Name:					
Roll No: e.g. 170001		**Dept.**: e.g. CHE		**Sect.**: e.g. A4	

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

Instructions:

Total: **30 X 6 = 180 marks**

1. This question paper contains a total of 24 pages (24 sides of paper). Please verify.
2. Write your name, roll number, department, section on **every side of every sheet** of this booklet
3. Write final answers **neatly with a pen** in the given boxes.
4. Do not give derivations/elaborate steps unless the question specifically asks you to provide these.

C Precedence Table:

Operator Category	Operators	Associativity
unary operators	`- ++ -- ! sizeof (type)`	Right to Left
arithmetic multiply, divide and remainder	$*$ / %	Left to Right
arithmetic add and subtract	+ −	Left to Right
relational operators	$< \ \leq \ > \ \geq$	Left to Right
equality operators	== !=	Left to Right
logical *and*	&&	Left to Right
logical *or*	||	Left to Right
conditional operator	? :	Right to Left
assignment operators	= += −= *= /= %=	Right to Left

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

Problem 1 (Back to the Basics: 23 + 7= 30 marks**).**

1. Riya wrote the following programs, but since she did not revise the Pre-Midsem syllabus for ESC101, she is a bit confused as to what their outputs will be. Help her clear the confusion by providing the outputs of each of the programs. If she has made an error due to which a program does not compile or crashes while running, tell her where she has made an error. If the program is correct, write its output in the corresponding space provided, else write *'ERROR'* and explain briefly why it occurred.
(7 + 5 + 4 + 3 + 4 = 23)

(a)

```
#include<stdio.h>

int main(){
    int i = 10, j = 1000;
    for(; i < j; i += 2){
        printf("Looping in!\n");
        i = i/2;
        for(; 2*i < j; j /= 2){
            printf("%d %d\n", i, j);
            i = i + 2;
            j = j - 4;
        }
    }

    return 0;
}
```

```
Looping in!
5 1000
7 498
9 247
11 121
13 58
Looping in!
8 27
```

(b)

```
#include<stdio.h>

int stat_loop(){
    int static num = 800;
    return (num = num/2 -1);
}

int main(){
    for(stat_loop(); stat_loop() + 1;){
        printf("%d\n", stat_loop());
    }

    return 0;
}
```

```
98
23
4
-1
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

(c)

```
#include<stdio.h>

int main(){
    int a = -6, b = -5;
    if(++a = b++)
         printf("Babbling bumbling band of baboons!\n");
    else
         printf("Dobby is free!");

    return 0;
}
```

```
Error. ++a returns a constant
-5 and b++ returns a constant
-5.
-5=-5 is not a valid expression
as we cannot assign value to a
constant.
```

(d)

```
#include<stdio.h>
#define lamb 2
#define cat 1

int main(){
    int a = 2, b = -1;
    switch(a?--a : b++){
        case lamb:
            a = a || b++;
            printf("%d %d\n", a, b);
            printf("Baba Black Sheep!\n");
        default:
            a = a && b--;
            printf("%d %d\n", a, b);
            break;
        case cat:
            printf("%d %d\n", a, b);
            printf("Meow!");
    }

    return 0;
}
```

```
1 -1
Meow!
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

(e)

```
#include<stdio.h>

int main(){
    int c = 5, d = 10, e;
    e = c*d++;
    printf("%d\n", e);
    if(++c||++d)
        printf("%d %d\n", c, d);
    else
        printf("%d %d\n", c * d, c +
    e);

    return 0;
}
```

```
50
6 11
```

2. A sorted rotated array is obtained by rotating an originally sorted(ascending) array by some unknown steps i.e. array {1,2,3,4,5,6} may become {4,5,6,1,2,3} or {6,1,2,3,4,5}. Complete the following program which takes as input an integer N and a sorted rotated array of size N with integer entries as input and outputs the value of the minimum element of the array.(7)

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

int main(){
    int i, N;
    scanf("%d\n",&N);
    int *num;
    num = (int *)malloc(sizeof(int)*N);
    for(i = 0; i < N; i++)
        scanf("%d", &num[i]);
    int Mid, Low = 0, High = N- 1;
        while(Low < High){
              int Mid = (Low + High)/2;
              if(num[Mid] > num[High])
                  Low = Mid+1;
              else
                  High = Mid;
        }
    printf("%d", num[Low]);
    return 0;
}
```

Name:

Roll No: e.g. 170001 **Dept.:** e.g. CHE **Sect.:** e.g. A4

Problem 2 (Fun with Arrays: 6 + 4 + 8 + 4 + 5 + 3 = 30 marks).

1. **Write T or F in the box for True and False respectively**

   1. **F** For a 1D array, `sizeof(a)/sizeof(a[0])` will be equal to the size of array `a`, no matter how the array is declared.
   2. **F** "`int 6[a];`" is a valid declaration statement in C.
   3. **F** Elements of different columns in a 2D array can have different types.
   4. **F** Arrays cannot contain strings(2D character arrays) as elements.
   5. **F** We can declare array size as a negative number.
   6. **T** It is not necessary to typecast the address. returned by `malloc`.

2. **Match the output of column 1 with column 2 in the box provided.**

   You can assume that `sizeof(int*) = 8` and `sizeof(int) = 4` for solving this question. `ERR` stands for compilation error and `GAR` stands for garbage output.

   1. **d**
   ```
   int a[6];
   printf("%d", sizeof(a)/sizeof(a[0]);
   ```
   a. 2

   2. **a**
   ```
   int *b = (int *) malloc(6*sizeof(int));
   printf("%d", sizeof(b)/sizeof(b[0]));
   ```
   b. 0 5

   3. **d**
   ```
   int a[6];
   a = {1, 2, 3, 4, 5, 6};
   printf("%d", a[4]);
   ```
   c. 6

   4. **d**
   ```
   int a[] = {0, 1, 2, 3, 4};
   int b[] = {5, 6, 7, 8, 9};
   int *temp;
   temp = a;
   a = b;
   b = temp;
   printf("%d %d", b[0], a[0])
   ```
   d. ERR

   e. 4

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

3. **Fill in the blanks**

Below you can find the skeleton of a function `search` that searches for a number `x` in a row-wise and column-wise sorted 2D array `mat` with dimensions `n x n`.

Row-wise Sorted : $\mathbf{i} < \mathbf{j} \implies \mathbf{A}[\mathbf{R}][\mathbf{i}] < \mathbf{A}[\mathbf{R}][\mathbf{j}]$
Column-wise Sorted: $\mathbf{i} < \mathbf{j} \implies \mathbf{A}[\mathbf{i}][\mathbf{C}] < \mathbf{A}[\mathbf{j}][\mathbf{C}]$

(Hint: Start from the top-right and move down or left to search through the array)

```
#include <stdio.h>

int search(int **mat, int n, int x)
{
  int i = (1)___, j = (2)___;  //set indexes for top right element
   while ( i < (3)___ && j >= (4)___ )
   {
     if ( mat[i][j] == (5)___ )
      {
         printf("Found at %d, %d", i, j);
         return 1;
      }
      if ( (6)___ > x )
        (7)___;
      else
        (8)___;
   }

   printf("Element not found");
   return 0;
}
```

1	`0`
2	`n-1`
3	`n`
4	`0`
5	`x`
6	`mat[i][j]`
7	`j--`
8	`i++`

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

4. **Fill in the blanks** Below you can find the skeleton of a function `merge` that takes as input, two 1D sorted arrays (sorted in increasing order) and their respective lengths. You have to fill the code such that the function returns a single 1D sorted array (sorted in increasing order).

```
#include <stdlib.h>

int* merge(int* a, int* b, int len_a, int len_b){
  int *c, i=0, j=0, k=0;
  c = (int*) malloc((1)___ * sizeof(int));
  while (i < (2)___ && j < (3)___){
    if(a[i] (4)___ b[j])
      c[k++] = a[i++];
    else
      c[k++] = b[j++];
  }

  while ((5)___ < len_a)
    c[k++] = a[(6)___];

  while ((7)___ < len_b)
    c[k++] = b[(8)___];

  return c;
}
```

```
1  len_a + len_b

2  len_a

3  len_b

4  <=, <

5  i

6  i++

7  j

8  j++
```

5. **Write output in the box provided**

Please ensure you take care of newline characters in the output by correctly putting answers in different lines. If you think the **code will not compile**, please write `CTE` and nothing else. If you think during the execution of any line, the program **will face an error**, please write `RTE` for that line only, and nothing beyond that. If you think a **garbage value will be printed**, please write `GAR` for every such instance.

```
#include <stdio.h>
int main()
{
  int A[3][3] = {{1, 2, 3},
                 {4, 5, 6},
                 {7, 8, 9}};
  printf("1 %d\n", **(A + 2 + 2)    );
  printf("2 %d\n", **(A + 2) + 2    );
  printf("3 %d\n", *(*A + 2 + 2)    );
  printf("4 %d\n", *(*(A + 2) + 2) );
  printf("5 %d"  , (**A + 2 + 2)    );

  return 0;
}
```

```
1 GAR / RTE
2 9
3 5
4 9
5 5
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

6. **Write output in the box provided**

Please ensure you take care of newline characters in the output by correctly putting answers in different lines. If you think the **code will not compile**, please write CTE and nothing else. If you think during the execution of any line, the program **will face an error**, please write RTE for that line only, and nothing beyond that. If you think a **garbage value will be printed**, please write GAR for every such instance.

```c
#include <stdio.h>
#define R 3
#define C 3

void modifyMatrix(int mat[][C]){
   mat[1][1] = 100;
   mat++;
   mat[1][1] = 200;
}

void printMatrix(int mat[][C]){
    int i, j;
    for (i = 0; i < R; i++){
        for (j = 0; j < C; j++)
            printf("%3d ", mat[i][j]);
        printf("\n");
    }
}

int main(){
  int mat[R][C] = {{1},
                   {2, 3},
                   {4, 5, 6}};
    modifyMatrix(mat);
    printMatrix(mat);

    return 0;
}
```

```
  1   0   0

  2 100   0

  4 200   6
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

Problem 3 (No Strings Attached: 6 + 8 + 4 + 12 = 30 marks).

1. **Write T or F in the box for True and False respectively**

   1. **F** `strcat(str1, str2);` appends str1 at the end of str2.

   2. **F** There is no difference between defining str as `char str[] = {'A','B','C','D','E'};` and `char str[] = "ABCDE";`

   3. **T** It is syntactically correct to write `int a[] = {1,2,3,4,5};` and you need not specify the size of array.

   4. **T** While reading a string using %s option in `scanf`, a null character is placed at the end of the string by scanf.

   5. **F** `gets` stops scanning when it encounters whitespaces but `scanf` can be used to scan string containing whitespaces as well.

   6. **F** `strcasecmp("ESC","Esc")` will return -32 because 'S' - 's' is -32.

2. **You must have studied atoi function that extracts the integers out of strings. Below is an implementation of atoi() in C. Fill in the blanks to complete the code. There are total 8 blanks.**

```
int my_atoi(char *s)
{
    int i,num=0,sign=1;

    for(i=0;s[i]!='\0';i++)
    {
            if(s[i]==' ')
                continue;
            else if(s[i]=='-')
                sign=-1;
            else break;
    }

    for(i=i;s[i]!='\0' && s[i]>='0' && s[i]<='9';i++)
        num = num*10 + s[i]-'0';
    return num*sign;
}
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

3. **Write the output of following program in the box provided. ASCII value is 65 for 'A' and 97 for 'a'**

```
#include <stdio.h>

int main() {
    char str[] = {'F'-3, 111, 108, 100, 'A'*3,
    112, 108, 97, 121, 0};
    str[4] *= str[9];
    str[0] += str[9] + str[4];
    printf("%s\n", str);
    printf("%s\n", str+5);
}
```

```
Cold
play
```

4. A *Super Line* is a line in a file in which the number of vowels is strictly greater than an integer value `m` and the number of characters is strictly less than another integer value `n` (not counting the newline character at the end of the line).
The program given next is a partially filled code which takes names of two files `src` and `dest`, and two integers `m` and `n`, as command line arguments and copies all the Super Lines that are present in `src` into the file `dest`.
For example, if the following command is executed,
`./a.out src dest 2 27`
where `src` is the name of the first file, `dest` is the name of the second file, `m` is 2 and `n` is 27. In this case, all lines in the file `src` with strictly less than `n` characters and strictly greater than `m` vowels are copied to the file `dest`. For example, if the file `src` was as follows:

```
eSC101
1 computing
chemistry
fundamentals of computing esc101

time COMPLEXITY
  Algorithms 2
1 dynamic 2
Miles to go before i sleep
Miles to go before we sleep
```

then the file `dest` should be as follows after the command is executed:

```
1 computing
time COMPLEXITY
  Algorithms 2
Miles to go before i sleep
```

Note: Assume that each line is terminated by the newline character and there are no trailing whitespace characters. Complete the following program to complete the given task. There are a total of 12 blanks.

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

```c
#include <stdio.h>
#include <stdlib.h>

int isVowel(char c){
  if(c>='A' && c<='Z')
    c = c - 'A' + 'a';
  if(c=='a' || c=='e' || c=='i' || c=='o' || c=='u')
    return 1;
  return 0;
}

int main(int argc, char **argv){
  int c1, c2, i, vCount, chCount;
  FILE *f1, *f2;
  char *str, c;
  if(argc!=5)
    printf("Incorrect number of arguments\n");
  else{
    f1 = fopen(argv[1],"r");
    f2 = fopen(argv[2],"w");
    c1 = atoi(argv[3]);
    c2 = atoi(argv[4]);
    str = (char *)malloc(c2*sizeof(char));
    while(feof(f1)==0){
      fscanf(f1, "%c", &c);
      i = chCount = vCount = 0;
      while(c!='\n'){
        if(chCount<c2){
          str[i] = c;
          if(isVowel(c))
            vCount = vCount + 1;
          i = i + 1;
        }
        chCount = chCount + 1;
        fscanf(f1, "%c", &c);
      }
      if(chCount<c2 && vCount>c1){
        for(i=0;i<chCount;i++)
          fprintf(f2, "%c", str[i]);
        fprintf(f2, "%c", c);
      }
    }
    free(str);
    fclose(f1);
    fclose(f2);
  }
  return 0;
}
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

Problem 4 (Pointers: 6+15+2+7 = 30 marks). Give your answers in the space provided only.

1. **True or False: 6 X 1 = 6**
   For each of the following simply write **T** or **F** in the box.

   a. **F** Arithmetic involving a `void*` type varible is legal in C.
   b. **T** Passing an array to a function is exactly same as passing the pointer to the first element of the array.
   c. **F** A pointer is by default initialized to NULL after declaration.
   d. **T** A void pointer is used to handle raw data in C.
   e. **F** Memory leaks can happen in the stack as well as in the heap.
   f. **T** If a variable `c` is of type `char**`, `*(c+100)` will give us a value of type `char*`.

2. **Write output in the space provided: 4+4+7 = 15**
   Please ensure you take care of newline characters in the output by correctly putting answers in different lines. If you think the **code will not compile**, please write `CTE`. If you think during the execution of any line, the program **will face an error**, please write `RTE`. For both **Compile Time Error** and **RunTime Error**, mention which line(s) cause the error(s) and provide a brief explanation why the error(s) occur. If you think a **garbage value will be printed**, please write `GAR` for every such instance.

   (a)

```
1 #include <stdio.h>
2 #include <string.h>
3 int main(){
4   char *str1 = "Hello World!\n";
5   char *str2 = "Hello ESC101!\n";
6   strcpy(str2, str1);
7   printf("%s\n%s", str2, str1);
8   return 0;
9 }
```

RTE
Line 6
String literals are allocated space in read-only memory. `strcpy` cannot overwrite str2. Hence, segmentation fault.

   (b)

```
1 #include <stdio.h>
2 int main(){
3   char a[100] = "Programming is fun!";
4   *(a+=18) = '?';
5   printf("%s\n", a);
6   a++;
7   printf("%s\n", a);
8   return 0;
9 }
```

CTE
Line 4 and 6
Though internally represented as a pointer, array type is not assignable.

Name:

Roll No: e.g. 170001 Dept.: e.g. CHE Sect.: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

(c) Assume the size of the data types as follows:
int - 4 bytes
double - 8 bytes
double* - 8 bytes
int* - 8 bytes
Also assume all typecasts between pointer types are safe on the given platform.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  int (*a)[3];
  a = (int(*)[3])malloc(24); // a
   contains the address 0xa05010 after
    this statement
  double* b = (double*)a;
  for(int i=0; i<2; i++){
    (*(a+i))[0] = i+0;
    (*(a+i))[1] = i+1;
    (*(a+i))[2] = i+2;
  }
  printf("%u %u\n",sizeof(a),sizeof(b)
   );
  printf("%p %p\n", a+1, b+1);
  for(; (int(*)[3])b < (a+2); b++){
    printf("%p %d\n", b, *(int*)b);
  }
  return 0;
}
```

```
8 8
0xa0501c 0xa05018
0xa05010 0
0xa05018 2
0xa05020 2
```

3. **Dangling Pointer: 2 marks**
Briefly explain what a dangling pointer means. Give an example program where dangling pointer is created due to a variable going out of scope.
A pointer that points to memory that has already been freed.

```
#include <stdio.h>
int main(){
  int *a;
  {
    int b = 5;
    a = &b;
  }
  // a is a dangling pointer here.
  return 0;
}
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

4. **URL Encoding: 7 marks**
If you have ever noticed, the urls in your browser's url bar do not contain any spaces. This is because space is considered as an unsafe character. The browser automatically replaces all spaces with '%20' before making any request. This is called URL encoding.
Implement a C function `char * url_encode(char *url, int len)` that takes pointer to a string(url) and the length of the string, replaces all spaces with '%20' and returns the pointer to the new string.

```
char *url_encode(char *url, int len){
  int new_len = len;
  for(int i=0; i<len; i++)
    if(url[i]==' ')
      new_len+=2;

  char *new_url = (char *)malloc(new_len+1);

  for(int i=0, j=0; i<len; i++){
    if(url[i]==' '){
      new_url[j++] = '%';
      new_url[j++] = '2';
      new_url[j++] = '0';
    }
    else new_url[j++] = url[i];
  }
  new_url[new_len] = '\0';
  return new_url;
}
```

Name:					
Roll No: e.g. 170001		**Dept.**: e.g. CHE		**Sect.**: e.g. A4	

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

Problem 5 (The Search is On: 3 + 6 + 4 + 11 + 6 = 30 marks**).**

In a binary search tree (BST), each node has pointers to its `left` child, `right` child and `parent`, any of which could be NULL. Each node also stores an integer `value`, which is the item id. In a BST, a node's left child always has a smaller value than itself, and its right child always has a larger value than itself. Keeping this design in mind, we implement a BST and its search functionality as follows:

```
 1 struct Node {
 2   int value;
 3   struct Node *left, *right;
 4   struct Node *parent;
 5 };
 6
 7 typedef struct Node Node;
 8
 9 Node *head;
10
11 Node *create_new_node(int x) {
12   Node *new = malloc(sizeof(
      Node));
13   new->value = x;
14   new->left = NULL;
15   new->right = NULL;
16   new->parent = NULL;
17   return new;
18 }
```

```
 1 Node *search_node(int x) {
 2   Node *current = head;
 3   while (current != NULL &&
      current->value != x) {
 4     if (current->value > x) {
 5       current = current->left;
 6     } else {
 7       current = current->right;
 8     }
 9   }
10
11   if (current == NULL) {
12     return NULL;
13   }
14
15   if (current->value == x) {
16     return current;
17   }
18 }
```

1. **Write T or F in the box for True and False respectively**

   1. **T** The first line inside `create_new_node`(line 12 on the left) makes use of implicit type conversion.

   2. **F** If `current` is NULL, the condition on the `while` loop inside `search_node` throws an error since `current->value` is invalid access (line 3 on the right).

   3. **F** If the `if` statements on lines 11-13 and lines 15-17 are swapped, `search_node`'s functionality would remain unaltered.

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

2. As can be seen, the function for searching a value makes vital use of the property of a BST that everything smaller than the current value lies to the left and everything larger than the current value lies to the right. Now, we need to make use of this property and take help from the code for searching to complete the function for inserting a node into the BST by filling the blanks. Insertion of a node happens at one of the leaf nodes of the tree, and proceeds by searching for the most appropriate leaf or spot to insert at, starting from the root (or head). At every step, we move either left or right, until we reach the appropriate leaf. While at any node during insertion, if the current value is larger, go left. If left is NULL, you've found the spot for insertion, so adjust pointers accordingly. Similarly proceed for the right if necessary.

```
void insert_node(Node *ptr, Node *current) {
  if (current == NULL) {
    // if there is no node in the tree
    head = ptr;
    return;
  }

  if (current->value > ptr->value) {
    // new node (ptr) needs to go to the left
    if (current->left != NULL) {
      // recursively call insert_node
      insert_node(ptr, current->left);
    } else {
      // insert this node at this spot
      current->left = ptr;
      ptr->parent = current;
      return;
    }
  } else {
    // ptr needs to go to the right
    if (current->right != NULL) {
      // recursively call insert_node
      insert_node(ptr, current->right);
    } else {
      // insert the node at this spot
      current->right = ptr;
      ptr->parent = current;
      return;
    }
  }
}
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

It can be observed that the time taken by a search or an insert operation depends on the depth of the BST, i.e. the maximum number of nodes on any path from the head to a leaf. Now that we have searching and insertion in place, we need a function to print the values stored in the BST in sorted order. For an analysis of how much time each of the search / insert operations take, we also need to find out the depth of the tree, which we can do by creating an `inorder` traversal function as shown below.

```
int inorder(Node *current, int depth) {
  if (current == NULL) {
    return depth-1;
  }

  int ldepth = inorder(current->left, depth+1);
  printf("%d ", current->value);
  int rdepth = inorder(current->right, depth+1);
  return ldepth > rdepth ? ldepth : rdepth ;
}
```

3. Find out the output for two instantiations of the tree and subsequent inorder traversals as follows:

```
void make_tree(ids[5]) {
  for (int i=0; i<5; i++) {
    Node *new = create_new_node(ids[i]);
    insert_node(new, head);
  }

  int depth = inorder(head, 1);
  printf("\ndepth: %d\n", depth);
}

int ids[5] = {5, 3, 7, 1, 4};
make_tree(ids);
head = NULL;
int ids2[5] = {1, 2, 3, 4, 5};
make_tree(ids2);
```

```
1 3 4 5 7
depth: 3
1 2 3 4 5
depth: 5
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

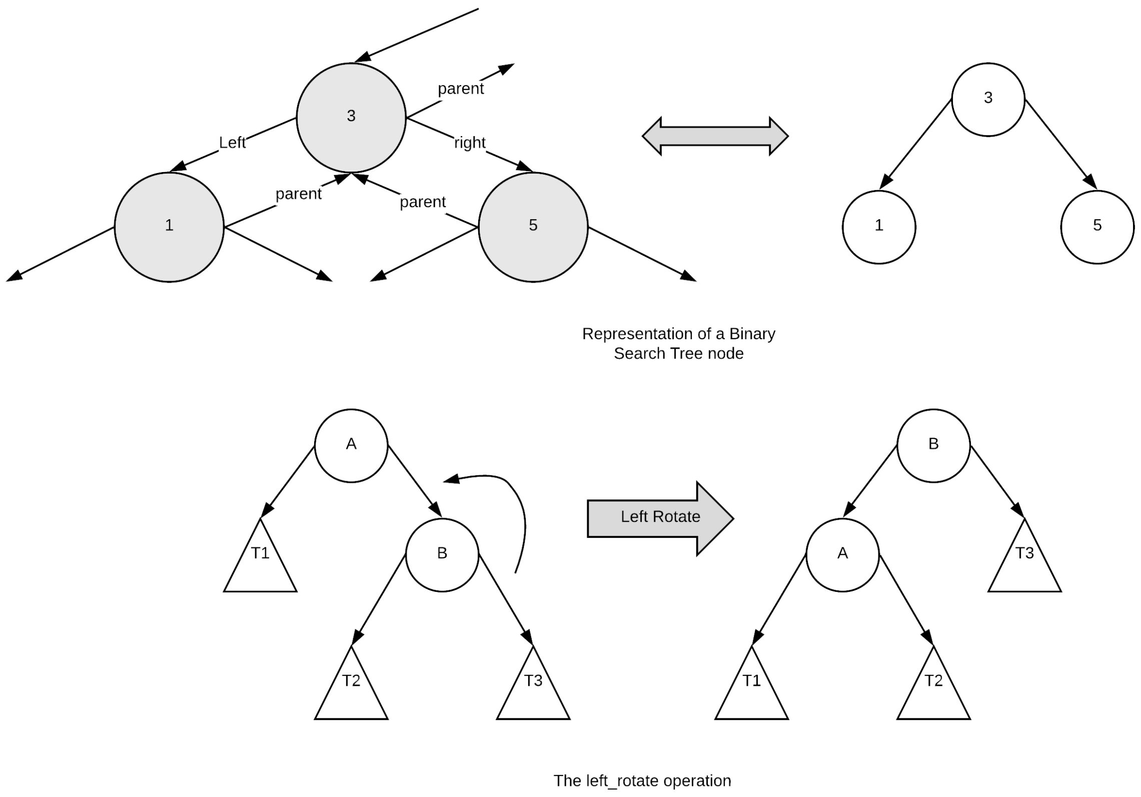

parent
3
Left
right
parent
parent
1
5
3
1
5
Representation of a Binary Search Tree node
A
T1
B
T2
T3
Left Rotate
B
A
T3
T1
T2
The left_rotate operation

Figure 1: Representation of a binary search tree node (above), and illustration of the left_rotate operation (below).

4. An image of how a binary search tree node, as defined above, can be visualized is as shown in Fig 1. What it also shows is an operation called rotation, which is very widely used for balancing skewed binary search trees. I'll describe the operation here for further clarity:

   We say that a node A has a left child and a right child, which could be nodes B and C respectively. Alternatively, we can also say that the node A's left child is another BST with its root at B, and similarly for its right child. Thus, we are defining the binary search tree in a recursive manner, saying that a BST has a root node, which has two children, both of them being BSTs. With this view in mind, we look at the BST shown at the bottom left of Fig 1, with node A as the root and a subtree rooted at B as its right child. $T1$ is the subtree that makes up its left child, and B's children are named as $T2$ and $T3$.

   The operation `left_rotate` does the following: it changes the parental relationship between A and B, without violating any property of a BST. After a `left_rotate`, A becomes B's child, instead of being its parent, and subtrees $T1$, $T2$ and $T3$ undergo changes in parentage to maintain the fact that every node can have at most two children and that all the elements to the left of a node are smaller than it and vice versa.

   Notice that $T2$ was the left child of B but in the right of A, thus having all its elements greater than A but smaller than B. This relationship can be seen from the state of the tree after the rotation (bottom right in Fig 1) as well. Your task is to implement the function `left_rotate` by filling in the blanks in the code below:

Name:

Roll No: e.g. 170001 **Dept.:** e.g. CHE **Sect.:** e.g. A4

```c
void left_rotate(Node *A) {
  // A is the node as shown in the figure
  if (A == NULL) {
    return;
  }

  if (A->right == NULL) {
    /* A->right is its child that is soon to become its parent, i.e.
        B in the figure */
    return;
  }

  /* save A's parent and T2:
     they are going to have some pointers altered. */
  Node *par = A->parent, *t2 = A->right->left;

  if (par != NULL) {
    // assign B as the new child to A's parent
    if (par->left == A) {
      par->left = A->right;
    } else {
      par->right = A->right;
    }
    // assign A's parent as the new parent to B
    A->right->parent = par;
  }

  // assign B as the new parent to A
  A->parent = A->right;
  // assign A as the new child of B
  A->right->left = A;  //A->parent->left is also fine here
  // reassign A's right child
  A->right = t2;

  if (t2 != NULL) {
    // reassign T2's parent
    t2->parent = A;
  }

  if (par == NULL) {
    // if A was earlier the root, now B would be the root
    head = A->parent;
  }
}
```

Name:

Roll No:
e.g. 170001

Dept.:
e.g. CHE

Sect.:
e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

5. Finally, find the output of the following piece of code, and observe the change in the depth of the tree after a few `left_rotate` operations. Thus, if used wisely, rotations can reduce the depth of the tree and hence speed up subsequent insert / search operations.

```
int ids[5] = {1, 2, 3, 4, 5};
make_tree(ids);
Node *found = search_node(3);
left_rotate(found);
int depth = inorder(head, 1);
printf("\ndepth: %d\n", depth);

found = search_node(1);
left_rotate(found);
depth = inorder(head, 1);
printf("\ndepth: %d\n", depth);
```

```
1 2 3 4 5
depth: 5
1 2 3 4 5
depth: 4
1 2 3 4 5
depth: 3
```

Name:

Roll No: e.g. 170001 **Dept.**: e.g. CHE **Sect.**: e.g. A4

Problem 6 (To Iterate is Human, to Recurse, Divine: 2 + 12 + 13 + 3 = 30 marks**).** Iteration and recursion are two approaches that you have studied for solving problems. But they have the same expressive power. It means that any recursive solution can be converted to an iterative solution, and vice versa. For example, the recursive implementation of the function **f** on the left is equivalent to the iterative implementation on the right. Here equivalent means that both will return the same answer if called with the same arguments.

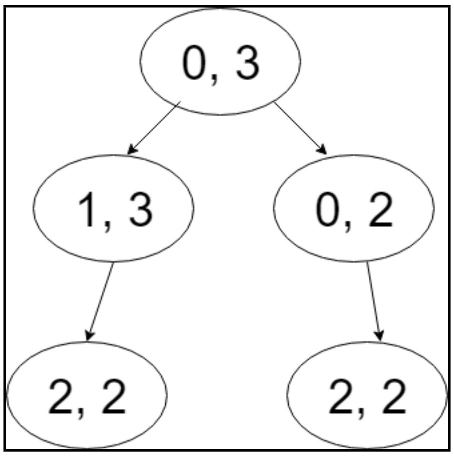
```
int f(int n){
  if (n==0) return 1;
  return f(n/2) + f(n/3) + 1;
}
```

```
int f(int n){
  int arr[n+1];
  arr[0] = 1;
  for(int i=1; i<=n; i++)
    arr[i] = arr[i/2] + arr[i/3] + 1;
  return arr[n];
}
```

Note one of the key differences is that in recursion, the process is top-down; it starts at n and gradually converges to 0. While with iteration, we build the solution bottom-up; starting at 0 and incrementally reaching to n.

1. For the recursive function **g** given below, draw the recursion tree when g is called with arguments (0, 3), and the first 4 elements of *arr* are $1, 2, 3, 1$. You would have seen recursion tree for the Fibonacci function. Write the arguments in the internal nodes of the tree, and edges denote recursive calls to g.

```
int arr[100]; //assume the array is initialized
int g(int l, int r){
  if(l > r) return 0;
  if(l == r) return 1;
  if(arr[l] == arr[r]) return g(l+1, r-1) + 2;
  return max(g(l+1, r), g(l, r-1)); //assume max(a,b) is defined
}
```



Name:

Roll No:
e.g. 170001

Dept.:
e.g. CHE

Sect.:
e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

2. *Divine → Human.* Complete the code for an **iterative** implementation of g. You need to fill total six blanks. Iterative version of g should return the same value as the recursive version.

```
int arr[100]; //assume the array is initialized
int g(int l, int r);  //declaration
//Assume 0 <= l, r < 50

/*Idea- We need to iterate in a way that when we reach (l,r), we have
    the value for (l+1,r-1), (l+1,r) and (l,r-1) ready.
One way to ensure this is first get the value for all (i,i), then all
    (i,i+1), then all (i,i+2), and so on.
ie, first get the value for all size 1 chunks, then all size 2 chunks
    and so on.
*/

int g_iter(int l, int r){
  if(l>r) return 0;
  int result[50][50];
  for(int size=0; size<=r-l; size++){ //iterate over the size of
   chunk
    for(int left=0; left<=r; left++){ //iterate over start of chunk
      int right = left+size; //we are at (left,right)
      if(left == right){
        result[left][right]=1;
      }

      else if(arr[left]==arr[right]){
        result[left][right] = result[left+1][right-1] + 2;
      }

      else{
        result[left][right] = max(result[left+1][right], result[left
   ][right-1]);
      }
    }
  }
  return result[l][r];
}
```

Name:					
Roll No: e.g. 170001		**Dept.**: e.g. CHE		**Sect.**: e.g. A4	

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

3. *Human → Divine*. Defined below is an iterative function **h**, which is similar in structure to the iterative version of the function **f** defined earlier. Your task is to write a **recursive** implementation (code/pseudo-code) for h. You are **not** allowed to create any arrays, but you may create constant number of local variables. You can also create new functions if needed. Recursive version of h should return the same value as the iterative version.

```
int h(int n){
  int a[n+1], b[n+1];
  a[0]=1, b[0] = 2;
  for(int i=1; i<=n; i++){
    a[i] = a[i/2] + b[i/3];
    b[i] = a[i/3] + b[i/2] + i;
  }
  return b[n];
}
```

```
int h(int n);
//write the definition of h in the space below. Statements should be
    unambiguous. Avoid overwriting.

/*
h is like the b array. g is like the a array. This is also called
    mutual recursion when two functions are defined in terms of each
    other.
There is also a solution possible using struct pair.
*/

int h(int n){
  if(n==0)return 2;
  return h(n/2)+g(n/3)+n;
}


int g(int n){
  if(n==0)return 1;
  return h(n/3)+g(n/2);
}
```

Name:

Roll No: e.g. 170001 | Dept.: e.g. CHE | Sect.: e.g. A4

IIT Kanpur
ESC101 Fundam. of Comp.
Endsem Set A
Date: April 26, 2018

4. For the function **f**, we needed an extra array (named *arr*) of size n+1 for the iterative implementation. The recursive version of f did not require explicitly creating any such array. Does this imply that recursion uses lesser memory, and is thus more memory-efficient than iteration in this case? Why or why not?

It is not the case that since you did not declare any array thus recursion uses no (or constant) amount of memory. In the recursive version, memory is allocated for arguments and local variables etc on the stack during recursive calls. Just writing no with a similar explanation will fetch you complete score.
Bonus (+2)- The amount of memory for recursion will be proportional to the maximum depth of recursion. In this case, the maximum depth of recursion will be $O(log_2 n)$, so the memory required will also be $O(log_2 n)$, which is less than the $O(n)$ memory required for creating the array in iteration, and recursion is indeed more memory-efficient (but due to a different reason).