Array

Recursion

Recursion

- C program allows a function to call itself
- When a function fun calls itself
  - a copy of the function fun is created (fun1)
  - variables in fun and fun1 have the same names but are unrelated
  - when fun1 returns, it returns to fun at the end of the line where it has been called

If the standard input is

abcd

what is the output?

```
#include <stdio.h>
#include <stdlib.h>
void mystery(void);

main() {
        mystery();
}
void mystery( void) {
        int c;
        if( (c=getchar() ) != EOF ) {
                mystery();
                putchar(c);
        }
}
```

output:dcba

Example 1 factorial function

```
int fact (int num);//declaration
int num=3;
int ans;
ans=fact(num);//invoke fact
int fact(int n)
{
if(n<=1)
  return 1;
else
  return n*fact(n-1);
}
```

In fact, n=3; fact1 created

in fact1, n=2; fact2 created

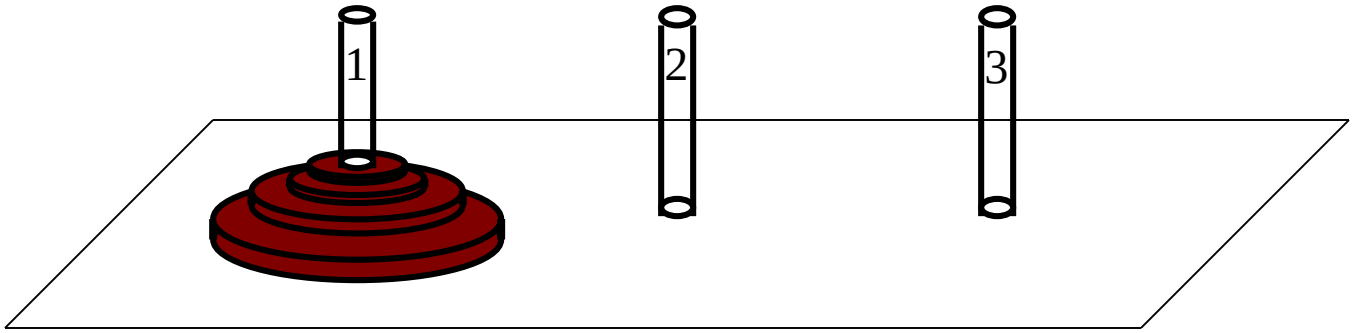
in fact2, n=1

returns to fact1, n=2, fact2=1,fact1=2*1

returns to fact, n=3, fact1=2*1, fact=3*2*1

♦ ***Problem***

The Tower of Hanoi is a puzzle consisting of three pegs mounted on a board and n disks of various sizes with holes in their centers (see follow figure. If a disk is on a peg, only a disk of smaller diameter can be placed on top of it. Given all the disks properly stacked on one peg as in following Figure, the problem is to transfer the disks to another peg, by moving one disk at a time.



The Tower of Hanoi puzzle

♦ ***Solution***

Example 2 Tower of Hanoi

- assume that hanoi(m,ori,dest,spare) is the solution to the m-disk problem
  - ori being the peg where m disks are originally stacked
  - dest = the peg where m disks are to be moved
  - spare = spare peg
  - hanoi(6,A,B,C) is the solution for the case where 6 disks are originally on peg A and finally moved to peg B; C is spare
  - hanoi(6,A,C,B) is then the solution for the case where 6 disks are moved from peg A to peg C; B is spare

Tower of Hanoi solution

The (m+1)-disk problem: m+1 disks on peg A to be move to peg B

- If we know the solution for the m-disk problem, we can construct a solution for the m+1 disk problem
- move the top m disks from A to C
  - hanoi(m,A,C,B)
- move the largest (m+1)disk from A to B
  - printf"..."
- move the m disks on C to B
  - hanoi(m,C,B,A)
- Therefore the (m+1)-disk solution can be expressed in terms of the m-disk solution, which can be expressed in terms of the (m-1)-disk solution, etc.
- Finally, m+1 disk problem is expressed in terms of the one-disk problem for which the solution is obvious.

Robot Walk
（one step is only 1 or 2 meters）

- Let walk(n) be the number of ways the robot can walk n meters
  - For the n-meter case assume that the robot's first step is 1 m
    - then there are n-1 meters left
    - number of ways the robot walks n-1 meters is walk(n-1)
  - if the robot's first step is 2 meters,
    - then there are n-2 meters left
    - number of ways the robot walks n-2 meters is walk(n-2)
  - the sum of the above 2 cases is the number of ways the robot can walk n meters
    - walk(n) = walk(n-1) +walk(n-2)
  - the base solutions are
    - walk(1)=1
    - walk(2)=2

```
walk(n)
if(n<=2)
return n;
return (walk(n-1)+walk(n-2));
```

- Assume n=4
- walk(4):
  - n=4
  - call walk1(3)
- walk1:
  - n=3
  - call walk2(2)
- walk2:
  - n=2; return 2 to walk1
- walk1
  - call walk3(1); return 1 to walk1
- walk1 return 2+1 to walk

continue

- walk call walk4(2)
- walk4 return 2 to walk
- walk return 3+2

➢ Monte Carlo Integration

♦ ***Problem***

Approximate the integral

$$\int_0^1 e^{-\frac{x^2}{2}}\,dx$$

by using the Monte Carlo technique.

♦ ***Sample Input/Output***

Input is in color, output is in black.

Run again (1 = yes, 0 = no)? 1
How many trials? 100
Integral = 0.820000

Run again (1= yes, 0 = no)? 1
How many trials? 1000
Integral = 0.865000

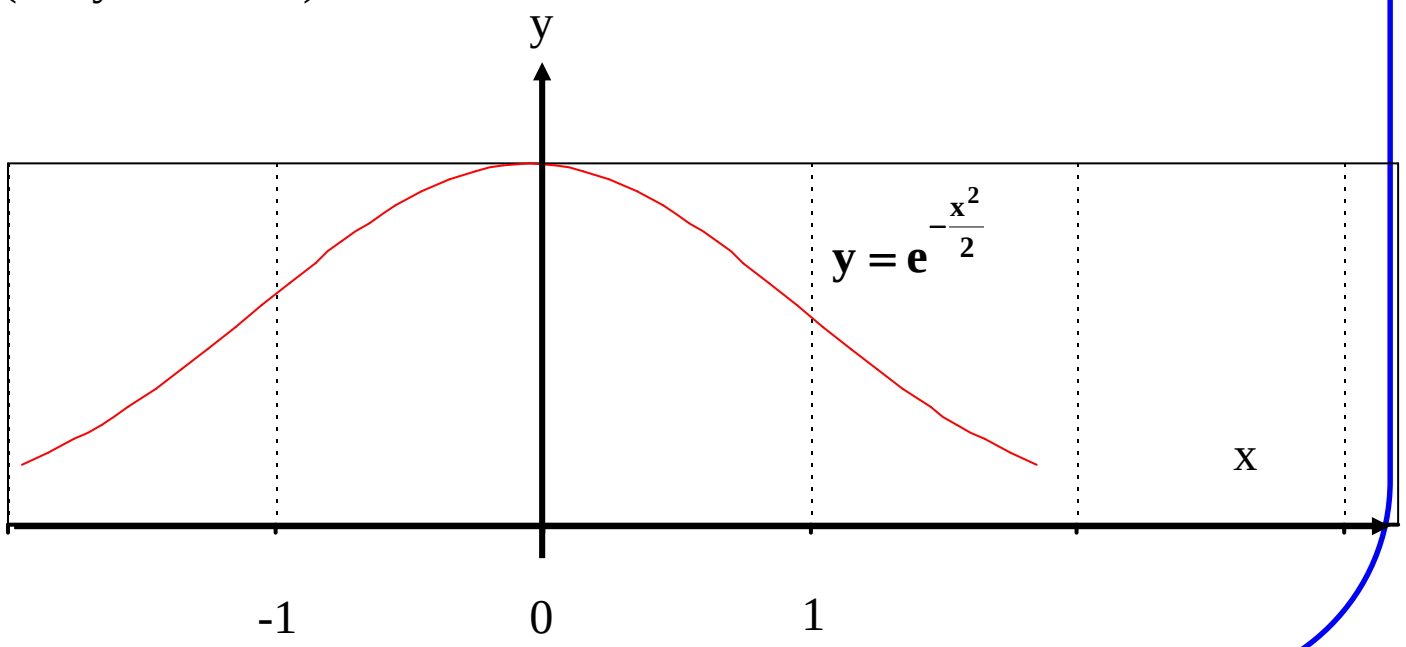
Run again (1= yes, 0 = no)? 1

How many trials? 5000
Integral = 0.855200

Run again (1 = yes, 0 = no)? 1
How many trials? 10000
Integral = 0.861400

Run again (1 = yes, 0 = no)? 0

♦ ***Solution***



♦ ***C Complementation***

```
/* This program approximates the integral e^(-x^2/2) from 0 to 1 by using
Monte Carlo integration. */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
/* rand( )-random number generators; RAND_MAX- Maximum random
   integer. Macro RAND_MAX  is defined in stdlib.h */
#define dran( dr )       ( ( rand( ) / (double) RAND_MAX ) * (dr) )
#define Prompt( p )     printf( p);
#define Response( r )  scanf( "%d", &r)
#define Answer( a )     printf( "Integral = %f\n", a )
```

```
double area ( int trials );
main( ) {
        int resp, trials;
        do {
                Prompt ( “Run again (1 = yes, 0 = no)? ”);
                Response ( resp );

                if ( resp ) {
                        Prompt ( “How many trials ? “ );
                        Response ( trials );
                        Answer (  area( trials ) );
                }
        } while ( resp );
}

double area ( int trials ) {
                double x, y;
                int count, hit;
```

```
        for ( count = 0, hit = 0; count < trials; count++) {
                x = dran( 1.0 );
                y = dran( 1.0 );
                if ( y <= exp( -x * x / 2 ) )
                        hit++;
        }
        return ( double ) hit / (double) trials;
}
```

Functions with arbitrary number of arguments (*skip this section if necessary)

```
#include <stdio.h>
…
int max(int n, …);
main()
{
int max_i;
int i1,i2,i3,i4;
```

enter 1 to 4 integers and generate EOF when done

if 1 integer +EOF, then max_i = max(1,i1)

if 2 integers +EOF, max_I=max(2,i1,i2)

etc.

Definition of the max function

```
int max(int n,...)
{
    int nmax=INT_MIN set nmax = lowest integer in the system
    int i;
    va_list  arg_addr; defined in stdarg.h; arg after n is located at arg_addr
    int next_I;
    va_start(arg_addr,n); set arg_addr to the address of the first arg after n
    in max
    for loop
    next_I=va_arg(arg_addr,int) return the int value of the next arg in max;
    update arg_addr; the 2nd arg in va_arg specifies the type (int) being
    read in max
    if next_I > nmax; then nmax=next_I
    va_end(arg_addr) mop up
}
```

Examples

- int printf(char* format_string,…)
  - printf expects one argument (format_string) followed by arbitrary number of arguments
  - char* is a pointer to char which is the address of the format_string
- int fprintf(FILE* fp,char* format_string,…)
  - fprintf expects 2 arguments, fp and format_string
  - followed by arbitrary number of arguments
  - FILE* is a pointer to fp, the filename for printing
  - char* is a pointer to the address of format_string