的整体链表的操作 vs 授课记录的一条记录的操作
 - ...



详细设计：存储接口

- 授课记录存储模块

- 函数设计：step 1

- 从指定的文本文件读取记录，在程序退出时保存记录到文件

```
Course *LoadCourseList( const char *filename, Course *list);
```

```
void SaveCourseList( const char *filename, const Course * const  
list);
```

- 函数设计：step 2

- 考虑到文本文件处理大量文本记录时速度较慢，在此约定了每次文件读写的记录数，并做为函数的参数之一，该参数可用于限制内存使用的规模

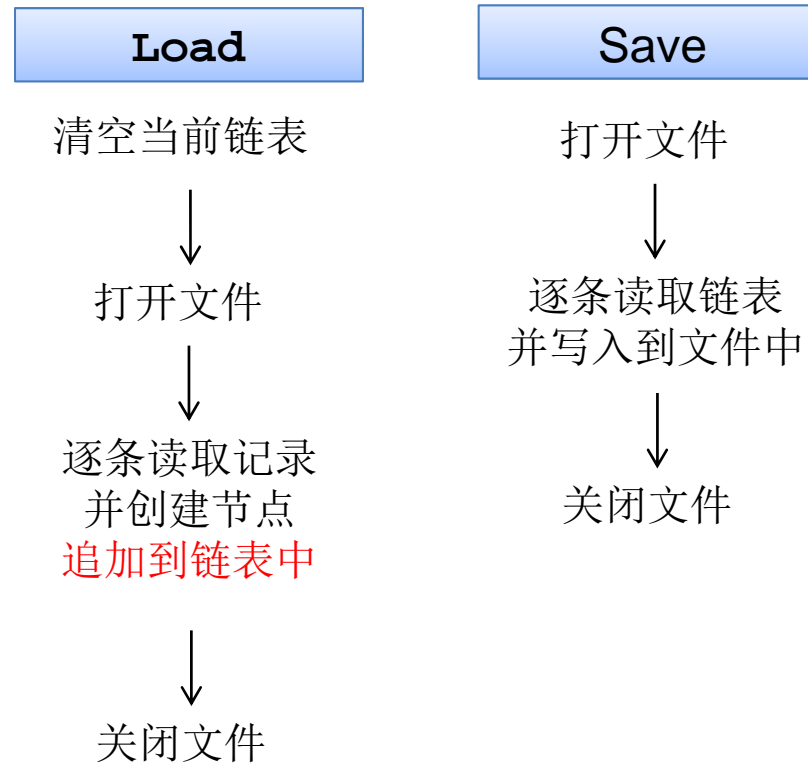
```
Course *LoadCourseList( const char *filename, Course *list, const  
int loadMax);
```

```
void SaveCourseList( const char *filename, const Course * const  
list, const int saveMax);
```



详细设计：存储接口

- 授课记录存储模块
 - 函数设计：step 3
 - 分析读取和写入文件对链表操作的需求，定义链表辅助函数



定义函数 `Course * AppendCourseNode(Course *list, Course *newCourse);`



详细设计：用户交互接口

- 用户交互接口

- 在每个回合都展示菜单，用户选择后调用相关功能函数，运行完毕返回主菜单；功能回合采用循环控制，直到用户退出该循环
- 是否需要单独放在一个.c文件里面，由程序员来决定
- 函数设计结果：

```
int main( void );
```

```
void displayMenu( void );
```

```
void cleanMenu( void );
```



详细设计：授课记录处理

- 授课记录处理模块的函数划分 step 1
 - 根据主要功能来区分，包括打印课程表、插入新课程记录、更新某课程记录、删除某课程记录

```
void DisplayCourseList(const Course *list);
```

```
Course* InsertCourseList(Course *list);
```

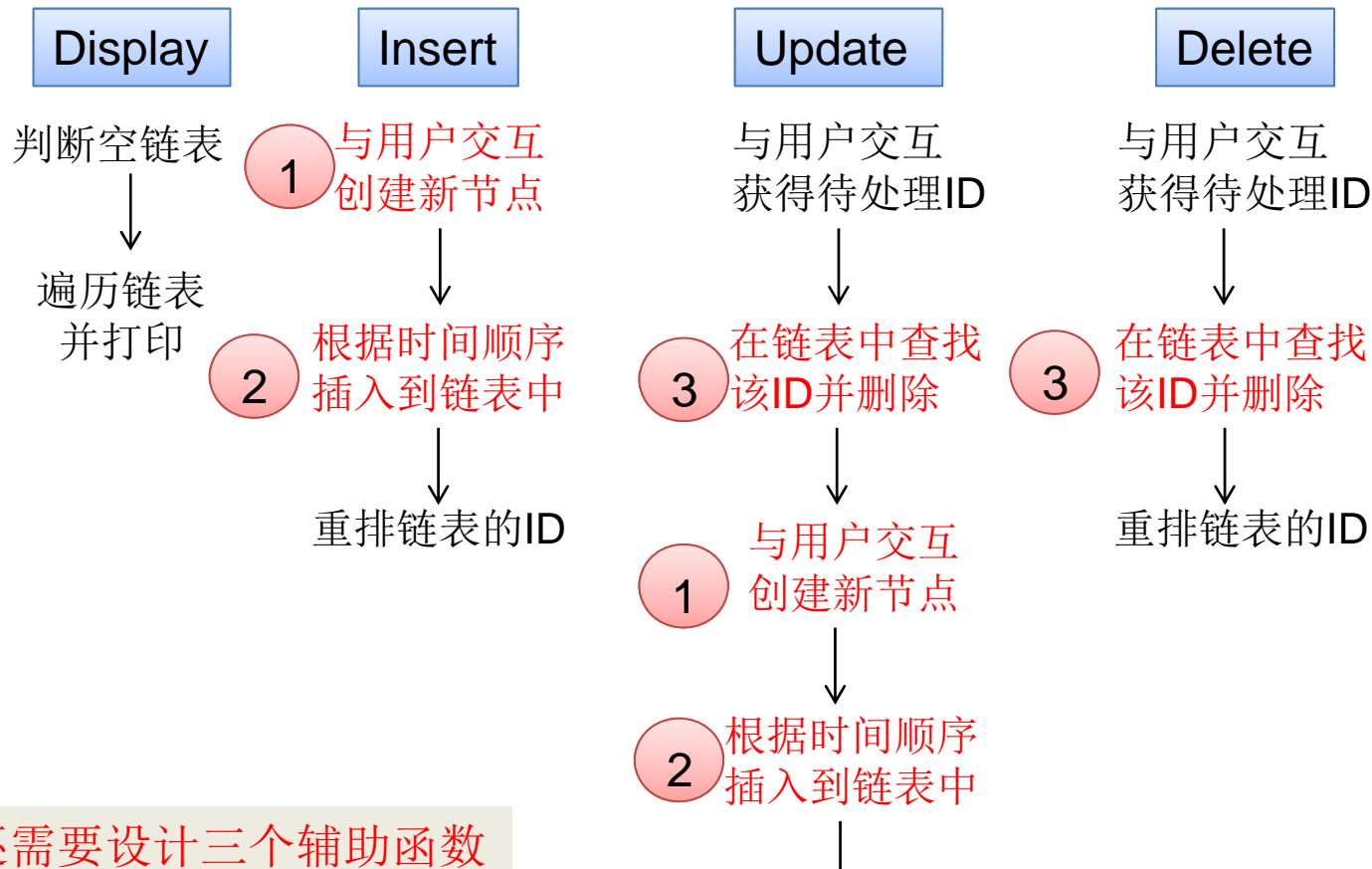
```
Course* UpdateCourseList(Course *list);
```

```
Course* DeleteCourseList(Course *list);
```



详细设计：授课记录处理

- 授课记录处理模块的函数划分 step 2
 - 分析各主要函数的流程，寻找其同性的流程，进一步设计辅助函数



```
Course * CreatCourseNode(Course *newCourse);
Course * DeleteCourseNodeByID( Course *list, int deleteID );
Course * InsertCourseNodeByTime( Course *list, Course *pNewNode );
```



详细设计：函数定义的小结 1

- 功能函数

```
void DisplayCourseList(const Course *list);  
Course* InsertCourseList(Course *list);  
Course* UpdateCourseList(Course *list);  
Course* DeleteCourseList(Course *list);
```

- 与用户交互函数

```
int main( void );  
void displayMenu( void );  
void cleanMenu( void );
```




详细设计：函数定义的小结 2

- 链表操作相关的辅助函数

// 链表的创建，该函数实际上由CreateCourseNode顶替了，不需要了

~~Course * CreateCourseList(void)~~

// 链表节点的创建，与用户交互并检查各值是否在区间内，返回新节点指针

Course * CreatCourseNode(Course *newCourse);

// 链表节点的显示

void DisplayCourseNode(Course *pCourse);

// 链表的追加，将新节点插入到链表的最后，返回首节点指针

Course * AppendCourseNode(Course *list, Course *newCourse);

// 链表的插入，按实际搜索新节点的插入位置，插入新节点，返回首节点指针

Course * InsertCourseNodeByTime(Course *list, Course *pNewNode);

// 链表的删除，按序号找到待删除节点，删除后返回首节点指针

Course * DeleteCourseNodeByID(Course *list, int deleteID);

// 链表的销毁

int DestroyCourseList(Course *list);

// 链表的长度

int GetCourseListLength (const Course * const list);

// 链表的重排

Course * ResortCourseListNodeID (Course * const list);




详细设计：函数实现的要求

- 异常处理的考虑
 - 链表辅助函数都需要考虑链表异常的情况（空链表、首节点被修改等），以便在无任何外在限制条件下就可以被其它的功能函数调用
 - 与用户交互函数都需要考虑输入值的范围（比如日期越界、记录ID越界等），以便保证程序数据的“干净”
 - 文件操作的函数都需要考虑文件访问失败的情况（比如文件被意外删除、文件找不到等），在任何情况下都不能因此死机
- 测试用例的设计
 - 测试符合预期的主要流程，比如正常的插入、删除、更新等
 - 测试特殊流程下的程序响应，比如链表的全部删除、首节点的修改等
 - 测试异常处理的流程，看程序能否正常报错和终止



软件项目的开发流程

- 需求分析
- 概要设计
- 详细设计
-  ● 技术实现 Implementation
- 测试



技术实现的原则

- 开发实现
 - 以模块、函数为单位逐一开发
 - 开发时循序渐进，步步为营，增量式的开发，不要一口气吃个胖子
 - 先开发辅助函数，并进行测试
 - 再开发功能函数，并进行测试
 - 最后开发与用户交互的函数，并进行测试



```
4 struct node
5 {
6     int value;
7     struct node * next;
8 };
9 typedef struct node Node;
10
11 int main()
12 {
13     // create a linked list
14     Node *n1, *n2, *n3 ;
15
16     n1 = malloc( sizeof(Node) );
17     n2 = malloc( sizeof(Node) );
18     n3 = malloc( sizeof(Node) );
19
20     n1->value = 10;
21     n1->next = n2;
22     n2->value = 20;
23     n2->next = n3;
24     n3->value = 30;
25     n3->next = NULL;
26
27     // navigate this linked list
28     Node * pNode;
29     printf("this linked list is:\n");
30     for ( pNode = n1 ; pNode != NULL ; pNode = pNode->next )
31     {
32         printf(" %4d ", pNode->value );
33     }
34     printf("\n");
35
36     // destroy this linked list
37     free(n1);
38     free(n2);
39     free(n3);
40
41 }
```

开发过程的建议:

1. 可以把单独的功能写在独立的小程序里面;

2. 程序指定输入, 测试后续代码能否达到意图;

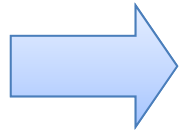
3. 如果代码达到意图, 则将其函数化, 巩固编程结果

4. 继续下一个功能



软件项目的开发流程

- 需求分析
- 概要设计
- 详细设计
- 技术实现
- 测试 Testing





测试的原则

- 白盒测试 vs 黑盒测试
 - 白盒测试：人工检查代码，设计测试用例，遍历每个可能的流程
 - 黑盒测试：请不熟悉程序的人员操作测试，随机输入，看看程序的响应

- 建议的测试步骤
 - 1. 功能流程的测试：先测试正常流程，看程序是否有预期的输出，再测试异常流程
 - 2. 用户交互的测试：请用户请不熟悉程序的人员操作测试，随机输入，看看程序的响应
 - 3. 压力测试与扩展性测试：输入边界值、输入超过预期数量的值等，测试程序能否正常响应
 -

讨论：排序、交换、数组、链表

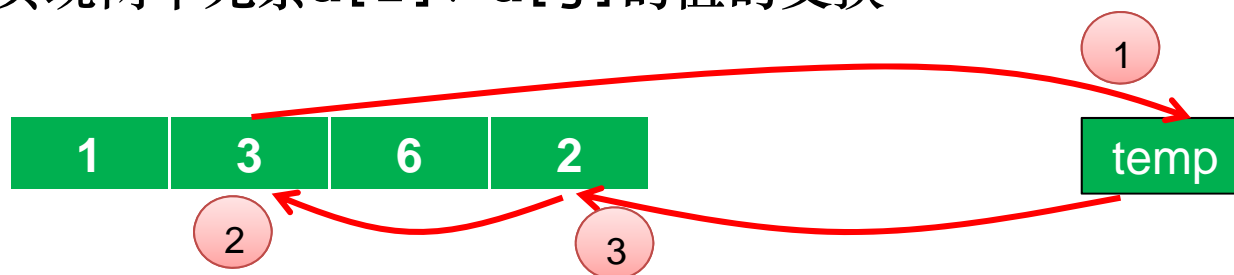


- 排序中的交换

- 排序是对数据序列进行处理的常用操作，并不局限于存储的实现方式。基于数组和链表的数据结构都可以实现典型的排序算法，比如冒泡排序，但是其交换节点的操作却很不同

- 基于数组的排序、交换

- 数组是顺序存储的，通过数组下标index，因此可以通过下标i和j实现两个元素 $a[i]$ 、 $a[j]$ 的值的交换



- 基于链表的排序、交换

- 链表是非连续存储的，没有现成的index可以用，只能通过修改指针指向的方式来实现两个节点在链表中“顺序”的交换
- 在本程序中，没有必要为每个节点保留其序号值，因此可以采用删除再插入的方式来处理



谢谢!



刘威 副教授

互联网技术与工程研究中心
华中科技大学电子与信息工程系

Email: liuwei@hust.edu.cn