# 数据结构实验报告——实验二

## 学号： 20201060330 姓名： 胡诚皓 得分：\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### 一、实验目的

1. 复习结构体、数组、指针；
2. 掌握数组的静态创建与动态创建；
3. 了解顺序存储的基本访问方法。

### 二、实验内容

1. （必做题）学生成绩信息存储

每个学生的成绩信息包括：学号、语文、数学、英语、总分、加权平均分；采用动态方法创建数组用于存储若干学生的成绩信息；输入学生的学号、语文、数学、英语成绩；计算学生的总分和加权平均分（语文占30%，数学占50%，英语占20%）；输出学生的成绩信息。

1. （必做题）追加和删除学生成绩信息

可以在数组末尾追加新学生的成绩信息；可以根据学号，删除该学生的成绩信息。

1. （选做题）对学生成绩信息进行排序

可以根据学号或总分，升序排序学生的成绩信息。

### 三、数据结构及算法描述

1. 学生成绩信息存储

使用结构体Stu来存储学生的成绩相关信息，其中long long id用于存储学生的学号，double chinese, math, english分别用来存储学生的语文成绩、数学成绩、英语成绩，double total用于存储学生的总分，double average用于存储学生的加权平均分。

为了避免命令行输入时回车键对输入判断的干扰，先读取将要输入的学生的个数，再读取每个学生的信息，在每次读入一个学生的成绩信息后，立刻计算该学生的总分与加权平均分存入。

1. 追加和删除学生成绩信息

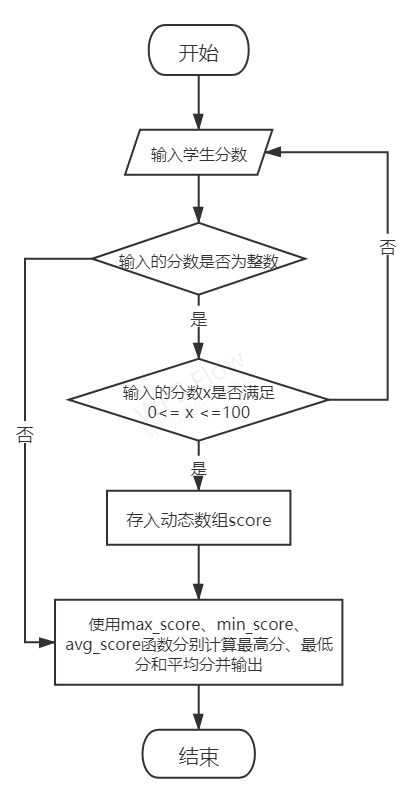
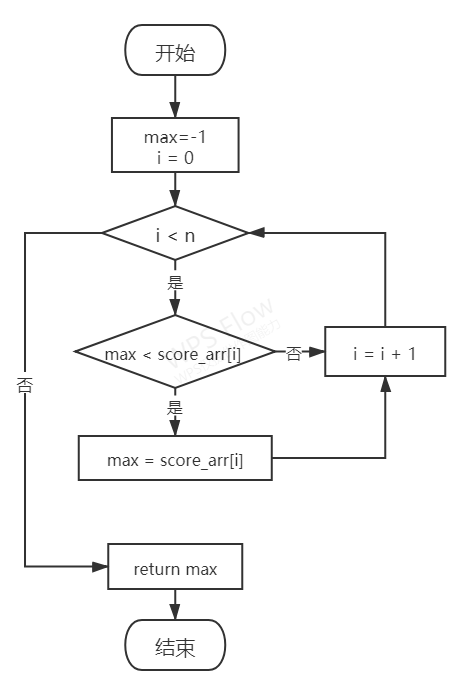
在追加学生的成绩信息时，直接将新追加的学生信息放在动态数组的最后。由于需要先查找到要删除id的学生的位置，先用快速排序对Stu数组进行升序排序，再通过二分查找找到需要删除的学生的位置，然后再进行删除。

1. 对学生成绩信息进行排序

此处根据总分进行升序排序，使用的是优化的快速排序算法。

### 四、详细设计

1. 学生成绩统计

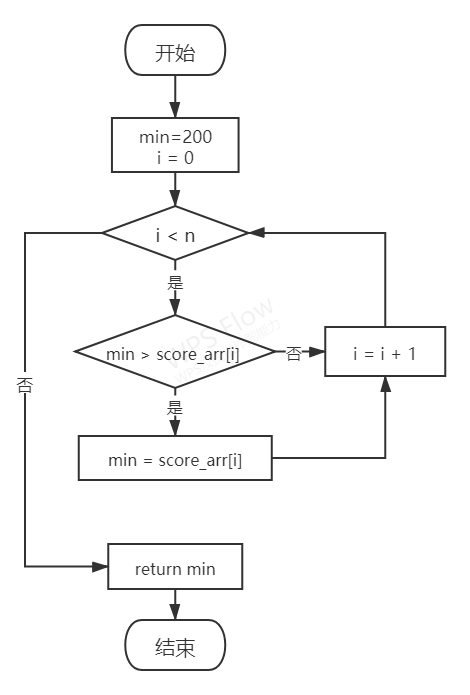
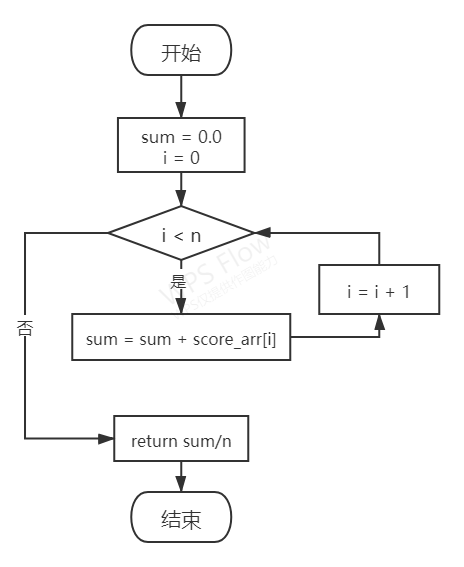
 

题1-1

max\_score流程图

题1-1

主函数流程图

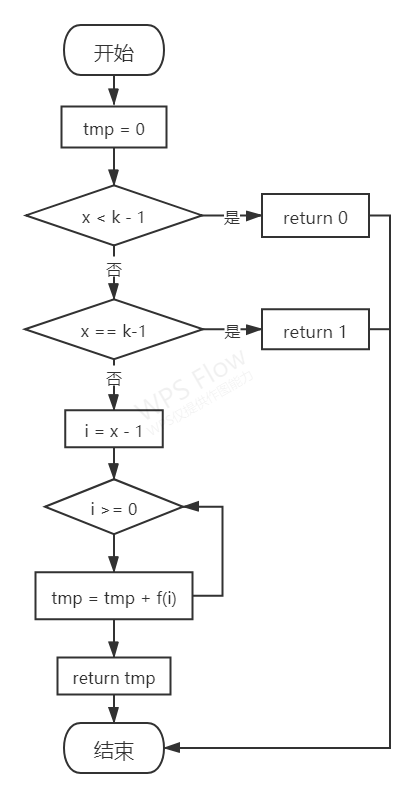
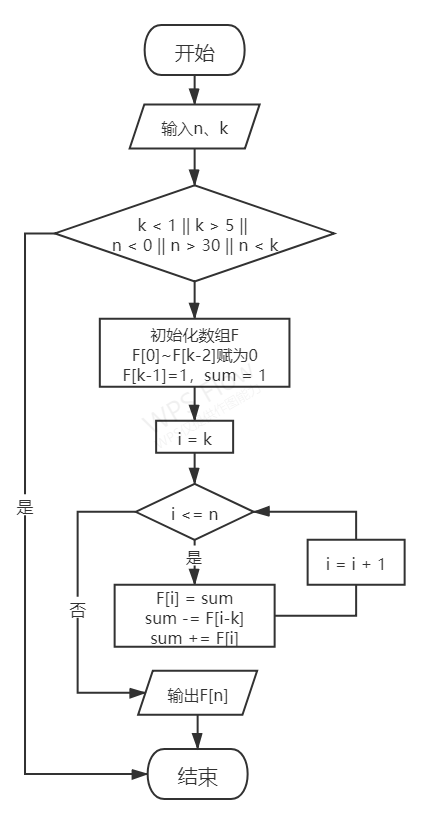
题1-1

min\_score流程图

题1-1

avg\_score流程图

1. 斐波那契数列的第n项

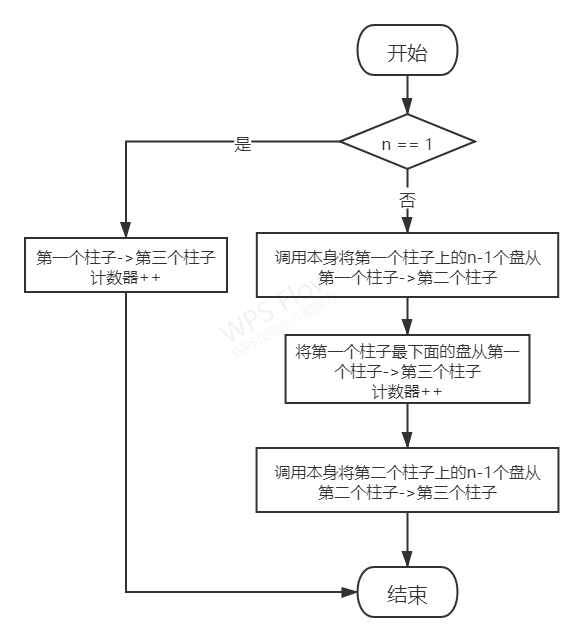
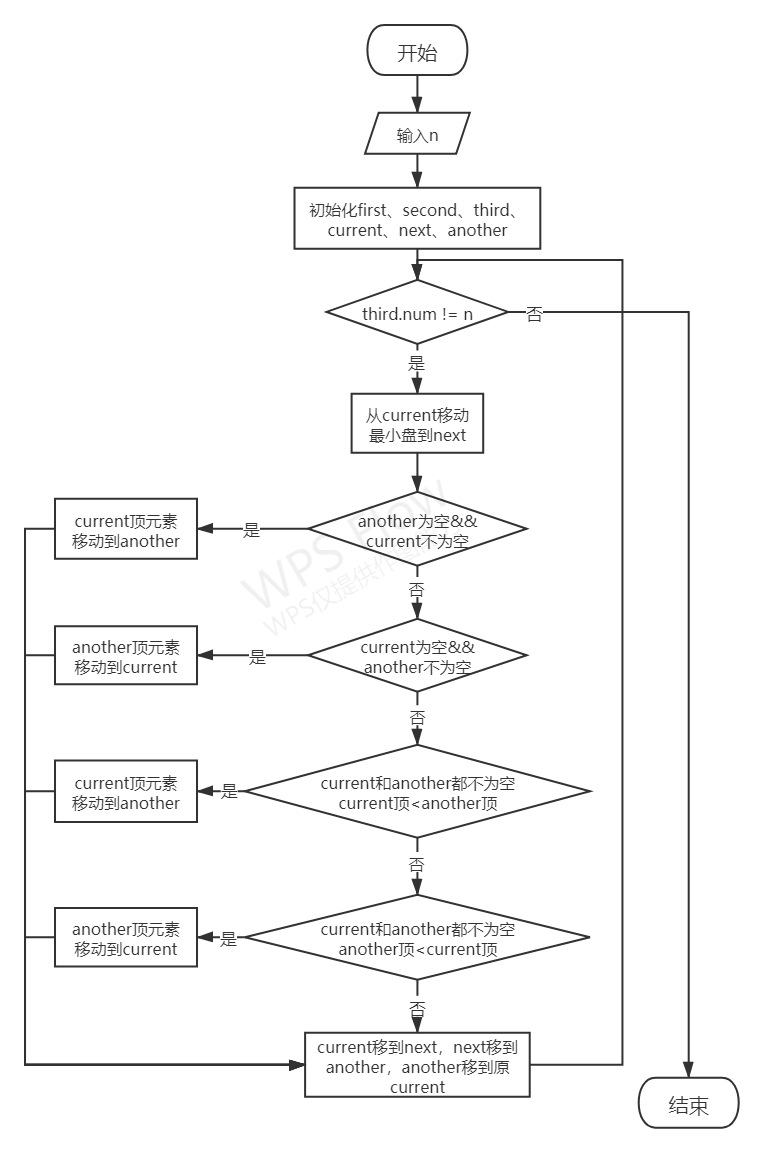
题1-2

循环解法流程图

题1-2

递归解法流程图

1. 汉诺塔问题

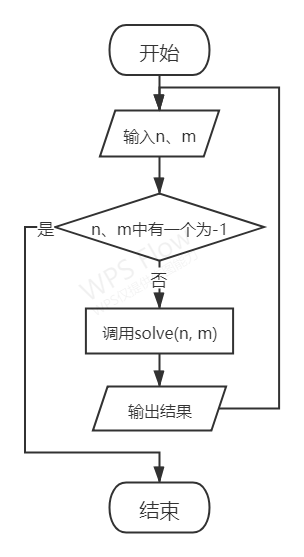
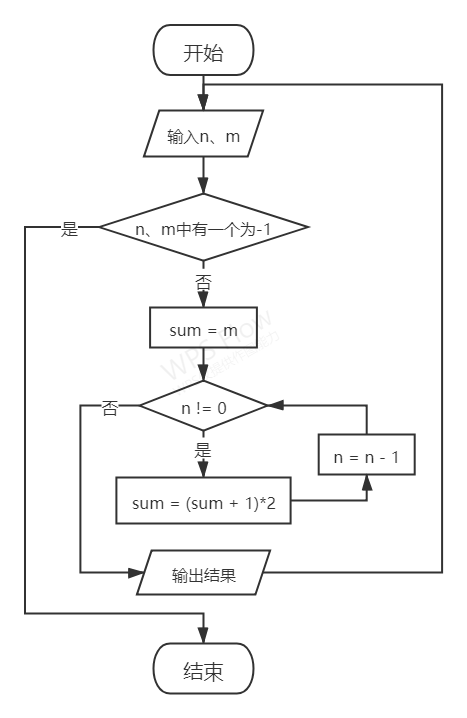
题1-3

循环解法流程图

题1-3

递归解法流程图

1. 猴子吃桃

题1-4

循环解法流程图

题1-4

递归解法流程图

### 五、程序代码

1. 学生成绩统计

（双击图标可以打开文件）

#include <stdio.h>

#include <stdlib.h>

s

typedef struct {

long long id;

double chinese, math, english;

double total, average;

} Stu;

int main() {

Stu \*students;

int count=0;

long long num;

double chi, math, eng;

int n;

students = (Stu \*) malloc(sizeof(Stu));

printf("How many students?\n");

scanf("%d", &n);

while (n--) {

scanf("%lld %lf %lf %lf", &num, &chi, &math, &eng);

(students+count)->id = num;

(students+count)->chinese = chi;

(students+count)->math = math;

(students+count)->english = eng;

(students+count)->total = chi + math + eng;

(students+count)->average = 0.3\*chi + 0.5\*math + 0.2\*eng;

count++;

students = (Stu \*) realloc(students, (count+1)\*sizeof(Stu));

}

printf("-----id-----|--Chinese--|--Math--|--English--|--Total--|--Avg--|\n");

for (int i = 0; i < count; i++) {

printf("%12lld|%11.2f|%8.2f|%11.2f|%9.2f|%7.2f|\n", students[i].id, students[i].chinese, students[i].math, students[i].english, students[i].total, students[i].average);

}

free(students);

return 0;

}

1. 斐波那契数列的第n项
2. 递归解法

（双击图标可以打开文件）

#include <stdio.h>

#include <stdlib.h>

int n, k;

int f(int);

int main() {

printf("input k: ");

scanf("%d", &k);

//对读入数据的有效性进行判断

if (k < 1 || k > 5) {

printf("error input k");

return 1;

}

printf("input n: ");

scanf("%d", &n);

if (n < 0 || n > 30) {

printf("error input n");

return 1;

}

printf("fn=%d", f(n));

return 0;

}

//递归计算fn的值，x <= k-1时为递归的边界

int f(int x) {

int tmp=0;

if (x < k-1)

return 0;

else if (x == k-1)

return 1;

else {

for (int i = x-1; i >= x-k; i--) {

tmp += f(i);

}

return tmp;

}

}

1. 循环解法

（双击图标可以打开文件）

#include <stdio.h>

#include <stdlib.h>

int n, k;

int main() {

int \*F;

int sum;

printf("input k: ");

scanf("%d", &k);

if (k < 1 || k > 5) {

printf("error input k");

return 1;

}

printf("input n: ");

scanf("%d", &n);

if (n < 0 || n > 30 || n < k) {

printf("error input n");

return 1;

}

//使用动态数组，不浪费空间

F = (int \*) malloc((n+1)\*sizeof(int));

//初始化条件

for (int i = 0; i < k-1; i++)

F[i] = 0;

F[k-1] = 1;

sum = 1;

//使用滑动窗口思想大幅减少时间复杂度

for (int i = k; i <= n; i++) {

F[i] = sum;

sum -= F[i-k];

sum += F[i];

}

printf("fn=%d", F[n]);

return 0;

}

1. 汉诺塔问题
2. 递归解法

（双击图标可以打开文件）

#include <stdio.h>

#include <stdlib.h>

void hanoi(int, char, char, char, int\*);

int main() {

int num=0, n;

printf("input n: ");

scanf("%d", &n);

hanoi(n, 'A', 'B', 'C', &num);

return 0;

}

//按照规则将n个盘借助形参b柱从形参a柱移至形参c柱

void hanoi(int n, char a, char b, char c, int\* times) {

if (n == 1) {

printf("%c -> %c\n", a, c);

(\*times)++;

} else {

//将形参a柱上的n-1个盘借助形参c移至形参b柱

hanoi(n-1, a, c, b, times);

printf("%c -> %c\n", a, c);

(\*times)++;

hanoi(n-1, b, a, c, times);

}

}

1. 循环解法

（双击图标可以打开文件）

#include <stdio.h>

#include <stdlib.h>

//使用栈存储每个柱子

typedef struct {

char name;

int \*arr;

int num;

} STK;

//使用一个字符作为标签初始化一个柱子

void STK\_init(STK \*, char);

void STK\_pop(STK \*);

int STK\_push(STK \*, int);

int main() {

int num = 0, n;

//绝对地存储三个柱子

STK first, second, third;

//current指向当前处理的柱子，next指向下一个要处理的柱子

//another指向作为辅助的其他柱子

STK \*current, \*next, \*another, \*tmp;

//初始化三个柱子

STK\_init(&first, 'A');

STK\_init(&second, 'B');

STK\_init(&third, 'C');

printf("input n: ");

scanf("%d", &n);

//奇数个盘时左移，即当前柱子为1号，下一个柱子为3号

//偶数个盘时右移，即当前柱子为1号，下一个柱子为2号

if (n % 2 == 1) {

current = &first;

next = &third;

another = &second;

} else {

current = &first;

next = &second;

another = &third;

}

//将盘放入1号柱子初始化，数字越大代表越大的盘，即1号盘是最小的

for (int i = n; i >= 1; i--) {

STK\_push(current, i);

}

//由于所有步骤都是在满足题目要求的情况下进行移动的

//则当3号柱有n个盘时就完成了汉诺塔游戏

while (third.num != n) {

//先将最小的盘移到下一个柱子

STK\_push(next, 1);

STK\_pop(current);

printf("%c -> %c\n", current->name, next->name);

//下一个柱子上已经有了最小盘，不再做移动

//在辅助柱子和当前柱子之间进行一次移动，分为4种情况

//情况1：辅助柱子为空，将当前柱子有盘

//情况2：当前柱子为空，辅助柱子有盘

//情况3：当前柱子和辅助柱子都有盘，但是当前柱顶的盘比辅助柱顶的盘小

//情况4：当前柱子和辅助柱子都有盘，但是辅助柱顶的盘比当前柱顶的盘小

if (another->num == 0 && current->num != 0) {

STK\_push(another, \*(current->arr + current->num - 1));

STK\_pop(current);

printf("%c -> %c\n", current->name, another->name);

} else if (current->num == 0 && another->num != 0){

STK\_push(current, \*(another->arr + another->num - 1));

STK\_pop(another);

printf("%c -> %c\n", another->name, current->name);

} else if (current->num != 0 && another->num != 0 && \*(current->arr + current->num - 1) < \*(another->arr + another->num - 1)) {

STK\_push(another, \*(current->arr + current->num - 1));

STK\_pop(current);

printf("%c -> %c\n", current->name, another->name);

} else if (current->num != 0 && another->num != 0 && \*(another->arr + another->num - 1) < \*(current->arr + current->num - 1)) {

STK\_push(current, \*(another->arr + another->num - 1));

STK\_pop(another);

printf("%c -> %c\n", another->name, current->name);

}

//循环交替进行

tmp = current;

current = next;

next = another;

another = tmp;

}

return 0;

}

void STK\_pop(STK \*stack) {

(stack->num)--;

stack->arr = (int \*) realloc(stack->arr, sizeof(int) \* (stack->num));

}

void STK\_init(STK \*stack, char name) {

stack->name = name;

stack->num = 0;

stack->arr = NULL;

}

int STK\_push(STK \*stack, int x) {

(stack->num)++;

if (stack->num == 1) {

stack->arr = (int \*) malloc(sizeof(int));

\*(stack->arr) = x;

} else {

stack->arr = (int \*) realloc(stack->arr, sizeof(int)\*(stack->num+1));

if (stack->arr == NULL)

return -1;

\*(stack->arr + stack->num - 1) = x;

}

return 1;

}

1. 猴子吃桃
2. 递归解法

（双击图标可以打开文件）

#include <stdio.h>

#include <stdlib.h>

int solve(int, int);

int main() {

int m, n;

while (1) {

//若输入的n和m中有一个为-1就退出程序

printf("input n: \n");

scanf("%d", &n);

if (n == -1) { return 0; }

printf("input m: \n");

scanf("%d", &m);

if (m == -1) { return 0; }

printf("origin peach: %d\n", solve(n, m));

}

return 0;

}

//返回第days天剩余peaches个桃时，一开始桃的个数

int solve(int days, int peaches) {

if (days == 0)

return peaches;

else

return solve(days-1, (peaches+1)\*2);

}

1. 循环解法

（双击图标可以打开文件）

#include <stdio.h>

#include <stdlib.h>

int main() {

int m, n, sum;

while (1) {

printf("input n: \n");

scanf("%d", &n);

if (n == -1) { return 0; }

printf("input m: \n");

scanf("%d", &m);

if (m == -1) { return 0; }

//初始化桃子数量

sum = m;

//通过倒推的方法不断将桃子数量+1再乘2，倒推n次即可

while (n--) {

sum = (sum + 1)\*2;

}

printf("origin peach: %d\n", sum);

}

return 0;

}

### 六、测试和结果

1. 学生成绩统计

**Input:**

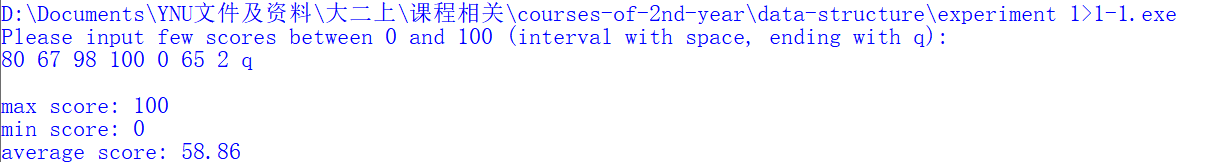
80 67 98 100 0 65 2 q

**Output:**

max score: 100

min score: 0

average score: 58.86



1. 斐波那契数列的第n项
2. 递归解法

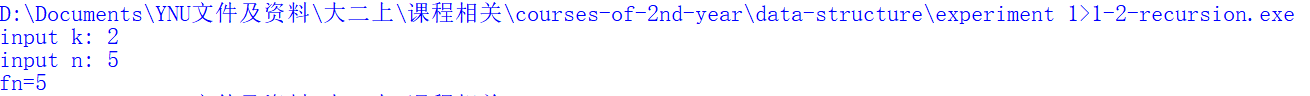
**Input:**

input k: 2

input n: 5

**Output:**

fn=5



1. 循环解法

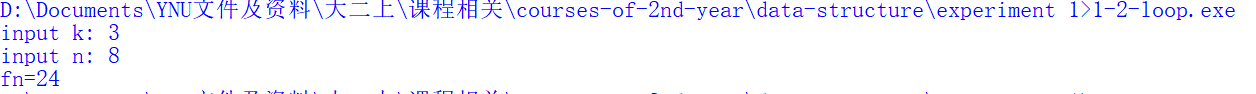
**Input:**

input k: 3

input n: 8

**Output:**

fn=24



1. 汉诺塔问题
2. 递归解法

**Input:**

input n: 4

**Ouput:**

A -> B

A -> C

B -> C

A -> B

C -> A

C -> B

A -> B

A -> C

B -> C

B -> A

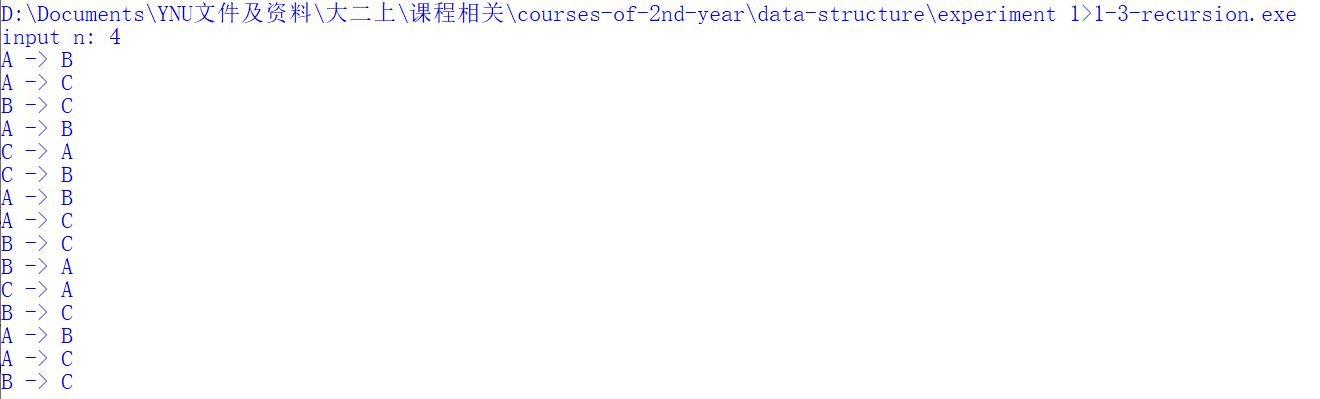
C -> A

B -> C

A -> B

A -> C

B -> C



1. 循环解法

**Input:**

input n: 3

**Output:**

A -> C

A -> B

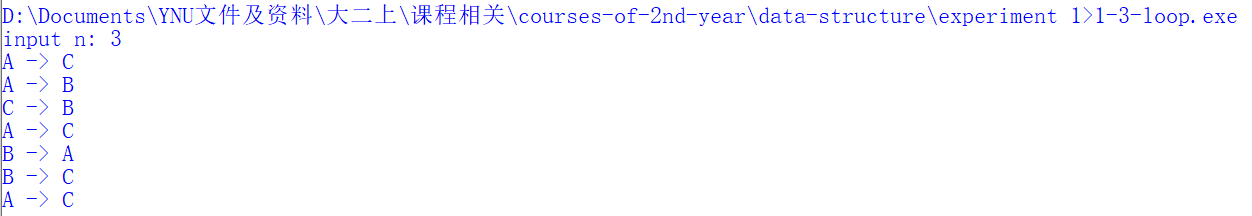
C -> B

A -> C

B -> A

B -> C

A -> C



1. 猴子吃桃
2. 递归解法

**Input:**

input n:

3

input m:

2

input n:

4

input m:

1

input n:

0

input m:

2

input n:

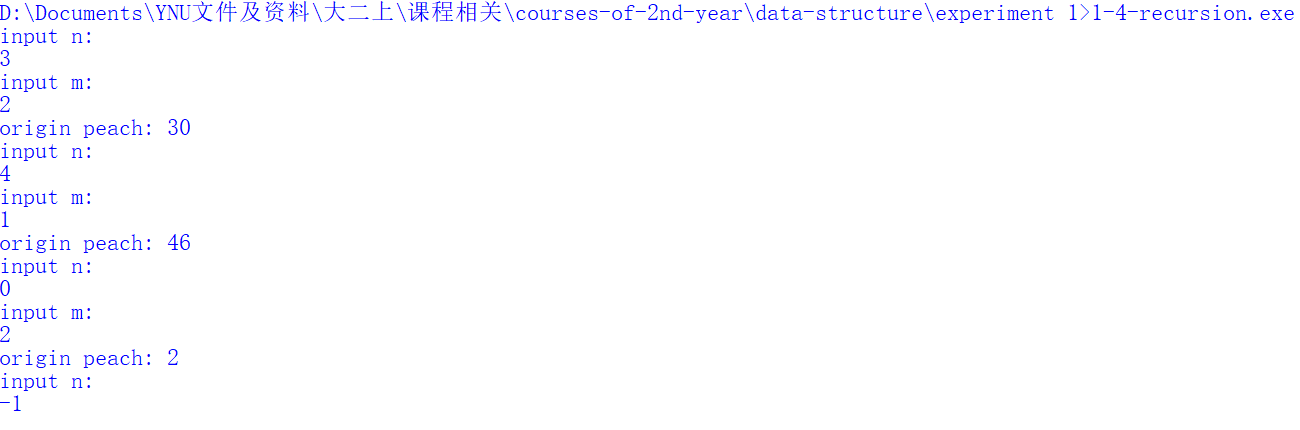
-1

**Output:**

origin peach: 30

origin peach: 46

origin peach: 2



1. 循环解法

**Input:**

input n:

3

input m:

1

input n:

2

input m:

2

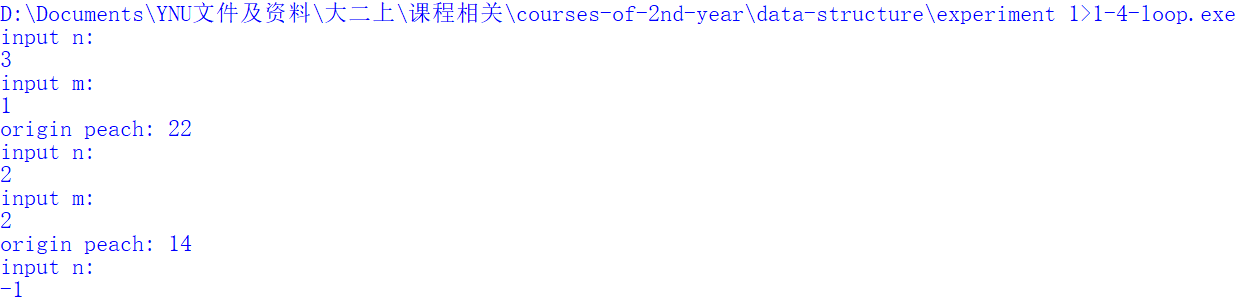
input n:

-1

**Output:**

origin peach: 22

origin peach: 14



### 用户手册

1. 学生成绩统计

0到100之间的整数被认为是有效的分数值，各个分数之间使用空格分隔，以q作为输入的结尾；也可以在输入一些数据后回车，再继续输入一些数据，只要最后以q结尾即可。输入的不在0到100之间的整数将会被自动忽略。

1. 斐波那契数列的第n项

输入的n与k需要满足1<=k<=5、0<=n<=30且n >= k，否则程序会在给出错误提示后退出。另外，需要注意fn中的n是从0开始的。

1. 汉诺塔问题

输入的n必须为正整数。

1. 猴子吃桃

输入的n、m必须均为正整数，由于sum使用的是int类型，经粗略估计，n不应30且m不应大于99。