CS 101 Exam 2 **Summer 2016** **Solution**

1. Write the output of this C++ program on the indicated blanks. **[10 points]**

```
#include <iostream>
#include <deque>       // same as deck
using namespace std;
int main( ) {
    deque<int> D1, D2;
    D1.push_front(0);
    D1.push_front(1);
    D1.push_back(2);
    D1.push_front(3);
    D1.push_front(4);
    D2.push_back(5);
    D2.push_back(6);
    D2.push_front(7);
    D2.push_back(8);
    D2.push_back(9);
    cout << D1.back( ) << endl;
    D1.pop_back( );
    cout << D2.front( ) << endl;
    D2.pop_front( );
    cout << D1.back( ) << endl;
    D1.pop_back( );
    cout << D2.front( ) << endl;
    D2.pop_front( );
    cout << D1.front( ) << endl;
    D1.pop_front( );
    cout << D2.back( ) << endl;
    D2.pop_back( );
    cout << D1.back( ) << endl;
    D1.pop_back( );
    cout << D2.front( ) << endl;
    D2.pop_front( );
    cout << D1.back( ) << endl;
    D1.pop_back( );
    cout << D2.front( ) << endl;
    D2.pop_front( );
    return 0;
}
```

Output:

2
7
0
5
4
9
1
6
3
8

2. Complete the methods in the C++ class List as specified below. Note: you will write some of the methods using two different approaches with different requirements. **[30 points]**

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

class List {        // Singly-linked circular list with header node
    Node *header;
public:
    List( );
    Node *insertAfter (int, Node *);
    Node *insertBefore (int, Node *);
    Node *remove (Node *);
    ...
};

List::List( ) {
    header = new Node;
    header->next = header;
}

Node * List::insertAfter (int x, Node *p) {
    // Add value x into the list so that x is after the data in node p.
    // Return the node that contains value x.
    // Runs in O(1) time.

    Node *q = new Node;
    q->data = x;
    q->next = p->next;
    p->next = q;
    return q;

}

Node * List::insertBefore (int x, Node *p) {
    // Add value x into the list so that x is before the data in node p.
    // Return the node that contains value x.
    // It is not permitted to modify data fields of pre-existing nodes.
    // Runs in O(n) time.

    Node *t = header;
    while (t->next != p)  t = t->next;
    Node *q = new Node;
    t->next = q;
    q->data = x;
    q->next = p;
    return q;

}
```

```
Node * List::insertBefore (int x, Node *p) {
     // Add value x into the list so that x is before the data in node p.
     // Return the node that contains value x.
     // It is permitted to modify data fields of pre-existing nodes.
     // Runs in O(1) time.

     Node *q = new Node;
     q->data = p->data;
     q->next = p->next;
     p->data = x;
     p->next = q;
     return p;

}

Node * List::remove (Node *p) {
     // Remove the data value in node p from the list.
     // Delete the node which is unlinked from the list.
     // Return the node that contains the data which had been after node p in the list.
     // It is not permitted to modify data fields of pre-existing nodes.
     // Runs in O(n) time.
     if (p==header || p==NULL)  return NULL;

     Node *t = header;
     while (t->next != p)  t = t->next;
     t->next = p->next;
     delete p;
     return t->next;

}

Node * List::remove (Node *p) {
     // Remove the data value in node p from the list.
     // Delete the node which is unlinked from the list.
     // Return the node that contains the data which had been after node p in the list.
     // It is permitted to modify data fields of pre-existing nodes.
     // Runs in O(1) time.
     if (p==header || p==NULL)  return NULL;

     Node *q = p->next;
     p->data = q->data;
     p->next = q->next;
     delete q;
     return p;

}
```

3. Write a C++ program as follows. First write a class template called Pair that can be instantiated to hold any pair of values, such that each value can be any type, and such that *the two values can have two different types*. The template should provide a constructor, two simple access methods, and also a mutator method that modifies the values. Next write a main function that creates and uses two pairs. The first pair should hold an integer value and a string value, and the second pair should hold a character value and a double value. Invoke the access methods and print the two values in each pair. Also invoke the mutator method on each pair instance. **[20 points]**

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

template <typename Type1, typename Type2>
class Pair {
        Type1 x;
        Type2 y;
public:
        Pair (Type1 a, Type2 b) { x=a; y=b; }
        Type1 getX ( ) { return x; }
        Type2 getY ( ) { return y; }
        void setXandY (Type1 a, Type2 b) { x=a; y=b; }
};

int main ( ) {
        Pair <int, string> p (7, "hello");
        cout << p.getX( ) << endl;
        cout << p.getY( ) << endl;
        p.setXandY (11, "goodbye");
        cout << p.getX( ) << endl;
        cout << p.getY( ) << endl;

        Pair <char, double> q ('A', 3.14159);
        cout << q.getX( ) << endl;
        cout << q.getY( ) << endl;
        q.setXandY ('Z', 2.71828);
        cout << q.getX( ) << endl;
        cout << q.getY( ) << endl;
        return 0;
}
```

4. Write the output of this C++ program on the indicated blanks. **[20 points]**

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

class Ex1 {
      const char *msg;
public:
      Ex1 (const char *m): msg(m) { }
      const char *what( ) const {return msg;}
};
class Ex2 {
      int value;
public:
      Ex2 (int v): value(v) { }
      int getValue ( ) { return value; }
};
class Ex3 { };

int F (int a, int b) {
      if (a==0) throw new Ex3;
      if (a==3) return 10*b;
      if (a>b) throw new Ex2(a-b);
      if (a<b) throw new Ex1("Less");
      return a+b;
}
int main ( ) {
      for (int j=-1; j<=3; j++) {
            for (int k=-1; k<=2; k++) {
                  try { cout << F(j,k); }
                  catch (Ex1 *ex1) { cout << ex1->what( ); }
                  catch (Ex2 *ex2) { cout << ex2->getValue( ); }
                  catch ( ... ) { cout << "Error"; }
                  cout << '\t';
            }
            cout << endl;
      }
}
```

Output:

```
-2      Less    Less    Less
Error   Error   Error   Error
2       1       2       Less
3       2       1       4
-10     0       10      20
```

5. The two hash tables shown below each have capacity N = 11 and hash function H(x) = x%11. Note that the keys 16 and 20 have already been inserted. Now insert these additional keys into each hash table in the specified order: 60, 38, 71, 75, 35. **[10 points]**

Quadratic probing

0	**35**
1	
2	**75**
3	**38**
4	
5	16
6	**60**
7	
8	
9	20
10	**71**

Double hashing with H′(x) = 7 – x%7

0	**71**
1	**35**
2	**38**
3	
4	**75**
5	16
6	
7	
8	**60**
9	20
10	

Suppose we want to remove the key 16 from these hash tables. Explain the usual method for removing a key, and state its main advantage(s) and disadvantage(s). **[5 points]**

Replace key 16 with a special value DELETED that is distinct from an empty slot. Advantage is simplicity, because the alternative might require rearranging many keys. Disadvantage is inefficiency because eventually no empty slots will remain in the table, and so searching for a given key can require examining every slot in the table.

Now suppose we allow insertions but not removals. Let n denote the current number of keys in the hash table, and let N denote the current capacity of the table. Suppose we want to always maintain that $2n \le N \le 4n$ (so the *load factor* always remains between 25% and 50%). Explain how to vary the capacity of the hash table to accomplish this. **[10 points]**

After doing each insertion:

```
n++;
if (2*n > N) {
    allocate a new hash table with capacity 2*N;
    rehash all n keys into the new hash table;
    N = 2*N;
    discard the old hash table;
}
```