Answer all questions in a separate document and email that document to abraham-sonk@ecu.edu as an attachment. Use subject CSCI 2530 final exam. You can attach graphics files for hand-written answers, but be sure that they are readable. Make all file names, including graphics files, begin with your last name, followed by your first name. Write your name in your answer file. Make your answers clear, concise and precise.

For the multiple-choice questions (marked [MC]), write the letter of the best answer in your solution page, even if no answer ideal.

Read each question twice. Be sure that you are answering the question that is asked. Check your answers.

1. [MC] What is the value of C++ expression 26/10 + 3/5? Be careful. C++ arithmetic has some differences from common mathematical notation.
   (a) 3.2
   (b) 4.2
   (c) 2
   (d) 3
   (e) 4

2. [MC] Expression new int has type
   (a) int
   (b) int*
   (c) int**
   (d) int&
   (e) It has no type at all, since it is not an expression.

3. [MC] Expression new int[20] has type
   (a) int*
   (b) int**
   (c) int[]
   (d) int[]*
   (e) It has no type at all, since it is not an expression.
4. [MC] A contract of a function
   (a) tells how the function works.
   (b) tells what the types of the parameters are.
   (c) tells which other functions use this function and how those other functions
       choose parameters to send to this function.
   (d) tells exactly what the function accomplishes and how its parameters affect
       what it accomplishes.
   (e) tells all of the above

5. [MC] If variable \( p \) has type double*, then expression *\( p \) has type
   (a) double**
   (b) double
   (c) double*
   (d) *double
   (e) **double

6. [MC] What is the value of C++ expression 5 > 6 || 4 > 2?
   (a) false
   (b) true
   (c) 2
   (d) 5
   (e) It does not have a value since it is a test.

7. [MC] When you create a variable \( p \) using declaration

   ```
   char* p;
   ```

   (a) \( p \) initially points to newly allocated memory in the heap.
   (b) \( p \) will have an initial value, but you have no way of knowing what that
       value will be when the program runs.
   (c) \( p \) has an initial value of NULL.
   (d) \( p \) has no initial value, and using \( p \) before you set its value will always cause
       an immediate run-time error.
   (e) \( p \) initially points to memory that is in the stack frame of the current
       function.
8. [MC] When an array \( A \) is passed as a parameter to a function in a C++ program,

(a) a copy of \( A \) is passed to the function.

(b) the function must make a copy of \( A \) before using \( A \).

(c) a size must be written in brackets after \( A \) in the function’s parameter list so that the function knows how large the array is.

(d) if the function wants to return \( A \), it should say

\[
\text{return } A[];
\]

(e) \( A \) is passed as a pointer to the caller’s array, since an array is a pointer.

9. [MC] What is the value of variable \( r \) after the program fragment below is finished? Do a careful hand simulation.

- (a) 5
- (b) 6
- (c) 16
- (d) 32
- (e) 64

```c
int n = 1;
int r = 1;
while(n < 10)
{
    if(n % 2 == 1)
    {
        r = r + r;
    }
    n = n + 1;
}
```

10. [MC] What is the value of variable \( m \) after the following program fragment is finished? Do a careful hand simulation.

- (a) \( 1 + 1 + 1 + 1 + 1 \)
- (b) \( 3 + 3 + 3 + 3 \)
- (c) \( 3 + 3 + 3 + 3 + 3 \)
- (d) \( 3 + 6 + 9 \)
- (e) \( 3 + 6 + 9 + 12 \)

```c
int m = 0;
int n = 0;
while(n < 13)
{
    m = m + n;
    n = n + 3;
}
```
11. [MC] Suppose that $r$ has been declared by

```c
int r[4];
```

What is in array $r$ after expression `test(r)` is performed, using the definition of `test` shown? Do a careful hand simulation.

(b) $r[0] = 1$, $r[1] = 2$, $r[2] = 3$, $r[3] = 4$

```c
void test(int A[])
{
    for(int i = 0; i < 4; i++)
    {
        A[i] = i + 1;
    }
    for(int i = 3; i >= 0; i--)
    {
    }
}
```

12. [MC] Which of the following will create an array of 50 characters in the heap, and make variable `parrot` point to it?

(a) `char** parrot = new char[50];`
(b) `char** parrot = new char*[50];`
(c) `char* parrot = new char[50];`
(d) `char* parrot = new char*[50];`

13. [MC] Which of the following makes all variables in array `parrot` from the question 12 hold 0?

(a) `for(int k = 0; k < 50; k++) parrot[k] = 0;`
(b) `for(int k = 0; k <= 50; k++) parrot[k] = 0;`
(c) `for(int k = 1; k <= 50; k++) parrot[k] = 0;`
(d) `for(int k = 1; k < 50; k++) parrot[k] = 0;`

14. [MC] Which of the following stores 22 into the first slot in array `parrot` from question 12?

(a) `parrot + 0 = 22;`
(b) `(*parrot)[0] = 22;`
(c) `(parrot*)[0] = 22;`
(d) `parrot[0] = 22;`
15. [MC] A search algorithm
   (a) is suitable for computing the sum of a sequence of numbers.
   (b) is suitable for determining if an array contains only positive numbers.
   (c) is suitable for finding the smallest value in a sequence of numbers.
   (d) always looks at each thing in a sequence exactly once.

16. [MC] Making use of a dangling pointer in a program
   (a) is normal behavior for a program, as long as the pointer points into the heap.
   (b) is normal behavior for a program, as long as the pointer points into the run-time stack.
   (c) is a mistake that can lead to a memory fault and is often difficult to locate using a debugger.
   (d) is a mistake that can lead to a memory fault but that is usually easy to locate and fix using a debugger.

17. [MC] A memory leak occurs when a program
   (a) deletes memory that it still needs.
   (b) does not delete memory that it no longer needs.
   (c) allocates an array that is smaller than what it really needs.
   (d) forgets to allocate memory for an array before using that array.

18. [MC] A min-heap is a data structure that is useful for efficient implementation of
   (a) a set of integers with lookup, insert and remove operations.
   (b) an equivalence relation with merge and equivalent operations.
   (c) a binary search tree.
   (d) a priority queue.
19. [MC] Function \( g \) is defined below. What is the value of expression \( g(g(g(3))) \)?

(a) 2

```c
int g(int n)
{
    if(n < 3)
    {
        return 6;
    }
    else
    {
        return n-1;
    }
}
```

(b) 3

(c) 4

(d) 5

(e) 6

20. [MC] To within a constant factor, how much time does it take, in the worst case, to insert a value into a min-heap that has \( n \) values in it?

(a) \( \Theta(1) \)

(b) \( \Theta(\log_2(n)) \)

(c) \( \Theta(n) \)

(d) \( \Theta(n \log_2(n)) \)

(e) \( \Theta(n^2) \)

21. [MC] To within a constant factor, how long does it take, in the worst case, to compute the length of a linked list of length \( n \)?

(a) \( \Theta(1) \)

(b) \( \Theta(\log_2(n)) \)

(c) \( \Theta(n) \)

(d) \( \Theta(n \log_2(n)) \)

(e) \( \Theta(n^2) \)

22. [MC] Suppose that \( T \) is a hash table that uses chaining to resolve collisions. How much time does it take, on the average, to insert an entry into hash table \( T \), if \( T \) has \( n \) values it in at the time of the insertion, and if the array holding the hash table has size \( n \)?

(a) \( \Theta(1) \)

(b) \( \Theta(\log_2(n)) \)

(c) \( \Theta(n) \)

(d) \( \Theta(n \log_2(n)) \)

(e) \( \Theta(n^2) \)
23. [MC] To within a constant factor, how long does it take, *in the worst case*, to look up a value in a height-balanced binary search tree that has $n$ values in it?

(a) $\Theta(1)$
(b) $\Theta(\log_2(n))$
(c) $\Theta(n)$
(d) $\Theta(n \log_2(n))$
(e) $\Theta(n^2)$

24. [MC] Suppose that the following structure type definition is given.

```c
struct Dwarf
{
    int Grumpy;
    int Happy;

    Dwarf(int G, int H)
    {
        Grumpy = G;
        Happy = H;
    }
};
```

Which of the following will create a variable $s$ of type Dwarf*, create a new Dwarf in the heap with its 'Grumpy' variable holding 60 and its 'Happy' variable holding 8, and make variable $s$ point to that new Dwarf?

(a) Dwarf* s = new Dwarf(60, 8);
(b) Dwarf* s = new Dwarf*(60, 8);
(c) Dwarf* s = Dwarf(60, 8);
(d) Dwarf* s = Dwarf*(60, 8);
(e) Dwarf* s(8, 60);

25. [MC] A scan algorithm on a sequence

(a) sometimes stops before it reaches the end of the sequence.
(b) always looks at each thing in a sequence exactly once.
(c) cannot be used to find the largest value in a sequence of numbers.
(d) sometimes needs to go through the sequence several times.
26. [MC] What is the value of funny(3), where funny is defined as follows? (Hint. Compute funny(0), funny(1), funny(2) and funny(3), in that order.)

(a) 1
(b) 3
(c) 7
(d) 15
(e) 31

```c
int funny(int n)
{
    if(n == 0)
    {
        return 1;
    }
    else
    {
        return 2*funny(n-1) + 1;
    }
}
```

27. [MC] To within a constant factor, how long does it take, in the worst case, to compute strlen(s) if s is a null-terminated string of length n?

(a) Θ(1)
(b) Θ(log₂(n))
(c) Θ(n)
(d) Θ(n log₂(n))
(e) Θ(n²)

28. [MC] The equivalence manager data structure in the minimal spanning tree program

(a) keeps track of the weights of the edges that have been added to the minimal spanning tree.
(b) is responsible for deciding which edge to try adding next to the minimal spanning tree.
(c) helps determine whether two vertices are connected by a path using the edges that have been added so far.
(d) helps to sort the array of edges.
29. [MC] The height-balancing mechanism for binary search trees
   (a) keeps a tree height-balanced, even in the worst case.
   (b) is used to rebalance the tree after every few insertions or removals.
   (c) keeps a tree height-balanced for average sequences of insertions and re-
       movals, but in the worst case leads to a tree that is not height-balanced.
   (d) keeps a tree height-balanced in the best case, but for average insertions
       and removals leaves the tree not height-balanced.

30. Suppose that you insert 1 into the following min-heap. What does the heap
    look like after the insertion, and after restoring it to a heap?

31. Suppose that you remove the smallest value from the min heap above. What
    does the heap look like after doing the removal and restoring the heap order?
    Use the heap that you started with in question 30, not the result that you got
    in question 30.
32. Suppose that you insert 29 into the following binary search tree using the algorithm that does rotations to keep the tree height-balanced. What is the resulting tree? Be sure to do the rotations correctly.

```
      50
     /  \
    24   75
   /  \
  15   42
     /  \
   28   45
```

33. Using the same starting tree as the preceding problem (not the tree that you got as your answer), suppose that 49 is inserted, using the height-balancing algorithm. What does the tree look like after doing that insertion and rebalancing?
34. **Read this question twice before you start to work on it.**

You are given a function `isPerfectSquare(n)` that returns true if $n$ is perfect square and returns false if $n$ is not perfect square. Do not write `isPerfectSquare`; assume that it is already available.

As indicated below, write two C++ definitions of function `perfectSquareInRange(i, j)`, which returns true if at least one of the values of $n$, where $i \leq n \leq j$, is perfect square. By definition, if $i > j$, `perfectSquareInRange(i, j) = false`. For example,

- `perfectSquareInRange(8, 12)` returns true since 9 is perfect square and $8 \leq 9 \leq 12$,
- `perfectSquareInRange(26, 30)` returns false since none of the numbers 26, 27, 28, 29 or 30 is a perfect square.
- `perfectSquareInRange(12, 9)` returns false.

**Note.** `perfectSquareInRange` must work by testing numbers to see if they are perfect squares, using `isPerfectSquare`.

**Note.** Once `perfectSquareInRange` has found a perfect square $k$ where $i \leq k \leq j$, it is clear that the answer must be true. In that case, `perfectSquareInRange` is required to return before asking if any other number is a perfect square.

**Note.** `perfectSquareInRange(i, j)` must not ask if any number $n$ is a perfect square if $n < i$ or $n > j$. Also, `perfectSquareInRange` must not compute `isPerfectSquare(n)` for the same number $n$ more than once.

**Note.** `perfectSquareInRange` must not change either of its parameters $i$ and $j$.

**Note.** Do not create any arrays or extraneous data structures for this problem.

(a) Write a definition of `perfectSquareInRange` using a loop. Do not use recursion. A heading is given.

```cpp
bool perfectSquareInRange(int i, int j)
```

(b) Write a definition of `perfectSquareInRange` using recursion. Do not use a loop. A heading is given.

```cpp
bool perfectSquareInRange(int i, int j)
```
35. **Read this question twice before you start to work on it.** Be sure that you are solving the problem that is asked.

Write a function called `everyOtherCharacter(s)` that takes a null-terminated string `s` and returns a null-terminated string containing the characters of `s` at odd indices, in their original order. For example, `everyOtherCharacter("abcdefghi")` must return "bdhli" and `everyOtherCharacter("abcd")` must return "bd". Remember that the first character is at index 0, and 0 is not odd.

**Note.** EveryOtherCharacter must return the result as a null-terminated string that is stored in a newly allocated array in the heap. The size of the result array must be just large enough to hold the result string, no smaller and no larger.

**Note.** Parameter `s` is marked const. EveryOtherCharacter must not change the parameter string at all, even temporarily.

**Note.** Do not use the C++ type `string` or any types from the C++ Standard Template Library. Do the work yourself. You are allowed to use functions from the `cstring` library, such as `strlen`, `strcpy` and `strcat`, if you find them helpful. A heading is given.

```c
char* everyOtherCharacter(const char* s)
```
The next three questions use the following type Node, which is used in implementing binary trees.

```c
struct Node
{
    int item;
    Node* left;
    Node* right;
    Node(int i, Node* L, Node* R)
    {
        item = i;
        left = L;
        right = R;
    }
};
```

36. Write a function `sumEven(t)` that returns the sum of all of the even integers that occur at nodes in tree `t`. Function `sumEven(t)` must not alter any nodes in tree `t`. If `t` is an empty tree then `sumEven(t)` is defined to be 0. A heading is given.

```c
int sumEven(const Node* t)
```

37. Write a C++ definition of a function called `tripleAll(T)` that replaces each value `k` in tree `T` by `3k`. For example, if tree `T` contains `2`, `−7` and `9` before, then after doing `tripleAll(T)`, `T` should contain `6`, `−21` and `27`.

This function does not produce a new tree, but modifies the existing tree, changing the values stored in its nodes. Do not assume that any tree functions are predefined for you. A heading is given.

```c
void tripleAll(Node* T)
```
38. Write a C++ definition of a function called `triples` that takes a binary tree \( T \) and returns a tree that has the same tree structure as tree \( T \), but whose nodes contain the triples of corresponding nodes in \( T \). For example, if the root of \( T \) contains 3, then the root of `triples(T)` contains 9. If the root of \( T \) contains \(-4\), then the root of `triples(T)` contains \(-12\). (If \( T \) is an empty tree, `triples(T)` is also an empty tree.)

Function `triples` must not modify \( T \). It must produce the result tree using newly allocated nodes. Do not assume that any tree functions are predefined for you. A heading is given.

```cpp
Node* triples(const Node* T)
```

The remaining questions use types `ListCell` and `List`, shown below, for implementing linked lists.

```cpp
struct ListCell
{
    int    head;
    ListCell* tail;

    ListCell(int h, ListCell* t)
    {
        head = h;
        tail = t;
    }
};
typedef ListCell* List;
```

Lists can be thought of in conceptual form, using notation \([2, 4, 6]\) for a linked list that contains three values 2, 4 and 6, in that order. head(\( L \)) is the first value in nonempty list \( L \), tail(\( L \)) is the list of all members of \( L \) excluding the first member, and \( h:t \) is the list whose head is \( h \) and whose tail is \( t \). [] indicates an empty list.
39. You are given a function isPrime(n) that returns true if n is prime and false if n is not prime. Do not write isPrime; assume that it is already available.

Suppose that firstPrime(L) is supposed to return the first integer in linked list L that is prime. For example, if L is [8, 11, 5, 6], then firstPrime(L) must return 11. If there are no prime integers in list L, then firstPrime(L) must return 0.

(a) What is firstPrime([4, 8, 5])?
(b) What is firstPrime([3, 4, 5])?
(c) What is firstPrime([])?
(d) Using conceptual notation (not C++ notation), write equations that, taken together, define firstPrime(L) for all lists L, by filling in the blanks below. Be sure that the equations are true and cover every case. You will receive no credit for any answer that is not an equation.

\[
\begin{align*}
\text{firstPrime([])} &= {} &
\text{firstPrime(L)} &= {} \\
\text{firstPrime(L)} &= \begin{cases} 
& \text{when } L \neq [] \text{ and head(L) is prime} \\
& \text{when } L \neq [] \text{ and head(L) is not prime}
\end{cases}
\end{align*}
\]

(e) Write a recursive definition of firstPrime(L) in C++. Do not use a loop. Base your definition closely on the equations from part (d). You can assume that functions head, tail and isEmpty, and constant emptyList, are predefined. Be sure to use C++ notation. FirstOdd(L) must not change list L. A heading is given.

```cpp
int firstPrime(const ListCell* L)
```

(f) Write a definition of firstPrime(L) using a loop. Do not use recursion for this version. FirstOdd(L) must not change list L.

```cpp
int firstPrime(const ListCell* L)
```
40. Function removeFirstOccurrence($x, L$) is intended to return a list that you get by removing the first occurrence of $x$ from linked list $L$. For example, using conceptual list notation,

\[
\begin{align*}
\text{removeFirstOccurrence}(2, [2, 3, 5, 6]) &= [3, 5, 6] \\
\text{removeFirstOccurrence}(2, [2, 2, 5, 6]) &= [2, 5, 6] \\
\text{removeFirstOccurrence}(5, [2, 3, 5, 6]) &= [2, 3, 6]
\end{align*}
\]

By definition removeFirstOccurrence($x, L$) must return an empty list when $L$ is an empty list.

(a) Using conceptual notation (**not C++ notation**), write equations that, taken together, define removeFirstOccurrence($L$) for all lists $L$. Make sure that they are all true. You will receive no credit for any answer that is not an equation.

\[
\begin{align*}
\text{removeFirstOccurrence}([], L) &= \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \q
41. [MC] Suppose that linked list \texttt{lst} starts off with length \( n \). How much time does it take in terms of \( n \), in the worst case, to run the following code, where \texttt{removeFirstOccurrence} \((x, L)\) is the function from the preceding problem?

\begin{verbatim}
ListCell* L = lst;
for(int i = 1; i <= n; i++)
{
    L = removeFirstOccurrence(i, L);
}
\end{verbatim}

(a) \( \Theta(1) \)
(b) \( \Theta(\log_2(n)) \)
(c) \( \Theta(n) \)
(d) \( \Theta(n \log_2(n)) \)
(e) \( \Theta(n^2) \)

42. Draw a linked list diagram, as shown in the class notes, of list [5, 4, 3].