

Name: _____	CS Username: _____
Grade: _____	Blackboard Username: _____

- Be sure to begin by printing your *name* and both *usernames* clearly in the spaces provided above.
- If you find a question ambiguous: write the most reasonable assumptions you can think of near the question, and then answer the question under these assumptions.
- The exam is worth 200 points. The points available for individual questions are in parentheses at the end of the question.

1. Suppose w is an 8-bit word with bits numbered from right to left starting with bit 1, as: $w = b_8b_7b_6b_5b_4b_3b_2b_1$. You are to complete the implementation for the function `BitValue(w,i)` that returns an integer representation of bit b_i in word w :

```
int BitValue(char w, int i)
{
    int rval;
    m = _____ ; // a) 1 line of code defining mask

    rval = _____ ; // b) 1 line of code defing rval

    return rval;
}
```

- (a) Show a calculation of a mask m for the i^{th} bit b_i of w . (*Insert answer in code above.*) (10 pts)
- (b) Show a calculation of an integer that is zero if and only if $b_i = 0$. (*Insert answer in code above.*) (10 pts)

2. This question concerns the template class `TVector<T>`. Please ignore namespace issues. (*A declaration of `TVector<T>` is appended to this exam.*)

- (a) Declare a vector object V with `int` `value_type`, size 20, and initial element values equal to 0: (10 pts)

- (b) Complete the following loop code so that V stores user input until 0 is entered: (10 pts)

```
while (cin >> entry)
{
    if (entry != 0)
        _____ // add 1 line of code here

} // V holds user input
```

3. This question concerns the template class `TList<T>`. Please ignore namespace issues. (*A declaration of `TList<T>` is appended to this exam.*)

(a) Declare a list object `L` of value_type `int`: (10 pts)

(b) Complete the code implementing `TList<T>::PushBack()`: (20 pts)

```
template < typename T >
int TList<T>::PushBack (const T& t)
// Insert t at the back (last) position.
{
    TLink* newLink = new TLink(t);
    if (!newLink)
    {
        std::cerr << "** TList error: memory allocation failure\n";
        return 0;
    }

    if (lastLink == 0)
    // Insert into an empty TList.
    {
                                                // insert code here (1 or 2 lines)
    }
    else
    // Insert at back of a non-empty list
    {
                                                // insert code here (3 lines)
    }
    return 1;
} // end PushBack()
```

4. The following represents output from the call `D.Dump()` of a `TDeque<char>` object `D`:

```
c d z s g k
0 1 2 3 4 5
e      b
```

(a) What is the result of the output statement `std::cout << D; ?` (10 pts)

(b) Show the result of `D.Dump()` after the two operations `D.PushFront('x')`; and `D.PopBack()`;.
(10 pts)

5. Name one public operation (or minimal set of public operations) the ADT has that the other two do not have: (30 pts)
- (a) `TVector<>`:
 - (b) `TDeque<>`:
 - (c) `TList<>`:
6. Use a stack to evaluate postfix expressions:
- (a) Use `CStack<>` to declare a list-based stack `S` of value_type `int`: (10 pts)
 - (b) Show, either graphically or with calls to the public interface of `S`, the process of using `S` to evaluate the expression `1 2 3 * +`. (10 pts)

7. Abstract Data Types (ADTs)

- (a) Name the three components of ADT. (10 pts)
 - 1.
 - 2.
 - 3.
- (b) What are the critical properties of ADTs Stack and Queue that distinguish them from each other? (Name and briefly describe both properties.) (10 pts)

8. Binary Search. This is an illustration of a vector of characters to be used in answering the questions:

```
element:  b d g k p v
index:    0 1 2 3 4 5
```

- (a) What is the return value of the call `lower_bound('f')`? (10 pts)
- (b) Show (by carefully underscoring) the search range in each step of the lower bound algorithm set in motion by the call above. Show one range for each iteration of the loop, and leave the remaining unmarked. (10 pts)

```
element:  b d g k p v
index:    0 1 2 3 4 5
```

```
element:  b d g k p v
index:    0 1 2 3 4 5
```

```
element:  b d g k p v
index:    0 1 2 3 4 5
```

```
element:  b d g k p v
index:    0 1 2 3 4 5
```

```
element:  b d g k p v
index:    0 1 2 3 4 5
```

```
element:  b d g k p v
index:    0 1 2 3 4 5
```

9. What is the runtime of each of the following algorithms, assuming the n is the size of the container? (Choose from among the answers provided.) (5 pts each)

- (a) `TVector<T>::PushBack(const T& t)`
- (b) `TList<T>::Insert(Iterator& I, const T& t)`
- (c) `lower_bound (const TVector<T>& V, const T& t)`
- (d) `sequentail_search (const TList<T>& V, const T& t)`

Possible Answers	
A. $O(1)$	B. Amortized $O(1)$
C. $O(\log n)$	D. Amortized $O(\log n)$
E. $\Theta(\log n)$	F. Amortized $\Theta(\log n)$
G. $O(n)$	H. Amortized $O(n)$
I. $\Theta(n)$	J. Amortized $\Theta(n)$
K. None of the above	

```

template <typename T>
class TVector
{
    friend class TVectorIterator<T>;

public:
    // terminology support
    typedef T          value_type;
    typedef TVectorIterator<T> Iterator;

    // constructors - specify size and an initial value
    TVector ();
    explicit TVector (size_t);
    TVector (size_t, const T&);
    TVector (const TVector<T>&);
    virtual ~TVector ();

    // member operators
    TVector<T>& operator = (const TVector<T>&);
    TVector<T>& operator += (const TVector<T>&);
    T&          operator [] (size_t) const;

    // other methods
    int      SetSize      (size_t);
    int      SetSize      (size_t, const T&);
    int      SetCapacity  (size_t);
    size_t   Size         () const;
    size_t   Capacity     () const;

    // Container class protocol
    int      Empty        () const;
    int      PushBack     (const T&);
    int      PopBack      ();
    void     Clear        ();
    T&       Front        () const;
    T&       Back         () const;

    // Iterator support
    Iterator Begin        () const;
    Iterator End          () const;
    Iterator rBegin       () const;
    Iterator rEnd         () const;

protected:
    // data
    size_t size, capacity;
    T* content; // pointer to the primitive array elements

    // method
    static T* newarray (size_t); // safe space allocator
};

```

```

template < typename T >
class TList
{
    friend class TListIterator<T>;

public:
    // terminology support
    typedef T          value_type;
    typedef TListIterator<T>  Iterator;

    // constructors and assignment
    TList (); // default constructor
    virtual ~TList (); // destructor
    TList (const TList<T>& L); // copy constructor
    TList<T>& operator = (const TList<T>& L); // assignment

    // modifying List structure
    int      PushFront (const T& t); // Insert t at front of list
    int      PushBack  (const T& t); // Insert t at back of list
    int      Insert    (Iterator& I, const T& t); // Insert t at I
    Iterator Insert    (const T& t); // Insert t

    int      PopFront  (); // Remove the Tval at front
    int      PopBack   (); // Remove the Tval at back
    int      Remove    (Iterator& I); // Remove item at I
    size_t   Remove    (const T& t); // Remove all copies of t

    void     Clear     (); // Empty the list, deleting all elements

    // information about the List
    size_t   Size      () const; // return the number of elements on the list
    int      Empty     () const; // true iff list has no elements

    // accessing values on the List
    T&       Front     () const; // return reference to Tval at front of list
    T&       Back      () const; // return reference to Tval at back of list

    // locating places on the list
    Iterator Begin     () const; // return iterator to front
    Iterator End       () const; // return iterator past the back
    Iterator rBegin    () const; // return iterator to back
    Iterator rEnd      () const; // return iterator past the front
    Iterator Includes  (const T& t) const; // position of first occurrence of t

// end of TList public interface

```

```

// continuing with TList implementation data

protected:

// A scope TList<T>:: class usable only by its friends (all members are
private)
class TLink
{
    friend class TList<T>;
    friend class TListIterator<T>;

    // TLink data
    T      value;          // data
    TLink * prevLink;     // ptr to predecessor Link
    TLink * nextLink;     // ptr to successor Link

    // TLink constructor - parameter required
    TLink(const T& Tval);
} ;

// structural data
TLink * firstLink, // pointer to the first element in the list
      * lastLink;  // pointer to the last element in the list

// protected method -- used only by other methods
void Clone (const TList<T>& L); // makes *this a clone of L
} ;

```