This test has the following sections:

I. True/False .......................... 28 points; (14 questions, 2 points each)
II. Short Multiple Choice ...... 30 points; (10 questions, 3 points each)
III. Long Multiple Choice ..... 42 points; ( 7 questions, 6 points each)

---------------
100 points total

This test is worth 20% of your final grade. You must put your answers on the answer form using a #2 pencil. You must turn in both the answer form and the test in order to get a grade. Be sure to put your name on your test as well as on your answer form. Be sure to have your UIN written on your answer form under the space for code number, left justified, no spaces. You may assume code is in C++ unless specified otherwise. This test is open book and open notes. You may assume that segments of code are part of a larger working program. You have two hours.

I. True/False: (2 points each) On your answer form fill out A for true and B for false.

T F 1. The code shown below will compile and run.

```c++
char problem1()
{
    return 'a';
}

int main()
{
    int x;
    x = problem1();
    // rest of program...
}
```

T F 2. The output from the following code segment is: Problem 2: 6

```c++
void foo2( int *pArray)
{
    pArray = pArray + 2;
    (*pArray)++;
}

void Problem12()
{
    int theArray[  ] = {1,3,5,7,8,9,10};
    foo2( theArray);
    cout << "Problem 2: " << theArray[ 2] << endl;
}
```

T F 3. A sequence of if statements can always be rewritten as a switch statement.
4. Assume we had the following declarations in a C++ program:

```cpp
struct Person {
    char name[25];
    int age;
};

Person p1 = {"Pat", 13};
Person *pPerson = &p1;
```

The following code would compile and run and allow storing first name and age properly.

```c
printf("Enter first name and age: ");
scanf("%s %d", p1.name, &(pPerson->age));
```

5. Given the following function declaration:

```c
void add( int newNumber, int * &pArray, int &size)
{
    int *pNewArray = new int[ size + 1];
    for( int i=0; i<size; i++) {
        pNewArray[ i] = pArray[ i];
    }
    pNewArray[ size] = newNumber;
    delete( pArray);
    pArray = pNewArray;
    size++;
}
```

function add(…) shown above could be called and used with the following code:

```c
int size = 3;
int *pNumbers = new int[ size];
pNumbers[ 0] = 3;
pNumbers[ 1] = 5;
pNumbers[ 2] = 7;
add( 7, pNumbers, size);
```

6. The following function declaration would compile successfully:

```c
void fooA( int values[][], int key)
{
    // More code here ...
}
```

7. Consider the function declaration shown below:

```c
void fooB( char names[][15], int salary)
{
    // More code in here ...
}
```

Changes to array names will automatically be reflected back to the calling part of the program.
For the following 3 problems, consider the declarations:

```c
struct Employee {  
    int age;  
    float hourlyWage;  
};  
Employee anEmployee;  
Employee *pEmployee = &anEmployee;  
anEmployee.age = 21;  
anEmployee.hourlyWage = 8.0;
```

T F 8. The age of anEmployee can be displayed using: cout << pEmployee.age;
T F 9. The age of anEmployee can be displayed using: cout << *pEmployee.age;
T F 10. The age of anEmployee can be displayed using: cout << anEmployee.age;

For the next problem assume the code shown at right below, where function `swapValues` is called.

```c
void swapValues(int x, int y)  
{  
    int temp = x;  
    x = y;  
    y = temp;  
}
```

```c
int x = 3;  
int y = 5;  
swapValues(x, y);  
cout << "Values are: "  
    << x << " " << y << endl;  
// ... other code
```

T F 11. Output of the above segment of code is: Values are: 5 3

For the next three problems assume the following struct declaration is used and a list is initialized using a sentinel node, so there is always at least that one node on the list.

```c
struct Node {  
    int data;  
    Node *pNext;  
};
```

T F 12. The function header to prepend nodes to the beginning of a list could be implemented as:

```c
Node * prepend(int value, Node *pHead)  
{  
    // More code would be here...
}
```

T F 13. The function declaration to prepend nodes to the beginning of a list could be implemented as:

```c
void prepend(int value, Node * &pHead)  
{  
    // More code would be here...
}
```

T F 14. The function declaration to append nodes to the end of a list could be implemented as:

```c
void append(int value, Node * pHead)  
{  
    // More code would be here...
}
II. Multiple Choice: Short Problems (3 points each)

15. What is the output from the following segment of code?

```cpp
char text[] = "All generalizations are false";
char first[81];
char second[81];
char *pCurrent = text;
char *pSpace;

pSpace = strrchr(pCurrent, ' ');
strcpy(first, pSpace);
*pSpace = '.';
pCurrent = strrchr(pCurrent, ' ');
*pSpace = NULL;
strcpy(second, pCurrent);

cout << first << " " << second << endl;
```

a) All generalizations
b) generalizations All
c) are false
d) false are
e) None of the above

16. When a 2-D array is passed to a function, in the function declaration the size of the first parameter may be left blank, but the second dimension must be supplied. Why is this?

a) Code in C can overwrite the end of an array, however for a 2-D array the formula used to find the ith row needs to know how many columns are on each row.
b) The size of the first dimension is always automatically supplied, even when not specified by the user, since it is included as part of the definition of every array.
c) A NULL character is always inserted at the end of every row so that the compiler can tell where one row ends and the next begins, so the size of the first dimension is not necessary.
d) The total size of the array divided by the size given in the second dimension is used internally to calculate the number for the first dimension, so the user need not supply it.
e) None of the above

17. Consider the function shown at right below. For positive integers, how would you best describe the return value of calling function `first`?

```cpp
double first( int x, int n)
{
    if( n<=0 ) {
        return 1;
    }
    return first( x, n-1 ) * x;
}
```

a) x
b) x + y
c) x * x
d) x^n
e) None of the above
The interface that we see for a data structure (operations that we can perform) are different from the implementation of the data structure (how it actually works under the surface.) For each of the following three scenarios select the implementation that would be most efficient when working with a series of nodes such as we used in class.

18. Adding (push) and deleting (pop) the most recent nodes to allow undoing a move

a) Singly-linked list with only a head pointer, where nodes are prepended to the beginning.
b) Singly-linked list, with head and tail pointers, where nodes are appended to the end of the list.
c) Doubly-linked list, with head and tail pointers
d) Dynamically growing array, where list items are stored in order.
e) Circularly linked list, where the last node points to the first node

19. Accessing a location by an index value (e.g. 3rd node, or nth node)

a) Singly-linked list with only a head pointer, where nodes are prepended to the beginning.
b) Singly-linked list, with head and tail pointers, where nodes are appended to the end of the list.
c) Doubly-linked list, with head and tail pointers
d) Dynamically growing array, where list items are stored in order.
e) Circularly linked list, where the last node points to the first node

20. Inserting in-order into existing nodes

a) Singly-linked list with only a head pointer, where nodes are prepended to the beginning.
b) Singly-linked list, with head and tail pointers, where nodes are appended to the end of the list.
c) Doubly-linked list, with head and tail pointers
d) Dynamically growing array, where list items are stored in order.
e) Circularly linked list, where the last node points to the first node

21. Consider the struct declaration at left below that is used to implement a doubly-linked list. If there is already at least one node on the list and we want to insert a new node before the current node, which of the following is true about the insertNode(...) function?

| struct Node { |
| Node* pPrev; |
| int data; |
| Node* pNext; |
| } | void insertNode( Node * &pCurrent, int input) |
| { Node *pTemp = new Node; |
| pTemp->data = input; |
| pTemp->pPrev = pCurrent->pPrev; |
| pTemp->pNext = pCurrent; |
| pCurrent->pPrev = pTemp; |
| }

a) It works correctly as shown.
b) It does not work correctly but could be fixed by reordering the current instructions.
c) It does not work correctly and to fix it would require modifying one instruction.
d) It does not work correctly and to fix it would require adding one instruction.
e) It does not work correctly and to fix it would require modifying or adding more than one instruction.
22. Consider the program segment shown below that declares and uses a multi-dimensional array of characters:

```cpp
const int Maximum = 15;
char names[ Maximum][ Maximum];
// ... assume name values are appropriately stored into each array element
cout << names[ 15];
```

What is the output of this segment of code?

a) 15 characters, one from the first letter of each name  
b) The last name entered  
c) There is no output since it gives a compiler error  
d) Whatever happens to be in memory after the array  
e) None of the above

23. Consider the following function:

```cpp
int handle( int x)
{
    if ( x < 10)
        return x;
    else
        return handle( x/10) + x%10;
}
```

What is the output if this function is called using: `cout << handle(2468);`

a) 8  
b) 20  
c) 2468  
d) 8642  
e) None of the above

24. Consider the following section of code:

```cpp
int x, y, z;  
int *xPtr, *yPtr, **zPtr;  
x = 2; y = 7; z = 9;  
xPtr = &x;  
yPtr = &y;  
zPtr = &xPtr;
```

The result of the statement

```
cout << **zPtr;
```

is:

a) 9  
b) the address in memory of x  
c) the address in memory of xPtr  
d) the address in memory of zPtr  
e) None of the above
III. Multiple Choice: Longer Problems (6 points each)
Choose the best answer for each. Be careful!

25. What is the output of the following C++ code segment called with `confuse4()`, where the top variables declared are global variables?

```cpp
int x=2, y=6;     // global variables

void confuse1(int y, int &x)
{
    x++; 
    y++; 
}

void confuse2(int *b, int x)
{
    x = ++(*b); 
    y = ++(*b); 
}

void confuse3(int &a, int *x)
{
    a = *x; 
    (*x)++; 
}

void confuse4()
{
    int x=4;
    confuse1( x, y);
    confuse2( &y, x);
    confuse3( x, &y);
    cout << "x + y = " << x + y << endl;
}
```

a) \(x + y = 14\)
b) \(x + y = 16\)
c) \(x + y = 17\)
d) \(x + y = 19\)
e) None of the above

On the next page and for the following 3 problems consider a maze program segment similar, though not exactly the same as what we discussed in class, called as: `makeMove( start);`
26. What is the output from running this code?

a) A solution path in order, displayed once.
b) A solution path in reverse order, displayed once.
c) A solution path in order, along with numbers of extra squares visited along the way.
d) A solution path in order, followed by an infinite loop going around again to the solution.
e) Numbers that do not include the solution, displayed in an infinite loop due to a cycle.

27. What is the output from running this code if the line: //done = 1; is uncommented?

a) A solution path in order, displayed once.
b) A solution path in reverse order, displayed once.
c) A solution path in order, along with numbers of extra squares visited along the way.
d) A solution path in order, followed by an infinite loop going around again to the solution.
e) Numbers that do not include the solution, displayed in an infinite loop due to a cycle.

28. What is the output from running this code if in addition to the line //done = 1; being uncommented, the moves[] array is changed to: {10, -1, -10, 1}

a) A solution path in order, displayed once.
b) A solution path in reverse order, displayed once.
c) A solution path in order, along with numbers of extra squares visited along the way.
d) A solution path in order, followed by an infinite loop going around again to the solution.
e) Numbers that do not include the solution, displayed in an infinite loop due to a cycle.
For the following two problems consider the program segment shown in the two columns below using a singly-linked list to implement a first-in-first-out (FIFO) queue, called with:  

```c
// typedef makes a synonym called pNode for: struct Node *
typedef struct Node * pNode;
struct Node {
    int data;
    Node *pNext;
};

void displayQueue( Node *pHead)
{
    if( pHead == NULL ) {
        return;
    }
    displayQueue( pHead->pNext);
    cout << pHead->data << " ";
}

pNode process( pNode pTemp)
{
    while( pTemp->pNext != NULL ) {
        pTemp = pTemp->pNext;
    }
    return pTemp;
}

void addToEnd( int number, pNode &pFront)
{
    pNode pTail = NULL;
    pNode pNewNode = new Node;
    pNewNode->data = number;
    pNewNode->pNext = NULL;
    if( pFront == NULL ) {
        pFront = pNewNode;
    } else {
        pTail = process( pFront);
        pTail->pNext = pNewNode;
    }
    pTail = pNewNode;
}

int removeFromFront( pNode &pFront)
{
    pNode pTemp = pFront;
    int value;
    if( pFront != NULL ) {
        value = pTemp->data;
    } else {
        cout << "Invalid. Exiting";
        exit( -1);
    }
    pFront = pFront->pNext;
    delete pTemp;
    return value;
}

void fifo()
{
    int number = 0;
    pNode pFront = NULL;
    int menuChoice;
    do {
        cout << "\n";
        cout << "Select option: \n";
        cout << "1. Add to tail\n";
        cout << "2. Remove from front\n";
        cout << "3. Exit\n";
        cout << "Your choice: ";
        cin >> menuChoice;
        switch( menuChoice ) {
            case 1:
                cout << "Enter value: ";
                cin >> number;
                addToEnd( number,pFront);
                break;
            case 2:
                number=removeFromFront(pFront);
                cout << "Retrieved "
                << number << endl;
                break;
            case 3: cout << "Exiting... ";
                return;
                break;
        }
    } while ( true);
}
```
29. Which function if any in the previously shown program is incorrect?
   
a) displayQueue(...)  
b) addToEnd(...)  
c) removeFromFront(...)  
d) More than one function is incorrect.  
e) None of the above. All the code works correctly.

30. Assuming all the rest of the code is fine, which of the following best describes function displayQueue?
   
a) It compiles and runs correctly as shown.  
b) It compiles and runs, but it displays queue nodes in the reverse order.  
c) It compiles and runs, however the last node is not terminated correctly and causes an error.  
d) It compiles and runs, but does not display the recently added node in the queue.  
e) None of the above.

31. Carefully consider the C/C++ program segment given below, called with:  

   ```
   void fooE() {
       int number = 0;
       Node *pHead = NULL;
       Node *pTemp;
       cout << "Enter numbers, then -1: ";
       while ( number != -1 ) {
           cin >> number;
           if (number != -1) {
               pTemp = new Node;
               pTemp->data = number;
               pHead->pNext = pHead;
               pHead = pTemp;
           }
       }
       pTemp = pHead;
       pHead = fcn31( pHead);
       fcn31a( pHead);
   }
   ```

   If the input is: 1 3 5 2 4 -1  
   then the output is:  
   
a) The reverse of the input, excluding -1  
b) The same as the input, excluding -1  
c) The same as the input, including -1  
d) It does not compile correctly.  
e) None of the above