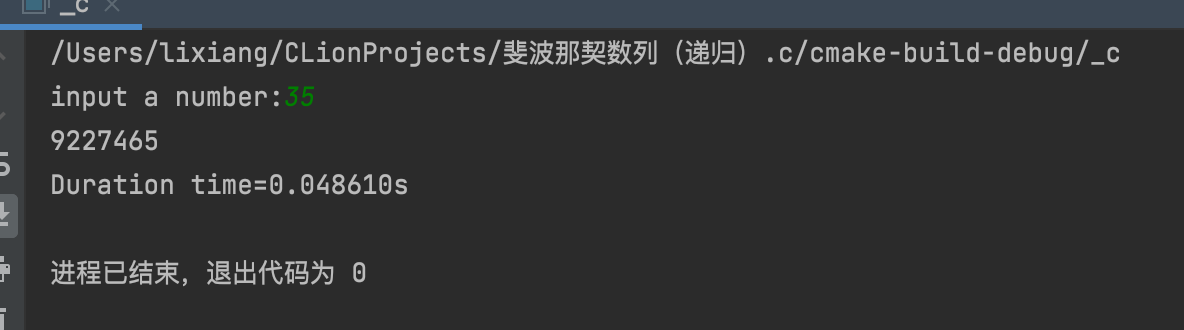
12.1

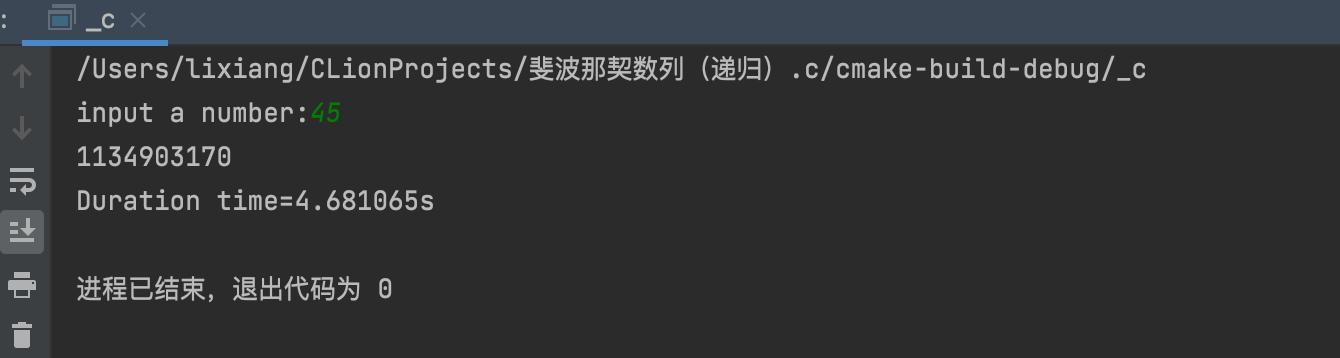
程序源代码：

#include <stdio.h>  
#include <time.h>  
int fibonacci(int n)  
{  
 if(n==1||n==2) return 1;  
 else return fibonacci(n-1)+ fibonacci(n-2);  
}  
int main()  
{  
 int n;  
 printf("input a number:");  
 scanf("%d",&n);  
 double start=clock();  
 fibonacci(n);  
 double end=clock();  
 printf("%d\n", fibonacci(n));  
 printf("Duration time=%fs\n", (end-start)/CLOCKS\_PER\_SEC);  
 return 0;  
}

程序运行截图







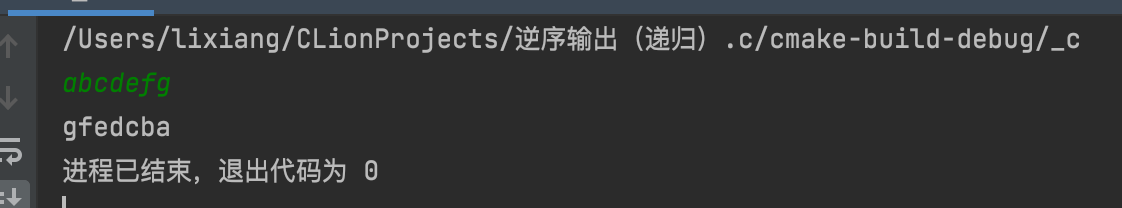
根本原因：冗余计算，每个递归调用都触发另外两个递归调用，而这两个调用的任何一个还将触发两个递归调用，再接下去的调用也是如此。这样，冗余计算的数量增长得非常快。例如，在递归计算Fibonacci(10)时，Fibonacci(3)的值被计算了21次。但是，在递归计算 Fibonacci(30)时，Fibonacci(3)的值被计算了317811次。当然，这317811次计算所产生的结果是完全一样的，除了其中之一外，其余的纯属浪费。这个额外的开销真是相当恐怖。如果使用一个简单循环来代替递归则可以解决这个缺陷！

12.3

程序源代码

#include <stdio.h>  
void Reverse() {  
 char c;  
 scanf("%c",&c);  
 if( c != '\n')  
 {  
 Reverse();  
 printf("%c", c);  
 }  
}  
int main()   
{  
 Reverse();  
}

程序运行截图



12.5

程序源代码

#include<stdio.h>  
#include<string.h>  
int i=0;  
int fun(char c[] , int length)  
{  
 if(length -1 - i <= 1) return c[i]==c[length-1];  
 if(c[i++] == c[length - 1])  
 {  
 fun(c , length -1 );  
 return 1;  
 }  
 else return 0;  
}  
int main(void){  
 char c[100];  
 scanf("%s",c);  
 int length= strlen(c);  
 if(fun(c , length))  
 {  
 printf("yes");  
 }  
 else printf("no");  
}

程序运行截图：

