Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Practice Sheet #06

Topic: Recursion in C

Date: 30-01-2017

1. What string does the following program print?

```
#include <stdio.h>
#include <string.h>

void strFunc2 (char A[], int n)
{
  char t;
  if (n <= 1) return;
     t = A[0]; A[0] = A[n-1]; A[n-1] = t;
     strFunc2(&A[1],n-2);
}

int main ()
{
   char A[10] = "PDS 2005";
   strFunc2(A,strlen(A));
   printf("%s", A);
}
```

   (a) 5002 SDP
   (b) 5DS 200P
   (c) PDS 2005
   (d) SDP 2005

   Ans. a

2. In this exercise, we deal with complex numbers of the form $(\frac{a}{b}) + i(\frac{c}{d})$ with a, b, c, d integers such that b and d positive. The following structure represents such a complex number.

```
typedef struct {
     int a; /* Numerator of the real part */
     int b; /* Denominator of the real part */
     int c; /* Numerator of the imaginary part */
     int d; /* Denominator of the imaginary part */
} complexNumber;
```

   Our aim is to compute the sum and product of two such complex numbers with each fraction reduced to the standard form (i.e., reduced to the form $\frac{m}{n}$ with gcd(m, n) = 1, n > 0).

   Fill in the blanks.

```
int gcd(int a, int b) /*Compute the gcd of two integers*/
{
    if (a < 0) a = -a; if (b < 0) b = -b;
    if (a == 0) return b; if (b == 0) return a;
    return gcd(b,___________ );
}
```

```
void reduce(int *a, int *b) /* Reduce a fraction to lowest terms */
{
   int d;
   d = gcd(______,_______); *a =_______; *b = ______;
}

void sum(int a1, int b1, int a2, int b2, int *a, int *b)

/* (a1/b1)+(a2/b2) */
{
*a = a1*b2 + a2*b1; *b = b1*b2;
reduce( ___________, _________);
}

void mul(int a1, int b1, int a2, int b2, int *a, int *b)
/* (a1/b1)*(a2/b2) */
{
*a = a1 * a2; *b = b1 * b2;
reduce( __________, __________);
}

complexNumber compAdd(complexNumber z1, complexNumber z2)
/* Compute z1 + z2 */
{
complexNumber z;
                             /* Set the real part of z */
sum (_____________ );
                            /* Set the imaginary part of z  */
sum (_____________ );
return z;
}

complexNumber compMul(complexNumber z1, complexNumber z2)
                /* Compute z1 * z2 */
{
/ *Some code for complex number multiplication */
}
```

3. What value does the call `h(5)` return, where `h` is defined as follows?

```
int h ( int n )
{
    if (n == 0) return 1;
    return 2*h(n-1);
}
```

(a) 25
(b) 32
(c) 64
(d) 120

Ans. b

4. Let the function `g` be defined as follows:

```
int g ( int n )
{
if (n < 2) return n;
return g(n/2);
}
```

What is the value returned by the call `g(142857)`?

(a) 0
(b) 1
(c) 2
(d) 71428
Ans. b

5. A two-dimensional character array is used to store a list of names. Each name is stored in a row. An empty string indicates the end of the list. Complete the ***recursive*** function `printNames()` to print the names stored in the two-dimensional array supplied as `p`.

```
void printNames (char (*p)[100])
{
______________________________________
}

int main()
{
   char names[20][100] = {"Bombay", "Delhi", "Guwahati",
"Kanpur","Kharagpur", "Madras", "Roorkee", "" };
printNames(names);
return 0;
}
```

6. Write a *recursive* C function `findMed(…)`, which returns the median of a one-dimensional array of integers, that is, the element which is larger than half of the elements and smaller than half of the elements. Suppose $a_1 \leq a_2 \leq \cdot \cdot \cdot \leq a_n$ is the sorted version of the input array A. If $n = 2k + 1$, then the element a_{k+1} is the median of A. On the other hand, if $n = 2k$, we take a_k as the median of A. Your function should use a partitioning technique as in Quick sort. Use the following function prototype:

```
int findMed (int A[], int startidx, int endidx, const int
medidx);
```

Here, the second and the third parameters are the start and end indices of the current partition; the fourth parameter is the index of the median in the sorted version of A and is kept constant across the calls.

7. Write a recursive C function `char *lastOccur (char *s, char c)` which takes a string `s` and a character `c` as input; it returns a pointer to the last occurrence of `c` in `s`, or `NULL` if the character `c` does not occur in `s`. *Do not use any string library functions.*

8. Consider the following recursive C function.

```
unsigned int f(unsigned int n)
{
   if (n < 10) printf("%d",n);
   else {
      printf("%d", n%10);
      f(n/10);
      printf("%d", n%10);
   }
}
```

What does the call `f(351274)` print?
Ans. 47215351274

9. Consider the following piece of C code.

```
int S(int n, int k)
{
   if (k > n) return 0;
   if ((k == 1) || (k == n)) return 1;
   return S(n-1,k-1) + k * S(n-1,k);
}
```

(a) How many times is `S(…)` called (including the outermost call) to compute `S(5,3)`?
(b) What is the value returned by `S(5,3)`? Show your calculations.
(c) Write a recursive function `SMul()` to count the number of multiplications in the call `S(n,k)`.

10. Complete the following recursive function that returns the floating point value of a continued fraction $<a_0, a_1, . . . , a_{n-1}>$.
Note that $<a_i, a_{i+1}, . . . , a_{n-1}> = a_i + 1/(<a_{i+1}, a_{i+2}, . . . , a_{n-1}>)$ for i = 0, 1, . . . , n − 2.

```
double cfracrec ( int i, int n)
{
   int a;
   printf("Enter a_%d: ", i); scanf("%d", &a);
                 /* Read ai in a */

   if ( i == ________) return__________ ;
            /* The terminating case, no recursive call */

   return_________ ;
/* Make a recursive call and return */
}
```

The outermost call for computing $<a_0, a_1, . . . , a_{n-1}>$ should be:
`cfracrec( ___,___ )`

11. Consider the following piece of C program.

```
#include <stdio.h>

int F[10];

int fib(int n)
{
   printf("*");
   if (n <= 1) return 1;
   if (F[n] != 0) return F[n];
   F[n]=(fib(n-1) + fib(n-2));
   return F[n] ;
}

int main( )
{
   int j;
   for (j=0; j< 10; j++) F[j] = 0;
   fib(5);
}
```

How many `*s` will be printed by the following program?

Ans. 9

12. For an integer k we have $a^{2k} = (a^{k})^2$, and $a^{2k+1} = (a^{k})^{2} * a$.

Use this observation to write a **recursive** function that, given a positive real number a and a non-negative integer n, computes a^n.
The function should be efficient in terms of the total number of multiplications required to find the answer.

13. The following incomplete code is for a **recursive function** RecursiveArraySum, which calculates the sum of all the n-elements of an array passed as function argument. Write the missing pieces of the code such that the sum is calculated by recursively calling the function. The function RecursiveArraySum is invoked with a 4-element array of [5, 10, 20, 40] as argument. The base address of the array is at 10,000 (decimal). At every recursive call, write the values of the parameters passed and the return value from the function.

```
int RecursiveArraySum(int *arr, int n){
   if . . . . . . . . . return . . . . . . ;
   . . . . RecursiveArraySum(...............);
}
```

Ans.
if (n == 0) return 0;
else
return (a[0] + RecursiveArraySum(a+1,n-1));
OR
return (a[n-1] + RecursiveArraySum(a,n-1));

14. The following function is expected to compute the factorial of a positive number **n**. Is the given code error free? If not, then using one sentence state the reason why it will not work and also give the corrected code.

```
long fact(int n)
{
   long k=1;
   do
      k*=fact(i--);
   while(n>0);
   return k;
}
```

Ans. No, it uses both recursion and iteration.
long fact(int n)
{
long k=1;
for(i=0;i<n;i++)
k*=i;
return k;
}
Or
long fact(int n)
{
if (n<=1) return 1;
return n*fact(n-1);
}

15. What value will the following function return when called as `recur(3)`?

```
int recur(int data)
{
   int k;

   k=(data>2)?(recur(data-1)-recur(data-2)):1;
   return k;
}
```

Ans. 0

16. Consider the following recursive function:

```
unsigned int f (unsigned int n)
{
    if (n <= 2) return 1;
    return f(n-3) + f(n-1);
}
```

What is the maximum height to which the recursion stack grows when the outermost call is `f(10)`? Assume that the stack is empty just before this outermost call.

(a) 5
(b) 9
(c) 13

(d) 32

17. Mathematically express the return value f (n) of the function supplied below. Your mathematical expression for f (n) must hold for all integers n > 0.

```
unsigned int f(unsigned int n)
{
if (n == 0) return 0;
return 2 * f(n - 1) + 1;
}
```

Ans. $f(n)=1+2+2^2+2^3+\ldots\ldots..+2^n=2^{n+1}-2$

18. The Fibonacci sequence is defined as

```
F(0) = 0,
F(1) = 1,
F(n) = F(n-2)+F(n-1) for n > 2.
```

The following function defines another sequence G(n) for n = 0, 1, 2, 3, …. .
How is the sequence G(n) mathematically related to the Fibonacci sequence F(n)? (Express G(n) in terms of F(n). Your expression should hold for all integers n > 0.)

```
int G (unsigned int n)
{
if (n == 0) return 0;
if (n == 1) return 1;
return G(n-2) - G(n-1);
}
```

19. Convert the recursive program in the left box to an iterative program in the right box.

```
int sum(int n)
{
   if (n < 1) return 0;
   return sum(n - 1) + sum(n - 2) + n;
}
```

20. What is printed by the following program?

```
#include<stdio.h>

void t(int n, char fp, char tp, char ap)
{
   if(n==1){
      printf("%c%c",fp,tp);
      return;
   }
   t(n-1,fp,ap,tp);
   printf("%c%c",fp,tp);
   t(n-1,ap,tp,fp); return;
}

void main()
{
 t(2,'x','y','z');
}
```

Ans. xzxyzy

21. If our universe consists of only the set of nonnegative integers, the even and odd numbers can be characterized as follows: a number is *even* if its predecessor is odd, a number is *odd* if is not even, the number 0 is even by definition. Complete the C functions below which return 1 when the input number *n* is even or odd respectively. Both the functions are defined in the same program and they call each other.

```
int IsEven(unsigned int n)
{
   if (n == 0) {
      return 1;
   }
   else {
      return_________;
   }
}

int IsOdd(unsigned int n)
{
   return__________;
}
```

22. A recursive algorithm may require more computation time and memory than its iterative version.

(a) TRUE
(b) FALSE
Ans. a

23. Let us assume xn is defined as follows:

```
xn = x n/2*x n/2 if n is even
   = x*x n/2*x n/2 if n is odd
```

(a) Write a recursive C function to compute xn based on the above definition. Clearly mention suitable base cases.
(b) Find out how many function calls are made to compute x16 using your function.

--*--