Indian Statistical Institute, Bangalore

CS1 – Final Exam, 2014-15 (Backpaper)

Total Marks: 100

1. **(10)** Write a function that finds primes from 2 to L (assumes L **>=** 2) using the following method:

   Let p_0***=2; m=0;*** */* 2 is the first prime. */*
   for each number ***n*** *in {3 .. L }:*
   If n is divisible by any of the numbers from p_0 *..* p_m*, then*
   n is not prime.
   Else n is prime, therefore,
   set p_{m+1}***=n, m=m+1***.

   It takes two parameters: the limit L and an array p in which the function stores the list of primes found. It returns the number of primes found from 2 to L.

2. **(5x3=15)** Consider the following function f :

   $f(0)=1$
   $f(1)=1$
   $f(n)=f(n-1)+f(n-2)$ if $n>1$

   a. Write a *recursive* C function: **`int rfib(int n);`** that implements the above definition of f.
   b. Once **`rfib(4)`** is called, list the ordered sequence of invocations of the function **`rfib`** mentioning the parameter values.
   c. Write a function **`int lfib(int n);`** that also computes the same function, except that it does so with a loop and no recursion. It simply keeps track of the last two values of f to compute the next value.

3. **(3x3+6=15)** Assume you are given a linked list of structures of the following kind:

```
struct student {
   int id;
   struct student * next;
};
```

   Write each of these functions:

   a. **`struct student * findMinAfter(struct student *p`**); Given a pointer **`p`** to a structure, it looks at all the structures *strictly after* the structure pointed to by **`p`** in the list and returns a pointer to the node *previous* to the one among them with minimum value of **`id`**. For example if we have 0->2 ->9->14->6->19 in the list and if we are given a pointer to node with 2, the function would return a pointer to the node with **`id`** 14 since 6 is the minimum to the right of 2.

b. **`struct student *removeStudentAfter(struct student *p);`** so that it removes the structure *immediately after* the one pointed to by **`p`** from the list, and returns a pointer to that removed structure.

c. **`void inserStudentAfter(struct student *p, struct student *q)`**; so that it inserts the structure pointed to by **`q`** *immediately after* **`p`** in the linked list pointed to by **`p.`**

d. **`void sortStudentListAfter( struct student * head)`**; so that if it is called as **`sortStudentList(head),`** where **`head`** points to the first structure in the linked list; then after the call returns, the structures in the linked list *after* the item pointed to by head are sorted by **`id`**. Use the idea from the Selection Sort method. Use the above three functions to do this.

4. **(2x3+9=15)** This question is about Binary Search.
   a. What assumption does binary search make about elements of the array that is being searched?
   b. Binary search is more complicated to program than simple sequential search, yet it is often used; Why?
   c. Does Binary search work for a linked list? Why or why not?
   d. Write a recursive function to implement binary search for an array **`a`** with **`n`** integers searching for an element **`x`**. It returns the position where it found **`x`**; if **`x`** is not in **`a`** then it returns **`-1`**.

5. **(5+5=10)** Assume **`struct student`** is defined as in question 3 above. Consider the following functions:

```
struct student * foo(struct student * h, int n){
       if ( h == NULL )
           return NULL;
       if ( h-> id == n )
              return h;
       return foo(h->next, n);
}
struct student * goo(struct student * h, struct student * m){
       if ( h == NULL )
            return m;
       h->next=goo(h->next,m);
       return h;
}
```

`If h` is a pointer to the linked list:: (h->5->7->3->2->6->4->9->8, where the values are the ids), **`n`** is the integer:: 3 and **`m`** is a pointer to a newly **`malloc-`**ed and initialized structure (with id set to 3)**`,`** say what these two calls do and what they return):

   a. **`foo(h,n)`**
   b. **`goo(h,m)`**

6. **(15)** Write the function

   `void sortIndices(int a[], int n, int d[]);`

   The function takes in an array **`a of n`** different integers and constructs the array **`d`** called the sorted index array to **`a`**. Elements in **`d`** are a permutation of the numbers **`0..n-1`** and they represent positions in the array a. When the function returns, the elements of **`d`** are arranged so that: **`a[d[0]] < a[d[1]] < a[d[2]] < .. < a[d[n-1]].`** For example if **`n=4`** and **`a`** is given as: **`20,4,5,2`** then after the function returns, d would contain: **`3,1,2,0`** ie **`a[3], a[1], a[2], a[0]`** is in sorted order.
   Your function should not move the elements of **`a`**. You should use bubble sort.
   Hint: Use bubble sort on d instead of **`a`**; with one change - to compare two items of **`d`**, use the values of the corresponding items of **`a`** instead.

7. **(10)** Write a function named **`transpose`** that takes two parameters – a two dimensional array **`A`** and a value **`n`** indicating that it stores an **`n`** *x* **`n`** matrix. The function transposes the matrix, i.e., **`A`** is replaced by $\mathbf{A}^T$. It has no return value. Assume there are no more than 20 columns in a row for **`A`**.

8. **(10)** For each of the following mention True or False:
   a. The a.out file produced by the compiler is text that is human readable.
   b. The .h file used in include statements is text that is human readable.
   c. The .h file contains source code of functions commonly used.
   d. We link libraries with our program (eg use **`-lm`** to link the math library when using **`gcc`**), these libraries contain source code of functions.
   e. In the **`for`** loop with a pattern: **`for( e1 ; e2 ; e3) { … } ,`** e_2 can be an assignment expression.
   f. In the **`for`** loop with a pattern: **`for( e1 ; e2 ; e3) { … }`** , e_1 is executed only once.
   g. In the **`while`** loop: **`while(1){ … }`** , the body of the while loop is executed only one time.
   h. Assume **`n`** is a positive integer, then the **`while`** loop: **`while(n--) printf("%d\n",n);`** prints the numbers from **`n`** down to 1.
   i. Assume **`n`** is 10 , then the loop: **`while(n) { if(n%7) break; else continue; n--;}`** , the loop will terminate with **`n`** having value 7.
   j. The following code prints "0123": **`char *s="0123"; while(*s){ printf("%c",*s); s++; }`**