CS 101 Exam 2 **Summer 2017** **Solution**

This exam has 120 points possible (includes 20 built-in extra credit points).

1. Write the output of this C++ program on the indicated blanks. **[14 points]**

```
#include <iostream>
#include <deque>
using namespace std;
int main( ) {
    deque<int> D1, D2;
    D1.push_front(0);
    D2.push_back(1);
    D1.push_front(2);
    D2.push_back(3);
    D1.push_back(4);
    D2.push_front(5);
    D1.push_front(6);
    D2.push_back(7);
    D1.push_front(8);
    D2.push_back(9);
    deque<int>::iterator it;
    for (it=D1.begin( ); it!=D1.end( ); it++)
        cout << *it;
    cout << endl;
    deque<int>::reverse_iterator rev;
    for (rev=D2.rbegin( ); rev!=D2.rend( ); rev++)
        cout << *rev;
    cout << endl;
    cout << D1.back( ) << endl;
    D1.pop_back( );
    cout << D1.back( ) << endl;
    D1.pop_back( );
    cout << D1.front( ) << endl;
    D1.pop_front( );
    cout << D1.back( ) << endl;
    D1.pop_back( );
    cout << D1.back( ) << endl;
    D1.pop_back( );
    cout << D2.front( ) << endl;
    D2.pop_front( );
    cout << D2.front( ) << endl;
    D2.pop_front( );
    cout << D2.back( ) << endl;
    D2.pop_back( );
    cout << D2.front( ) << endl;
    D2.pop_front( );
    cout << D2.front( ) << endl;
    D2.pop_front( );
    return 0;
}
```

Output:

86204

97315

4

0

8

2

6

5

1

9

3

7

2. We want to convert the C++ class Array given below into a template class that can be instantiated using different array element types and also different array lengths. Show precisely all the additions and changes that are needed in the code below. **[20 points]**

Solution one:

```
template <typename T>
class Array {
      T *data;
public: int n;
      Array (int m);
      T & operator[ ] (int k);
};

template <typename T >
Array<T> :: Array (int m) {
      data = new T[m];
      for (int k=0; k<m; k++) data[k] = T( );
      n = m;
}

template <typename T>
T & Array<T> :: operator[ ] (int k) {
      return data[k];
}

template <typename T>
ostream & operator<< (ostream & os,
                              Array<T> & z) {
      os << "[ ";
      for (int k=0; k<z.n; k++) os << z[k] << " ";
      os << "]";
      return os;
}
int main( ) {
      Array <int> a (1000);
      a[999]=5;
      cout << a[999] << endl;
      cout << a << endl;
}
```

Solution two:

```
template <typename T, int N>
class Array {
      T data[N];
public:
      Array( );
      T & operator[ ] (int k);
};

template <typename T, int N>
Array<T, N> :: Array( ) {
      for (int k=0; k<N; k++) data[k] = T( );
}

template <typename T, int N>
T & Array<T, N> :: operator[ ] (int k) {
      return data[k];
}

template <typename T, int N>
ostream & operator<< (ostream & os,
                              Array<T, N> & z) {
      os << "[ ";
      for (int k=0; k<N; k++) os << z[k] << " ";
      os << "]";
      return os;
}
int main( ) {
      Array <int,1000> a;
      a[999]=5;
      cout << a[999] << endl;
      cout << a << endl;
}
```

3. Write the exact output of this C++ program when the input is: `-2 7 0` **[18 points]**

```
class SomeError {
        string msg;
public:  SomeError (string m):  msg(m) { }
        string what( ) { return msg; }
        string to_string( ) { return "SomeError (" + msg + ")"; }
};
class ZeroError {
        string msg;
public:  ZeroError( ):  msg("arg is zero") { }
        string what( ) { return msg; }
        string to_string( ) { return "ZeroError (" + msg + ")"; }
};
int my_function (int param) {
        if (param < 0) {
                throw SomeError("param is negative");
                cout << "Leaving if statement" << endl;
        }
        else if (param == 0) {
                throw ZeroError( );
                cout << "Leaving else clause" << endl;
        }
        cout << "Leaving my_function" << endl;
        return param/2;
}
int main ( ) {
        int k;
        while (cin >> k)
                try {
                        int n = my_function (k);
                        cout << "Leaving try block with n = " << n << endl;
                } catch (SomeError &e) {
                        cout << "Exception occurred: " << e.to_string( ) << endl;
                } catch (ZeroError &e) {
                         cout << "Received exception: " << e.to_string( ) << endl;
                } catch (...) {
                        cout << "Catches all exceptions" << endl;
                }
}
```

Exception occurred: SomeError (param is negative)
Leaving my_function
Leaving try block with n = 3
Received exception: ZeroError (arg is zero)

4. Suppose the class Queue is defined as follows.

```
class Queue {
    Queue( );
    bool isEmpty( );
    void enqueue (int);
    int dequeue( );
}
```

Write a C++ program that solves this problem: Read an unknown number of integer values from standard input stream. If the number of input values is odd, output the values at the odd positions (1st value, 3rd value, 5th value, etc). But if instead the number of values is even, output the values at the even positions (2nd value, 4th value, 6th value, etc). Your program should use the class Queue, but do not use arrays or lists or any other data structure except Queue. Examples: If the input stream is 10 20 30 40 50, then the output stream should be 10 30 50. But if the input stream is 10 20 30 40 50 60, then the output stream should be 20 40 60. **[12 points]**

```
int main( ) {
    Queue q;
    int k;
    bool odd=false;
    while (cin >> k) {
        q.enqueue (k);
        odd = !odd;
    }
    while (! q.isEmpty( )) {
        k = q.dequeue( );
        if (odd) cout << k << " ";
        odd = !odd;
    }
    return 0;
}
```

5. Suppose the class Stack is defined as follows.

```
class Stack {
    Stack( );
    bool isEmpty( );
    void push (int);
    int pop( );
}
```

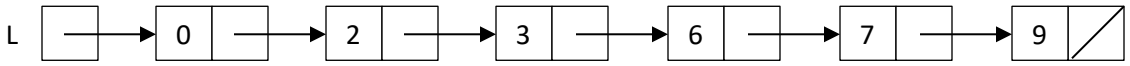
Write a C++ program that solves this problem: Read an unknown number of integer values from standard input stream. Next output these same values in the reverse order, and then again in original order. Your program should use the class Stack, but do not use arrays or lists or any other data structure except Stack. Example: if the input stream is 10 20 30 40, then the output stream should be 40 30 20 10 10 20 30 40. **[12 points]**

```
int main( ) {
    Stack s1, s2;
    int k;
    while (cin >> k)
        s1.push (k);
    while (! s1.isEmpty( )) {
        k = s1.pop( );
        cout << k << " ";
        s2.push (k);
    }
    while (! s2.isEmpty( )) {
        k = s2.pop( );
        cout << k << " ";
    }
    return 0;
}
```

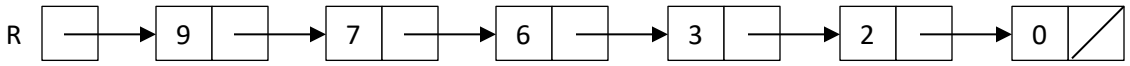
6. Suppose this definition is provided for a singly-linked list:

```
struct Node  {  int data;  Node *next;
                  Node (int d, Node *x)  { data = d; next = x; }
};
class List {  Node *head;
public:       List (Node *h=NULL)  { head = h; }
              List reverse( ) { … }
};
```

Complete the method reverse( ) so that it returns a new list with the data values in reversed order. However, this method must not modify the original list. Example: Suppose the list L is as follows.



Next suppose we call R = L.reverse ( ). List L remains the same, and the list R is built as follows.



a. First write the method reverse( ) in C++ using iteration. Do not use recursion. **[10 points]**

```
List reverse( ) {
      Node *q = NULL;
      for (Node *p=head; p!=NULL; p=p->next) {
            Node *t = new Node (p->data, q);
            q = t;
      }
      return List (q);
}
```

b. Next write the method reverse( ) in C++ using recursion. Do not use iteration. **[10 points]**

```
List reverse( ) {
      Node *r = rev (head, NULL);
      return List (r);
}
Node *rev (Node *p, Node *q) {
      if (p==NULL) return q;
      Node *t = new Node (p->data, q);
      return rev (p->next, t);
}
```

7. Given an initially empty hash table with capacity 13 and hash function H (x) = x % 13.
Suppose keys 36, 37, 38, 39 are already inserted as shown.
Next insert keys 50, 51, 49, 24, 23, 25, 65, 78 (in that order). **[24 points]**

a. Separate chaining

0		→39→**65**→**78**
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		→36→**49**→**23**
11		→37→**50**→**24**
12		→38→**51**→**25**

b. Linear probing

0	39
1	**50**
2	**51**
3	**49**
4	**24**
5	**23**
6	**25**
7	**65**
8	**78**
9	
10	36
11	37
12	38

c. Quadratic probing

0	39
1	**49**
2	**50**
3	**51**
4	**65**
5	
6	**23**
7	**24**
8	**25**
9	**78**
10	36
11	37
12	38

d. Double hashing using
H’ (x) = 7 – (x % 7)

0	39
1	**65**
2	**24**
3	
4	**50**
5	**49**
6	**78**
7	**23**
8	**25**
9	**51**
10	36
11	37
12	38