Computing Laboratory 2025 Class Test 1

Maximum Marks: 25 August 08, 2025 Total Time: 45mins

Roll No: ____________ Name: ________________

[Write your answers in the question paper itself. Answer all questions.]

1.

(a) What does the following program print? (2)

```
int main()
{
    int i = 100, m, n;
    m = 1 + (i++);
    n = 1 + (++i);
    printf("m = %d, n = %d\n", m, n);
    return 0;
}
```

m =101
n =103

(b) What does the following program print? (2)

```
int eval(int q)
{
    q *= 10;
    q -= 100;
    return q;
}

int main()
{
    int p = 10, q = 100;
    q = eval(p);
    printf("%d %d ", p, q);
    q = eval(q);
    printf("%d %d\n", p, q);
    return 0;
}
```

10 0 10 -100

(c) Describe in one English sentence what the following function outputs. Assume $b \neq 0$. (2)

```
int what(int a, int b)
```

```
{
    double q1, q2;
    q1 = a / b;
    q2 = (double)a / (double)b;
    return (q1 == q2);
}
```

Returns whether a is an integral multiple of b.

(d) What does the following program print? (2)

```
int main () {
    int n = 100, s;
    for (s = 2; n > 2; --n) {
        s += 2; n -= 2;
    }

    printf ("%d", s);
}
```

68

(e) What does the following program print? (2)

```
int main ()
{
    int m = 5, n = 5, x;
    char p = 'p', q = 'q';
    x = !((m >= n) || (m <= n) && (p > q));

    printf ("%d", x);
}
```

0

2. You are given an array A of n integers. It is given that the elements of A satisfy the following inequalities

$$A[0] < A[1] < \cdots < A[m-1] < A[m] > A[m+1] > A[m+2] > \cdots > A[n-1]$$

for some (unknown) index m in the range $1 \leqslant m \leqslant n-2$. Let us call such an array a *hill-valued* array. The sequence $A[0], A[1], \ldots, A[m-1], A[m]$ is called the ascending part of the hill, and the remaining part $A[m], A[m+1], \ldots, A[n-1]$ is called the descending part of the hill. The element $A[m]$ is the peak of the hill and is the largest element in the array.

Your task is to locate the peak (that is, $A[m]$) in the hill-valued array A. Initially, start searching in the entire array. Subsequently, in each iteration of the loop, reduce the search space to a subarray of half the size of the subarray in the previous iteration. In order to achieve that, compute the middle index in the current search space. Compare the element at this index with its two neighbors. If you are at the peak, break the loop, else adjust the search space appropriately. (This procedure is similar to binary search in a sorted array.) Complete the following C program that implements this idea. Do not use new variables. **(5)**

```
#include <stdio.h>

#define MAX 1000

main () {
    int A[MAX], i, n, L, R, M, found;

    printf("Enter a hill-valued array.\n");
    printf("Number of elements = "); scanf("%d", &n);
    for (i=0; i<n; ++i) { printf("A[%d] = ", i); scanf("%d", &A[i]); }

    /* Initialize the left and right boundaries L and R of the search space
       so as to encompass the entire array A */

    L = _______0_______ ; R = _______n-1_______ ;
    found = 0; /* Initialize flag to false */
    while (!found) { /* Loop as long as the maximum is not located */
        /* Compute the middle index M */

        M = ____(L+R)/2____ ;

        if ( (A[M-1]<A[M]) && (A[M+1] <A[M]) ) {
            /* if the top of the hill is located, */
            /* set the flag to break the loop before the next iteration */
            found = 1;

        } else if ( (A[M-1]< A[M]) && (A[M]< A[M+1]) ) {
            /* if the middle index is in the ascending part of the hill, */
            /* discard a suitable half of the search space */

            ____________L=M____________ ;
        } else {
            /* if the middle index is in the descending part of the hill, */
            /* discard a suitable half of the search space */

            ____________R=M____________ ;
        }
    }
    printf("Maximum = %d\n", ______A[M]______ );
}
```

3. A paragraph is stored in a two-dimensional array of characters. Each line of the paragraph is stored in a row of the array as a null-terminated string. An empty line indicates the end of the paragraph.

(a) Complete the following function that, given a paragraph stored in the format mentioned above, prints the paragraph line by line, and stops when the terminating empty line is encountered. **(4)**

```
void prnPara ( char para[] [MAXLEN] )
{
   int i; /* Do not use any other variable */
   /* Print each line as a string until a blank line is found */
   for (i=0; strlen( para[i] ); ++i) printf("%s\n", para[i] );
}
```

(b) Complete the following function that right justifies a paragraph with respect to a target length T as follows. For each line in the paragraph, its length l is computed. If $l > T$, then right justification fails. If $l \leqslant T$, you shift the line by $T - l$ positions to the right, so that the line now ends at exactly the index $T - 1$ (excluding the terminating null character). The first $T - l$ positions are then filled with spaces. The right justification loop terminates whenever an empty line is found (indicating the end of the paragraph). **(6)**

```
void rightJustify ( char para[] [MAXLEN], int T )
{
   int i, j, len; /* Do not use any other variable */

   /* Repeat so long as an empty line is not found */

   for (i=0; strcmp( para[i] , "" ); ++i) {
      /* Compute in len the length of the i-th line */
      len = strlen(para[i]) ;
      if (len > T) {
         printf("Line %d is too wide to fit in %d characters\n", i, T);
         continue /* Skip the rest of the current iteration */
      }
      /* Write a loop to shift characters by T-len positions to the right */
      for(j=len-1; j>=0; --j) para[i][j+(T-len)] = para[i][j];
      /* Write a loop to fill the first T-len characters by spaces */
      for(j=0; j<T-len; ++j) para[i][j]=' ';
      /* Terminate the i-th line by the NULL character */
      para[i] [ T ] = '\0' ;
   }
}
```