

软件开发过程



目录 | CONTENTS



01

结构化程序设计

02

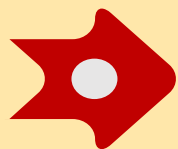
软件开发过程

目录 | CONTENTS



01

结构化程序设计



自顶向下的分解

模块划分

库的设计与实现

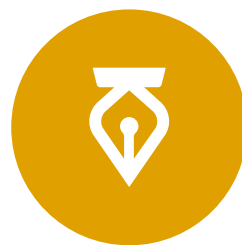
库的应用

猜硬币的游戏

功能：



提供游戏指南



计算机随机产生正反面，
让用户猜，报告对错结果

重复此过程，直到用户不想玩了为止。

顶层分解

程序要做两件事：显示程序指南；模拟玩游戏的过程。

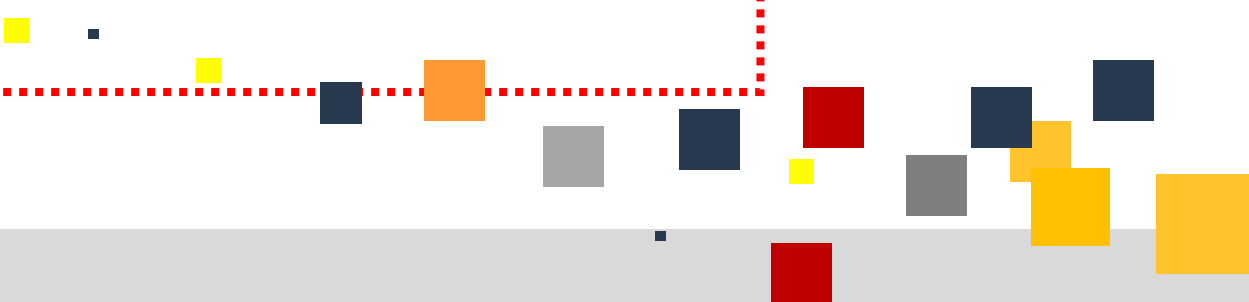
```
main( )  
{  
    显示游戏介绍；  
    玩游戏；  
}
```

主程序的两个步骤是相互独立，没有任何关联，因此可设计成两个函数：

```
void prn_instruction()  
void play( )
```

顶层分解

```
int main()  
{  
    prn_instruction();  
    play();  
  
    return 0;  
}
```

A decorative graphic at the bottom of the slide consisting of a horizontal line of small squares in various colors (yellow, dark blue, orange, grey, red) that tapers off to the right, with additional squares scattered below it.

prn_instruction的实现

prn_instruction函数的实现非常简单，只要一系列的输出语句把程序指南显示一下就可以了。

```
void prn_instruction()
{
    printf("这是一个猜硬币正反面的游戏。\\n");
    printf("我会扔一个硬币，你来猜。\\n");
    printf("如果猜对了，你赢，否则我赢。\\n");
}
```

play函数的实现

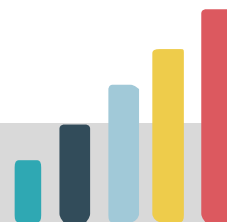
Play函数随机产生正反面，让用户猜，报告对错结果，然后询问是否要继续玩。

```
void play()
{ char flag = 'y' ;
  while (flag == 'Y' || flag == 'y' )
  { coin = 生成正反面;
    输入用户的猜测 ;
    if ( 用户猜测 == coin )
        报告本次猜测结果正确 ;
    else 报告本次猜测结果错误 ;
    询问继续玩吗？输入flag
  }
}
```


play函数的细化

生成正反面：

如果用0表示正面，1表示反面，那么生成正反面就是随机生成0和1两个数。



play函数的细化

输入用户的猜测。如果不考虑程序的鲁棒性，这个问题也可以直接用一个输入语句即可。但想让程序做得好一点，就必须考虑得全面一些。比如，用户可以不守规则，既不输入0也不输入1，而是输入一个其他值，程序该怎么办？因此这个任务还可以进一步细化，所以再把它抽象成一个函数 `get_call_from_user`。

```
void play() {  
    int coin ;  
    char flag = 'Y';  
    srand(time(NULL));  
    while (flag == 'Y' || flag == 'y') {  
        coin = rand() * 2 / (RAND_MAX + 1);  
        if (get_call_from_user() == coin)  
            printf("你赢了");  
        else printf("我赢了");  
        printf("\n继续玩 ( Y或y ) ,输入任意其他字符表示不玩 : ");  
        scanf("\n%c", &flag);  
    }  
}
```



get_call_from_user函数的实现

该函数接收用户输入的一个整型数。如果输入的数不是0或1，则重新输入，否则返回输入的值。

```
int get_call_from_user()
{
    int guess;
    do {
        printf( "\n输入你的选择
        ( 0表示正面 , 1表示反面 ) :");
        scanf("%d", &guess);
    } while (guess != 0 && guess != 1);
    return guess;
}
```

运行实例

这是一个猜硬币正反面的游戏。
我会扔一个硬币，你来猜。
如果猜对了，你赢，否则我赢。



运行实例

输入你的选择（ 0表示正面， 1表示反面 ）:1 我赢了
继续玩吗（ Y或y ） ? y

输入你的选择（ 0表示正面， 1表示反面 ）:6

输入你的选择（ 0表示正面， 1表示反面 ）:1 你赢了
继续玩吗（ Y或y ） ? n

Press any key to continue



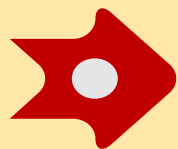
目录 | CONTENTS



01

结构化程序设计

自顶向下的分解



模块划分

库的设计与实现

库的应用

模块划分

■ 当程序由很多函数组成时，要在一个单独的源文件中处理如此众多的函数会变得困难。

■ 把程序再分成几个小的源文件。每个源文件都包含一组相关的函数。一个源文件被称为一个模块。



模块划分

模块划分标准：



同一模块中的
函数比较类似。



块内联系尽可能大，块间联系尽可能小。

石头、剪刀、布游戏

游戏规则：

布覆盖
石头

石头砸坏
剪刀

剪刀
剪碎布

石头、剪刀、布游戏

游戏的过程为：

游戏者选择出石头、剪子或布，计算机也随机选择一个，输出结果，继续游戏，直到游戏者选择结束为止。在此过程中，游戏者也可以阅读游戏指南或看看当前战况。



第一层的分解

```
While ( 用户输入 != quit ) {  
    switch ( 用户的选择 ) {  
        case paper, rock, scissor: 机器选择 ;  
                                     评判结果 ;  
                                     报告结果 ;  
  
        case game: 显示目前的战况 ;  
        case help: 显示帮助信息 ;  
        default: 报告错误 ;  
    }  
    显示战况 ;  
}
```



函数抽取

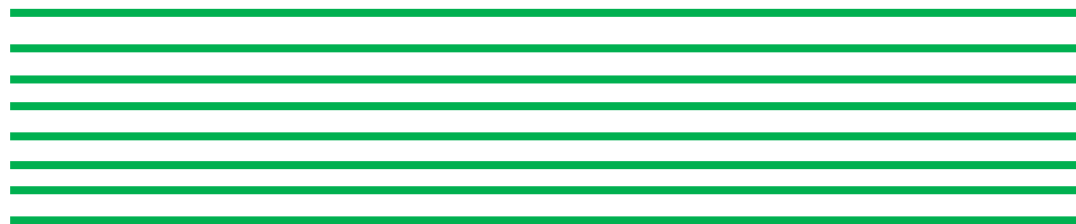
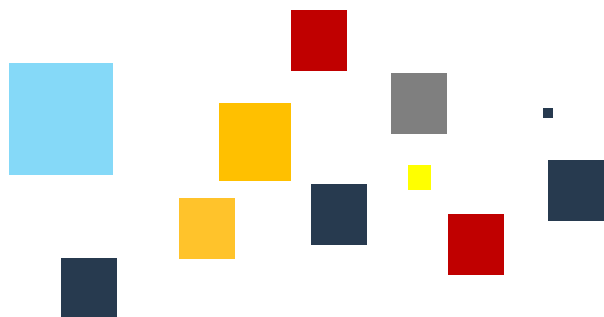
- ▶ 获取用户输入selection_by_player
- ▶ 获取机器输入selection_by_machine
- ▶ 评判结果compare
- ▶ 报告结果并记录结果信息report
- ▶ 显示目前战况prn_game_status
- ▶ 显示帮助信息prn_help六个函数

枚举类型的定义

为了提高程序的可读性，我们定义两个枚举类型：

```
enum p_r_s {paper, rock, scissor, game,  
            help, quit} ;
```

```
enum outcome {win, lose, tie} ;
```



模块划分

分成四个模块：

1.主模块

main函数

2.获取选择的模块

Selection_by_player

selection_by_machine

模块划分

3.比较模块

compare

4.输出模块

Report、
prn_game_status和
prn_help函数



Select模块的设计

selection_by_player从键盘接收用户的输入并返回此输入值。因此，原型为：

```
enum p_r_s selection_by_player ( ) ;
```

selection_by_machine函数由机器产生一个石头、剪子、布的值，并返回。因此，原型为：

```
enum p_r_s selection_by_machine ( ) ;
```

Compare模块的设计

- compare函数比较用户输入的值和机器产生的值，确定输赢。
- 它要有两个参数，都是p_r_s类型的，它也应该有一个返回值，就是判断的结果。
- 原型为：

```
enum outcome compare ( enum p_r_s ,  
                        enum p_r_s ) ;
```

print模块的设计

- ➔ `prn_help`显示一个用户输入的指南，告诉用户如何输入他的选择。因此，它没有参数也没有返回值。
- ➔ `Report`函数报告输赢结果，并记录输赢的次数。因此它必须有四个参数：输赢结果、输的次数、赢的次数和平局的次数，但没有返回值。



print模块的设计

- ➔ `prn_game_status`函数报告至今为止的战况，因此需要三个参数：输的次数、赢的次数和平的次数，但没有返回值。



print模块的进一步考虑

01

OPTION

输的次数、赢的次数和平局的次数在Report和prn_game_status两个函数中都出现。

02

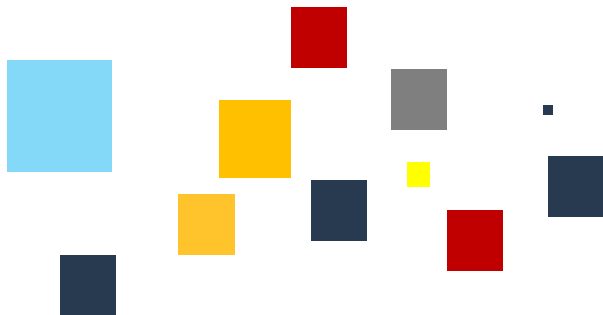
OPTION

Report函数修改这些变量的值，
prn_game_status函数显示这些变量的值。

03

OPTION

这三个函数的原型和用户期望的原型不一致，
用户不希望原型中有这些参数。



print模块的进一步考虑

04
OPTION

输的次数、赢的次数和平局的次数和其他模块的函数无任何关系，因此可作为该模块的**内部状态**。

05
OPTION

内部状态可以作为该模块的全局变量。

06
OPTION

这样report和prn_game_status函数中都不需要这三个参数了。

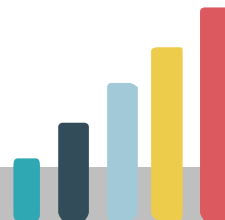
头文件的设计



为方便起见，我们把所有的符号常量定义、类型定义和函数原型声明写在一个头文件中，让每个模块都include这个头文件。



那么，每个模块就不必要再写那些函数的原型声明了。



头文件的设计

但这样做又会引起另一个问题，当把这些模块连接起来时，编译器会发现这些类型定义、符号常量和函数原型的声明在程序中反复出现多次。

解决方法：

需要用到一个新的编译预处理命令：

`#ifndef` 标识符

...

`#endif`

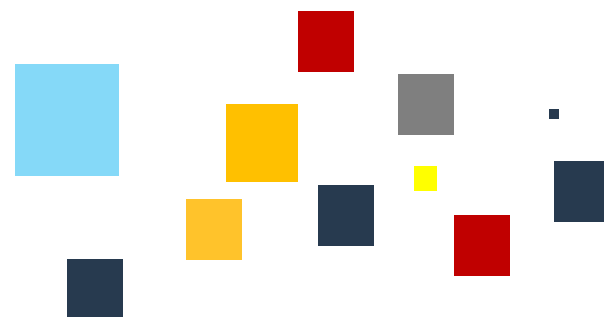
头文件的格式

```
#ifndef _name_h
```

```
#define _name_h
```

头文件真正需要写的内容

```
#endif
```



石头、剪子、布游戏的头文件

```
// 文件 : p_r_s.h  
  
// 本文件定义了两个枚举类型，声明了本程序包括的所有函数原型  
  
#ifndef P_R_S  
  
#define P_R_S  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
enum p_r_s {paper, rock, scissor, game, help, quit} ;  
enum outcome {win, lose, tie} ;
```



石头、剪子、布游戏的头文件

```
enum outcome compare(enum p_r_s player_choice,  
                     enum p_r_s machine_choice);  
  
void prn_final_status();  
void prn_game_status();  
void prn_help();  
void report(enum outcome result);  
enum p_r_s selection_by_machine();  
enum p_r_s selection_by_player();  
  
#endif
```



主模块的实现

注意，用
双引号

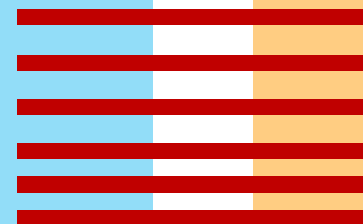
```
#include "p_r_s.h "  
  
int main()  
{  
  
    enum outcome result;  
  
    enum p_r_s player_choice, machine_choice;  
  
    srand(time(NULL));
```

```
while((player_choice = selection_by_player()) != quit)
    switch(player_choice) {
        case paper:    case rock:    case scissor:
            machine_choice = selection_by_machine();
            result = compare(player_choice, machine_choice);
            report(result); break;
        case game: prn_game_status(); break;
        case help: prn_help();}
    prn_game_status();
    return 0;
}
```



select模块的实现

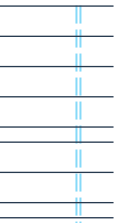
```
#include "p_r_s.h"
enum p_r_s selection_by_machine( ) {
    int select = (rand( ) * 3 / (RAND_MAX + 1));
    printf(" I am ");
    switch(select){
        case 0: printf("paper. "); break;
        case 1: printf("rock. "); break;
        case 2: printf("scissor. "); break; }
    return ((enum p_r_s) select);
}
```



```
enum p_r_s selection_by_player()
{
    char c;
    enum p_r_s player_choice;
    prn_help();
    printf("please select: "); c = getchar();
    switch(c) {
        case 'p': player_choice = paper;
                 printf("you are paper. ");
                 break;
```



```
    case 'r': player_choice = rock;
               printf( "you are rock. ");
               break;
    case 's': player_choice = scissor;
               printf( "you are scissor. ");
               break;
    case 'g': player_choice = game; break;
    case 'q': player_choice = quit; break;
    default: player_choice = help; break;
}
return player_choice;
}
```



Compare模块的实现

```
#include "p_r_s.h"

enum outcome compare(enum p_r_s player_choice, enum p_r_s
machine_choice)
{ enum outcome  result;
  if (player_choice == machine_choice)  return tie;
  switch(player_choice) {
    case paper: result = (machine_choice == rock) ? win : lose; break;
    case rock: result = (machine_choice == scissor) ? win : lose; break;
    case scissor: result = (machine_choice == paper) ? win : lose; break;
  }
  return result;
}
```



Print模块的实现

```
#include "p_r_s.h"
```

```
static int win_cnt = 0, lose_cnt = 0, tie_cnt = 0;

void prn_game_status()
{
    printf("\nGAME STATUS:\n");
    printf("win: %d\n", win_cnt);
    printf("Lose: %d\n ", lose_cnt );
    printf("tie:  %d\n ", tie_cnt);
    printf("Total: %d\n ", win_cnt + lose_cnt + tie_cnt);
}
```



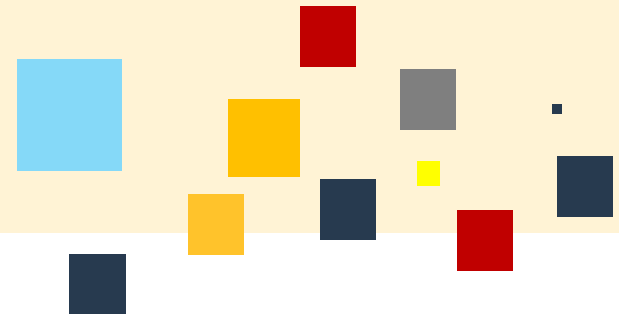
Print模块的实现

```
void report(enum outcome result){  
    switch(result) {  
        case win: ++win_cnt;  
                printf("You win. \n");  
                break;  
        case lose: ++lose_cnt;  
                printf("You lose.\n");  
                break;  
        case tie: ++tie_cnt; printf("A tie.\n");  
    }  
}
```



Print模块的实现

```
void prn_help()
{
    printf("\nThe following characters can be used:\n");
    printf("  p  for paper\n");
    printf("  r  for rock\n");
    printf("  s  for scissors\n");
    printf("  g  print the game status\n");
    printf("  h  help, print this list\n");
    printf("  q  quit the game\n");
}
```



目录 | CONTENTS



01

结构化程序设计

自顶向下的分解

模块划分



库的设计与实现

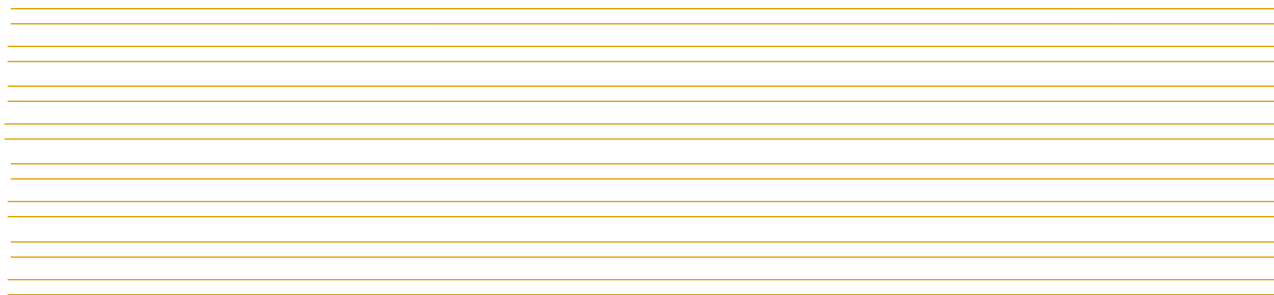
库的应用

库的设计与实现

- ➡ 如果你的工作经常要用到一些特殊的工具，你可以设计自己的库
- ➡ 一个库应该有一个主题。一个库中的函数都应该是处理同一类问题。如标准库stdio包含输入输出功能，math包含数学运算函数。我们自己设计的库也要有一个主题。

库的设计与实现

设计一个库还要考虑到它的通用性。库中的功能应来源于某一应用，但不局限于该应用，而且要高于该应用。在某一应用程序中提取库内容时应尽量考虑到兼容更多的应用，使其他应用程序也能共享这个库。



库的设计与实现

设计库的接口：



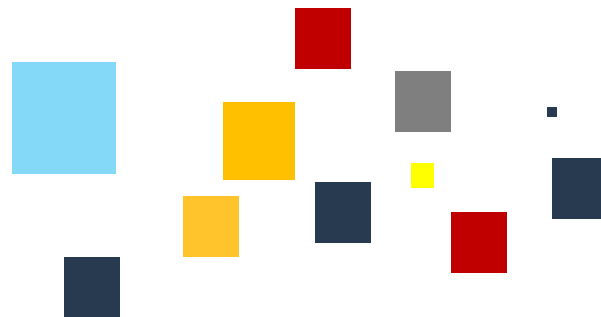
A

库的用户必须了解的内容，包括库中函数的原型、这些函数用到的符号常量和自定义类型。



B

接口表现为一个头文件。



库的设计与实现

设计库中的函数的实现：表现为一个源文件。

一般，源文件与头文件有相同的前缀名

库的这种实现方法称为信息隐藏。



随机函数库的设计与实现

设计了一个掷硬币的程序。该程序用到了随机数的一些特性。如果我们的工作经常需要用到随机数，我们可以把随机数的应用写成一个库。

随机函数库的设计与实现

库的功能：

在自动出题中，用到了随机生成0和3及随机生成0到9。

用一个函数概括：生成low到high之间的随机数

```
int RandomInteger(int low, int high)
```

初始化函数RandomInit()实现设置随机数种子的功能。

接口文件

头文件的格式

与石头、剪子、布游戏中的头文件格式一样。

头文件中，每个函数声明前应该有一段注释，告诉用户如何使用这些函数。



库接口的设计

```
/* 文件：Random.h  
   随机函数库的头文件  */
```

```
#ifndef _random_h  
#define _random_h  
/*
```

函数：RandomInit

用法：RandomInit()

作用：此函数初始化随机数种子

```
*/
```

```
void RandomInit();
```

```
/*
```

函数：RandomInteger

用法：n = RandomInteger(low, high)

作用：此函数返回一个low到high之间的随机数，包括low和high。

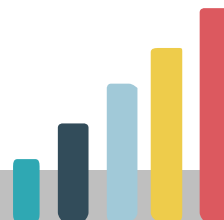
```
*/
```

```
int RandomInteger(int low, int high);
```

```
#endif
```

库的实现

库的实现文件和头文件的名字是相同的。
如：头文件为Random.h，则实现文件为Random.c。



库的实现

实现文件的格式：

- ➡ 注释：这一部分简单介绍库的功能。
- ➡ `include`此源文件所需的头文件。
- ➡ 每个实现要包含自己的头文件，以便编译器能检查函数定义和函数原型声明的一致性。
- ➡ 每个函数的实现代码。在每个函数实现的前面也必须有一段注释。

库的实现

```
/* 文件：Random.c  
   该文件实现了Random库 */
```

```
#include <stdlib.h>  
#include <time.h>  
#include "Random.h"
```

```
/* 函数：RandomInit  
   该函数取当前系统时间作为随机数发生器的种子 */
```

```
void RandomInit()  
{  
    srand(time(NULL));  
}
```



库的实现

/* 函数：RandomInteger

该函数将0到RAND_MAX的区间的划分成 $high - low + 1$ 个子区间。当产生的随机数落在第一个子区间时，则映射成low。当落在最后一个子区间时，映射成high。当落在第 i 个子区间时（ i 从0到 $high - low$ ），则映射到 $low + i$

*/

```
int RandomInteger(int low, int high)
{
    return (low + (high - low + 1) * rand() / (RAND_MAX + 1));
}
```



目录 | CONTENTS



01

结构化程序设计

自顶向下的分解

模块划分

库的设计与实现



库的应用

库的应用 -- 龟兔赛跑

动物	跑动类型	占用时间	跑动量
乌龟	快走	50%	向前走3点
	后滑	20%	向后退6点
	慢走	30%	向前走一点
兔子	睡觉	20%	不动
	大后滑	20%	向后退9点
	快走	10%	向前走14点
	小步跳	30%	向前走3点
	慢后滑	20%	向后退2点

龟兔赛跑解题思路

01
OPTION

分别用变量tortoise和hare代表乌龟和兔子的当前位置。

02
OPTION

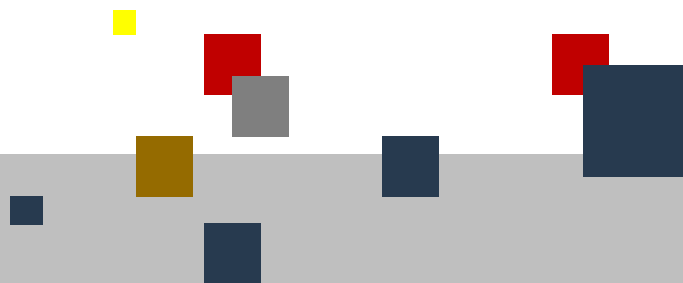
时间用秒计算。

03
OPTION

用随机数来决定乌龟和兔子在每一秒的动作，根据动作决定乌龟和兔子的位置的移动。

04
OPTION

跑道的长度设为70个点。



第一层分解

```
main() {  
    int hare = 0, tortoise = 0, timer = 0;  
    while (hare < RACE_END && tortoise < RACE_END)  
        { tortoise += 乌龟根据他这一时刻的行为移动的距离 ;  
          hare += 兔子根据他这一时刻的行为移动的距离;  
          输出当前计时和兔子乌龟的位置;  
          ++timer;  
        }  
    if (hare > tortoise) 输出 " hare wins!";  
    else 输出 " tortoise wins!";  
}
```



抽取函数

乌龟在这一秒的移动距离：

```
int move_tortoise();
```

兔子在这一秒的移动距离：

```
int move_hare();
```

输出当前计时和兔子乌龟的位置

```
void print_position(int timer, int tortoise,  
int hare);
```

模块划分

主模块
01



移动模块
02

move_tortoise
move_hare

输出模块
03

print_position



主模块

```
#include "Random.h"  
#include <stdio.h>  
#define RACE_END 70  
int move_tortoise();  
int move_hare();  
void print_position(int, int, int);
```



```
int main()
{
    int hare = 0, tortoise = 0, timer = 0;
    RandomInit();
    printf("timer tortoise hare\n " );

    while (hare < RACE_END && tortoise < RACE_END) {
        tortoise += move_tortoise();
        hare += move_hare();
        print_position(timer, tortoise, hare);
        ++timer;
    }

    if (hare > tortoise)
        printf("\n hare wins!\n");
    else printf("\n tortoise wins!\n");
    return 0;
}
```



Move模块

如何用随机数模拟概率，以乌龟为例，它的三种情况和概率右表：

快走	50%
后滑	20%
慢走	30%

为此我们可以生成0-9之间的随机数，当生成的随机数为0-4时，认为是第一种情况，5-6是第二种情况，7-9是第三种情况。这样就可以根据生成的随机数确定发生的事件。

Move模块

```
#include "Random.h"
int move_tortoise()
{
    int probability = RandomInteger(0,9);
    if (probability < 5) return 3;
    else if (probability < 7) return -6;
        else return 1;
}
```



Print模块

```
#include <stdio.h>
```

```
void print_position(int timer, int t, int h)
{
    if (timer % 6 == 0)
        putchar('\n');
    printf("%d\t%d\t%d\n", timer, t, h );
}
```



目录 | CONTENTS



01

结构化程序设计

02

软件开发过程

目录 | CONTENTS



02

软件开发过程



软件工程

软件生命周期

软件开发实例

软件危机

个人时代

- ➡ 20世纪60年代中期以前。
- ➡ 针对具体应用，规模较小，编写者和使用者是同一人。
- ➡ 存档资料只有程序。

小作坊时代

- ➡ 60年代中期到70年代中期。
- ➡ 生产软件产品。

软件危机

01
OPTION

如何合作完成一个软件？

02
OPTION

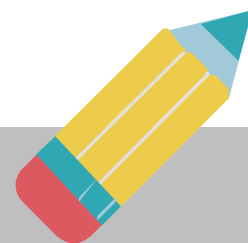
产品需要维护，需要移植



软件工程闪亮登场！

软件工程

- 借鉴现代化工业生产的生产模式生产软件。
- 从管理和技术两方面研究如何更好地开发和维护计算机软件。
- 原理



	程序设计时代	软件时候	软件工程时代
名称	程序	软件	软件产品
生产方式	个人	项目组	软件组织
质量	取决于个人水平	取决于小集团水平	软件生产管理
设计对象	以硬件为中心	硬件/软件为中心	以软件为中心
开发工具	无	无系统工具，工具为个人所有	软件生成器

	程序设计时代	软件时候	软件工程时代
维护	无	由开发者进行维护，在设计中不重视维护设计问题。	设计与制作过程中均考虑维护问题，维护成本占很大比重。
设计方法	无	自顶向下	结构设计、原型设计
开发特点	以技巧为主	有质量保证问题和持续性问题	

软件工程-----原理

01

OPTION

用分阶段的生命周期计划严格管理。

02

OPTION

坚持进行阶段评审：设计错误63%，编码错误37%。

03

OPTION

严格的产品控制：严格按照设计文档工作。

04

OPTION

采用现代程序设计技术，提高软件质量。

05

OPTION

结果能清楚地审查。

06

OPTION

开发成员少而精。



目录 | CONTENTS



02

软件开发过程

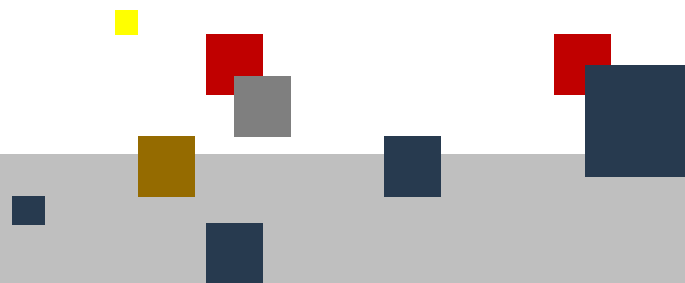
软件工程



软件生命周期

软件开发实例

软件生命周期



可行性研究

确定问题能否解决？是否值得解决？

系统分析员一般从以下3个角度论证：

技术可行性

01

现有技术能
解决这个问题吗？

经济可行性

02

有能力做吗？
效益超过成本吗？

操作可行性

03

系统操作方式
在用户组织中
行得通吗？

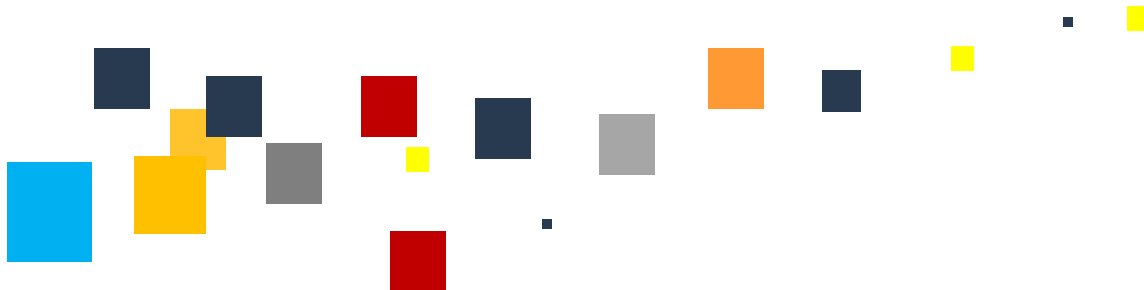
可行性研究

结果

可行性研究报告




系统开发计划



需求分析

- 对目标系统提出完整、准确、清晰、具体的要求，包括功能、性能、可靠性、可用性、将来可能的修改等，还要包括每个功能处理的数据以及处理的流程
- 由系统分析员和用户共同完成
- 结果：



软件开发规格说明书
修正系统开发计划

总体设计

如何实现目标系统

具体工作

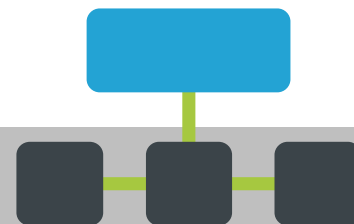
- 1.确定系统的实现方案：如开发环境、开发工具。
- 2.设计系统的结构，即模块划分。
- 3.数据库设计
- 4.确定测试计划

结果

- 1.系统说明
- 2.用户手册
- 3.数据库设计
- 4.测试计划
- 5.详细实现计划

详细设计

- ➡ 也称模块设计
- ➡ 确定实现模块功能所需的算法和数据结构，在编码阶段可直接转换成程序
- ➡ 相当于普通产品设计中画出所有的零件设计图



编码和单元测试

- 📎 根据详细设计编写程序
- 📎 测试每个程序模块，保证每个模块的正确性

**编程者
自己测试：**

**质量控制
人员的验收测试：**

白盒法

黑盒法

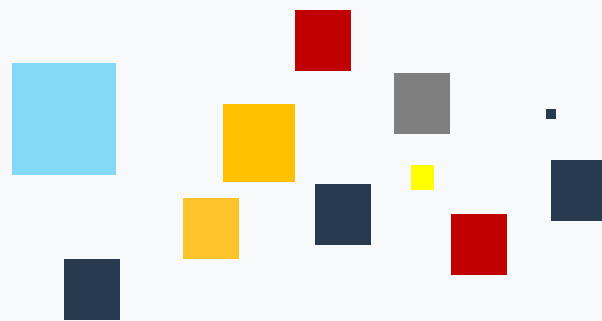
编码和单元测试

测试设计

分析测试情况

- ➡ 正常情况
- ➡ 非正常情况
- ➡ 边界情况

设计测试用例



综合测试

对完整的系统进行测试，分为：

集成测试

根据软件结构把各个程序模块连接起来进行测试

验收测试

按照需求阶段的规格说明书由用户对系统进行验收

目录 | CONTENTS



02

软件开发过程

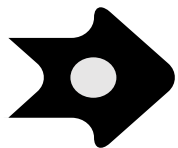
软件工程

软件生命周期



软件开发实例

目录 | CONTENTS



软件开发实例

■ 学生管理系统

网上书店

学生管理系统

某个学校的学生管理系统必须满足以下需求

01

OPTION

输入学生基本信息：学生基本信息包括学号、姓名、性别和班级。

02

OPTION

输入学生考试成绩：任课教师输入某门课程学生成绩。

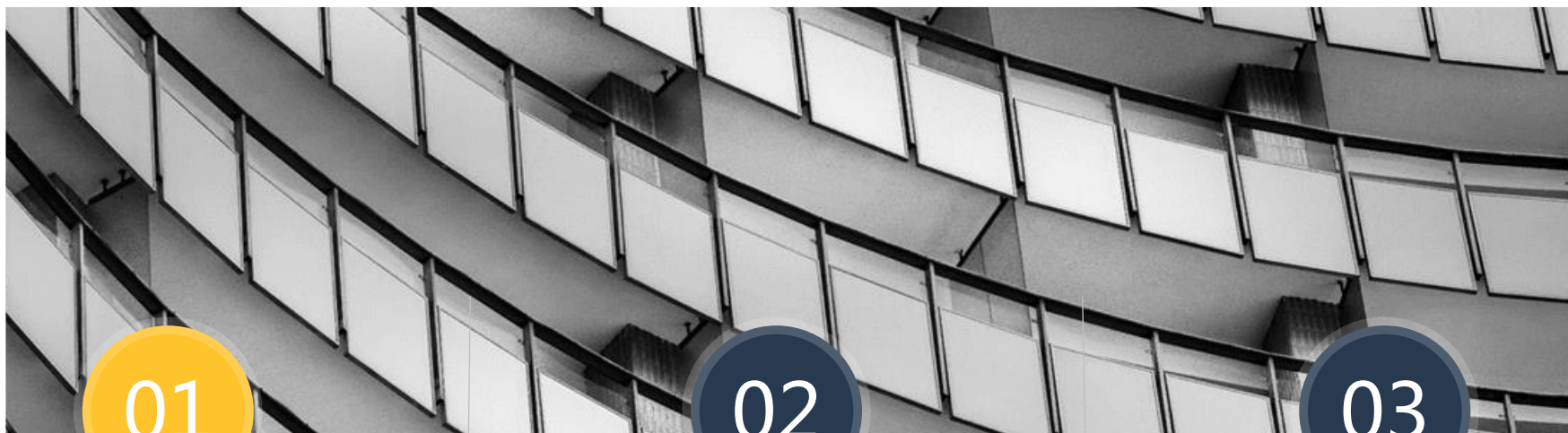
03

OPTION

显示学生信息：根据学号显示该生的概要信息和学习状况。



学生管理系统



01

需求分析

02

总体设计

03

详细设计

需求分析

学生基本信息输入

学生的基本信息包括：学号、姓名、性别和班级。学号是一个整数。姓名是长度为10的字符串。性别用一个字符表示，男生用M，女生用F。班级是一个长度为10的字符串。学号是一个流水号。第一个入学的学生学号为1，第二个为2，以此类推。



需求分析

学生基本信息输入

一般学生都是成批入学。进入这个功能可以输入一批学生的基本信息，直到用户选择输入结束。由于学号是流水号，可以由系统自动生成。输入时只需要输入姓名、性别和班级。

需求分析

学生基本信息输入

进入此功能，屏幕上显示新输入学生的学号，然后提示用户输入姓名、性别和班级。在接受完输入后，显示该生的信息供用户检查是否有错。如果有错，则重新输入

需求分析

输入学生的考试成绩

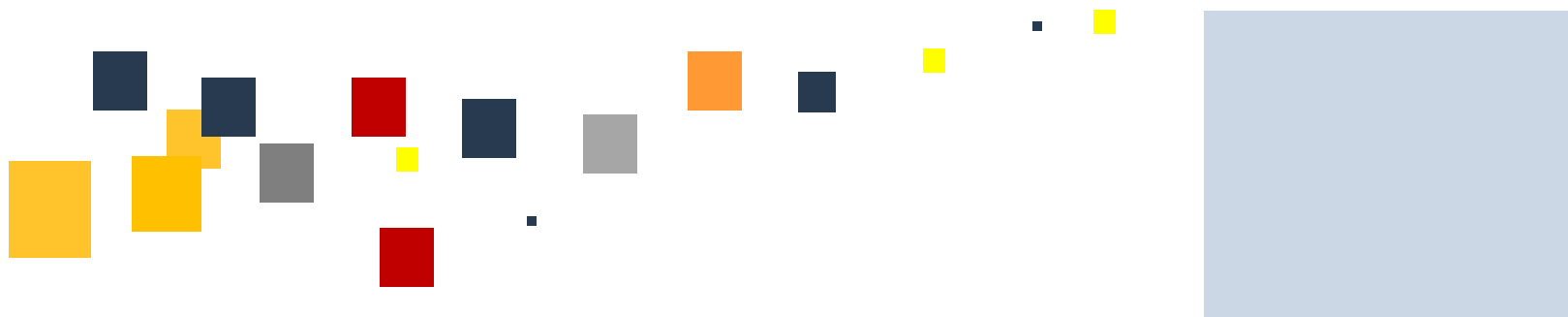
- 考试成绩是由任课老师输入的。每次输入的是同一门课的成绩，所以不管有多少学生参加考试，课程名字只需要输入一次。课程名是长度为10的字符串。
- 进入该功能，首先提示输入课程名。然后逐个输入学生的学号和成绩，直到输入结束。



需求分析

显示学生信息

进入该功能，输入学号，显示该学生的基本信息
和所有课程的成绩。



学生管理系统



01

需求分析

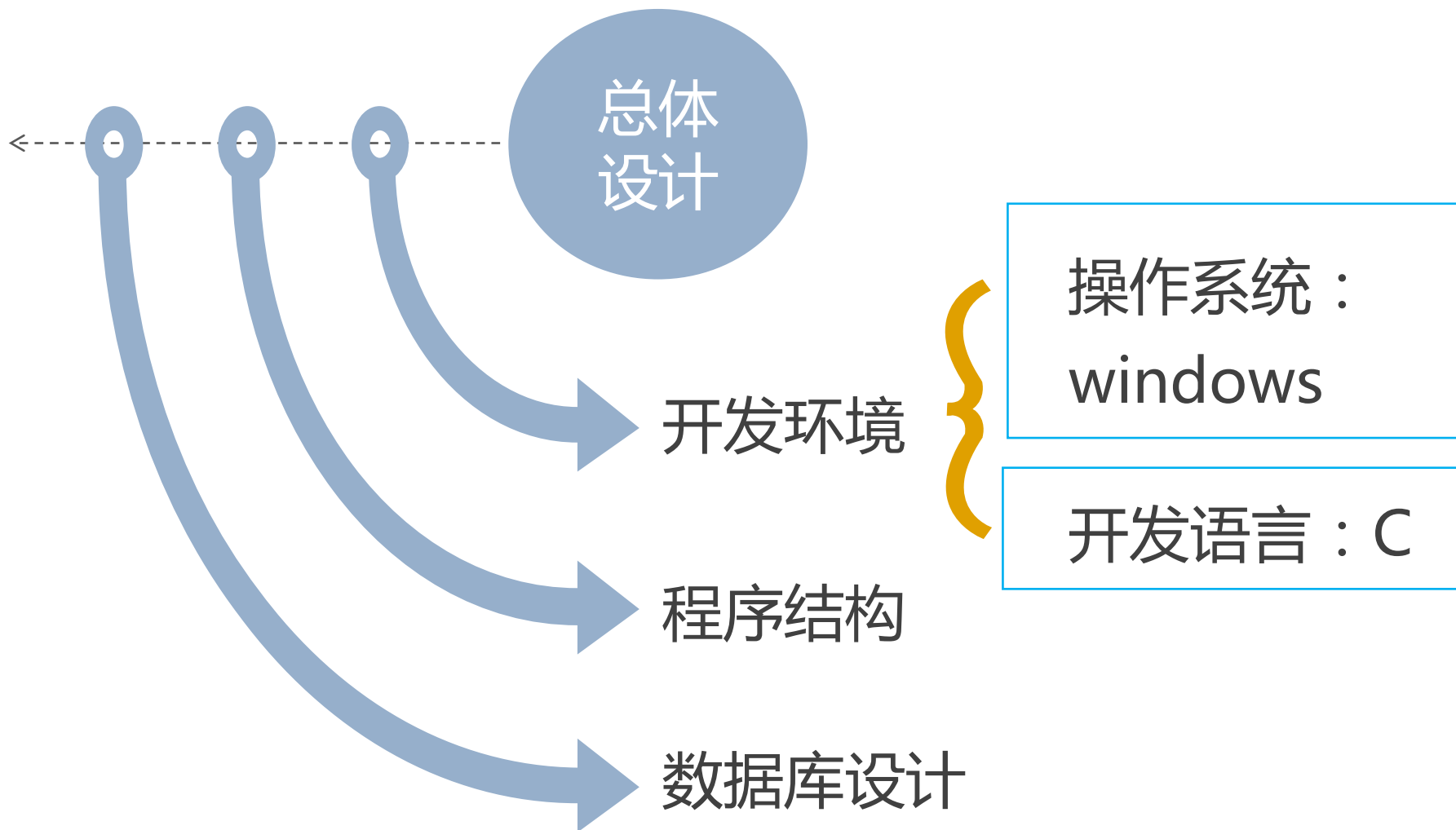
02

总体设计

03

详细设计

总体设计



程序结构

需求分析中的每个需求都必须有一个对应的程序模块。

需求中有一个隐含功能：对保存信息的文件进行初始化

学生管理系统

系统初始化

学生基本信息输入

考试成绩输入

学生信息查询

数据库设计

一个文件一般保存一类数据，用二进制文件。

每个学生的信息包括两个部分。

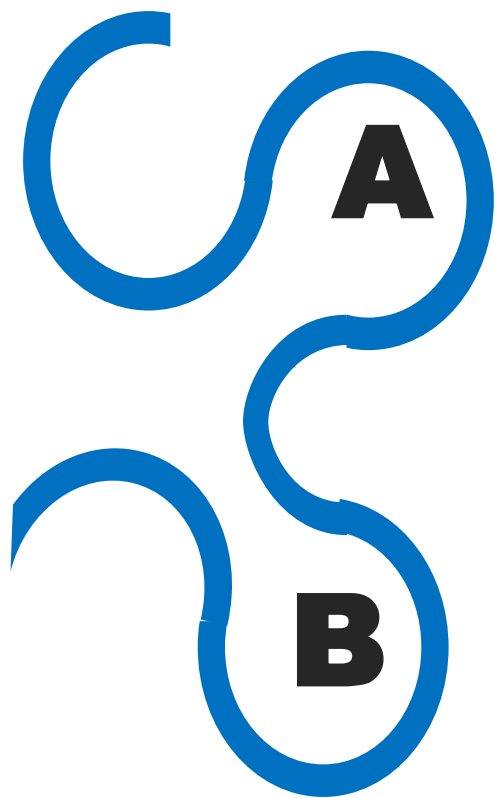
基本信息

每个学生都必须有的信息，包括学号、姓名、性别和班级。

考试成绩信息

考试成绩信息的多少对每个学生而言都不相同。有些学生选修的课程比较多，有些学生选修的课程比较少。学生文件该为每个学生预留多少保存考试成绩信息的空间？

考试成绩存储



把单链表的思想扩展到文件

将每个学生信息分成两部分

考试成绩存储

将每个学生信息分成两部分

- ➡ 基本信息形成一个文件。每个学生的基本信息是文件中的一个记录。
- ➡ 每个学生考试成绩组织成一个单链表，每次考试的成绩是单链表的一个结点，所有单链表的结点可以存储在一个文件中。
- ➡ 基本信息文件中的每个记录保存对应单链表的首节点。



考试成绩存储

student

学号	姓名	性别	班级	单链表第一个结点地址
1	aaa	F	F9089	
2				

score

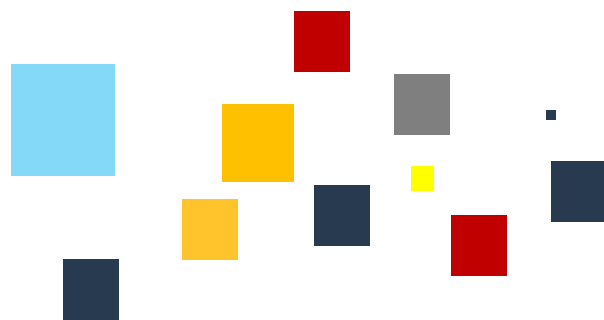
课程名	成绩	Next
		^
		^
		^

一种改进方案：score的一个记录存储若干个成绩

学生基本信息文件

由于学号是自动生成的，每输入一个学生，基本信息文件添加一个记录。所以基本信息文件是按学号的递增次序排列。

查询可以通过随机访问基本信息文件实现。



数据库设计总结

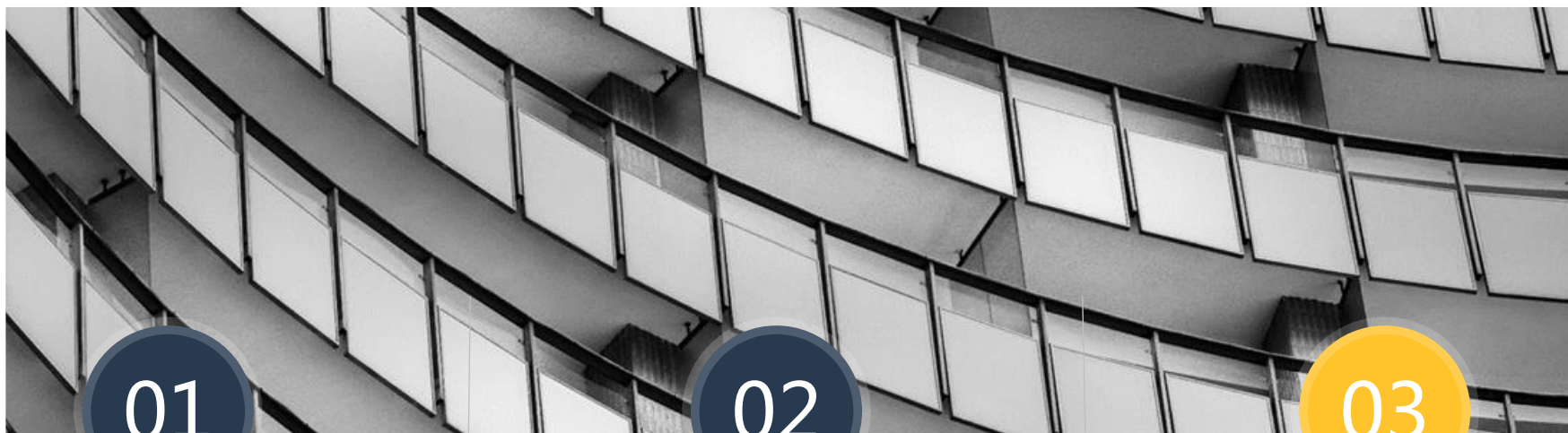
**基本
信息
文件**

每个记录保存一个学生信息以及指向对应单链表首结点的指针。按学号次序排列。

保存成绩的单链表中的结点。
每个记录有3个字段：课程名、
成绩、next指针

**成绩
文件**

学生管理系统



01

需求分析

02

总体设计

03

详细设计

结构体
Student_T

学生基本信
息文件中的
记录类型



结构体
score_T

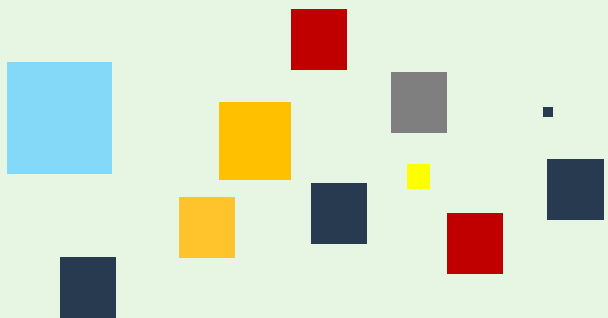
score文件的
记录类
型

总控模块

功能： 显示所有功能菜单，根据用户选择的功能调用相应的函数。

过程：

```
while (1)
{
    显示功能菜单
    输入用户选择
    if (选择 “退出” ) return
    根据选择执行相应函数
}
```



初始化模块

功能： 显示所有功能菜单，根据用户选择的功能调用相应的函数。

过程：

创建文件 student、score，并把他们设置为空文件。

即以out方式打开文件，然后关闭。

基本信息输入模块

以app方式打开文件

获取文件长度last

计算新添加学生的学号

```
new_no = ( last + 1 ) /  
          sizeof(studentT) + 1
```

While(输入没结束) {

 输入学生的姓名、性别、班级

 学号 = new_no

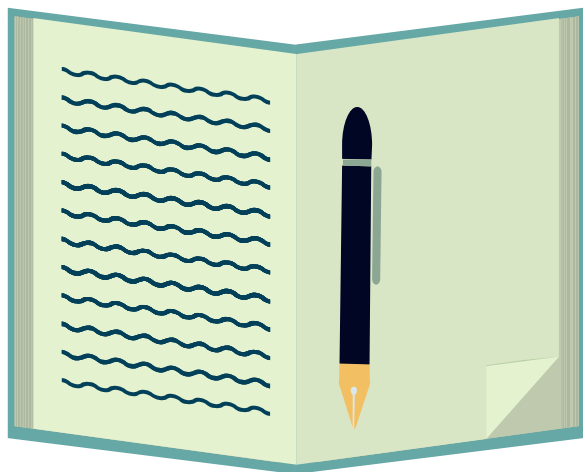
 first = -1

 显示完整的学生信息供用户检查

```
if ( 信息正确 ) {  
    写入文件  
    ++new_no  
    询问是否继续输入  
    if ( 继续输入 ) continue ;  
    else break ;  
}  
}  
关闭文件
```

成绩输入模块

输入课程名到tmp.course
以rb+方式打开文件student
以app方式打开文件score
Sp = score文件的长度



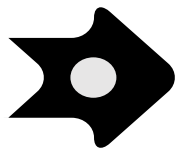
```
While ( 有成绩需要输入 ) {  
    输入学号到currentNo  
    从文件student读入学号为  
        currentNo的学生信息到tmpS  
    输入成绩到tmp.score  
    tmp.next = tmpS.first  
    tmpS.first = sp  
    将tmpS重新写入文件student  
    将tmp写入文件score  
}  
关闭所有文件
```

查询模块

```
if (student文件不存在)
    输出“无学生信息”，函数执行结束
以rb方式打开文件student
以rb方式打开文件score
输入所需查询的学号
```

```
if ( 查询学号 > 最大学号 )
    输出“无此学号”，
    函数执行结束
从文件student读入此学生的基
    本信息，显示在显示器上
next = 单链表第一个结点地址
while ( next != -1 ) {
    从文件score的第next字节
    读入一个记录
    显示课程名及考试成绩
    next = 新读入记录的next
}
```

目录 | CONTENTS

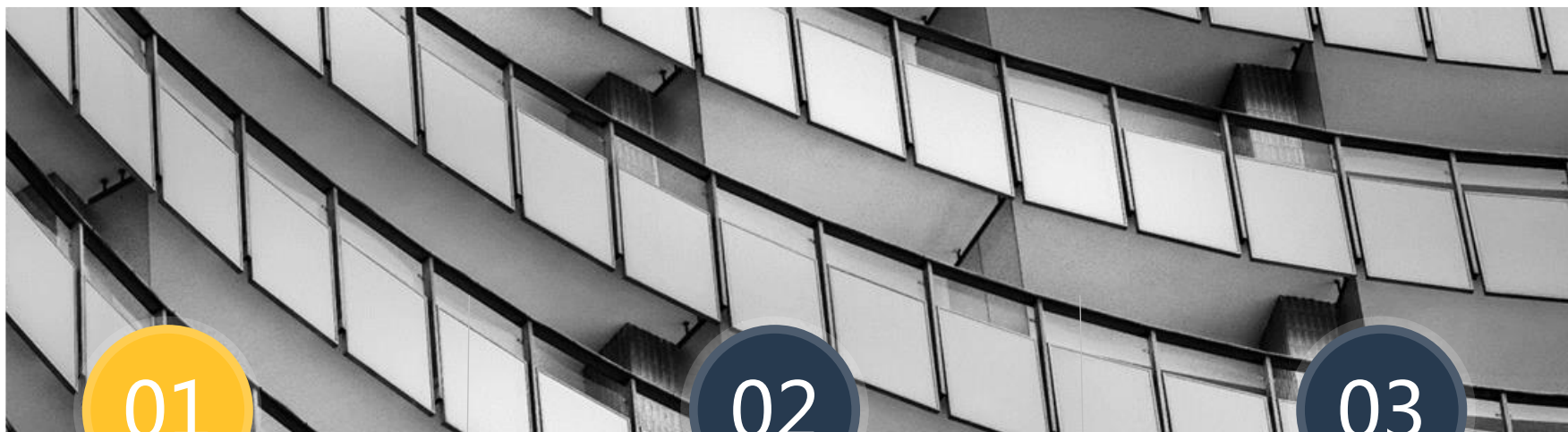


软件开发实例

学生管理系统

■ 网上书店

网上书店



01

需求分析

02

总体设计

03

详细设计

需求分析—与相关人员的交流

老板的需求
01

每个月的
盈利情况

员工的需求
02

查询库存
进货

消费者的需求
03

查询图书
购买图书

需求分析—保存信息

盈利情况：月份、本月收入、本月收入

书目信息

书店角度



**ISBN号、
库存、
销售价格**

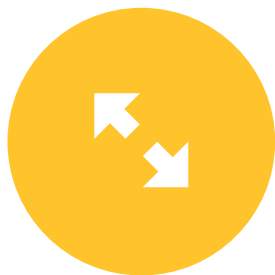
消费者角度



**书名、作者、
ISBN号、
关键字**

需求分析—保存信息

用户信息:



登录号



密码



姓名



角色

需求分析—工作内容及过程

老板：查询某月盈利情况

员工：

根据ISBN号查询
库存

A decorative graphic consisting of several small squares in various colors (blue, yellow, red, grey, dark blue) arranged in a scattered pattern at the bottom of the left box.

进货：输入某本
书进货数量及总
金额，增加库存
量、和本月支出。

需求分析—工作内容及过程

消费者



根据书名、作者、关键字或ISBN号查询某本书。



购书：输入ISBN号及数量，减少库存，增加本月输入。



网上书店



01

需求分析

02

总体设计

03

详细设计

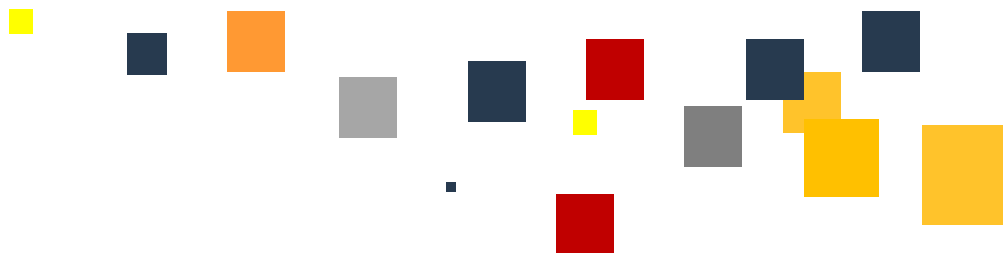
总体设计

➡ 开发环境:



➡ 程序结构

➡ 数据库设计



程序结构

需求分析中的每个需求都必须有一个对应的程序模块。

需求中有四个隐含功能：



A

用户权限管理



B

如何实现按ISBN号查询？
解决方案：对ISBN号进行索引

程序结构



C

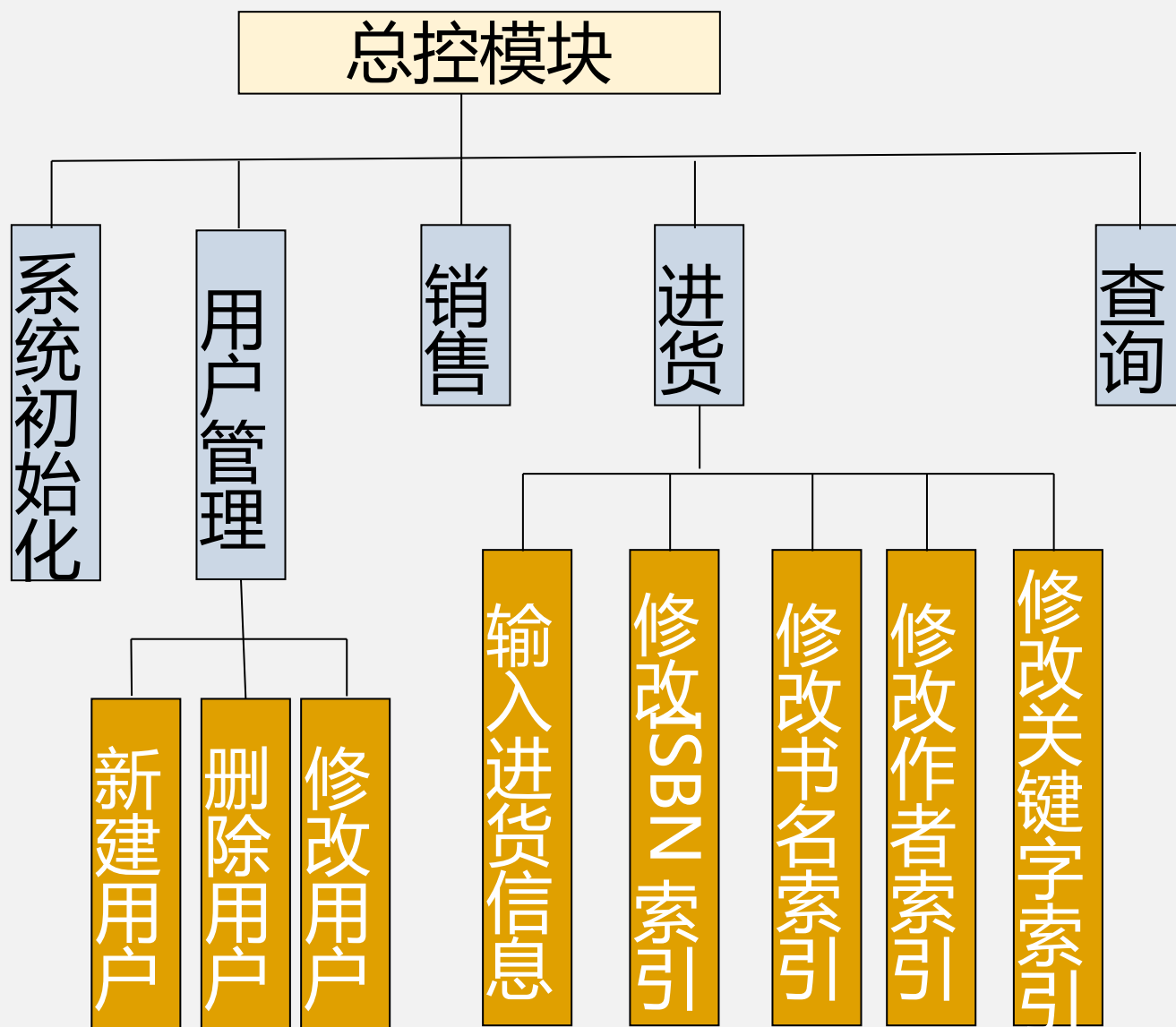
如何实现书名、作者和关键字查询？对应于同一书名、作者、关键字有很多书。解决方案：用倒排文档。



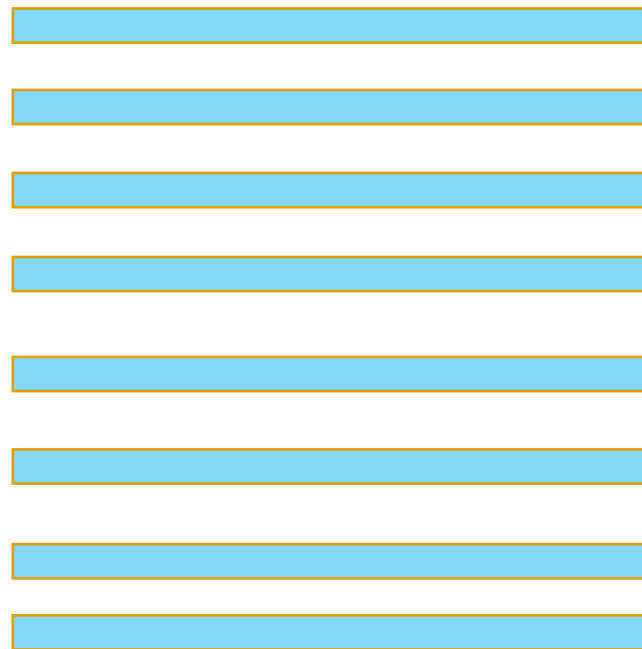
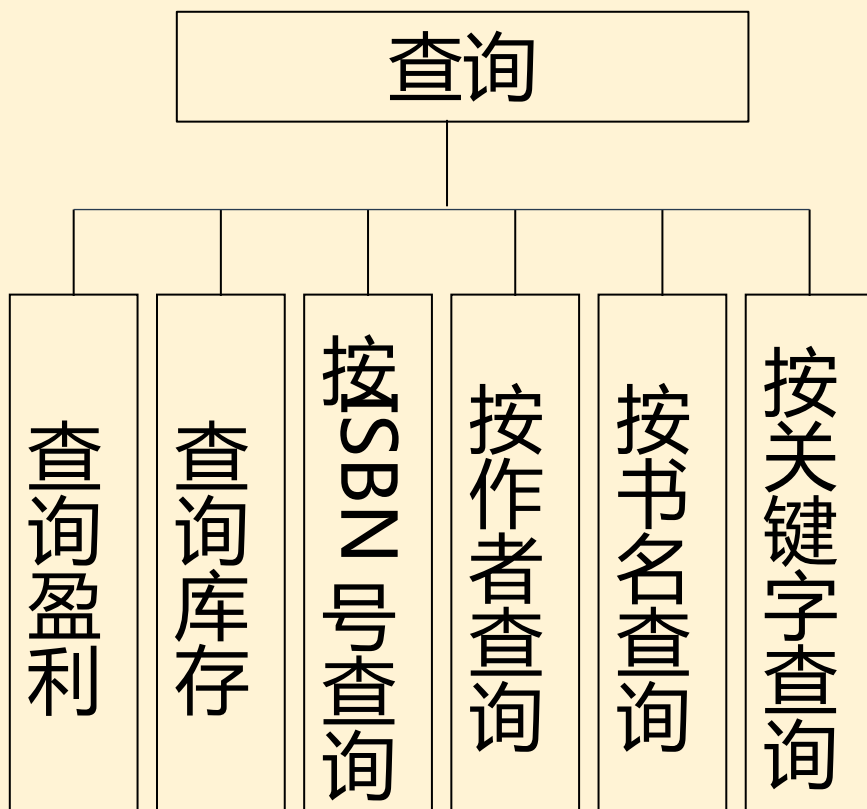
D

对保存信息的文件进行初始化。





可拓展组合查询



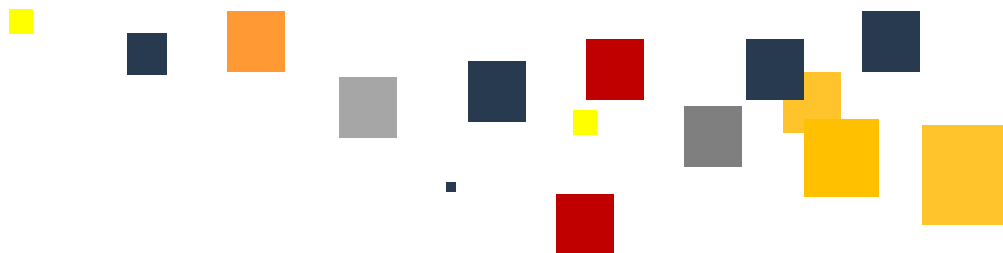
总体设计

➡ 开发环境:



➡ 程序结构

➡ 数据库设计



数据库设计

主要三类信息，每类信息一个文件

用户信息：用户名、密码、真实姓名、角色（系统管理员、老板、普通员工）。系统管理员可以添加或修改用户信息。老板可以查询所有信息。普通员工可以查询除财务以外的所有信息及进货。

图书信息：书名、作者、ISBN号、出版社、关键字、库存量、售价。

财务信息：日期、本月收入、本月支出。



数据库设计

问题：如何实现按书名、作者、关键字查找？

答案：用倒排文档



顺排文档与倒排文档

C++程序设计	张三	C++、程序设计	50
算法与数据结构	李四	算法、数据结构	20
C++实战训练	王五	C++、算法	12
C++入门	张三	C++	21
.....			
顺排文档			

顺排文档与倒排文档

C++	1、 3、 4
程序设计	1
算法	2、 3
数据结构	2

倒排文档

每个查找途径都要有一个倒排文档：书名、作者、关键字、ISBN号。

每个倒排文档都需要对应的索引以加快查找过程。

倒排文档的存储设计

除了ISBN号以外，其余的倒排文档中的每一项都对应了多本书。可以把某一个索引项对应的信息联成一个单链表。如C++对应的单链表是1->3->4



倒排文档的存储设计

问题：查找速度慢。读链表的每个元素都是一次I/O操作。

解决方案：块状链表

倒排文档的索引文件

01

OPTION

倒排文档按索引项的字母序排列。每个倒排文档对应有个索引文件。

02

OPTION

索引文件的组织方式与学生管理系统类似。可以用定长块或非定长块，如姓张作者组成一块。

03

OPTION

在新增一本原来没有的图书时，可能会在倒排文档中插入新的行，这将会造成大量的数据移动。想一想，有什么好办法？



隐含信息

每次添加新的书目后都要修改倒排文档，所以需要记录倒排文档中最后一条书目是哪一条。每次修改倒排文档都从这条信息以后的书目开始。这个信息可以用记录在文件中的位置表示。

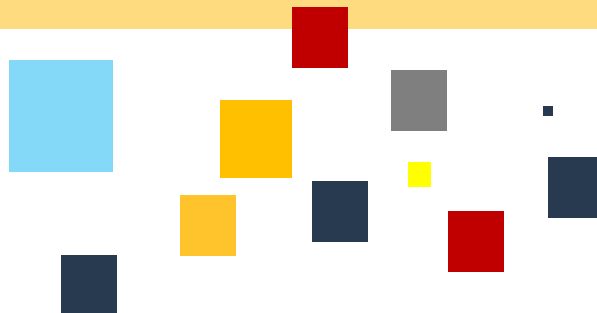
隐含信息

解决方法：

A. 可以专门设一个文件记录四个倒排文档最后的书目信息。

缺点：该文件只有四个整数

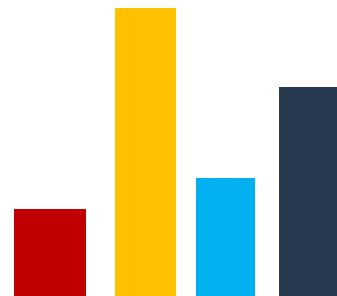
B. 将这个信息分别记录在每个倒排文档的头上。建议用这种方法。



数据库设计总结

共有3个主要的**数据文件**：财务信息文件、用户信息文件、以及书目信息文件。

共有4个**倒排文件及对应的索引文件**：ISBN号、书名、作者、关键字。



网上书店



01

需求分析

02

总体设计

03

详细设计

总控模块

01

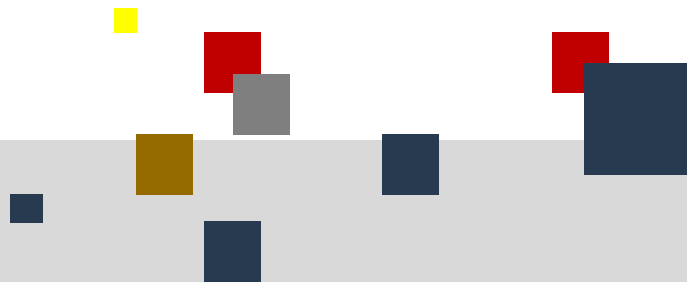
OPTION

将用户文件读入内存（假设用户很少）。

02

OPTION

登录：输入用户码、密码。如果是合法用户，保存用户角色，否则重试。也可以跳过登录过程，默认是消费者。消费者可以查询书目信息及买书。



总控模块

A.显示所有
功能菜单

B.输入用
户选择

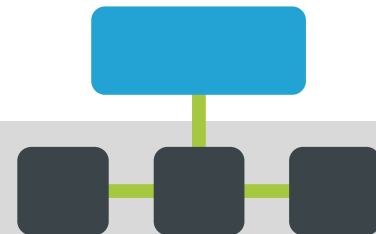
C.检查用户是
否有此权限

D.根据选择调
用相应的函数



初始化模块

- 创建所有文件。
- 在用户文件中添加一个默认的系统管理员账号。
- 在四个倒排文件中添加一个-1，表示上次倒排的最后一一条书目的位置是-1。



用户管理

重复下面工作：



A

显示功能菜单



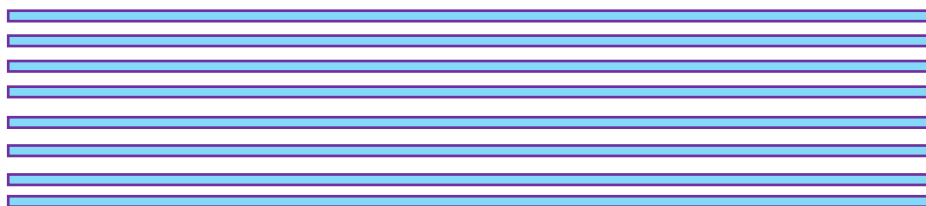
B

接受用户选择



C

根据选择分别新增用户、
修改用户和删除用户内存
中的用户信息。



销售

重复下面工作：

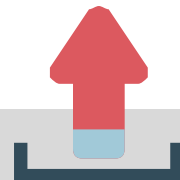
输入ISBN号
和数量

减少对应
书目的库存

获取时间
增加当月的收入

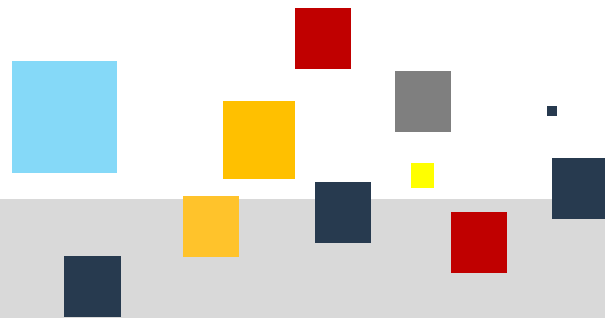
进货

- ➡ 获取当前时间。
- ➡ 输入ISBN号，按ISBN号查询。
- ➡ 如果书已存在，显示书目信息，接着输入数量和总价，修改对应的库存和本月支出
- ➡ 如果书不存在，输入书目信息，接着输入数量和总价，在书目文件中添加信息。
- ➡ 修改本月支出。



修改ISBN索引

- ✦ 从ISBN到排文件头上读入一个long int的数。检查书目文件的长度是否大于这个整数。
- ✦ 如果不大于，退出本函数。
- ✦ 否则从这个位置开始逐条读入书目信息，将对应的ISBN号添加到ISBN号倒排文档。
- ✦ 修改此倒排文件的索引。



查询模块

重复下面工作



显示所有查询
功能



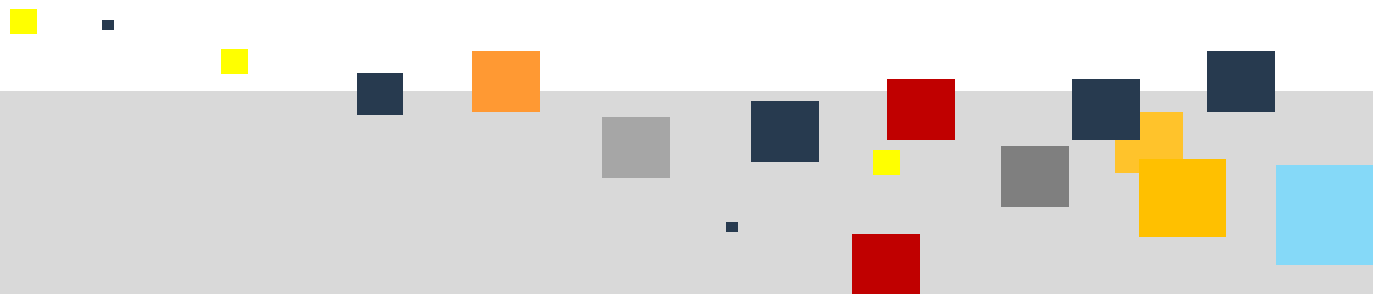
接收用户选择



根据用户选择
调用相应的查
询函数

盈利查询

- ➔ 输入年月。
- ➔ 读入财务文件的第一条记录。
- ➔ 根据第一条记录确定查询年月对应的记录位置。
- ➔ 读取该纪录并显示。



库存查询

- ➡ 输入ISBN号
- ➡ 在ISBN号倒排文件中找到对应的书目位置。在书目文件中读取该书目信息并显示。



作者查询

01

OPTION

输入作者名

02

OPTION

在作者倒排文件中找到对应的条目

03

OPTION

对条目中对应的每一个书目的位置，在书目文件中读取该书目信息并显示



学有所获

