CS 141, Final Exam, Spring 2017

CS 141 – Programming Design II
Final Exam - Prof. Reed
Spring 2017

You may take this test with you after the test, but you must turn in your answer sheet.
This test has 33 multiple-choice questions, each worth 3 points, for a total of 99 points, plus one for free.

This test is worth 20% of your final 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 two hours.

1. Consider using Bubble Sort to put the following elements into ascending order:

   7 6 12 2 9 4 3

What of the following best describes the relationship between the data and the passes?

A) The number of passes required before the elements are in order is smaller when doing our passes starting from the left.
B) The number of passes required before the elements are in order is smaller when doing our passes starting from the right.
C) The number of passes for this data in particular is the same regardless of whether we start our passes through the data starting from the left or from the right.
D) The number of passes is always the same for all data regardless of starting our passes from the left or from the right.

2. Consider the code segment shown at right. What would most likely happen when running this code?

   ```c
   int value = 0;
   printf("Enter a number: ");
   scanf("%d", value);
   ```

A) It would not compile
B) It would compile, but give a runtime error every time
C) It would compile and run, and cause a problem sometimes
D) It would compile and run correctly

3. Consider the code segment shown at right. What would most likely happen when running this code?

   ```c
   char userInput[81];
   printf("Enter a word: ");
   scanf("%s", userInput);
   ```

A) It would not compile
B) It would compile, but give a runtime error every time
C) It would compile and run, and cause a problem sometimes
D) It would compile and run correctly

4. Consider the code segment shown at right. What would most likely happen when running this code?

   ```c
   char userInput[81];
   char *pWord = userInput;
   printf("Enter a word: ");
   scanf("%s", pWord);
   ```

A) It would not compile
B) It would compile, but give a runtime error every time
C) It would compile and run, and cause a problem sometimes
D) It would compile and run correctly
5. Consider using *binary search* to find a particular number within a *sorted* array of 100 unique random numbers. On average how many numbers will have to be examined before the number you are searching for is found? Include the number found in your count.

A) 6  
B) 7  
C) 10  
D) 50  
E) 100

6. What is the minimum number of lines of code shown below that would need to be corrected in order to get the program segment shown below to compile and run correctly?

```c
char salary;
printf("Enter your hourly salary in dollars and cents: ");
scanf("%4c", salary);
if( salary > 10.5) {
    printf("More than minimum wage.\n");
} else {
    printf("Not greater than minimum wage.\n");
}
```

A) 2 lines  
B) 3 lines  
C) 4 lines  
D) 5 or more lines

7. What is the output of the code segment shown at right when function `testf1()` is called?

A) 0 0 0  
B) 1 1 1  
C) 0 1 2  
D) 1 2 3

```c
void f1( int y)
{
    static int x=1;
    printf("%d ", x++);
}
void testf1() {
    for( int x=0; x<3; x++) {
        f1( x);
    }
}
```

8. Consider the following two code options to dynamically allocate space for an array of integers:

*Option A:* \[ \text{int *pNewArray1} = \text{new int}[\text{size}]; \]

*Option B:* \[ \text{int *pNewArray2} = (\text{int *})\text{malloc( sizeof(int) * size);} \]

Assuming variable `size` is initialized, which of the descriptions below of the above two lines is most accurate?

A) Neither option is correct  
B) Only Option A is correct  
C) Only Option B is correct  
D) Both are correct
9. Consider the following declaration:

```c
struct Person {
    char name[25];
    int age;
};
Person p1 = {"Erin", 23};
Person p2 = {"Darin", 37};
Person *pPerson = &p1;
```

Which of the following would compile and run, allowing storing first name and age properly?

A) ```c
printf("Enter first name and age: ");
scanf("%s %d", pl.name, (*pPerson).age);
```

B) ```c
printf("Enter first name and age: ");
scanf("%s %d", name.pl, age.(*pPerson));
```

C) ```c
printf("Enter first name and age: ");
scanf("%s %d", pl.name, *(pPerson.age));
```

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

10. What are the results of calling functions \texttt{fa} and \texttt{fb} in the code shown below:

Option A:

```c
void \texttt{fa}( int x) {
    for( int i=0; i<x; i++) {
        for( int j=0; j<x; j++) {
            printf("*");
        }
    }
    printf("\n");
}
```

Option B:

```c
void g1( int x) {
    if(x==0) return;
    printf("*");
g1(x-1);
}

void g2( int x, int y) {
    if(y==0) return;
g1( x);
    printf("\n");
g2(x,y-1);
}

void \texttt{fb}( int x) {
    g2( x,x);
}
```

A) For the same parameter value \texttt{fa} will give more output than \texttt{fb}
B) For the same parameter value \texttt{fb} will give more output than \texttt{fa}
C) \texttt{fa} will give exactly the same output as \texttt{fb}
D) One version will give output, but the other results in an infinite loop
11. 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) 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.
B) The first array dimension is left blank so that arrays can grow dynamically.
C) C programs can overwrite the end of an array, however for a 2-D array the formula used to find the \(i^{th}\) row needs to know how many columns are on each row.
D) 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 row begins, so the size of the first dimension is not necessary.

12. Consider the code segment shown below. If after the function call to changeLetters(...) the value of \textit{number} has changed, what is the most likely cause?

```c
int number = 5;
printf(" %d", number);
char letters[]="ABCD";
changeLetters( letters);
printf(" %d", number);
```

A) \textit{number} is a global variable instead of a local variable as it should be
B) Although \textit{number} is not passed to function changeLetters(), function changeLetters() itself calls a second function which changes \textit{number}
C) There is some ASCII control characters that are present in the code even though they are not visible
D) Function changeLetters() overwrites the end of array letters

13. Consider the struct declaration (shown below at right) used to implement a list which has a sentinel node, so there is always at least that one node on the list. Consider the following function declarations used to prepend nodes at the beginning of the list:

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

I. Node * prepend(int value, Node *pHead)
II. void prepend(int value, Node *&pHead)
III. void prepend(int value, Node **pHead)

How many of the above functions could be used to prepend nodes at the beginning of the list?

A) None
B) One
C) Two
D) Three

14. Given a Node structure as discussed in class and pHead is a pointer to the head of the list:

```
6 5 4 3 2 1
```

What is the output from the statement: \(\text{printf}("%d", \text{pHead}->\text{pNext}->\text{pNext}->\text{data});\)

A) 5
B) 4
C) 3
D) 2
15. What is the best choice to declare the type of a dictionary where you don't know ahead of time the number of words or the size of the words?

A) char dictionary;
B) char dictionary[];
C) char *dictionary;
D) char **dictionary;

16. 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:

   char dictionary[ numberOfWords * (wordSize+1)];

A) It would not be possible to pull out individual words
B) It would be possible to store words this way, however C string functions could not be used
C) It would be possible to store words this way, however the whole dictionary would basically be one extremely large C string
D) It would be possible to store words this way and still use C string functions, doing arithmetic to find the address of each word starting position.

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

A) It uses less memory
B) It can cluster together elements of different types
C) An element can be inserted in the middle without having to shift other elements
D) All of the above

18. Consider a linked list where we only have a head pointer and we make it circularly linked by making the tail point to the head, rather than pointing to NULL. What other changes would we be likely to need to make in a linked list program that uses this circularly linked list?

A) We would need an additional pointer to keep track of the starting node when displaying the list
B) The code to reverse the list would no longer work correctly
C) We would only be able to insert after the head node or alternatively we would need to traverse the entire list
D) All of the above

19. Given the code shown below, what is stored in array letters after calling function Problem19()

```c
int eval( char *pC)
{
    if( *pC == NULL) return 0;
    return eval( ++pC) + 1;;
}
void modify(char letters[], int index1)
{
    char temp = letters[ index1];
    int index2 = eval( letters) - 1;
    letters[ index1] = letters[ index2];
    letters[ index2] = temp;
}
void f19( char letters[], int index)
{
    int x = eval(letters);
    if( index >= x) return;
    modify( letters, index);
    f19( letters, ++index);
}
void Problem19()
{
    char letters[]="abcd";
    f19( letters, 0);
    printf("%s", letters); 
}
```

A) abcd
B) dcba
C) cdab
D) dabc
20. Consider the two approaches shown below to implement a linked list, where sometimes we *prepend* at the beginning of the list, and other times we *append* at the end of the list.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td><img src="image1.png" alt="Diagram A" /></td>
</tr>
<tr>
<td>B</td>
<td><img src="image2.png" alt="Diagram B" /></td>
</tr>
</tbody>
</table>

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, along with some additional code necessary for A

21. Consider a version of a 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 line to order food at a fast-food restaurant. Consider the two head-and-tail pointer implementations shown below:

<table>
<thead>
<tr>
<th>Approach</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td><img src="image3.png" alt="Diagram A" /></td>
</tr>
<tr>
<td>B</td>
<td><img src="image4.png" alt="Diagram B" /></td>
</tr>
</tbody>
</table>

Which of these two approaches is preferable?

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

22. Consider the following declaration of a function intended to grow an array of integers by dynamically allocating memory:

```c
void growArray( int oldSize, int newSize, int *&pTheArray)
```

Assume the part of the program that calls this function has the following:

```c
int theArray[ 10 ] = {1,2,3,4,5,6,7,8,9,10};
int oldSize = 10;
int newSize = 20;
growArray( oldSize, newSize, theArray);
```

What is the best description of whether or not this function call will work?

A) The function will not compile correctly  
B) The function will compile correctly, however will crash when running  
C) The function will compile correctly and could run, however the calling code will not have access to the new larger array after returning from the function  
D) The function will compile correctly and could run correctly
For the next four problems consider the code shown at right, similar to the maze program discussed in class. The code is called using: `makeMove(start);`

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

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.

24. For this maze what is the relationship between the ordering of the values in the `moves` array and the `start` and `goal` values?

A) If an ordering of the `moves` array works, then it will still work if the `start` and `goal` values are reversed.
B) If an ordering of the `moves` array works, then it will not work if the `start` and `goal` values are reversed.
C) An ordering of the `moves` array will work for both the original and reversed `start` and `goal` values only if we always do the `left` and `right` moves before doing the `down` and `up` moves.
D) An ordering of the `moves` array will work for both the original and reversed `start` and `goal` values only if we always attempt to do the `down` and `up` moves before doing the `left` and `right` moves.

25. What would be the result if we set `movesSize = 8;` and we set the `moves` array to be:
   `{11,10,9,-1,-11,-10,-9,1}

A) The results of running the program would be unchanged
B) A solution would still be found, but it would take more steps
C) A solution would still be found, but it would take fewer steps
D) A solution would no longer be found.

26. What would be the result in the original code above if we uncomment the line of code at `// Point A`

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.
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

27. 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

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

    void sortDescending(Node *&pHead);

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

29. 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) insertInOrder, sameAsNext, prepend

30. Consider the function shown at right below used to traverse and display the values on a linked list. What is the best description of this function when called with pHead pointing to a non-empty list?

    A) It works correctly
    B) It traverses and displays the list, however it destroys the list head pointer in the process
    C) It does not compile
    D) It compiles but gives a run-time error

```c
void displayList(Node *pHead) {
    while (pHead != NULL) {
        cout << pHead->data << " ";
        pHead = pHead->pNext;
    }
    cout << "\n\n";
}
```
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.

```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: ";

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);
}
```

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

A) The reverse of the input, excluding -1
B) The same as the input, excluding -1
C) Only part of the input
D) It compiles and runs, but goes into an infinite loop

32. 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) The reverse of the input, excluding -1
B) The same as the input, excluding -1
C) Only part of the input
D) It compiles and runs, but goes into an infinite loop

33. 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) The reverse of the input, excluding -1
B) The same as the input, excluding -1
C) Only part of the input
D) It compiles and runs, but goes into an infinite loop