Time: 3 Hours + 30 minutes for uploading. **Max Points:** 100

Section 1 – Write Code snippets **Marks:** 20 points (20%)

Q1. Write code snippets for the following scenarios.

Note:

i. No marks will be awarded for text based explanations.

ii. Write useful code only i.e. variables declarations and problem-solving logic. Avoid glue code like #includes, main (), etc. at all costs.

a) Suppose you have received a log file with 200 transactions that was created by an ATM machine. Each transaction consists of six fields: timestamp, card number, amount, 5000 note count, 1000 note count, 500 note count. Store each transaction in an element of a dynamic array. Also, show code snippet that checks that the amount should tally with the note count. [10]

```
struct data_type{
        char timestamp[10] ;
        char card_number[10];
        float amount;
        int cnt5000;
        int cnt1000;
        int      cnt500;
};
// Read from file
data_type *arr; // = new data_type[10]; // allocation at once
int i, n=10;
ifstream ifile ("log.txt");
for ( i = 1; i < n; i++) {
        arr[i] = new data_type; // per element allocation
        ifile >>        arr[i].timestamp >> arr[i].card_number >> arr[i].amount >>
                                arr[i].cnt5000  >> arr[i].cnt1000 >> arr[i].cnt500;
}
ifile.close();
// Check amount with currency notes. Covert_amt function converts notes into amount
for ( i = 1; i < n; i++) {
        if (arr[i].amount != covert_amt(arr[i].cnt5000,arr[i].cnt1000,arr[i].cnt500))
                cout << "Recorded amount tally error.\n";
        else
                cout << "Recorded amount tally error.\n";
}
```

b) Assume that a mobile app stores (i) all incoming 160 characters long SMS messages and (ii) FIVE telephone numbers of family. Suggest a suitable data structure for both data. Now, show code snippet that checks all messages and generate alerts if the message belongs to a family member. [10]

```
struct sms_type {
   int msg_id;
   int phone_num;
   char msg_text [169];
};
struct fam_list_type {
   int fam_id;
   int phone_num;
};
//
sms_type sms[500]; // static array
fam_list_type fam_list[4]; // five elements
int j, k;
// suppose messages are placed by some background rountine into sms
array.
for (j=0; j < n; j++)
   for (k=0; k <= 4; k++)
      if (sms[i].phone_num == fam_list[k].phone_num)
         generate_alert(sms[i],k); // call routine to generate alerts.
```

Section 2 – Short reasoning (No diagram or code here) **Marks:** 20 points (20%)

Q2. Give short and to-the-point (3-4 lines max) reasons for the following. [2x10=20]

i. State benefits of chaining over linear probing when implementing hash tables.
Chaining evaluate hashing function and allocate or read the linked list element. In linear probing, hashing function is evaluated to locate the position and then you search hash table to find the value (search) or empty space (insert) after the collision i.e. more collisions will increase searching.

ii. State differences between Kruskal, Prim and Dijkstra algorithms. Don't write how they function.
Dijkstra calculates shortest path tree w.r.t a starting node, while krushal and Prims target calculating minimum spanning tree.

iii. State a scenario where you would avoid the use of linked lists.
If the overhead of storing and maintaining the linked list grows beyond a threshold then they become unaffordable. For example, storage of pointers and operations to access, insert, delete, and update elements.

iv. Graph or Tree. Choose and explain which you use to plot a water distribution scheme in cities.
Tree as we don't want any loops. Also, a minimum spanning tree can be cost effective.

v. Explain the algorithm you would use to describe the 2nd best path in a 5 node graph.
Dijkstra is used to select the shortest path (1st best path). If it is not available it will give you 2nd best path. Best path = Shortest path or Longest Path depends on the weights.

vi. How sorting can be performed without making comparisons?
Creating buckets and distributing number into buckets based on radix, like radix sort.

vii. Why would you favor breadth-first search over depth-first search while doing graph traversal?
Depends on the type of problem. If the requirement is visiting adjacent nodes w.r.t source, then BFS definitely favored.

viii. State benefits of using a built-in Tree Abstract Data Type (ADT) in your code.
Use of built-in ADT results in writing compact code and optimize execution time. Writing your ADT will be lot of coding and later debugging and optimizations.

ix. If the time comes and you have to prefer one over the other, how you justify the time complexity over space complexity and vice versa. Write arguments on both preferences.
Prefer time complexity. If you can pay experience programmer to code a time efficient system by sacrificing memory (which you have) without compromising requirements. Prefer space complexity. When you have limited memory you will tend to accept system with relative high complexity to meet the requirements.

x. Why do you think allocating multiple linked list elements at a time is a good idea? Assume that the linked list is actively used for incoming data.
If you need to allocate 1000 elements per second, the system will make 1000 calls to new. If you allocate 100 elements per call you need to make only 10 called for 1000 elements.

Section 3 – Dry run with tracing **Marks:** 20 points (20%)

Q3. Dry runs the following code snippets. Also trace key variables.

Note: Marks will be awarded if the grader feels satisfied with your understandings of the code snippets.

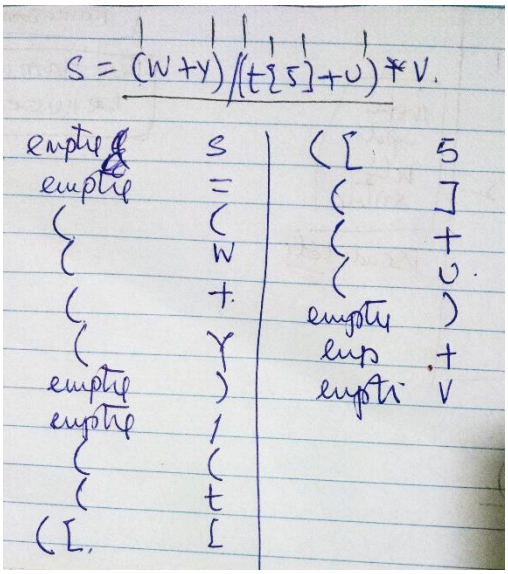
```
void code_snippetONE() {
    char what[80];
    register int p = 0;
    cin.get(what[p]);
    while (what[p] != '\n') cin.get(what[++p]);
    for (p -= 2; p >= 0; cout.put(what[p--]));
}
```

a) Assume suitable values where initializations are required or if they are provided by the user.

The char array what [80] will be filled from user standard input stream until a newline character is encountered. It then print the characters in reverse. Why the FOR loop doing p-=2; this probable means that we are reading characters which user terminated by a carriage return line feed. Detailed working need to be shown for convincing the grader in which he student needs to at least keep track variable p and able to identify what this function does.

b) Processing the statement s = (w+y)/(t[5]+u)*v read from a file; with the algorithm below.

```
delimiterMatching(file)
  read character ch from file;
   while not end of file
      if ch is '(', '[', or '{'
            push(ch);
      else if ch is ')', ']', or '}'
            if ch and popped off delimiter do not match
               failure;
      else if ch is '/'
            read the next character;
            if this character is '*'
                  skip all characters until "*/" is found and report an error
                  if the end of file is reached before "*/" is encountered;
            else ch = the character read in;
                  continue; // go to the beginning of the loop;
   // else ignore other characters;
      read next character ch from file;
   if stack is empty
        success;
   else failure;
```



See Figure 4.2 page 134 of text book. The statement has been changed. The student should be able to trace stack with respect to the input character as shown in the above working.

Section 4 – Scenarios (Do as directed) **Marks:** 40 points (40%)

Q4. Imagine A group of friends started a fund raising campaign for needy families. They set stalls at different locations in their city. They also make use of social media platforms to increase the reach. Many people started funding through their program. They have a list of needy families. This list is created with some priorities. Such as, if the family members are greater than 10 in one home, they should be funded earlier and better as compared to the home whose family members are only 3. Many of the needy families register themselves using their stalls. At the end of the day their task is to give priority to the families having more family members.

a) Your task is to explain which data structure is suitable for such type of scenario.
Priority Queue is suitable. The underlying data structure for priority queue is Binary heap.

b) Show step by step, how the following record of families (see Table 1) will get prioritized?

Day1	Fam1	Fam2	Fam3	Fam4	Fam5	Fam6	Fam7	Fam8	Fam9	Fam10
	4	5	2	8	12	3	11	9	10	13

Table 1: Record of families.

First all the records will be saved into an array using the logic of complete binary tree.
Secondly, the build heap function will be used to convert the complete binary tree into binary max heap.
In third step, binary heap sort will be applied to get the prioritized data.
All these steps must be performed by the students in the form of dry run.

Q5. Study the graph (see Figure 1) given below:

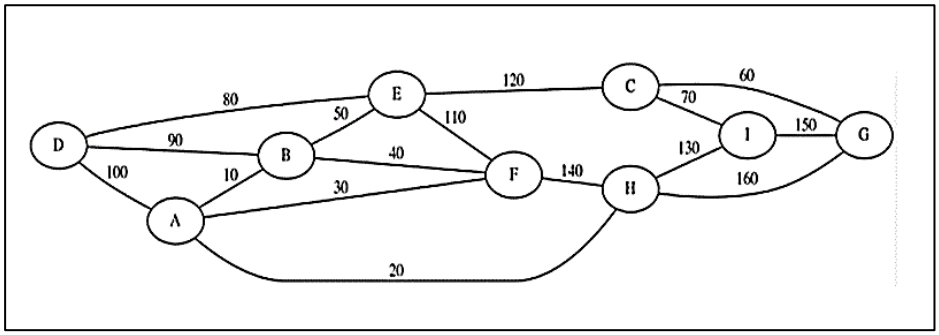


Figure 1: Graph

a) In the task **you are required to compute the MST using Prim's algorithm**. Specifically, record the edges of the MST in the request wherein Prim's algorithm adds them to the MST.

A-B	A-H	A-F	B-E	D-E	E-C	C-G	C-I
-----	-----	-----	-----	-----	-----	-----	-----

b) In the task **you are required to compute the MST using Kruskal's algorithm**. Specifically, record the edges of the MST in the request wherein Kruskal's algorithm adds them to the MST.

A-B	A-H	A-F	B-E	C-G	C-I	D-E	C-E
-----	-----	-----	-----	-----	-----	-----	-----

Working to drive the above values.

Q6. Consider the hash table with thirteen inserted values in it, as shown in Figure 3.

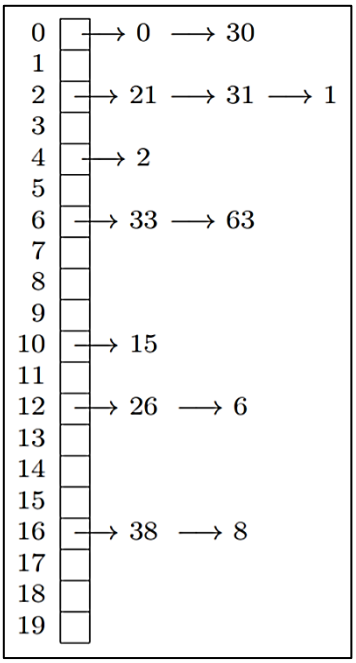


Figure 3: Hash Table

a) Identify the problem associated with the given hash table?
1- Unevenly distributed values over the entire hash table.
2- Numerous clashes which lead to a lot of vacant positions.

b) Provide an example of hash function that could lead to the problem that you identified in part a.
Findings:
1 – Values that are placed at the same indices have same last digit, for instance "21, 31, 01".
2 – Odd indices are vacant.
Example of a hash function that leads to above findings:
H (n) = 2n mod 20

c) Provide a better hash function that could resolve the problem.
By just discarding the coefficient "2" from the hash function H (n) = 2n mod 20, we can have completely different hash table that would not have a lot of clashes and vacant positions.

Q8. Imagine there are lots of people within a room and some of them are having corona. Let's say we have 1000 people overcrowded in this room with having 450 people having corona.

Now our task is to separate corona patients from non-corona patients so that the ones having corona can be shifted to another room and doctors can then deal with these patients accordingly. In order to send these corona patients in some sorted manner by having non-corona patients residing in the same room also having same sorted order. Your task is to explain a step by step procedure that what we should follow in order to achieve the task described above.

Restrictions: We CANNOT send these corona patients to another room one by one, thereby, we need to first sort these corona patients up, then only we can shift all these patients to some other room together.

Following questions should be answered while explaining the procedure. You can have some extra variables associated with each patient.

a) What sorting criteria do you intend to use and why? This means what value of patient will be taken into consideration for sorting them.
b) Which sort do you intend to use and why?
c) How do you intend to separate corona patients from non-corona patients?
d) What data structure would you prefer to store all corona and non-corona patients separately and why?

Answer:

The question mainly checks analytical skill of the student. We need to first sort the patients up. If some students sorts patients by name then this is wrong as to handle patients we always consider them on FCFS basis. In hospitals we are given a slip number which indicates this, so students should have this much sense that each patient node should have an extra variable named as patient# so that we may sort them accordingly. Next, the sorting approach should be merge sort as its stable and complexity is also O(nlogn). To separate corona patients from non corona ones each patient can have an extra Boolean variable which indicates it for example 1 for corona and 0 for non corona. Through this we can create two separate arrays/ linked list/trees which will store corona and non corona patients separately. Students should justify this data structure.

Q7. Implement a city database using a BST to store the database records. Each database record contains the name of the city (a string of arbitrary length) and the coordinates of the city expressed as integer x- and y-coordinates. The BST should be organized by city name. Your database should allow records to be inserted, deleted by name or coordinate, and searched by name or coordinate.

Answer:

Column # 1	Column # 2	Column # 3

```
class Node{
   public:
      string name;
      int x, y;
      Node *left, *right;
      Node(string city_n, int x_cord,int
y_cord) {
         this->name = city_n;
         this->x = x_cord;
         this->y = y_cord;
         this->left = nullptr;
         this->right = nullptr;
      };
};

class BST{
   private:
   public:
      Node *root;
      BST(): root(nullptr){}

      void InsertNode(Node*
&node,string name, int x, int y){
         if(node == nullptr)
            node = new Node(name, x ,
y);
         else if( node-
>name.compare(name) <= 0)
            InsertNode( node->left,
name, x, y);
         else
            InsertNode( node->right,
name, x, y);
      }
      Node* DeleteByName(Node*
&node, string name)
      {
         if(node == nullptr)
            return node;
         if(node->name.compare(name)
<= 0)
            node->left =
DeleteByName(node->left, name);
         else if(node-
>name.compare(name) > 0)
            node->right =
DeleteByName(node->right, name);

         else
         {
            if(node->left == nullptr)
               return node->right;

            else if (node -> right ==
nullptr)
               return node->left;

            Node *temp =
FindSmallest(node->right);
            node->name = temp->name;
            node->right =
DeleteByName(node->right, name);
         }
         return node;

      }
      Node* DeleteByCo(Node*
&node,int x, int y){
         if(node == nullptr)
            return node;
         Node * temp =
SearchByCo(node,x, y);
         DeleteByName(node, temp-
>name);
      }

      Node* SearchByName(Node*
&node, string name){
         if(node == nullptr)
            return node;

         int find = node-
>name.compare(name);
         if(find == 0)
            return node;
         else if (find < 0)
            return SearchByName(node-
>left, name);

         else
            return SearchByName(node-
>right, name);

      }

      Node* SearchByCo(Node*
&node, int x, int y){
         if(node == nullptr)
            return node;
         bool found = (node->x == x) &&
(node-> y == y);
         if(found)
            return node;
         else{
            Node *temp;
            temp = SearchByCo( node-
>left, x,  y) ;
            if(temp == nullptr)
               temp = SearchByCo(node-
>right, x,  y);
            return temp;
         }
      }
      Node *FindSmallest(Node*
&node){
         if(node == nullptr)
            return nullptr;
         Node *ans = node;
         if(ans->left != nullptr)
            ans = FindSmallest(ans->left);
         return ans;
      }
};
```

---------- (X) ----------