PART (A)

You are not required to write any code for the questions given in part A.

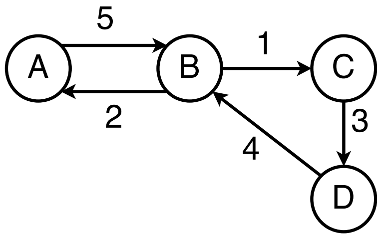
Q1. How an empty AVL tree will look like after inserting the following values in the given order: [4]
3, 2, 10, 8, 11, 9, 6, 4, 7, 1, 5.

Q2. Sort the following values using heap sort, show all the steps for heapification and sorting: [4]
3, 2, 10, 8, 11, 9.

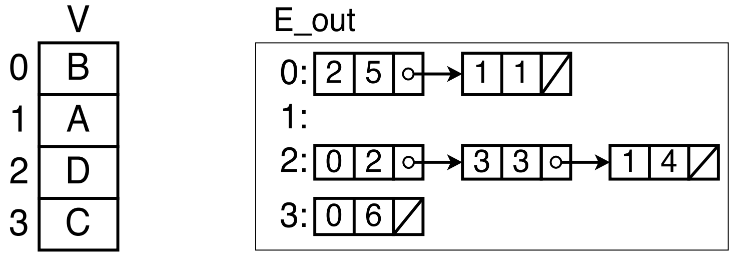
Q3. How an empty hash table of size 9 using open addressing, division method, and double hashing would look like after inserting the following values in the given order. For the second hash function, use the formula `1+(k%(m-2))`, where *m* is the size of the hash table. The values are: 23, 5, 50, 32, 77. [4]

Q4. [2+2=4]

a) Given the following graph, draw its corresponding adjacency matrix.



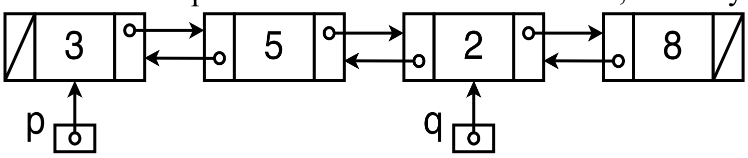
b) Given the following adjacency list, draw its corresponding graph.



Q5. The following code uses an STL based list to store values. How would the list look like at the end of the code? Use boxes to draw the list. [4]

```
list<pair<pair<int,int>,int> >  L;
for (int i=1;i<=5;i++) {
    pair<int,int> p(i*2,i*3);
    pair<pair<int,int>,int> q(p, p.first + p.second);
    L.push_back(q);
}
```

Q6. Given the following doubly linked list. What will be the result of the given expressions? In case of an error, mention whether it is a compile time error or a run-time error, and why? [5]



a) `p->next->prev->data`
b) `p->prev->next->data`
c) `p->next == q->prev`
d) `q->next->next->prev->data`
e) `q->prev->data == p->next`

f) `p->data + q->next->data`
g) `q->prev->next->data+p->next->prev->data`
h) `p->next->data * q->prev`
i) `p->prev == q->next->next`
j) `q->prev->data * q->next->data`

Q7. Given the following code. Draw the stack, queue, and tree on your paper wherever asked in the code. Assume that the BST implementation is unbalanced. [1+2+2=5]

```
BST<int> t;     stack<int> s;    queue<int> q;
for (int i=2; i<=10; i=i+2)
    s.push(i);
//(a) ***Show the stack s on your paper*** //

bool flip = true;
while (!s.empty()) {
    int n = s.top();
    if (flip)
        q.push(n*2);
    else
        q.push(n-1);
    flip = !flip;
    s.pop();
}
//(b) ***Show the queue q on your paper*** //

while (!q.empty()) {
    t.insert(q.front());
    q.pop();
}
//(c) ***Show the BST t on your paper*** //
```

PART (B)

For the part B, only write the code which is asked. Do not write the code for the whole ADTs.

Q8. Write ADT code for a function *int weighted_degree(vtype v)* for a Directed Graph ADT implemented using Adjacency List. The function should take a vertex *v* as input and return the sum of the weights of the edges going outwards from *v*. For example, for the graph given in Q4(a), calling *weighted_degree('B')* should return 3. Only write the code for the function *weighted_degree*. Do not write code for the whole ADT. [10]

Q9. Write ADT code for a function *void pop_k(int k)*, which removes the k^{th} element from a stack. Assume that the Stack ADT is implemented using linked structures. As an example, if the stack contains the values 3,7,8,5 (with 3 on top of the stack), calling *pop_k(2)* would remove the 2^{nd} element, i.e., '7' from the stack and the values 3,8,5 would remain on the stack. Only write the code for the function *pop_k*. Do not write code for the whole ADT. [10]