**5.37(汉诺塔：迭代解法)**

代码如下：

#include <stdio.h>

int main()

{

int rings, last, next, x, z[500], s[3];

printf("how many rings? ");

scanf("%d", &rings);//输入环数

int width = rings + 1;

for (x = 1; x <= rings; x++)

z[x] = width - x;

for (x = 0; x <= 2 \* width; x += width)

z[x] = 1000;

if (rings % 2 == 0)

{

last = 1; s[2] = 0; s[1] = 1;

z[width + 1] = z[rings];

}

else

{

last = 2; s[1] = 0; s[2] = 1;

z[2 \* width + 1] = z[rings];

}

printf("from 1 to %d\n", last + 1); s[0] = rings - 1;

while (s[0] + s[1])

{

if (last == 0) next = z[width + s[1]]<z[2 \* width + s[2]] ? 1 : 2;

if (last == 1) next = z[s[0]]<z[2 \* width + s[2]] ? 0 : 2;

if (last == 2) next = z[s[0]]<z[width + s[1]] ? 0 : 1;

if ((z[next\*width + s[next]]>z[last\*width + s[last]]) ||

((z[last\*width + s[last]] - z[next\*width + s[next]]) % 2 == 0)) last = 3 - next - last;

printf("from %d to %d\n", next + 1, last + 1);

s[next] = s[next] - 1; s[last] = s[last] + 1;

z[last\*width + s[last]] = z[next\*width + s[next] + 1];

}

}

真看不懂 = =

**5.48(研究项目：对递归计算fibonacci数的改进)**

1）递归法

代码如下：

#include<stdio.h>

int fibonacci(unsigned int n)

{

unsigned long long int result;

if(n == 1||n == 2)

return 1;

else{

result = fibonacci(n - 1)+fibonacci(n - 2);

return result;

}

}

int main(void)

{

unsigned int num;

scanf("%u",&num);

printf("%u\t%llu\n",num,fibonacci(num));

return 0;

}

特点：键入的num值（求第num个Fibonacci数）每增加1，则递归函数调用次数是原先的2倍。

在计算譬如45以上的Fibonacci数时运算极慢，效率很低。

代码易读，简洁。

2）迭代法

代码如下：

#include <stdio.h>

int fibonacci(unsigned int n)

{

unsigned long long int a = 1;

unsigned long long int b = 1;

unsigned long long int c = a;

for (int i = 3;i <= n;i++)

{

c = a + b;

a = b;

b = c;

}

return c;

}

int main(void)

{

unsigned int n;

scanf("%u",&n);

printf("%u\t%d",n,fibonacci(n));

return 0;

}

特点：运算快，键入的n值每增加1，则循环次数增加1(即运算次数增加1)。

在计算45及以上的数值时运算很快。

代码较普通递归冗长。

3）尾递归

代码如下：

#include <stdio.h>

double fib(double n, double a, double b)

{

if(n<=0.0)

return -1.0; //错误输入

else if(n==1.0 || n==2.0)

return b; //b记录最靠后的一项

else

{

while(n>2.0)

return fib(n-1.0, b, a+b);

}

}

int main(int argc, char \*argv[])

{

double d, n = 20.0;

d = fib(n, 1.0, 1.0);

printf("斐波那契数列第 %.f 项的值为 %.f。\n", n, d);

return 0;

}

特点：混合了迭代和递归，保证了运算速度，也提升了代码的简洁性和可读性。

不用尾递归，函数的堆栈耗用难以估量，需要保存很多中间函数的堆栈。而使用尾递归，这样则只保留后一个函数堆栈即可，之前的可优化删去。

**5.50（计算机辅助教学）**

代码如下：

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int question()//定义question函数产生随机一位整数之间的乘法题目

{

int a, b;

a = rand()%9 + 1;

b = rand()%9 + 1;

printf("How much is %d times %d?\n", a, b);

return a\*b;//返回a\*b的正确值与输入答案比对

}

int main()

{

srand(time(NULL));

int i = 0;

int answer = 0;

while(i != -1){//用i是否等于-1判断是否循环

int c = question();

scanf("%d",&answer);

while(answer!= c){

printf(“No. Please try again.”);

scanf("%d",&answer);

}

Printf(“Very good!”);

printf("Continue? (-1 to quit)\n");//输出-1以退出while语句

scanf("%d",&i);

}

return 0;

}

**5.51（计算机辅助教学：减轻学生疲劳）**

代码如下：

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int question()

{

int a, b;

a = rand()%9 + 1;

b = rand()%9 + 1;

printf("How much is %d times %d?\n", a, b);

return a\*b;

}

int main()

{

srand(time(NULL));

int i = 0;

int answer = 0;

while(i != -1){

int c = question();

scanf("%d",&answer);

while(answer!= c){

int num = rand()%4 + 1;//基于5.50，int num 来随机选择失败的提示语

switch(num){

case 1:

printf("No. Please try again.\n");

break;

case 2:

printf("Wrong. Try once more.\n");

break;

case 3:

printf("Don't give up!\n");

break;

case 4:

printf("No. Keep trying.\n");

break;

}

scanf("%d",&answer);

}

int num2 = rand()%4 + 1;

switch(num2){// 基于5.50，int num2 来随机选择成功的提示语

case 1:

printf("Very good!\n");

break;

case 2:

printf("Excellent!\n");

break;

case 3:

printf("Nice work!\n");

break;

case 4:

printf("Keep up the work!\n");

break;

}

printf("Continue? (-1 to quit)\n");

scanf("%d",&i);

}

return 0;

}