You may take this test with you after the test, but you must turn in your answer sheet. This test has 25 multiple-choice questions, each worth 4 points, for a total of 100 points.

The final exam is worth 20% of your course grade. You must put your answers on the answer form using a #2 pencil. This test is open book and open notes. For the multiple choice problems, select the best answer for each one and select the appropriate letter on your answer sheet. When a problem describes a segment or fragment of code you may assume the rest of the program is correct and would be supplied to make it work. You have 75 minutes.

1. Consider the code segment shown below:

   ```c++
   int m = 1;
   do {
       cout << m++ << " ";
   } while( m<7);
   ```

Which of the following two code segments will give the same output as the above code?

   **Option I:**

   ```c++
   int j=7;
   while( (j-1)>0 ) {
       cout << 8-j << " ";
       j--;
   }
   ```

   **Option II:**

   ```c++
   for( int j=0; j<6; ) {
       cout << ++j << " ";
   }
   ```

A) Neither I nor II will give the same output.
B) I will give the same output, but II will not
C) II will give the same output, but I will not
D) Both I and II will give the same output.

2. What is the output of the code segment below when called with:

   ```
   char string2[ 81 ] = "last class";
   f2( string2, 'a' );
   ```

   ```c++
   void f2( char *pString, char c)
   {
       char *pTemp = pString;
       while( pTemp ) {
           pTemp = strchr( pString, c);
           if( pTemp ) {
               strcpy(pTemp, (pTemp+1) );
           } 
       }
       cout << pString << endl;
   }
   ```

A) last class
B) lst class
C) last clss
D) lst clss
3. Consider the declaration shown at right. Which of the following would compile and run, allowing storing first name and age properly?

A) 
```cpp
cout << "Enter first name and age: ";
cin >> p1.name >> (*pPerson).age;
```

B) 
```cpp
cout << "Enter first name and age: ";
cin >> name.p1 >> age.(*pPerson);
```

C) 
```cpp
cout << "Enter first name and age: ";
cin >> pl.name >> *(pPerson.age);
```

D) 
```cpp
cout << "Enter first name and age: ";
cin >> pl.name >> &(pPerson->age);
```

E) None of the above

4. If we knew the number of dictionary words ahead of time and the words were all the same size, what would be the effect of declaring the dictionary as:

```cpp
char dictionary[ numberOfWords * (WordSize+1)];
```

A) It would not be possible to pull out individual words
B) It would not be possible to store words this way, because C string functions could not be used to search for a particular word within the dictionary
C) It would be possible to store words this way, finding a word using:
   ```cpp
cout << strstr(dictionary, wordToFind);
```
D) It would be possible to store words this way, printing the nth word using:
   ```cpp
char foundWord[ WordSize + 1];
strncpy( foundWord, dictionary[ n * (WordSize+1) ] );
cout << foundWord;
```

5. Assume we have a dictionary stored in a linked list of Nodes, where each Node has a string for the word and a pointer to the next Node. If we want to implement a skip list where we have an array of 26 pointers with each one jumping to a particular Node on the list, how should the skip list be declared?

A) Node skipList[ ];
B) Node skipList[ 26];
C) Node *skipList[ 26];
D) Node *skipList[ ] [ 26];

6. What advantage does a linked list have over an array?

A) It uses less memory
B) Accessing a linked list node is typically faster than accessing an array element
C) An element can be inserted in the middle without having to shift other elements
D) We know ahead of time what the maximum size of the list can be
7. What advantage does an array have over a linked list?
   
   A) It uses less memory  
   B) Accessing an array entry is typically faster than accessing a list element  
   C) We can use memory arithmetic to jump to a particular part of the array  
   D) All of the above

8. Consider the two approaches shown below to implement a linked list used, where we need to both prepend at the head of the list, as well as append at the tail of the list.

   | Approach A: | pHead |  |  |  |  |
   |--------------|-------|---|---|---|
   | Approach B: | pHead |  |  |  | pTail |

Which of the descriptions below is the most accurate?

   A) Neither A nor B can be used  
   B) Approach A can be used, but B cannot  
   C) Approach B can be used, but A cannot  
   D) Either A or B can be used, though one approach may be more efficient than the other

9. Consider a version of a linked list that implements a queue where we always add new nodes to the tail of the list and delete existing nodes from the head of the list, like in a movie theater ticket line. Consider the two head-and-tail pointer implementations shown below:

<table>
<thead>
<tr>
<th>Approach I:</th>
<th>pTail</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>pHead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach II:</td>
<td>pHead</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pTail</td>
</tr>
</tbody>
</table>

Which of these two approaches is preferable?

   A) Neither one would work correctly  
   B) Approach I is preferable  
   C) Approach II is preferable  
   D) Neither one is preferable, they are both equally well suited

10. What is the output of the function shown at right, when the input is: hey$

   A) yeh  
   B) yeh$  
   C) hey  
   D) hey$
11. What is the output of the function shown at right, when the input is: hey$

A) hey  
B) yeh  
C) hey$  
D) yeh$

12. What is the best description of the function shown at right?

A) It does not compile  
B) It compiles but gives a run-time error  
C) It compiles and runs successfully, deleting all the nodes on the list.  
D) It compiles and runs successfully, deleting only the last node on the list.

13. What is the best description of the function shown at right?

A) It does not compile  
B) It compiles but gives a run-time error  
C) It deletes all the nodes on the list  
D) It traverses the list and deletes the last node

14. Given the code shown below, what is stored in array letters after calling function Problem14()

A) acbd  
B) badc  
C) dbca  
D) cadb
15. A queue is a type of linked list where we always add new nodes to the tail of the list and delete existing nodes from the head of the list, like in a movie theater ticket line. Consider implementing a queue using a circularly linked list, such as the one shown at right. Consider whether or not each of the following statements are true:

I. We can add to the tail of the list and delete from the head without having to traverse the list.
II. We can add to the tail of the list, but we must traverse the list to delete from the head of the list.
III. We can delete from the head of the list, but we must traverse the list to add to the tail.
IV. If we change pHead to instead be pTail, only then can we add to the tail of the list and delete from the head of the list without needing to traverse the list.

How many of the above statements are true?

A) 1  
B) 2  
C) 3  
D) 4  

16. Consider using function insertInOrder( number, pHead) to create a list where as each number is inserted, the list is maintained in ascending order. Assume we are using a doubly-linked list, where each Node has two pointers, one pointing to the next node, and one pointing to the previous node.

Which of the following statements is the best comparison between a circularly-linked list implementation versus this doubly-linked list implementation, where in both cases we have a pointer pTemp to a particular node in the middle of the list, and we want to be able to delete node pTemp. Assume that both are very large lists. Which of the following is the best description?

A) It requires roughly the same amount of computation for both a circularly-linked list and a doubly-linked list.
B) It requires much more computation with the circularly-linked list.
C) It requires much more computation with the doubly-linked list.
D) It is possible to do it with one of the list implementations, but not the other.
For the next three problems consider the code shown at right, similar to the maze program discussed in class. The code is called using: `makeMove( start);`

17. What is the result of running this program relative to the values in the `moves` array?

A) No matter the `moves` order, it will always still find the solution
B) Only one `moves` order will allow it to find the solution.
C) Multiple `moves` orders will allow it to find the solution.
D) No `moves` order will allow it to find the solution using given code.

18. What would be the result if we change the line at `Point A` to be:

```
!done && cameFrom[ next] == 0) {
```

A) No matter the `moves` order, it would always still find the solution
B) It would find the solution when using some `moves` orders, but not others.
C) It can only find the solution using one `moves` order
D) No `moves` order will allow it to find the solution using the code as shown.

19. With the original problem shown at right above, what would be the result if we set

```
movesSize = 8;
```

and we set the `moves` array to be:

```
{11,1,-9,-10,-11,-1,9,10}
```

and we set

```
maze[ 34] = 1;
```

A) The program will find a solution without any backtracking
B) The program will find the solution, but will need to go down some wrong paths and then backtrack to get to the solution.
C) The program will get stuck in an endless loop without finding the solution
D) The program will exhaust all possible valid moves without finding the solution.
For the next three problems carefully consider the C/C++ program segment given below, called with:

function doit(). Note the lines **Position A** and **Position B** which are referred to in the questions below. Assume input to the program is: 1 2 3 4 -1

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

void display( Node *pHead) {
    while( pHead != NULL) {
        cout << pHead->data << " ";
pHead = pHead->pNext;
    }
}

Node * modify( Node *pHead) {
    Node *pTemp;
    if (pHead->pNext == NULL) {
        return pHead;
    } else {
        pTemp = modify(pHead->pNext);
pHead->pNext->pNext = pHead;
        // Position B left
        return pTemp;
    }
}

void doit() {
    int number = 0;
    Node *pHead = NULL;
    Node *pTemp;

cout <<"Enter numbers, then -1: ";
// prepend nodes
while ( number != -1) {
    cin >> number;
    if (number != -1) {
        pTemp = new Node;
pTemp->data = number;
pTemp->pNext = pHead;
pHead = pTemp;
    }
}
pTemp = pHead;
pHead = modify( pHead);
// Position A right
display( pHead);
}
```

20. What is the output of the code as shown?

- A) 4 3 2 1
- B) 1 2 3 4
- C) Only part of the input
- D) It compiles and runs, but goes into an infinite loop

21. What is the output of the original code if we replace only the line of code at **Position A right** above with:

```
pTemp->pNext = NULL;
```

- A) 4 3 2 1
- B) 1 2 3 4
- C) Only part of the input
- D) It compiles and runs, but goes into an infinite loop

22. Starting again with the unchanged original code, what is the output if we replace the line of code at **Position B left** above with:

```
pHead->pNext = NULL;
```

- A) 4 3 2 1
- B) 1 2 3 4
- C) Only part of the input
- D) It compiles and runs, but goes into an infinite loop
For the following three linked list questions, assume we have already implemented the following functions, using the same Node structure we discussed in class. Assume list values are randomized to start.

I. Node *findLastNode( Node *pHead)    // find last node on list
II. void prepend(Node * &pHead, Node * pNew) // put new node at list front
III. void insertInOrder(Node * &pHead, Node * pNew) // insert in ascending order
IV. bool isInList(int data, Node * pHead)  // return true if data is in list
V. bool sameAsNext(Node * pTemp)         // true if node data is same as next's
VI. Node *reverseList( Node * pHead)  // reverse the list

23. Which of the above functions would be best to use with minimum additional code to implement:

   void concatenate(Node *& pListA, Node * pListB);
which should result in all the nodes of list B being added to the end of list A:

A) This can be done easily without using any of the above functions
B) prepend
C) insertInOrder
D) findLastNode

24. Which of the above functions would be best to use to implement:

   void sortDescending(Node *& pHead);
where the resulting code run time is in proportion to the list length?

A) findLastNode, prepend
B) isInList, prepend
C) insertInOrder, findLastNode
D) insertInOrder, prepend

25. Which set of the above functions would be best to use with minimum additional code to implement:

   void removeDuplicates(Node *& pHead);

A) findLastNode, prepend
B) findLastNode, insertInOrder
C) sameAsNext, prepend
D) sameAsNext, insertInOrder, prepend