**Q1a (Trace-code; parameter-passing):** What is the output of the following program? [3 Marks]

```cpp
int fun(int& a, int b) {
    a += 3;
    b = 3;
    cout << "a=" << a << " b=" << b << "\n";
    return a%b;
}

int main() {
    int a = 7;
    int b = 4;
    int c = fun(a,b);
    cout << "a=" << a << " b=" << b << " c=" << c << "\n";
    return 0;
}
```

**Space for Rough Work:**

**Marking Scheme:**
- 1 Mark for correct first output.
- 2 Marks for correct second output.
- No other partial credit.

**Output:**
```
a=10 b=3
a=10 b=4 c=1
```

**Q1b (Trace-code; pointers):** Show the contents of the memory locations at the specified steps. [3 Marks]

```cpp
int main () {
    int num[4]; int* p;
    for (int i=0; i < 4; i++) num[i] = 0;
    p = num;  // Show the memory here ... Step 1
    *p = 10;
    p++; *(p++) = 20;
    *p = 30;  // Show the memory here ... Step 2
    p = &num[3]; *p = 40;
    *(p-1) = 30;
    for (int n=0; n<4; n++)
        cout << num[n] << ", ";
    return 0;
}  // Show the final output ... Step 3
```

**Step1:** Contents of the memory locations are:
- num[0] has memory byte address X, value 0.
- num[1] has address X+4, value 0, and so on.
- P has address Y, value X. Or P should be drawn as pointing to num[0]. Note: Y could be X+16.

All items are shown correctly – 1 Mark
Items are shown correctly but some details are missing – 0.5 Marks.
Anything else – 0 Marks.

**Step 2:** Contents of the memory locations are:

All values are correct – 1 Mark. Else 0 Marks.

**Step 3:** The output of the program is:
- 10, 20, 30, 40

All values are correct – 1 Mark. Else 0 Marks.

Run: quiz2-pointers-parameters.cpp to verify answers to both the above programs.
Q2 (Write code; Strings): Write a C++ function that takes an array of strings as input, and returns the smallest string in dictionary order. For example, if the array has strings \{abcd, abc, bc, acd\}, your function should return 'abc'. You should use C++ string type and its functions. If you are not familiar with C++ string, then you may use the C strings library functions, or any other way, with a penalty of two marks. State any additional assumptions that you need to make.

---

### Solution using C++ string type

```cpp
#include <iostream>
#include <string> // C++ string class
using namespace std;

const int n = 8; // Number of strings

string findSmallestString(string A[], int n) {
    string msg1, msg2;
    msg1 = A[0];
    for (int i=1; i<n; i++) {
        msg2 = A[i];
        if (msg1.compare(msg2) > 0) msg1 = msg2;
    }
    return msg1;
}

int main() {
    string A[n], msg;
    cout << "Give n strings\n";
    for (int i=0; i<n; i++) cin >> A[i];
    msg = findSmallestString(A, n);
    cout << "The smallest string is: " << msg << endl;
} // Main program is for showing call to function
```

### Solution using C string library functions

```cpp
#include <iostream>
#include <cstring> // C library of string functions
using namespace std;

const int n = 8; // Number of strings
const int m = 8; // length of longest string

void smallest() { //Using C library functions
    char s1[n], s2[n], B[n][m];
    cout << "Give n strings\n";
    for (int i=0; i<n; i++) cin >> B[i];
    for (int j=0; j<m; j++) s1[j] = B[0][j];
    for (int i=1; i<n; i++) {
        for (int j=0; j<m; j++) s2[j] = B[i][j];
        if (strcmp(s1, s2) > 0) strcpy (s1, s2);
    }
    cout << "The smallest string is: " << s1 << endl;
}

int main() {
    smallest();
} // This is not required as part of answer
```

### Marking Scheme:
The above is one valid solution.
- `#include <string>` - 1 Mark
- Correct function declaration – 2 Marks
- Correct setup of loop – 2 Marks
- Use of string.compare function – 2 Marks
- Correct return statement – 1 Mark

If any other solution is shown, use TA discretion, to give marks roughly as per above scheme.

---

Run: quiz2-strings.cpp to verify both the above programs.
Write your Roll Number here: ____________________

**Q3 (Write code; Linked Lists):** Write a function, `sortedIntersect()`, that takes two lists sorted in increasing order, and returns a new linked list having the intersection of the two lists. For example, if the original lists are given as {1,2,3,4} and {2,3, 23}, then `sortedIntersect()` should return a list containing {2, 3}.

You may use assume the following as given:

```cpp
class node {
    int num;
    node * next;
}; // each node of the list has one number and a pointer.

node* append(node *list, int item); // function that appends item at the end of a given list and returns a pointer to its head.

int main(); // Main program that creates two lists L1 and L2 and calls sortedIntersect() appropriately.
```

You need to do:

1. Write the pseudo-code (logic) for sortedIntersect().  
   **[4 Marks]**
2. Write the C++ code for above: `node* sortedIntersect(node* L1, node* L2);`  
   **[6 Marks]**

---

**// psuedo-code**

1. Get pointers to the head of L1 and L2  
2. Get a pointer to an empty output list  
3. If num of L1 ptr == num of L2 ptr, then  
4. append the num to the output list  
5. Else if L1's num is smaller, advance L1 ptr  
6. Else advance L2 ptr  
7. Repeat 3--6 until one of the lists is empty  
8. Return pointer to head of output list

**Marking Scheme:**  
There is no fixed way of writing psuedo-code. It is enough if you can follow the logic of the steps.

Start with an empty output list – 1 Mark  
Condition true (steps 3-4) then append – 1 Mark  
Condition false (steps 5-6) then advance – 1 Mark  
Loop until one of the lists is empty – 1 Mark  
Partial marking as per TA discretion.

---

**// C++ code**

```cpp
node* sortedIntersect(node* L1, node* L2) {
    node* head = NULL;
    while (L1!=NULL && L2!=NULL) {
        if (L1->num == L2->num) {
            head = append (head, L1->num);
            L1 = L1->next; L2 = L2->next;
        } else if (L1->num < L2->num) L1 = L1->next;
        else L2 = L2->next;
    }
    return head;
}
```

**Marking Scheme:**  
Initialization and return statement – 1 Mark  
Setup of loop with correct exit condition – 1 Mark  
If condition check for intersection – 1 Mark  
Call to append done correctly – 1 Mark  
Advance pointers correctly in each case – 2 Mark  
Partial marking as per TA discretion.

---

Run: quiz2-linklist.cpp to verify the above program.
Q4 (Debug-code, matrices): Your friend wrote the program below to find all the zeros in a 8*8 matrix and replace the corresponding rows and columns with zeros. A sample input and output are shown below. Fill in the two places of the missing code as indicated.

[6 Marks]

Example Input Matrix | Example Output Matrix
---|---
1 2 3 4 5 6 7 8 | 1 0 3 4 0 6 0 8
2 0 2 3 4 5 6 9 | 0 0 0 0 0 0 0 0
3 1 5 7 8 4 2 3 | 3 0 5 7 0 4 0 3
4 3 8 7 0 3 3 5 | 0 0 0 0 0 0 0 0
5 2 3 4 5 6 1 7 | 5 0 3 4 0 6 0 7
6 7 6 9 1 8 9 | 6 0 6 6 0 1 0 9
7 0 7 4 3 2 0 2 | 0 0 0 0 0 0 0 0
8 9 5 2 3 4 5 6 | 8 0 5 2 0 4 0 6

```c
#include <iostream>
using namespace std;

const int n = 8;
struct locations { //keep track of rows and columns that need to be nullified
    int row[n], col[n];
};

void nullifyRow(int A[n][n], int row) { for (int i=0; i<n; i++) A[row][i] = 0; }
void nullifyCol(int A[n][n], int col) { for (int i=0; i<n; i++) A[i][col] = 0; }

locations findZeros(int A[n][n]); //Find the zeros in the given matrix and record their location in the struct.
// It first initializes locations to all 1s and then changes a particular entry to 0 as appropriate.
// This is the function declaration. Fill in this function definition on the next page.

int main() {
    locations h; int Matrix[n][n]; int i, j;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) cin >> Matrix[i][j];
    h = findZeros(Matrix);
    // Call nullifyRow, nullifyCol as per the entries in h.
    // Fill in this part of the code
    for (i=0; i<n; i++) {
        if (l.row[i] == 0) nullifyRow(Matrix, i);
        if (l.col[i] == 0) nullifyCol(Matrix, i);
    }
    for (i=0; i<n; i++) { //Output the matrix
        for (j=0; j<n; j++) cout << Matrix[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

Marking Scheme:
Correct condition checks – 1 Mark
Correct call to nullify – 1 Mark

Marking Scheme:
Initialization of locations variable k – 1 Mark
Setup of nested for loops – 1 Mark
If condition check and actions – 1 Mark
Return statement – 1 Mark

Note: The simplest solution is to scan every cell of the matrix, as above. If someone has optimized the scanning, it is good, but not essential.