+===+
| Indian Institute of Technology Kharagpur |
Department of Computer Science and Engineering
CS10003 : Programming and Data Structures (Theory)
Autumn 2020	Long Test 3 (ALL Q&A)	Marks: 40
Date: 10-Mar-2021 (Wednesday)	Time: 09:00am-10:15am (75 min)
+===+

==
++++++++++++++++ Pointers, Arrays and Dynamic Memory Allocation ++++++++++++++++

******** Question ********
In the following C code snippet, fill in the blanks so that the program will pri
nt the values as mentioned in the comment section.

```
#include<stdio.h>
int main()
{
  int a[] = {10,5,21,51,3,2,19}, *p;
  p = ____(1)____ ;
  printf("%d\n",*p);    // It will print 10
  p=p+*(p+4);
  printf("%d\n",*p);    // What will be printed here? ____(2)____
  p = ____(3)____ ;
  printf("%d\n",*p);    // It will print 3
  p= ____(4)____ ;
  printf("%d\n",*p);    // It will print 19
  p=p-*(a+1);
  printf("%d\n",*p);    // What will be printed here? ____(5)____
  return 0;
}
```

******** Answer ********
(1) a
(2) 51
(3) p+1
(4) p+2
(5) 5

******** Question ********
What will be the output if the following C program executes:

```
#include<stdio.h>
int main()
{
    int i;
    int a[5] = {1,2,3,4,5}, *p = a;
    for(i=0; i<5; i++,p++) {
        printf("%d %d",a[i],*(a+i));
        printf(" %d %d %d\n",*(i+a),a[i],*p);
    }
    return 0;
}
```

Note 1: Consider each iteration as one blank.
Note 2: There is a single blank space in between the %d symbols under the format

specifier.

******** Answer ********
(1) 1 1 1 1 1
(2) 2 2 2 2 2
(3) 3 3 3 3 3
(4) 4 4 4 4 4
(5) 5 5 5 5 5

******** Question ********
Fill in the blanks of the following program so that on successful execution it p
rints at the terminal the following output:

40 bytes allocated. Storing ints: 0 1 2 3 4 5 6 7 8 9
4000000 bytes more allocated, first 10 ints are: 0 1 2 3 4 5 6 7 8 9

The corresponding C Program from which the above output is supposed to come out
is as follows (assume that, int will occupy 4 bytes):

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  ____(1)____ ;
  int *pb, n;
  pa = (int *)malloc(10 * sizeof *pa);
  if(pa) {
       printf("%zu bytes allocated. Storing ints: ", 10*sizeof(int));
       for(n = 0; n < 10; ++n)
           printf("%d ", pa[n] = n);
  }
  /* void *realloc(void *ptr, size_t size) resizes the memory block
     that was previously allocated by malloc or calloc */
  pb = (int *)realloc(pa, 1000000*sizeof(*pb));
  if( ____(2)____ ) {
       printf("\n%zu more bytes allocated, first 10 ints are: ", 1000000 * ____(
3)____ );
       for(n = 0; n < 10; ++n)
           printf("%d ", pb[n]);
           printf("\n");
           ____(4)____ ;
  }
  ____(5)____ ;
  return 0;
}
```

******** Answer ********
(1) int *pa
(2) pb OR pb!=NULL
(3) sizeof(int)
(4) free(pb)
(5) free(pa)

******** Question ********
Fill in the blanks to complete the following program to print all prime numbers
between 1 and 100.

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

```
int main()
{
  int A[ ____(1)____ ], i, j=2;
  for(i=2; i<=100; i++)
    A[i]=i;
  while(1)
  {
    for( ; A[j]==0 && j<=100; j++);
    if( ____(2)____ )
      break;
    for(i=2; i*j<=100; i++)
      ____(3)____ ;
    j++;
  }
  for( ____(4)____; i<=100; ____(5)____ )
    if(A[i]!=0)
      printf("%d ", i);
  return 0;
}
```

******** Answer ********
(1) 101 (something bigger than 101 also correct; the size of A must be at least 101)
(2) j==101 (j>=101 also correct)
(3) A[i*j]=0
(4) i=2
(5) i++

******** Question ********
Fill in the blanks to complete the following program that applies a clockwise cyclic shift of 's' positions on an array of 10 integers.

```
#include<stdio.h>
int main()
{
  int A[10], i, j, s, t;
  for(i=0; i<10; i++)
  {
    printf("Enter element %d: ", i+1);
    scanf("%d", &A[i]);
  }
  printf("Enter shift (between 0 and 9): ");
  scanf("%d", &s);
  for(i=1; ____(1)____ ; i++)
  {
     t=A[9];
     for( ____(2)____ ; ____(3)____ ; ____(4)____ )
       A[j]=A[j-1];
     ____(5)____ ;
  }
  for(i=0; i<10; i++)
    printf("%d ", A[i]);
  return 0;
}
```

******** Answer ********
(1) i<=s
(2) j=9
(3) j>=1

(4) j--
(5) A[0]=t

******** Question ********
Fill in the blanks to complete the following program that calculates the sum as well as sum of the squares of the array elements.

```
#include<stdio.h>

void f(float A[], float *x, float *y)
{
  float s=0, ss=0;
  int i;
  for(i=0; i<10; i++)
  {
    s+=A[i];
    ss += ____(1)____ ;
  }
  ____(2)____ ;
  ____(3)____ ;
}

int main()
{
  int i;
  float A[10], sum, SumOfSquares;
  for(i=0; i<10; i++)
  {
    printf("Enter element %d: ", i+1);
    scanf("%f", &A[i]);
  }
  f(A, ____(4)____ , ____(5)____ );
  printf("The sum of the elements of A is %f", sum);
  printf("\nThe sum of the squares of the elements of A is %f", SumOfSquares);
  return 0;
}
```

******** Answer ********
(1) A[i]*A[i]
(2) *x=s
(3) *y=ss
(4) &sum
(5) &SumOfSquares

******** Question ********
Consider the problem of sentence splitting where one sentence is broken into multiple words on the basis of a delimiting character. For example, let us consider the following sentence represented as a 1D character array.
str[ ]="A quick brown fox jumps over the lazy dog"
If we consider the delimiting character to be space, the resultant set of words is a 2D character array tokens[ ][ ] = {"A", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"}. Similarly, for str[ ]="A&quick&brown&fox&jumps&over&the&lazy&dog" with delimiter '&', we shall get the same result.

The following function char **split_string(char str[], char delimiter) takes as input a character string str, a character variable delimiter, splits str into a set of words based on the delimiter and returns a 2D character array comprising the resultant words. Note the 2D array is allocated dynamically in an efficien

t manner i.e. each resultant word is allocated with minimum required memory. The function makes use of another function called count_words(char str[] ,char delimiter) which returns the number of resultant words in the sentence subject to the specified delimiter. The function when applied to the above sample sentence with 'q' as delimiter returns 2.

Complete the code/function by filling in the required blanks marked by the numbers.

```
char **split_string(char str[], char delimiter) {
  int i, j;
  int count = count_words(str, delimiter);
  char **tokens = (char **)malloc(sizeof(char*) * count);
  int token_id = 0;
  int word_len = 0;
  for (i = 0; i<=strlen(str); i++) {
      word_len++;
      if (str[i] == ____(1)____ || ____(2)____ ) {
        tokens[token_id] = (char *)malloc(sizeof(char) * ____(3)____ );
        for (int k = 0; k < word_len-1; k++) {
          tokens[token_id][k] = str[ ____(4)____ ];
        }
        tokens[token_id][ ____(5)____ ] = '\0';
        word_len = 0;
        token_id++;
      }
  }
  return tokens;
}
```

******** Answer ********
(1) delimiter
(2) str[i]=='\0'
(3) word_len
(4) i - (word_len-1) + k
(5) word_len-1

******** Question ********
Consider the following binary string pattern matching problem where we have a character array bitstring of size n comprising 0s and 1s only. We have another character array called pattern of size k which comprises 0,1 and the ? character where ? can mean 0 or 1.
Assume that n is a multiple of k. Consider that bitstring is divided into contiguous chunks of length k. The following recursive function count_pattern counts the number of k-size chunks of bitstring that match the contents of pattern. A chunk matches the pattern if all elements of the chunk match the elements of pattern.

For example,
bitstring: 001101000111, pattern: 0??1. Since ?=0/1, the pattern matches with strings {0001, 0011, 0101 and 0111}.
Here n=12 and k=4. The chunks to be investigated are 0011, 0100 and 0111. Out of these, chunks 0011 and 0111 match the required pattern. The function count_pattern will return 2.

```
int count_pattern(char bitstring[], char pattern[])
{
    int n=strlen(bitstring);
    int k=strlen(pattern);
```

```
    if( ____(1)____ )
        return 0;
    int match=0;
    for(int i=0; i<k; i++)
        if( bitstring[i] == ____(2)____ || ____(3)____ )
            match++;

    int c = ( ____(4)____ )? 1 : 0;

    return c + count_pattern( ____(5)____ );
}
```

******** Answer ********
(1) n==0
(2) pattern[i]
(3) pattern[i]=='?'
(4) match==k
(5) bitstring+k, pattern

******** Question ********
Let us consider the window based averaging operation for a 1D array of floating point numbers. Given an input array A of size n and a window of size k, the operation moves the window in strides of k, takes the average of k consecutive numbers in A and populates a reduced array B. For example, if n=6, A = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0} and k = 3, the operation creates the array B ={2.0, 5.0 }. where B[0]=(A[0]+A[1]+A[2])/3 and B[1]=(A[3]+A[4]+A[5])/3. For the same example if k=4, the operation would create the reduced array B=[2.5, 5.5] where B[0]=(A[0]+A[1]+A[2]+A[3])/4 and B[1]=(A[4]+A[5])/2. Note that the averaging windows do not overlap. The following incomplete program performs the required averaging operation. Complete the snippet by filling in the required blanks marked by the numbers.

```
#include<stdio.h>
int main()
{
    int t,n,k;
    // n: number of elements of A, k: window size
    scanf("%d %d",&n,&k);

    //Dynamically allocate A
    float *A = (float*)malloc(n*sizeof(float));
    for(int i=0;i<n;i++)  //scan elements of A
        scanf("%f",&A[i]);

    //Determine number of elements of B
    int size_b= (n%k==0) ? ____(1)____ : ____(2)____ ;

    //Dynamically allocate B
    float *B =(float*)malloc(size_b*sizeof(float));
    int counter = 0;
    for(int i=0;i<n;i+=k)
    {
        float sum=0.0;
        int num_elems = 0;
        // Compute average of a window
        for(int j=0; j<k && ____(3)____ ; j++)
        {
            sum+=A[i+j];
            num_elems++;
```

```
        }

        //Store result
        B[ ____(4)____ ++ ] = ____(5)____ ;
    }
    for(int j=0;j<counter;j++)
        printf("%.2f ",B[j]);
    printf("\n");
}
```

******** Answer ********
(1) n/k
(2) n/k+1
(3) i+j<n
(4) counter
(5) sum/num_elems

******** Question ********
Given an array, the reverseArray function tries to reverse it using pointers. Fill up the blanks in the below code to achieve the functionality.

Example:
Input Array is {-1, 5, -8, -4, 5, 3}
Output is {3, 5, -4, -8, 5, -1}

```
#include <stdio.h>

// Function to swap two memory contents
void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void reverseArray(int array[], int array_size)
{
    // leftPointer pointing at the beginning of the array
    int *leftPointer = ____(1)____ ;

    // rightPointer pointing at end of the array
    int *rightPointer = ____(2)____ ;

    while ( ____(3)____ )
    {
        swap(leftPointer, rightPointer);
        ____(4)____ ;
        ____(5)____ ;
    }
}
```

******** Answer ********
(1) array
(2) array + array_size - 1
(3) leftPointer < rightPointer
(4) leftPointer++
(5) rightPointer--

******** Question ********
Suppose you have a binary string str and two integers x>0 and y>0. The following code intends to create a new string finalString in such a way that '0' comes x times, then '1' comes y times, and so on until one of the '0' or '1' is finished. Then concatenate the rest of the string at the end and print the final string. Complete the following code to achieve the objective.

Example:
Input-> str = '00110100111010', x = 2, y = 3
Output-> Final String: 00111001110010

Input-> str = '00110100111010', x = 1, y = 2
Output-> Final String: 01101101101000

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

void arrangeString(char str[], int x, int y) {
    int count_0 = 0, count_1 = 0;
    char finalString[100] = "";

    // Count the number of 0's and 1's in the given string.
    for (int i = 0; i < strlen(str); i++) {
        if (str[i] == '0')
            count_0++;
        else
            count_1++;
    }

    // First concatenate all 0's x-times and decrement count of 0's x-times.
    // Then do the similar task with '1'.
    while ( ____(1)____ ) {
        for (int j = 0; ____(2)____ ; j++) {
            if (count_0 > 0) {
                ____(3)____ ;
                count_0 --;
            }
        }
        for (int j = 0; ____(4)____; j++) {
            if (count_1 > 0) {
                ____(5)____ ;
                count_1--;
            }
        }
    }
    printf("Final String: %s\n", finalString);
}

int main() {
    char str[100];
    int x,y;
    printf("Enter the binary string: ");
    scanf("%s", str);
    printf("Enter x and y: ");
    scanf("%d %d", &x, &y);
    arrangeString(str, x, y);
    return 0;
}
```

******** Answer ********

```
(1) count_0 > 0 || count_1 > 0
    [multiple solutions possible as long as the condition is satisfied]
(2) j < x && count_0 > 0
    [multiple solutions possible as long as the condition is satisfied]
(3) strcat(finalString, "0")
(4) j < y && count_1 > 0
    [multiple solutions possible as long as the condition is satisfied]
(5) strcat(finalString, "1")


******** Question ********
Complete the following C program that checks if an array is a palindrome or not
using pointers.

#include<stdio.h>

int main() {
    int n;
    printf("Enter no of elements: ");
    scanf("%d", &n);
    int *arr = (int *)malloc(n * sizeof(int));
    for (int i = 0; i <  n; i++) {
        scanf("%d", (arr+i));
    }

    int *start, *end;
    start = arr;
    end = ____(1)____ ;        //points to last element of array

    while( ____(2)____ ) {
        if( ____(3)____ ) {                //compares the integers
            printf("\nNot a Palindrome");
            return 0;
        }
        if( ____(4)____ ) {                //condition to check for an even pali
ndrome
            printf("An Even Palindrome");
            return 0;
        }
        start ++;
        end --;
    }
    if( ____(5)____ )        //condition to check for an odd palindrome
        printf("An Odd Palindrome");
}

******** Answer ********
(1) arr + (n)-1
(2) start!=end
(3) *start!=*end
(4) end==start+1
(5) start==end


================================================================================
++++++++++++++++ Sorting, Searching and Time Complexity Analysis ++++++++++++++++


******** Question ********
Following C program reads a list of first names (terminated by END string), and
```

sort them in dictionary order.

Example Input:
Amit
Abir
Kamal
Zarina
Mandira
END

Example Output:
Abir
Amit
Kamal
Mandira
Zarina

Fill in the blanks to complete the following program so that it can perform the above mentioned activity.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char nameList[100][100],name[100];
  int i,j,n;

  printf("Enter the firstname (terminate by END)\n");
  n = -1;
  do {
    scanf("%s",name);
    n++;
    strcpy(nameList[n],name);
  } while( ____(1)____ );
  for(i=0; i<n; i++) {
    for( ____(2)____ ) {
       if( ____(3)____ ) {
           strcpy(name,nameList[i]);
           ____(4)____ ;
           ____(5)____ ;
       }
    }
  }
  for(i=0; i<n; i++) {
    printf("%s\n",nameList[i]);
  }

  return 0;
}
```

******** Answer ********
(1) strcmp(name,"END")
(2) j=i+1; j<n; j++
(3) strcmp(nameList[i],nameList[j])>0
(4) strcpy(nameList[i],nameList[j])
(5) strcpy(nameList[j],name)

******** Question ********
A user defined structure stores the information of student record -- name and CG

PA. Write a program to read the information of N such students and print the nam
e of the students based on the decreasing order of their CGPA.

Example Input:
3
Jyoti
8.9
Kaberi
9.1
Gurunath
8.8

Example output:
Kaberi
Jyoti
Gurunath

Fill in the blanks to complete the following C program so that it can perform th
e above mentioned function.

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

struct STUD {
  char name[30];
  float CGPA;
};

int main()
{
  struct STUD stList[100],temp;
  char name[30];
  float cgpa;
  int i,j,n;

  printf("Enter the list of students\n");
  scanf("%d",&n);
  printf("Enter the name of the student and his/her CGPA\n");
  for(i=0; i<n; i++) {
    scanf("%s %f",name,&cgpa);
    ____(1)____ ;
    ____(2)____ ;
  }
  for(i=0; i<n; i++) {
    for(j=i+1; j<n; j++) {
       if( ____(3)____ ) {
           temp = stList[i];
           ____(4)____ ;
           ____(5)____ ;
       }
    }
  }
  for(i=0; i<n; i++) {
    printf("%s %3.1f\n",stList[i].name,stList[i].CGPA);
  }

  return 0;
}
```

******** Answer ********

(1) strcpy(stList[i].name,name)
(2) stList[i].CGPA=cgpa
(3) stList[i].CGPA<stList[j].CGPA
(4) stList[i]=stList[j]
(5) stList[j]=temp

******** Question ********
Fill in the blanks of the following C program so that it reads a list of integer elements and prints the integer element that occurs most frequently. In case of multiple elements with the same highest frequency the program will print only one.

```
#include <stdio.h>

int main()
{
  int list[100],n,i,j,maxFreq,maxFreqE,freq,elem;

  printf("Enter the number of elements: ");
  scanf("%d",&n);
  printf("Enter the elements: ");
  for(i=0; i<n; i++) {
    scanf("%d",&list[i]);
  }
  maxFreqE = list[0];
  maxFreq = 1;
  for(i=0; i<n; i++) {
    elem = list[i];
     ____(1)____ ;
    for(j=i+1; j<n; j++) {
      if( ____(2)____ )
         ____(3)____ ;
    }
    if(freq > maxFreq) {
      ____(4)____ ;
      ____(5)____ ;
    }
  }

  printf("%d occurs maximum %d times!!\n",maxFreqE,maxFreq);
  return 0;
}
```

******** Answer ********
(1) freq=1
(2) list[i]==list[j]
(3) freq++
(4) maxFreqE=elem
(5) maxFreq=freq

******** Question ********
Write whether the following are true or false.
(1) 2n^3 + 3n + 2 = O(n^3)
(2) 3n^3 = Ω(n^4)
(3) log n = θ(n)
(4) 2n^2 = O(n)
(5) 3n^4 - 4n^2 = Ω(n^3)

******** Answer ********
(1) True
(2) False
(3) False
(4) False
(5) True

******** Question ********
Complete the following program to arrange student records in alphabetical order of their names by quicksort. Fill in the given blanks.

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

struct student
{
  char name[50];
  int ID;
};

void exchange(struct student list[], int i, int j)
{
  int t;
  char temp[50];
  strcpy(temp, list[i].name);
  strcpy(list[i].name, list[j].name);
  strcpy(list[j].name, temp);
  t = list[i].ID;
  list[i].ID = list[j].ID;
  list[j].ID = t;
}

int partition (struct student list[], int left, int right)
{
  int i=left, j=right;
  char s[50], temp[50];
  strcpy(s, list[left].name);
  while(i<j)
  {
    for( ; strcmp(list[i].name, s)<=0 && i<=right; i++);
    for( ; strcmp(list[j].name, s)>0 && j>=left; j--);
    if(i<j)
      ____(1)____ ;
  }
  exchange(list, left, j);
  return ____(2)____ ;
}

void quicksort(struct student list[], int left, int right)
{
  int pivot_index;
  if(left>=right)
    ____(3)____ ;
  pivot_index = partition(list, left, right); /*partitions the array with the le
ftmost element as the  pivot.*/
  quicksort( ____(4)____ );
  quicksort( ____(5)____ );
}
```

```
int main()
{
  int i;
  struct student list[10];
  for(i=0; i<10; i++)
  {
    printf("Enter name and ID: ");
    scanf("%s %d",list[i].name, &list[i].ID);
  }
  quicksort(list, 0, 9);
  for(i=0; i<10; i++)
    printf("%s  %d\n", list[i].name, list[i].ID);
  return 0;
}
```

```
******** Answer ********
(1) exchange(list, i, j)
(2) j
(3) return
(4) list, left, pivot_index-1 (list, left, pivot_index also correct)
(5) list, pivot_index+1, right (list, pivot_index, right also correct)
```

```
******** Question ********
Fill in the blanks and complete the following program to search an element in a
sorted array using the ternary search algorithm. Ternary search is similar to bi
nary search. The difference is that we split the array into 3 nearly equal parts
 and recurse into one of them.
```

```
#include<stdio.h>

int ternary_search(int A[], int k, int left, int right)
{
  int d, t1, t2;
  if(left > right)
    return ____(1)____ ;
  if(left == right)
    return ____(2)____ ;
  if(right == left+1)
    return ____(3)____ ;
  d = (right-left+1)/3;
  t1 = left+d-1;
  t2 = t1+d-1;
  if(k <= A[t1])
    return ternary_search(A, k, left, t1);
  if(k <= A[t2])
    return ternary_search( ____(4)____ );
  return ternary_search(A, k, t2+1, right);
}

int main()
{
  int A[10], i, k;
  printf("Enter 10 elements in non-decreasing order\n");
  for(i=0; i<10; i++)
  {
    printf("Enter element %d: ", i+1);
    scanf("%d",&A[i]);
  }
  printf("Enter the search key: ");
```

```
  scanf("%d",&k);
  ternary_search( ____(5)____ )? printf("Present") : printf("Absent");
  return 0;
}
```

******** Answer ********
(1) 0
(2) (A[left]==k)
(3) (A[left]==k) || (A[right]==k)
(4) A, k, t1+1, t2. If somebody writes t1 instead of t1+1 and/or t2+1 instead of t2, give full marks.
(5) A, k, 0, 9

******** Question ********
Let us consider a function 'print_Missing_Repeated_Elements' that takes an unsorted integer array (arr) and the size of the array (n) as input. It is given that array elements are in the range from 1 to n and one number from the set {1, 2, …n} is missing and one number occurs twice in the array. The function prints these two elements. This can be done by sorting the input array and then traversing the array to check for missing and repeating elements. The time complexity for such an approach will be ____(1)____ . (give the time complexity in Big-Oh, O-notation)

The given incomplete version of the 'print_Missing_Repeated_Elements' function does not sort the input array, instead it traverses the array element wise. While traversing, the function uses the absolute value of every element as an index and makes the value at this index as negative to mark it as visited. For this you can use the abs() function in the code snippet where abs(n) returns n if n is >=0 and -1*n if n < 0.

If it finds something is already negative then that element is the repeating element. For example,
Input: arr[] = {4, 3, 6, 2, 1, 1}
Iteration for finding repetition:
Iteration 0: arr[0]= 4, value at the 4th position of the array i.e. arr[3]=2 will be updated as A[3]= -2.
Iteration 4: arr[4]= 1, value at the 1st position of the array i.e. A[0]=4 will be updated as A[0]= -4.
Iteration 5: arr[5]= 1, value at the 1st position of the array i.e. A[0]= -4, is found to be already marked as a negative value. Hence arr[5]=1 is the repeating element.

To find the missing element, traverse the array again. The index of the element with positive value is the missing element.
The time complexity for the given function will be ____(2)____ . (give the time complexity in Big-Oh, O-notation)

```
void print_Missing_Repeated_Elements(int arr[], int n)
{
  int i;
  /*absolute value of each element in the array is taken as an index and the value at that index is checked if it negative or positive*/
  for (i = 0; i < n; i++) {
     if ( ____(3)____ > 0)  // if the value is positive, negate the value
         arr[abs(arr[i]) - 1] = ____(4)____ ;
     else  // if the value is negative, print the repeating term
         printf("The repeating element is %d \n", abs(arr[i]));
   }
   /* traverse the array and the index of the element with positive value is the
```

```
 missing element*/
   for (i = 0; i <n; i++) {
       if ( ____(5)____ > 0)
           printf("The missing element is %d \n", i + 1);
    }
}
```

******** Answer ********
(1) O(nlogn)
(2) O(n)
(3) arr[abs(arr[i]) - 1]
(4) -arr[abs(arr[i]) - 1]
(5) arr[i]

******** Question ********
Let us consider a function 'merge' that takes an m × n 2D integer array (A) as input where each row of the 2D array A is a sorted 1D array of size n. It is given that each row of A does not contain repetition(s) of integers, that is, the n integers in each sorted list are distinct from one another. However, the integers may be repeated in different rows. The program merges the m sorted 1D arrays and stores the merged list in a sorted 1D array (B). During the merging step, all repeated elements of A are removed while storing in B. Finally, the function returns the size of the array B. For example,

A[0] = [1,4,5,6]
A[1] = [1,2,3,4]
A[2] = [3,5,6,7]
A[3] = [2,3,6,9]

B =[1,2,3,4,5,6,7,9]
Function returns 8 as size of B. Note, in the above example elements 1,2,3,4,5,6 are repeated across different rows of A and repetitions are removed while storing to B.

Complete the following 'merge' function that will operate as described above.

```
#define INFINITY 123456789
#define MAX 100

int merge ( int B[], int A[][MAX], int m, int n )
{
  int index[MAX], i, k, min;
  /*index[i] will point to the column with minimum value for i-th row */
  for (i=0; i< m ; ++i)
    index[i] = 0 ; // initialize the index array with 0

  k = 0 ;  // index for iterating array B

  while (1)
  {
   /* find the minimum element among the minimum of each of the row */
    min = INFINITY;
    for ( i=0; i<m; ++i )
   {
     if ( (index[i] < ____(1)____ ) && ( ____(2)____ < min) )
        min = A[i][index[i]] ;
   }
   if (min == INFINITY)
     return ____(3)____ ;
```

```
   // populate the array B with the min value
   ____(4)____ = min;
   /* increment the index for those rows where the value at current index is equ
al to min value */
   for ( i=0; i<m; ++i )
   {
      if ( (index[i] < n) && ( ____(5)____ ) )
        ++index[i] ;
   }
 }
}
```

```
******** Answer ********
(1) n
(2) A[i][index[i]]
(3) k
(4) B[k++]
(5) A[i][index[i]] == min
```

```
******** Question ********
Let us consider a function that takes an unsorted integer array A and two intege
rs n and K as inputs.  Here n is the size of the array A. The function will prin
t two elements in the array such that their sum is equal to the given integer K.
 If such a pair is not found, it will print nothing. For example,
Input: A[] = {0, -1, 2, -3, 1}
       K = -2
Output: -3, 1

This can be solved by iterating over each element in A, and checking whether the
re is any pair of elements (by inspecting all possible pairs) whose sum is K by
using two nested 'for-loops'.  The time complexity for such an approach will be
____(1)____ . (give the time complexity in Big-Oh, O-notation)

The incomplete version of the 'search' function given below does not use any for
 loop. Instead it uses a two-pointer technique. It is done by sorting the array
using a function quickSort() first. Once the array is sorted, the two pointers a
re taken which mark the beginning (low)  and end (high) of the array respectivel
y. If the sum is less than the sum of those two elements at the two pointers, sh
ift the left pointer to increase the value of the required sum and if the sum is
 greater than the required value, shift the right pointer to decrease the value.

For example,
A[] = {-3, -1, 0, 1, 2}
low=0, high=4, sum=-3+2= -1 > -2 So, high=3
low=0, high=3, sum=-3+1= -2 = K => Output: -3,1

The time complexity for such an approach is ____(2)____ . (give the time complex
ity in Big-Oh, O-notation)

void search (int A[], int n, int K)
{
   int low=0, high=n-1, sum;
   quick_sort(A,low,high);  // sort the array A

   /*check the sum of the two elements at the two pointers low and high and incr
ement or decrement the pointers after comparing the sum with K */
   while( ____(3)____ )
   {
     sum = ____(4)____ ;
```

```
     if(sum == K)
     {
       printf("Pair found: %d + %d = %d", A[low],A[high],K);
       return;
     }
     else if( ____(5)____ )
       low++;
     else
       high--;
   }
   return;
}
```

******** Answer ********
(1) O(n2)
(2) O(nlogn)
(3) low<high
(4) A[low]+A[high]
(5) sum<K

******** Question ********
You are given an array A[ ] consisting of integers. Your task is to sort the arr ay such that all even numbers are put first and then odd numbers. You do not nee d to keep the even and odd numbers in order.
Example:
Input = {12, 34, 45, 9, 8, 90, 3}
Output = {12, 34, 90, 8, 9, 45, 3}

```
void segregateOddEven (int A[ ], int n)
{
/* We initialize two variables left and right where left = 0 and right = n - 1.
Keep incrementing the left index until you see an odd number and similarly, keep
 decrementing the right index until you see an even number. If left < right, swa
p A[left] and A[right] */

    int left = 0; right = n - 1;
    while (left < right) {
        while ( ____(1)____ && left < right)
             left++;
        while ( ____(2)____ && left < right)
             right--;
        if (left < right) {
             int temp;
             temp = ____(3)____ ;
             A[left] = ____(4)____ ;
             A[right] = ____(5)____ ;
             left++;
             right--;
        }
    }
}
```

******** Answer ********
(1) (A[left] % 2) == 0   OR   (A[left] & 1) == 0
(2) (A[right] % 2) == 1   OR   (A[right] & 1) == 1
(3) A[left]
(4) A[right]
(5) temp

******** Question ********
Let A and B be two sorted arrays. Array A is of size m+n but contains only m elements whereas array B is of size n and contains n elements. The task is to merge these two arrays into the first array of size m+n such that output is sorted.
(Hint: The function xsort() starts filling the destination array A from the back with the largest elements. As a result, it finally ends up as a merged and sorted array)

```
void xsort(int A[], int m, int B[], int n)
{
    int count = m;
    int i = n-1, j = ____(1)____ , k = m-1;
    for(; k>=0; k--){
        if(B[i] > A[j] || j < 0){
            ____(2)____;
            i--;
            if( ____(3)____ )
                break;
        }
        else{
            ____(4)____ ;
            j--;
        }
    }
}
```

The running time of the above function xsort() is proportional to ____(5)____

******** Answer ********
(1) count - 1
(2) A[k] = B[i]
(3) i < 0
(4) A[k] = A[j]
(5) m+n   [Alternate Answer: O(m+n)]

******** Question ********
Compute the time complexities for the below 5 questions

[a] The worst case time complexity in terms of O (Big O) notation of the below code is ____(1)____ .

```
// search an element in an array
// list is already sorted
function search (list, item, start, end) {
    if (start > end) {
        return false;
    }
    const mid =floor((start + end) / 2);
    if (list[mid] < item) {
        return search(list, item, mid + 1, end);
    }
    if (list[mid] > item) {
        return search(list, item, start, mid - 1);
    }
    return true;
}
```

[b] The worst case time complexity in terms of O (Big O) notation of the followi

ng algorithm is ____(2)____ .

```
function A(n)
{
    If (n <= 2)
       return(1);
    else
       return A(ceil(n));
}
```

[c] An unordered list contains n distinct elements. The number of comparisons to find any one element in this list that is neither maximum nor minimum in terms of O (Big O) notation is ____(3)____ .

[d] The worst case time complexity in terms of O (Big O) notation of function fun is ____(4)____ .

```
int fun(int n)
{
  int count = 0;
  for (int i = 0; i < n; i++)
     for (int j = i; j > 0; j--)
        count = count + 1;
  return count;
}
```

[e] The worst case time complexity in terms of O (Big O) notationof the below code fragment/snippet is ____(5)____ .

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n / 2;
    }
}
```

******** Answer ********
(1) O(n log n)
(2) O(log log n)
(3) O(1)
(4) O(n^2)
(5) O(n log n)

==========
++++++++++ Structures and File Handling ++++++++++

******** Question ********
An input text file (input.txt) contains a list of integer elements. Read those elements and write them back to another file (output.txt) after removing the duplicate elements.

Fill in the blanks so that the following C program can perform the above-mentioned function. Note that, here the function, int feof(FILE *fp), is used to test the end-of-file indicator for file pointer fp.

```
#include <stdio.h>

int main()
{
 FILE *fpr,*fpw;
 int list[1000],n,i,j;
```

```
 fpr=fopen("input.txt","r");
 n=0;
 while( !feof(fpr) ) {
   ____(1)____ ;
   n++;
 }
 fclose(fpr);
 for(i=0; i<n; i++) {
   for(j=i+1; j<n; j++) {
     if( ____(2)____ ) {
        list[j] = -32768;
     }
   }
 }
 fpw = ____(3)____ ;
 for(i=0; i<n-1; i++) {
   if(list[i] == -32768) continue;
   ____(4)____ ;
 }
 ____(5)____ ;
 return 0;
}
```

```
******** Answer ********
(1) fscanf(fpr,"%d",&list[n])
(2) list[i]==list[j]
(3) fopen("output.txt","w")
(4) fprintf(fpw,"%d ",list[i])
(5) fclose(fpw)
```

******** Question ********

A user defined structure stores the information of student record -- name, roll, age and CGPA. Write a program to read the information of N such students and write them in a file (studRec.txt) as per the following order

Example Input:

```
3
Rohit
3
21
8.9
Ayush
2
20
9.1
Prashant
1
21
8.8
```

Example content of studRec.txt

```
3,Rohit,21,8.9
2,Ayush,20,9.1
1,Prashant,1,21,8.8
```

Fill in the blanks so that the following C program can perform the above-mentioned function.

NOTE: Two marks for blank 3. One mark each for blanks 1, 2, and 4.

```
#include <stdio.h>

struct SRECORD {
  char name[30];
  int roll,age;
  float cgpa;
};

int main()
{
 struct SRECORD sr[100];
 ____(1)____ ;
 int n,i;

 printf("Enter the number of students: ");
 scanf("%d",&n);

 printf("Enter the student details: \n");
 for(i=0; i<n; i++) {
   scanf("%s %d %d %f",sr[i].name,&sr[i].roll,&sr[i].age,&sr[i].cgpa);
 }
 fpw = ____(2)____ ;
 for(i=0; i<n; i++) {
   ____(3)____ ;
 }
 ____(4)____ ;
 return 0;
}

******** Answer ********
(1) FILE *fpw
(2) fopen("studRec.txt","w")
(3) fprintf(fpw,"%d,%s,%d,%f\n",sr[i].roll,sr[i].name,sr[i].age,sr[i].cgpa)
(4) fclose(fpw)


******** Question ********
Write a C program that will read comma separated elements (elements may be of an
y datatype) from an input file (csFile.txt) and write back to another file (tsFi
le.txt) where comma will be replaced by tab ('\t').

Fill in the blanks so that the following C program can perform the above-mention
ed function. Note that, here the function, int feof(FILE *fp), is used to test t
he end-of-file indicator for file pointer fp.

#include <stdio.h>

int main()
{
 FILE *fpr,*fpw;
 char ch;
 fpr = ____(1)____ ;
 fpw = ____(2)____ ;
 while(!feof(fpr)) {
   if( feof(fpr) ) {
     break;
   }
   ____(3)____ ;
   if(ch == ',') {
     ____(4)____;
```

```
   } else {
     ____(5)____ ;
   }
 }
 fclose(fpr);
 fclose(fpw);
 return 0;
}
```

```
******** Answer ********
(1) fopen("csFile.txt","r")
(2) fopen("tsFile.txt","w")
(3) fscanf(fpr,"%c",&ch)  OR   fputc function
(4) fprintf(fpw,'\t')    OR fputc function
(5) fprintf(fpw,"%c",ch)    OR  fputc function
```

```
******** Question ********
Fill in the blanks to complete the following program that creates a new file nam
ed records.txt in the current directory and writes records of 5 persons in it. T
he contents of the file after a sample run of the program is given below.

#include<stdio.h>

int main()
{
  ____(1)____ ;
  char s[50], t[50];
  FILE *fp;
  ____(2)____ = ____(3)____ ;
  for(i=0; i<5; i++)
  {
    printf("Enter name: ");
    scanf(" %s", s);
    printf("Enter age: ");
    scanf("%d", &age);
    printf("Enter ID: ");
    scanf(" %s", t);
    fprintf( ____(4)____ );
  }
  ____(5)____ ;
  return 0;
}

Content of "records.txt" file after a sample run:

Somesh 30 GTH778
Girish 56 HWK830
Swapna 66 VAH089
Benny 40 POA295
Fatima 45 GBY654
```

```
******** Answer ********
(1 ) int age, i
(2) fp
(3) fopen("records.txt", "w")
(4) fp, "%s %d %s\n", s, age, t
(5) fclose(fp)
```

******** Question ********
Complete the following program that creates a dynamic list of records of a variable number of persons, each person having a name, which is a sting of variable length, and an age, which is an integer. Use dynamic memory allocation when the data has a variable length. Fill in the following blanks.

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

struct info
{
  char *name;
  int age;
};

int main()
{
  int i, k, n;
  struct info *list;
  char s[100];;
  printf("Enter number of records: ");
  scanf(" %d", &n);
  ____(1)____ = ____(2)____ ;
  for(i=0; i<n; i++)
  {
    printf("Enter name of person %d: ", i+1);
    scanf("%s",s);
    k = strlen(s);
    ____(3)____ = ____(4)____ ;
    ____(5)____ ;
    printf("Enter age of person %d: ", i+1);
    scanf("%d", &list[i].age);
  }
  return 0;
}
```

******** Answer ********
(1) list
(2) (struct info *)malloc(n*sizeof(struct info))
(3) list[i].name
(4) (char *)malloc(k+1) or list[i].name=(char *)malloc((k+1)*sizeof(char)).
(5) strcpy(list[i].name, s)

******** Question ********
Complete the following program to first write the names and ages of 5 persons in a new file named information.txt, and then read the contents of the file and store them in an array of structure.

```
#include<stdio.h>

struct info
{
  char name[50];
  int age;
};

int main()
{
```

```
  int i, age;
  FILE *fp;
  struct info list[5];
  char s[50], t[50];
  fp = fopen( ____(1)____ );
  for(i=0; i<5; i++)
  {
    printf("Enter name of person %d: ", i+1);
    scanf("%s",s);
    printf("Enter age of person %d: ", i+1);
    scanf("%d", &age);
    ____(2)____ ;
  }

/*
After a sample run of the last for loop, the contents of the file "information.t
xt" should look as follows:
age: 30 name: Somesh
age: 60 name: Girish
age: 37 name: Subha
age: 71 name: Benny
age: 45 name: Fatima
*/

  ____(3)____ ;
  ____(4)____ ;
  i = 0;
  while(fscanf(fp, " %s %d %s %s ", s, &list[i].age, t, list[i].name)!=EOF)
    ____(5)____ ;
  return 0;
}

******** Answer ********
(1) "information.txt","w"
(2) fprintf(fp, "age: %d name: %s\n", age, s)
(3) fclose(fp)
(4) fp=fopen("information.txt", "r")
(5) i++


******** Question ********
Let us consider the following function void sort_files (FILE *fp[ ], FILE *out,
int n)   which takes as input an array of n file pointers fp. Each of the file p
ointers point to an input file containing a list of sorted integers. In each fil
e, each line contains one integer. The function merges the contents of the files
 stored in fp and populates the output file pointed by out such that the integer
s stored in the output file are also sorted.

For example, consider n=3 i.e we have 3 input files, say "input1.txt", "input2.t
xt" and "input3.txt". The contents of these files are as follows:

Input1.txt
7
9
input2.txt
1
2
Input3.txt
3
4
```

5
6

The contents of the output file populated by the function would be as follows:
1
2
3
4
5
6
7
8
9

Complete the following incomplete function by filling in the required blanks.

```
#define MAX 100
#define INF 9999
void sort_files(FILE *fp[], FILE *out, int n)
{
  int i = 0;
  int current[MAX];
  // current[i] contains the integer currently read from the i'th file
  int ptr[MAX];
  // ptr[i]=0 if i'th file has no integers to be read or 1 otherwise

  int minimum, min_index;
  int flag = 1;
  for (i = 0; i < n; i++) {
    fscanf(fp[i], "%d", &current[i]);
    ptr[i] = 1;
  }
  // Iterate till flag is set to 1
  while (flag) {
    minimum = INF;  // macro for largest integer

    // Obtain minimum and corresponding input file index
    for (i = 0; i < n; i++) {
      if ( ____(1)____ && current[i] < minimum) {
        minimum = current[i];
        min_index = i;
      }
    }
    //Write output to file
    fprintf(out, "%d\n", ____(2)____ );

    //Scan next number of relevant file if possible
    if (fscanf( ____(3)____ , "%d", &current[ ____(4)____ ] ) == EOF)
      ptr[min_index] = 0;

    // Set flag for while loop
    int check=0;
    for (i = 0; i < n; i++)
      check = ____(5)____ ;
    flag = flag & check;
  }
}
```

******** Answer ********
(1) ptr[i]==1 or ptr[i]

(2) minimum
(3) fp[min_index]
(4) min_index
(5) check | ptr[i] or check + ptr[i]

******** Question ********
Consider a polynomial with real (floating-point) coefficients in the form-
f(x) = c0 * x^d0 + c1 * x^d1 + ... + cn * x^dn where ^ denotes 'raised to the power'. Each non-zero term ci*x^di has integer degrees di where 0 <= d0 < d1 < ... < dn and coefficients ci are nonzero. Each such non-zero term is defined in a C structure as follows:

```
typedef struct {
  double coeff; /* The coefficient in a non-zero term */
  int degree; /* The degree of X in the term */
} term;
```

A polynomial is then stored as a structure instance of the following type:

```
typedef struct {
  int nterms; /* The number of non-zero terms */
  term *termlist; /* The list of terms, that is, (coefficient,degree) pairs */
} poly;
```

Complete the given C function that takes two such polynomials f and g such that f and g have same number of terms. If there exists a non zero term with degree di in f, then there must exist a non zero term with the same degree in polynomial g and vice-versa.
The function finally computes their sum in the same representation.
For example, h = f + g :
f(x) = 4.2 + 5.5   * x - 4.0 * x^2 - 7.098 * x^4
g(x) = 2.91 + 2.98 * x + 4.0 * x^2 + 3.12   * x^4
h(x) = 7.11 + 8.48 * x - 10.218 * x^4

```
poly add ( poly f, poly g )
{
   // f and g has same number of terms
   /* If there exists a non zero term with degree di in f, then there must exist a non zero term with the same degree in polynomial g  and vice-versa. */

   poly h;
   term* tmp_termlist;
   double tmp_coeff;
   int i, k=0, tmp_degree;
   tmp_termlist = (term*) malloc( ____(1)____ ); //allocate memory for termlist
   for(i=0;i<f.nterms;i++)
   {
     tmp_coeff = ____(2)____ ; //compute the coefficient of the term at index i
     tmp_degree = f.termlist[i].degree;
     //compute the degree, can also be g.termlist[i].degree
     if(tmp_coeff!=0) //populate the termlist
     {
       ____(3)____ = tmp_coeff;
       ____(4)____ = tmp_degree;
       k++;
     }
   }
   //update the h polynomial
   h.termlist = ____(5)____ ;
   h.nterms= k ;
```

```
   free(tmp_termlist);
   return h;
}
```

******** Answer ********
(1) f.nterms*sizeof(term) or g.nterms*sizeof(term)
(2) f.termlist[i].coeff + g.termlist[i].coeff
(3) tmp_termlist[k].coeff
(4) tmp_termlist[k].degree
(5) tmp_termlist

******** Question ********
Consider a structure of the following form: struct Student {int roll_no;char nam
e[20];}
It has the following load store routine methods:
a. char* write_entry(struct Student* stlist, int n) is the method to store an n
length student list stlist to a file and return the file name.
b. void fetch_entry(char* filename) is a function to fetch and print the details
 of students from the file name provided as its input.

Fill in the blanks properly to make them work as described.

```
char* write_entry(struct Student* stlist, int n)
{
  FILE *outfile;
  int fwrite= 0;
  char* fname= "studentlist.txt";
  outfile= fopen ( ____(1)____ )
  if (outfile == NULL) {
    fprintf(stderr, "\nError to open the file\n");
    exit (1);
  }
  for(int i=0;i<n;i++){
    fwrite= fprintf(outfile, "%d %s\n", ____(2)____ );
    if( ____(3)____ )
      printf("Contents to file written successfully !\n");
    else
      printf("Error writing file !\n");
  }
  fclose (outfile);
  return fname;
}

void fetch_entry(char* filename)
{
  FILE *infile;
  struct Student st;
  infile = fopen (filename, "r");
  if (infile == NULL) {
    fprintf(stderr, "\nError to open the file\n");
    exit (1);
  }
  while( (fscanf(infile,''%d %s", ____(4)____ ))!=EOF)
    printf ("Student roll_no = %d name = %s\n", ____(5)____ );
  fclose (infile);
}
```

******** Answer ********
(1) fname, "w"

(2) stlist[i].roll_no,stlist[i].name
(3) fwrite != 0
(4) &st.roll_no,&st.name[0]
 (&st.roll_no,&st.name -- might throw a warning for input char[] size against %s
, but will run)
(5) st.roll_no, st.name

******** Question ********
Given a text file in a directory, the following program prints all the odd line contents of the file first then prints all the even line contents. Assuming the program CAN NOT USE "r" mode to open the file, fill up the below code to achieve the above output.

Example: Suppose the "file1.txt" contains the following text:

```
This is Line Number 1
This is Line Number 2
This is Line Number 3
This is Line Number 4
This is Line Number 5
This is Line Number 6
```

Output:

```
This is Line Number 1
This is Line Number 3
This is Line Number 5
This is Line Number 2
This is Line Number 4
This is Line Number 6
```

```
#include <stdio.h>

void printLines(char x[ ])
{
    // The program CAN NOT use "r" mode to read the file

    FILE* fp = fopen( ____(1)____ );
    fprintf(fp, "\n");

    // set the file pointer to desired position
    ____(2)____ ;

    int LineType = 0;
    char buf[100];

    // Print Odd lines
    while (fgets(buf, sizeof(buf), fp))
    {
        if ( ____(3)____ )
        {
            printf("%s", buf);
        }
        LineType++;
    }
    LineType = 1;
    ____(4)____;

    // Print Even lines
```

```
    while (fgets(buf, sizeof(buf), fp))
    {
        if ( ____(5)____ )
        {
            printf("%s", buf);
        }
        LineType++;
    }

    fclose(fp);
    return;
}

int main()
{
    char x[] = "file1.txt";
    printLines(x);

    return 0;
}
```

```
******** Answer ********
(1) x, "a+"
(2) fseek(fp, 0, 0) or  rewind(fp)
(3) !(LineType % 2)
(4) fseek(fp, 0, 0) or  rewind(fp)
(5) !(LineType % 2)
```

```
******** Question ********
The objective of this program is to calculate the document statistics of a text
file 'sample.txt'. Suppose the text file contains only '.' and ',' as punctuatio
n characters. Complete the following code to calculate the number of lines and n
on-punctuation characters in the text file.

#include <stdio.h>

int main() {
    FILE *fp;
    int lcount = 0, ccount = 0;  // lcount for line counter and ccount for chara
cter counter
    char c;
    fp = fopen("sample.txt", "r");
    if (fp == NULL) {
        printf("Could not open file");
        return 0;
    }
    for ( ____(1)____ ; ____(2)____ ; ____(3)____ ) {
    // Extract characters from file and loop till end of the file
        if ( ____(4)____ ) { // Count line number
            lcount++;
            continue;
        }
        if ( ____(5)____ ) // Count non-punctuation characters
            ccount++;
    }
    fclose(fp);
    printf("%d %d\n ", lcount, ccount);
    return 0;
}
```

******** Answer ********
(1) c = getc(fp)
(2) c != EOF
(3) c = getc(fp)
(4) c == '\n'
(5) !(c == ' ' || c == '.' || c == ',')   [Alternate Solution: c != ' ' && c != '.' && c != ',']

******** Question ********
Consider a structure node that consists of an integer and a pointer to struct node. Let arr1 and arr2 be two arrays of struct node. The objective is to map the elements from arr1 to the elements in arr2 which have the same data.
For eg: arr1 = {1, 2, 3, 4, 5}   arr2 = {3, 4, 2, 5, 1}
arr1[0] should be mapped to arr2[4]
arr1[1] should be mapped to arr2[2]
arr1[2] should be mapped to arr2[0]
arr1[3] should be mapped to arr2[1]
arr1[4] should be mapped to arr2[3]

Complete the following C code that takes an integer n, creates two array of structures each of length n, maps the element of arr1 to arr2 and prints the elements as follows
1 -> 1
2 -> 2
3 -> 3
4 -> 4
5 -> 5

```
#include<stdio.h>

struct node {
    int data;
    struct node* map;
};

int main() {
    int n;
    scanf("%d", &n);
    struct node* arr1 = (struct node*)malloc(n*sizeof(struct node));
    struct node* arr2 = (struct node*)malloc(n*sizeof(struct node));

    for(int i=0;i<n;i++) {
        scanf("%d", ____(1)____ );  //enter data
        arr1[i].map = NULL;  // set the map pointer to NULL
    }
    for(int i=0;i<n;i++) {
        scanf("%d", ____(2)____ );  //enter data
        arr2[i].map = NULL;  // set the map pointer to NULL
    }
    //map elements from arr1 to elements in arr2 having the same data.
    for(int i=0;i<n;i++) {
        for(int j=0;j<n;j++) {
            if( ____(3)____ ) {  //compare data
                arr1[i].map = ____(4)____ ;  //map elements in arr1 to elements in arr2
            }
        }
    }
```

```
    for(int i=0;i<n;i++) {
        //prints elements in arr1 and the mapped elements in arr2
        printf("%d -> %d \n", arr1[i].data, ____(5)____ );
    }
}
```

******** Answer ********
1. &arr1[i].data
2. &arr2[i].data
3. arr1[i].data == arr2[j].data
4. &arr2[j]
5. arr1[i].map->data

==========

++++++++++++++++++++++++ Linked List, Stack and Queue ++++++++++++++++++++++++

******** Question ********
Fill in the following blanks in the C program that will perform following steps on a list of integers supplied as input using stack.

ASSUME: push(), pop(), top(), isFull(), isEmpty() functions are already defined and provided to you to access them as required.

(a) Read the list of integers to be entered.
(b) Push the first element into the stack
(c) Second elements onwards if the current element is less than the stack top el ement then print onto the terminal, else push the element into the stack.
(d) Pop all the elements from the stack once input list is exhausted.

Example Input:
Enter the number of elements: 10
Enter the elements:
20 10 3 2 4 1 5 62 9 1

Example Output:
10 3 2 4 1 5 9 1 62 20

```
#include <stdio.h>

struct STACK{
  int elements[1000];
  int top;
};

int main()
{
   int i,num,stk,n;
   ____(1)____ ;

   printf("Enter the number of elements: ");
   scanf("%d",&n);
   printf("Enter the elements: ");
   scanf("%d",&num);
   ____(2)____ ;
   for(i=1; i<n; i++) {
     scanf("%d",&num)
     if(!isFull(s)) break;   // stack full
     if( ____(3)____ ) {
```

```
      push(s,num);
    }  else {
      ____(4)____ ;
    }
  }
  do {
    ____(5)____ ;
    printf("%d ",num);
  } while(!isEmpty(s));   // until the stack is not empty
  return 0;
}
```

```
******** Answer ********
(1) struct STACK *s
(2) push(s,num)
(3) top(s)<num
(4) printf("%d ",num)
(5) num=pop(s)
```

******** Question ********

The following C function will take two arguments (i) A linked list which is already created and contains a list of integer elements, and (ii) an integer value. Then the function will delete the first occurrence of the integer value if it is present in the linked list.

Fill in the blanks so that the following C program snippet performs the abovementioned function.

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

struct NODE {
  int num;
  struct NODE *next;
};

struct NODE *nodeDel(struct NODE *head, int key)
{
  struct NODE *curr,*prev;
  int found=0;
  if(head->num == key) {
    ____(1)____ ;
    ____(2)____ ;
    found = 1;
  } else {
    prev = head;
    curr = prev->next;
    while( ____(3)____ ) {
       if(curr->num == key) {
          ____(4)____ ;
          found = 1;
          break;
       }
       prev = curr;
       curr = prev->next;
    }
  }
  if(found == 1) {
```

```
  ____(5)____ ;
  }
  return (head);
}
```

```
******** Answer ********
(1) curr=head
(2) head=head->next
(3) curr!=NULL
(4) prev->next=curr->next
(5) free(curr)
```

```
******** Question ********
Fill in the blanks so that the following C function that will read one integer e
lement and will insert that in a sorted linked list (sorted in non-decreasing/as
cending order) which is already created and supplied as an argument.
```

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

struct NODE {
  int num;
  struct NODE *next;
};

struct NODE *nodeIns(struct NODE *head, int ele)
{
  struct NODE *curr,*temp; int inserted=0;
  ____(1)____ ;
  temp->num = ele;
  temp->next = ____(2)____ ;
  if(head->num > ele) {
    ____(3)____ = head;
    head = temp;
  }
  else {
    curr = head;
    while(curr->next != NULL) {
       if(curr->next->num > ele) {
          ____(4)____ ;
          ____(5)____ ;
          inserted=1; break;
       }
       curr = curr->next;
    }
    if(inserted==0) {
       curr->next=temp;
    }
  }
  return (head);
}
```

```
******** Answer ********
(1) temp=(struct NODE *)malloc(sizeof(struct NODE)))
(2) NULL
(3) temp->next
(4) temp->next=curr->next->next
(5) curr->next=temp
```

******** Question ********
Complete the following program which manipulates some numbers within a linked li
st.

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

struct num
{
  int val;
  struct num *next;
};

int main()
{
  int i, n;
  char r;
  FILE *fp;
  struct num *head=NULL, *temp;
  printf("Do you want to enter a number (y/n)? ");
  scanf(" %c",&r);
  if(r != 'y')
  {
    printf("List is empty!");
    return 0;
  }
  printf("Enter the number: ");
  scanf("%d",&n);
  head = ____(1)____ ;
  head->val = n;
  head->next = ____(2)____ ;
  while(1)
  {
    printf("Do you want to enter a number (y/n)? ");
    scanf(" %c",&r);
    if(r != 'y')
      break;
    printf("Enter the number: ");
    scanf("%d",&n);
    ____(3)____ ;
    temp->val = n;
    ____(4)____ ;
    ____(5)____ ;
  }
  return 0;
}
```

******** Answer ********
(1) (struct num*)malloc(sizeof(struct num))
(2) NULL
(3) temp=(struct num*)malloc(sizeof(struct num))
(4) temp->next=head
(5) head=temp

******** Question ********
Complete the following program that computes the binary representation of an int
eger using a global stack. You must access the stack only through the functions
defined here. Fill in the following blanks.

```
#include<stdio.h>

struct Stack
{
  int A[50];
  int top;
} S;

void Init()
{
  S.top = -1;
}

int Isempty()
{
  return ((S.top) == -1);
}

void push(int n)
{
  S.A[++(S.top)] = n;
}

int pop()
{
  if(Isempty(S))
    return -1;
  else
    return S.A[(S.top)--];
}

int main()
{
  int i, n;
  printf("Enter number: ");
  scanf("%d",&n);
  Init();
  while(n != ____(1)____ )
  {
    ____(2)____ ;
    ____(3)____ ;
  }
  while( ____(4)____ )
    printf("%d", ____(5)____ );
  return 0;
}
```

******** Answer ********
(1) 0
(2) push(n%2)
(3) n/=2 or n=n/2 or anything equivalent
(4) !Isempty() or Isempty()==0 or Isempty()!=1 or anything else having the same effect here.
(5) pop()

******** Question ********
Fill in the blanks to complete the following program that first takes in some in tegers from the user and enqueues them in a queue, and then prints the second la

st element enqueued. You are permitted to access the queue only through the functions provided.

```
struct Queue
{
  int A[SIZE];
  int front;
  int rear;
} Q;

void Init()
{
  Q.front = 0;
  Q.rear = 0;
}

int Isempty()
{
  return (Q.rear == Q.front);
}

void enqueue(int n)
{
  Q.A[(Q.rear)] = n;
  Q.rear = (Q.rear+1)%SIZE;
}

int dequeue()
{
  int t;
  if(Isempty())
    return -1;
  t = Q.A[Q.front];
  Q.front = (Q.front+1)%SIZE;
  return t;
}

int main()
{
  int n,t;
  char c;
  Init();
  printf("Do you want to enter a number (y/n)?");
  scanf(" %c",&c);
  if(c != 'y')
  {
    printf("Empty queue");
    return 0;
  }
  while(c == 'y')
  {
    printf("Enter number: ");
    scanf("%d",&n);
    enqueue(n);
    printf("Do you want to enter a number (y/n)?");
    scanf(" %c",&c);
  }
  n = ____(1)____ ;
  if(Isempty())
  {
```

```
    printf("Only one element in the queue");
    return 0;
  }
  t = dequeue;
  while( ! ____(2)____ )
  {
    ____(3)____ = ____(4)____ ;
    ____(5)____ ;
  }
  printf("The second last element is %d", n);
  return 0;
}
```

```
******** Answer ********
(1) dequeue()
(2) Isempty()
(3) n
(4) t
(5) t=dequeue()
```

```
******** Question ********
We have the following structure,
struct Node {
  int data;
  struct Node *next;
};
to build each node of the linked list. Consider the following functions:
[a] void insert(struct Node **head_ref, int new_data) : By this function, you ca
n populate a linked list.
[b] void merge(struct Node **p, struct Node *q) : This function merges 2 such li
nked lists by placing elements from the first linked list alternately with eleme
nts from the second linked list. The merged list starts with the first element o
f the second linked list and the merged list will reside in the second list itse
lf.

See the sample inputs and outputs to understand in detail. Fill in the blanks pr
operly so that the methods behave as described.

When no of elements in list2 is more than list1:
    sample Input: Llist1 = [1,2,3], Llist2 = [4,5,6,7,8]
    merged Llist2 = [4,1,5,2,6,3,7,8]
When no of elements in list1 is more than list2 :
    sample Input: Llist1 = [10,2,13,4,5], Llist2 = [6,17,8]
    merged Llist2 = [6,10,17,2,8,13,4,5]

Note that, the merged list is always in Llist2 and it starts with the first elem
ent of Llist2 .
Also see in the sample examples, if both the lists are not same in size, the ext
ra elements (from Llist1 or Llist2) are always appended at the end of Llist2.

void merge(struct Node **p, struct Node *q)
{
  // *p points to the head of first linked list
  // q points to the head of the second linked list
  struct Node *p_curr = *p,
  struct Node *q_curr = q;
  struct Node *p_next, *q_next, *q_last;

  while ( ____(1)____ )
```

```
  {
    // Save next pointers
    p_next = p_curr->next;
    q_next = q_curr->next;

    // Point current element of p as next element of *q
    ____(2)____ = p_curr;
    // link q with current p
    ____(3)____ = q_next;

    // Update current pointer of p for next iteration
    p_curr = p_next;
    // store the last pointer of q
    q_last = q_curr;
    // Update current pointer of q for next iteration
    q_curr = q_next;
  }
  // Update pointer of the first list to point at its remaining elements
  *p = ____(4)____ ;
  // if p has more elements than q,
  // append the remaining elements at the end of q
  if (p_curr!=NULL)
  {
    ____(5)____ = *p;
  }
}
```

```
******** Answer ********
(1) p_curr != NULL && q_curr != NULL
(2) q_curr->next
(3) p_curr->next
(4) P_curr
(5) q_last->next
```

```
******** Question ********
Consider the following structure, used for a stack implementation:
struct stack {
  int data;
  struct stack* next;
};
uses following utility methods,
int top(struct stack** s) : to peek at the top element of stack s
int pop(struct stack** s) : to pop the top element of stack s
void push(struct stack** s, int i) : to push integer i in the stack s

The following method void sortStack(struct stack* op, struct stack* ip) populate
s the elements from the input stack ip into the stack op in descending order, i.
e. maximum element will be at the top of the stack op and so on. Complete the fo
llowing method by filling in the blanks properly.

Sample input: input stack (ip) = [30,20,2,1,38,-1], output stack (op) = [38,30,2
0,2,1,-1]
Note that, the first element of ip or op denotes the top element and the last el
ement is the bottom element. After sort, the first element should be the maximum
 one and the following elements should be in descending order (Last element is t
he minimum among the stack entries).

void sortStack(struct stack** op,struct stack** ip)
{
```

```
  struct stack* input= *ip;
  struct stack* tempstack= NULL;
  struct stack** head= &tempstack;

  while ( input!=NULL)
  {
    // pop the first element of input stack
    int iptop = pop(&input);

    while (tempstack!= NULL)
    {
      // if top of temp stack is greater than top of input stack
      if( ____(1)____ ){
        // pop from temp stack and push into the input stack
        int temp = ____(2)____ ;
        push(&input, ____(3)____ );
      }
      else{
        break;
      }
    }
    // push top element of input stack into temp stack
    push( ____(4)____ );
  }
  // return output using op pointer;
  *op = ____(5)____ ;
}

******** Answer ********
(1) top(&tempstack) > iptop
(2) pop(&tempstack)
(3) temp
(4) &tempstack,iptop
(5) tempstack


******** Question ********
Consider a queue defined using the following structure:
struct Queue {
  int front, rear, size;
  unsigned capacity;
  int* array;
};
Here front, rear signifies the index of the frontmost and rearmost elements in t
he queue. size signifies the current size of the queue and capacity is maximum a
llowable size of the queue.

There are also utility functions like:
struct Queue* createQueue(unsigned s) : to initialize a queue with capacity s.
void enqueue(struct Queue* q, int i) : to enqueue an integer i into the queue q
int dequeue(struct Queue* q) : to dequeue the front element from q
int front(struct Queue* q) : to return the frontmost element in q

Fill in the following blanks to write a function struct Queue* reverse(struct Qu
eue* q)using these functions, in order to return a queue that has all the elemen
ts of q in reversed order?

Sample input : q= [4,8,2,9]
Sample output: newq= [9,2,8,4]
Note that, the first element of the array is the front of the queue.
```

```
struct Queue* reverse(struct Queue* q)
{
  // Size of queue
  int s = q->size;

  // Second queue
  struct Queue* newq= createQueue(s);

  for (int i = 0; ____(1)____ ; i++)
  {
    // Get the last element to the front of old queue
    for (int j = 0; j < q->size - 1; j++) {
      int x = ____(2)____ ;
      enqueue( ____(3)____ );
    }
    // Get the last element and add it to the new queue
    enqueue(newq, ____(4)____ );
    dequeue(q);
  }
  return ____(5)____ ;
}
```

```
******** Answer ********
(1) i < s
(2) dequeue(q)
(3) q, x
(4) front(q)
(5) newq
```

```
******** Question ********
Suppose there are two singly linked lists both of which intersect at some point
and become a single linked list. The head or start pointers of both the lists ar
e known, but the intersecting node is not known. Also the number of nodes in eac
h of the lists before they intersect are unknown and both lists may have it diff
erent. Let's write an efficient C program for finding the merging point. The fun
ction will return the data stored in the intersection node. If no intersecting n
ode is found, it will return NULL.

typedef struct Node {
    int data;
    struct Node *next;
};

struct Node* FindIntersection(struct Node* list1, struct Node* list2) {
    /* L1 and L2 store the number of nodes counted in
        List1 and List2 and diff stores their difference. */
    int L1 = 0, L2 = 0, diff = 0;
    struct Node* head1 = list1;  /* start of list1 */
    struct Node* head2 = list2;  /* start of list2 */
    while(head1 != NULL) {  /* traverse list1 */
        L1++;
        ____(1)____ ;
    }
    while(head2 != NULL) {  /* traverse list2 */
        L2++;
         ____(2)____ ;
    }
    diff = L1 - L2;
```

```
    if (L1 < L2) {
        head1 = list2;
        head2 = list1;
        diff = ____(3)____ ;
    }
    for (int i = 0; i < diff; i++)
        head1 = ____(4)____ ;
    while (head1 != NULL && head2 != NULL) {
        if ( ____(5)____ )  /* the intersecting node is found */
            return head1->data;
        head1 = head1->next;
        head2 = head2->next;
    }
    return NULL;
}

******** Answer ********
(1) head1 = head1->next
(2) head2 = head2->next
(3) L2 - L1
(4) head1->next
(5) head1 == head2


******** Question ********
Given a string consisting of opening and closing parenthesis, the following func
tion, findMaxLen(char str[]), returns the length of the longest balanced parenth
esis in it.

For example,
Input: (()()     --> Output: 4
Input: (()())(() --> Output: 6
Input: ((((      --> Output: 2
Input: ((((      --> Output: 0

We have implemented the following algorithm in the given code:
-- Iterate over the string characters
-- If the character is an opening parenthesis push the index in a stack
-- If the current character is a closing parenthesis, pop the top index and push
 the current index if the stack is empty
-- The length is obtained by finding the difference between the current index an
d the index at the stack top.

Pre-Defined Functions:
void push(struct Stack* stack, int item):  Function to add an item to stack.  It
 increases top by 1
void pop(struct Stack* stack):  Function to remove an item from stack.  It decre
ases top by 1
int isEmpty(struct Stack* stack):  Return 1 when stack is empty else 0
int peek(struct Stack* stack):  Function to return the top from stack without re
moving it

Fill in the blanks to complete the function findMaxLen(char str[]) that implemen
ts the above algorithm. You can use the pre-defined functions stated.

// A structure to represent a stack
struct Stack {
    int top;
    unsigned capacity;
    int* array;
```

```
};

// function to create a stack of given capacity.
struct Stack* createStack(unsigned capacity)
{
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (int*)malloc(stack->capacity * sizeof(int));
    return stack;
}

// Function to find the length of the longest balanced parenthesis in a string
int findMaxLen(char str[])
{
    // Declare an empty character stack. By default a stack of capacity 100 is d
eclared
    struct Stack* stack = createStack(100);
   // initialize the stack with index -1
    push(stack,-1);
    // stores the length of the longest balanced parenthesis
    int len = 0;

    // iterate over the characters of the string
    for (int i = 0; str[i]!='\0'; i++)
    {
        // if the current character is an opening parenthesis
        if (str[i] == '(') {
            ____(1)____ ;
        }

        // if the current character is a closing parenthesis
        else
        {
            // if current character is closing parenthesis
            ____(2)____ ;

            // if the stack becomes empty
            if (isEmpty(stack))
            {
                ____(3)____ ;
                continue;
            }

            // get length of the longest balanced parenthesis ending
            // at current character
            int curr_len = ____(4)____ ;

            // update the length of the longest balanced parenthesis
            if ( ____(5)____ ) {
                len = curr_len;
            }
        }
    }

    return len;
}

******** Answer ********
(1) push(stack,i);
```

(2) pop(stack);
(3) push(stack,i);
(4) i - peek(stack);
(5) len < curr_len

******** Question ********
Let S1 and S2 be the two stacks used in the implementation of queue. We shall define the EnQueue and DeQueue operations for the queue. Fill in the following blanks:

Pre-defined functions are as follows:
-- int IsEmptyStack(struct Stack** S): Indicates whether any elements are stored in the stack or not
-- void Push(struct Stack** S, int data): Inserts data onto stack
-- int Pop(struct Stack** S): Removes and returns the last inserted element from the stack

```
struct Queue{
    struct Stack *S1;        //for EnQueue
    struct Stack *S2; // for DeQueue
}

void EnQueue(struct Queue *Q, int data)
{
    return ____(1)____ ;  //Pushes onto the stack
}

int DeQueue(struct Queue *Q)
{
    if(!IsEmptyStack(Q -> S2))
        return ____(2)____;  //Pops the required element
    else
    {
        while(!IsEmptyStack(Q -> S1))
             // Transfers all elements from S1 to S2 and
             // pops the top element from S2
             ____(3)____ ;

         return _____(4)____ ;  //Returns the popped element
     }
}
```

With reference to the above function, void EnQueue(struct Queue *Q, int data), the running time in Big-Oh (O) notation is ____(5)____ .

******** Answer ********
(1) Push(Q->S1, data)
(2) Pop(Q->S2)
(3) Push(Q->S2, Pop(Q->S1))
(4) Pop(Q->S2)
(5) constant [Alternate Answer: O(1)]

==========