# 数据结构实验报告——实验一

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

### 一、实验目的

1. 复习变量、数据类型、语句、函数；
2. 掌握函数的参数和值；
3. 了解递归。

### 二、实验内容

1. （必做题）采用函数统计学生成绩

输入学生的成绩，计算并输出这些学生的最低分、最高分、平均分。

1. （必做题）采用递归和非递归方法计算k阶裴波那契序列的第n项的值

序列定义如下：

f0=0，f1=0，…，fk-2=0，fk-1=1，fn= fn-1+fn-2+…+fn-k (n>=k)

要求：输入k(1<=k<=5)和n(0<=n<=30)，输出fn

1. （二选一）采用递归和非递归方法求解汉诺塔问题

问题描述如下：

有三根柱子A、B、C，在柱子A上从下向上有n个从大到小的圆盘，在柱子B和C上没有圆盘，现需将柱子A上的所有圆盘移到柱子C上，可以借助柱子B，要求每次只能移动一个圆盘，每根柱子上的圆盘只能大的在下，小的在上。

要求：输入n，输出移动步骤。

1. （选做题）采用递归和非递归方法求解

问题描述如下：

有一群猴子摘了些桃子放在仓库，每天吃掉所有桃子的一半还要多吃一颗。第n天时，还剩下m颗桃子，问一开始放入仓库的桃子几颗？

要求：可循环输入n，m求解，直到输入-1时退出程序，输出一开始的桃子总数sum。

### 三、数据结构及算法描述

1. 学生成绩统计

使用动态数组score来存储每个学生的成绩，count来记录学生的数量，0到100之间的分数值被认为是有效的，不在此区间内的分数值将会被忽略。以q结尾表示输入的结束。max\_score、min\_score、avg\_score函数分别用于计算最高分、最低分和平均分。

1. 斐波那契数列的第n项

对于k阶斐波那契数列，第k项（下标为k-1）为1，之前都为0。后面的某一项等于它的前k项之和。在递归解法中，边界条件为Fk-1=1，F0~Fk-2=0

在循环解法中，从第k+1项（即Fk）开始一项一项地往后计算。在循环中可以使用滑动窗口算法大大降低算法的时间复杂度。由于题目中给出了n与k的取值范围，并且满足n>=k，在获取输入后需要进行简单的验证。

1. 汉诺塔问题

解决汉诺塔问题，相较于非递归方法，递归方法更加便于理解且易于代码编写。递归边界条件为只有一个盘时，直接移动至目标柱子上即可。其他情况努力转换为除了最小的盘外都有序放在目标柱子上，最后只要把最小的盘直接移动至目标柱子上即可。其中hanoi(int n, char a, char b, char c, int\* times)表示按照汉诺塔的规则，将n个盘借助b柱子从a柱子移动至c柱子，times用来存储总共移动的次数，边界条件为n=1。以把n个盘从A移动至C为例，从总体上看，需要做的分为三步：①把A上方的n-1个盘借助C移动至B；②把A上剩下的一个盘（最大的）直接放到C；③把B上的n-1个盘借助A移动至C，由于C上只有一个最大的盘，所有其他的盘都能移到C上，相当于C上是空的。做完这三步就完成了整个汉诺塔游戏。

非递归的循环迭代法思考起来较为困难，为了达到最佳方案，需要找到其中的固定规律。经过观察，得出以下几个规律：①完成游戏的方案有多个，但是移动次数最少的最佳方案只有一种，且移动的步骤是唯一的；②对于奇数个的盘子，需要不断向左循环移动；对于偶数个的盘子，需要不断向右循环移动；③只要不断做两个动作就可以达到最佳方案，首先将最小的盘移到下一个柱子，其次将没放最小的盘的另两个柱子之间进行一次移动（能怎么移就怎么移）。每个柱子都是后进先出的，每个柱子都作为一个栈进行存储，STK.name存储该栈的标签（即为柱子编号），STK.arr指向栈底，STK.num记录栈中的元素数量。current、next、another三个STK的指针用来交替循环指向当前处理、下一个处理、辅助的柱子。first、second、third则作为柱子的绝对位置存储。其他细节见代码注释。

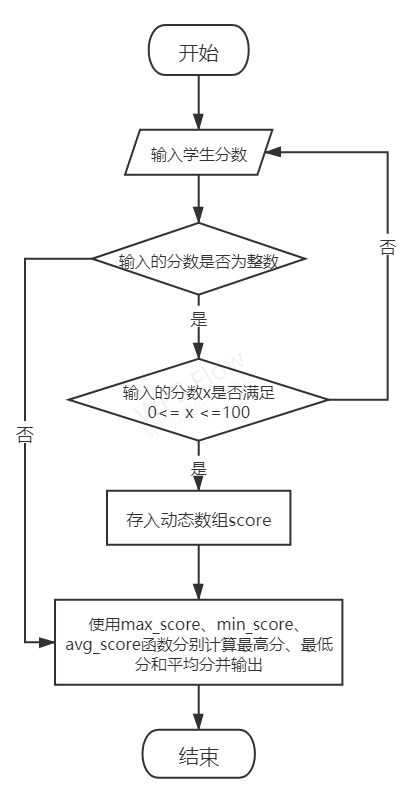
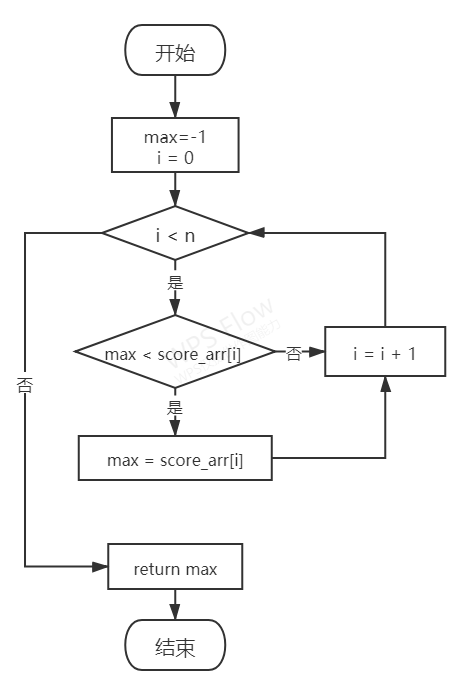
1. 猴子吃桃

在递归解法中，int solve(int days, int peaches)函数作为递归函数，用于解决“第n天时剩下peaches个桃子，求一开始桃子的数量”的问题，边界条件为days == 0，可以直接返回。“第n天时剩下peaches个桃子，求一开始桃子的数量”的问题可以转移为“第n-1天时剩下(peaches+1)\*2个桃子，求一开始桃子的数量”。

在循环解法中，先初始化桃子数量为m，将桃子数量不断+1再乘2。由于求解的是第n天，操作n次即可得到一开始的桃子数量

### 四、详细设计

1. 学生成绩统计

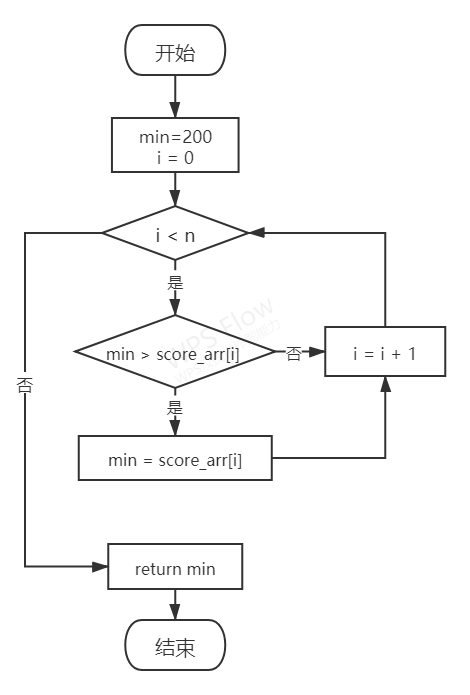
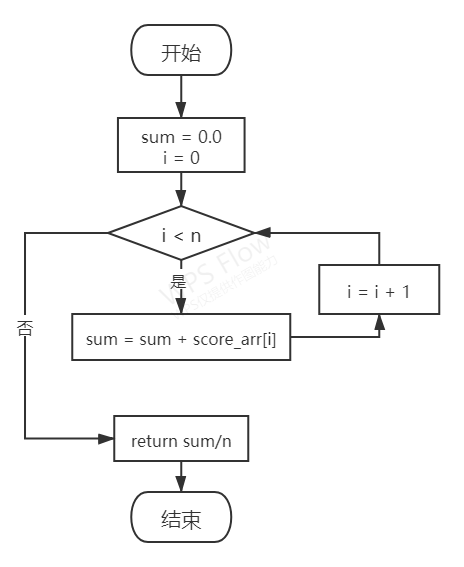
 

题1-1

max\_score流程图

题1-1

主函数流程图

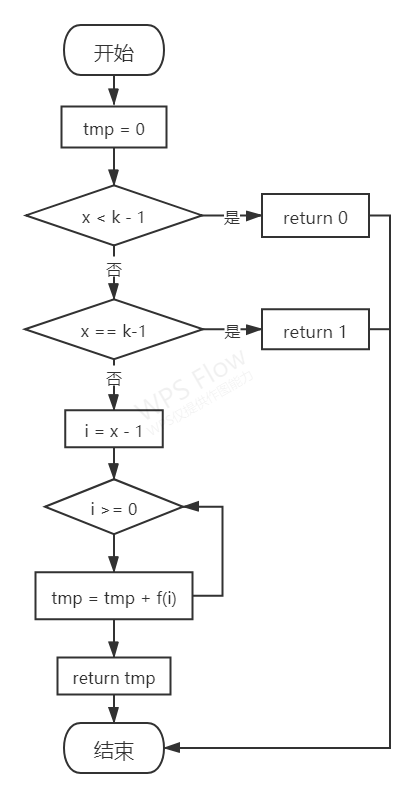
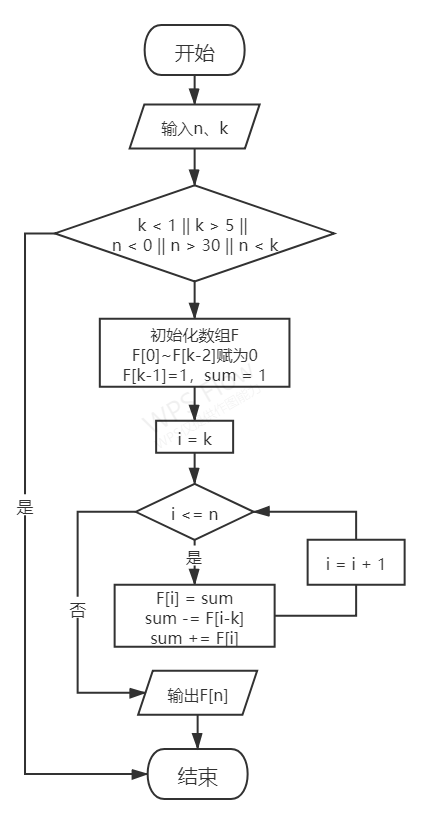
题1-1

min\_score流程图

题1-1

avg\_score流程图

1. 斐波那契数列的第n项

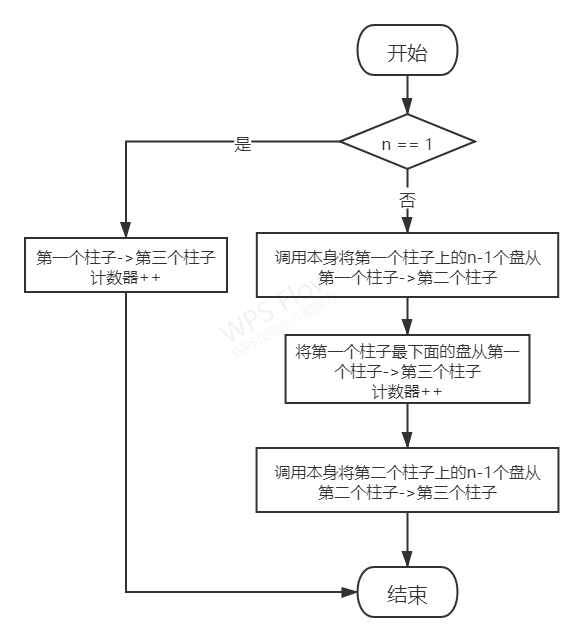
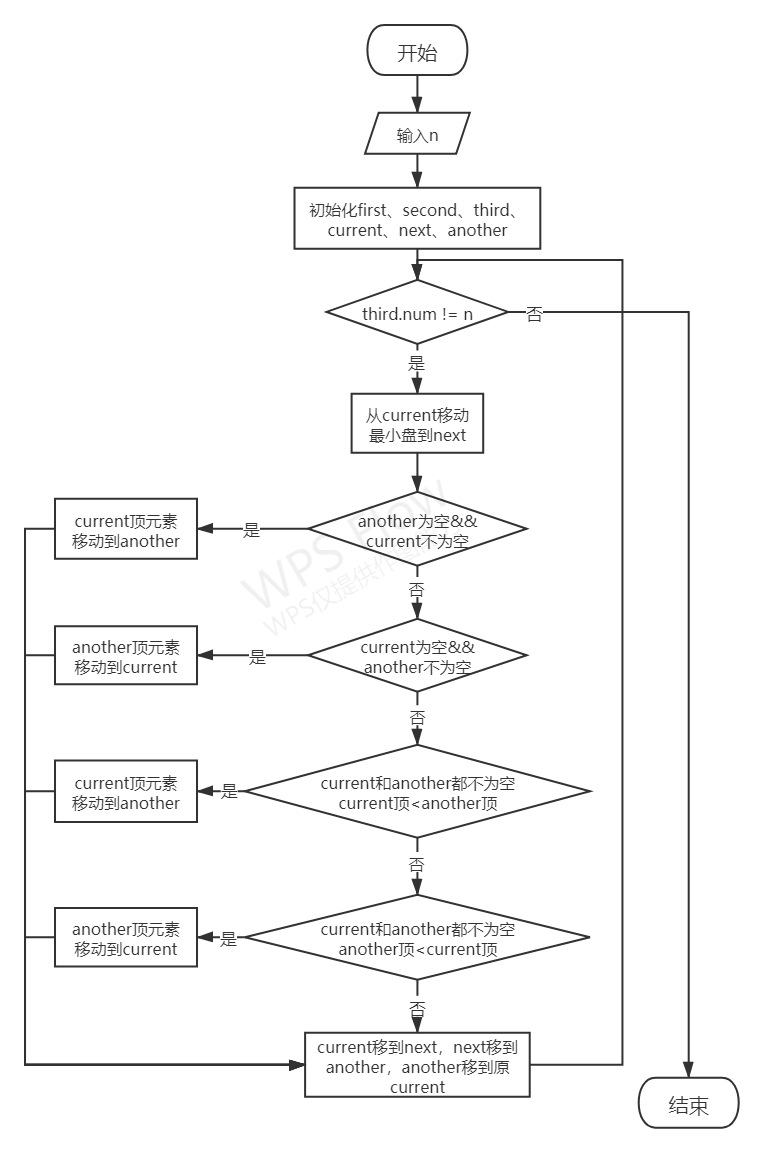
题1-2

循环解法流程图

题1-2

递归解法流程图

1. 汉诺塔问题

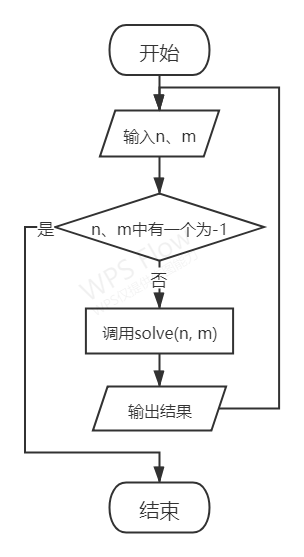
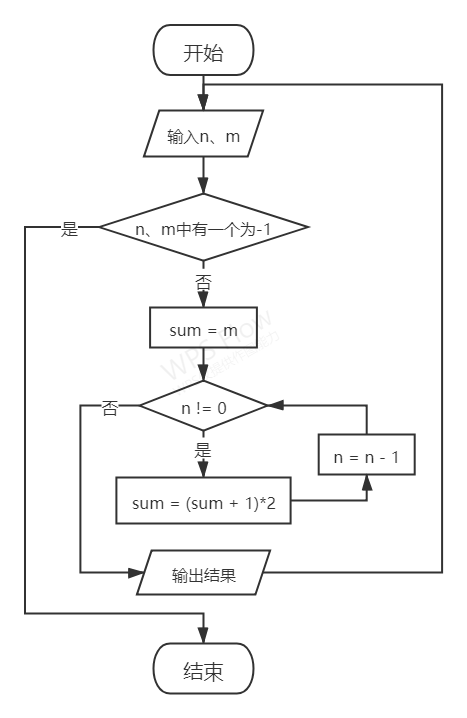
题1-3

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>

int max\_score(int\*, int);

int min\_score(int\*, int);

double avg\_score(int\*, int);

int main() {

//使用动态数组来存储分数，count记录数组中存储的分数的个数

int \*score, count=0;

score = (int \*)malloc(sizeof(int));

printf("Please input few scores between 0 and 100 (interval with space, ending with q):\n");

while (scanf("%d", score+count)) {

//忽略不在0到100之间的分数

if (\*(score+count) < 0 || \*(score+count) > 100)

continue;

else {

count++;

score = (int \*) realloc(score, count\*sizeof(int));

if (score == NULL)

return 1;

}

}

printf("\nmax score: %d", max\_score(score, count));

printf("\nmin score: %d", min\_score(score, count));

printf("\naverage score: %.2f", avg\_score(score, count));

return 0;

}

//在长为n的动态数组score\_arr中寻找最高分数

int max\_score(int\* score\_arr, int n) {

int max=-1;

for (int i = 0; i < n; i++)

max = score\_arr[i] > max ? score\_arr[i] : max;

return max;

}

//在长为n的动态数组score\_arr中寻找最低分数

int min\_score(int\* score\_arr, int n) {

int min=200;

for (int i = 0; i < n; i++)

min = score\_arr[i] < min ? score\_arr[i] : min;

return min;

}

////求取长为n的动态数组score\_arr中分数的平均值

double avg\_score(int\* score\_arr, int n) {

double sum=0.0;

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

sum += score\_arr[i];

}

return sum/n;

}

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:**

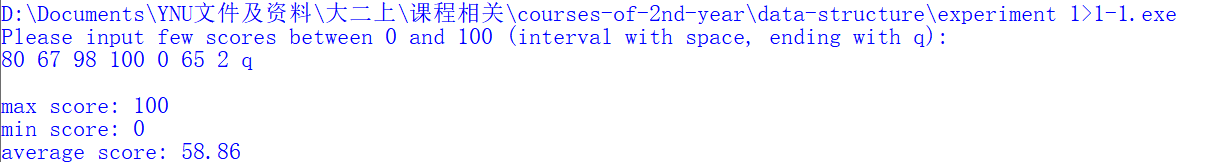
80 67 98 100 0 65 2 q

**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

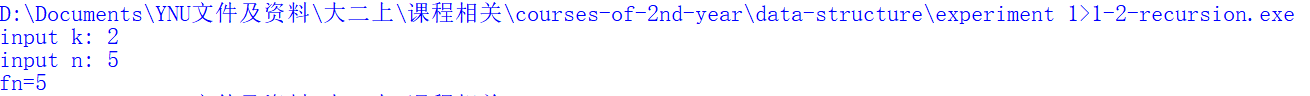
**Input:**

input k: 2

input n: 5

**Output:**

fn=5



1. 循环解法

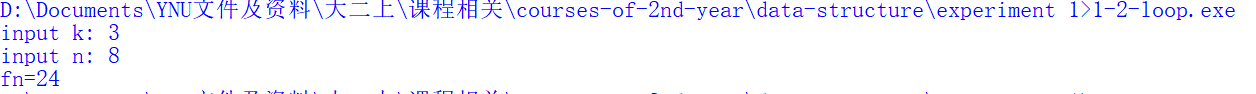
**Input:**

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

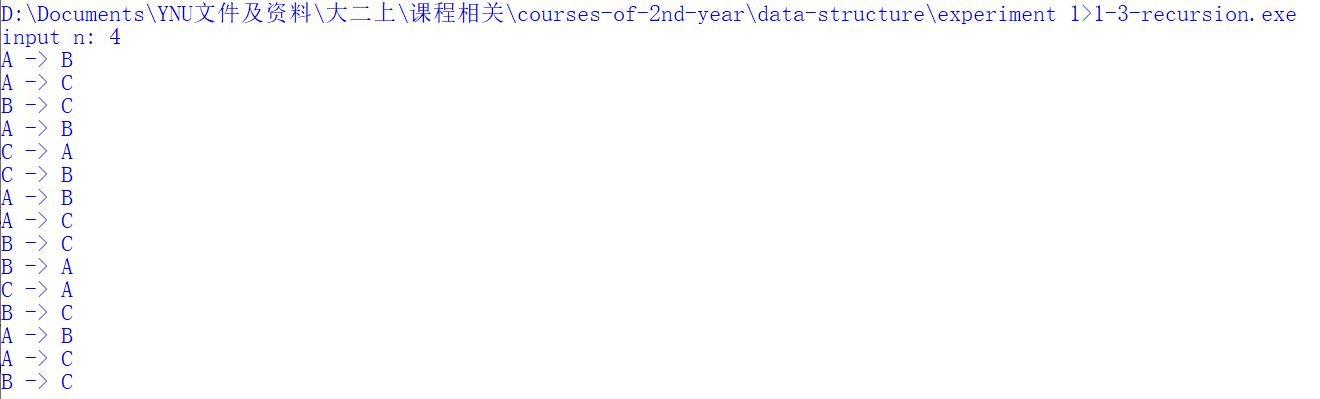
C -> A

B -> C

A -> B

A -> C

B -> C



1. 循环解法

**Input:**

input n: 3

**Output:**

A -> C

A -> B

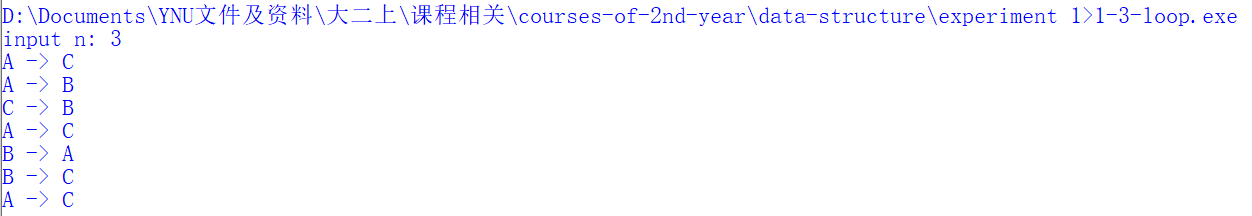
C -> B

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:

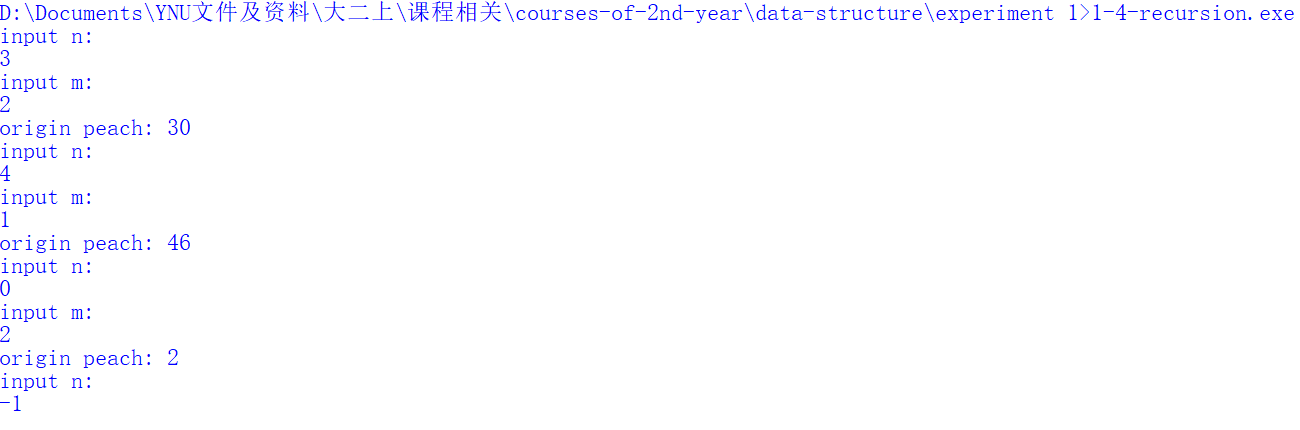
-1

**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

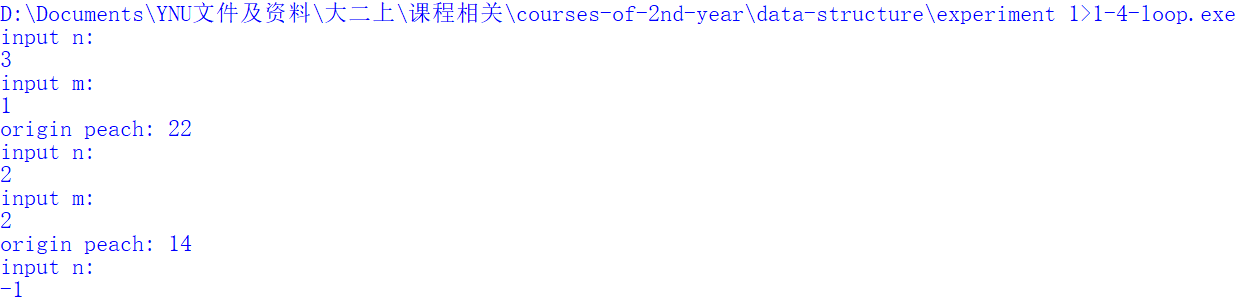
input n:

-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。