

RECURSION

In this material, we use the definition of singly linked list given in the exercise set of Linked Lists.

Short Answer

1. * Which repetition approach is less efficient, a loop or a recursive function? Justify your choice.
When should you choose a recursive algorithm over an iterative algorithm? And vice versa?
2. * What type of recursive function do you think would be more difficult to debug, one that uses direct recursion, or one that uses indirect recursion? Justify your choice.
3. * Explain what is likely to happen when executing a recursive function that has no way of stopping.
4. ** What does each of the following functions do for a given singly linked list with first node as `head`?

- ```
void func(Node* head){
 if(head == nullptr)
 return;
 else
 func(head->next);
 cout << head->key;
}
```
- ```
Node* func(Node* head, Node* x){
    if (head == NULL)
        return x;
    else
        head->next = func(head->next, x);
    return head;
}
```

5. ** Consider the following program, which compiles without warning, but crashes when run.

Looking at the source code, why does the program crash?

```
int factorial(int x) {
    if(x == 1)
        return 1;
    return x * factorial(x-1);
}
int main(int argc, char**argv) {
    int n = factorial(0);
    return 0;
}
```

6. ** For a given linked list with first node as `p`, in which cases does the following function returns 1?

```
int func(Node *p){
    return (    (p == NULL) ||
               (p->next == NULL) ||
               ((p->key <= p->next->key) && func(p->next))    );
}
```

7. ** Convert each of the following functions to one that uses recursion.

- ```
bool isPrime(int n) {
 if (n <= 1) // Corner case
 return false;
 for (int i = 2; i < n; i++) // Check from 2 to n-1
 if (n % i == 0)
 return false;
 return true;
}
```
- ```
void traverseList(Node* head) {
    // head: the head pointer of a singly linked list
    while (head != NULL) {
        cout << head->key << " ";
        head = head ->next;
    }
}
```

8. ** Convert each of the following functions to one that uses iteration.

- ```
long factorial(int n) {
 if (n == 0)
 return 1; // base case
 else
 return n * factorial(n-1); // recursive step
}
```
- ```
int fibonacci(int n) {
    if (n == 0 || n == 1)
        return 1;                // base cases
    else
        return fibonacci(n-1) + fibonacci(n-2); // recursive step
}
```

9. *** Is it always possible to turn any iterative function into a recursive one? If yes, give an example. Otherwise, give a counter example. Reversely, is it always possible to turn any recursive function into an iterative one? Give an example or a counter example.

10.*** Give examples to demonstrate the tail recursion and the non-tail recursion.

Is it true that the tail recursion is superior to non-tail recursion in some aspect? Explain.

Fill-in-the-Blank

1. * Fill in each of the following blanks with an appropriate terminology

- _____ recursion is when a function explicitly calls itself, while _____ recursion is when function A calls function B.
- The _____ case returns a value without making any subsequent recursive calls.
- _____ is a special case of recursion where the calling function does no more computation after making a recursive call.

2. ** What does each of the following programs output? Furthermore, mark the base case in each recursive function by putting X to the corresponding line.

Programs	Base case (X)	Output
<pre> • int func(int num){ if (num <= 0) return 0; else return func(num - 1) + num; } int main(){ cout << func(10) << endl; return 0; } </pre>		
<pre> • void func(int num){ if (num > 0){ for (int x = 0; x < num; x++) cout << '*'; cout << endl; func(num - 1); } } int main(){ func(10); return 0; } </pre>		
<pre> • void func(string str, int pos, int size){ if (pos < size){ func(str, pos + 1, size); cout << str[pos]; } } int main(){ string mystr = "Hello"; cout << mystr << endl; func(mystr, 0, mystr.size()); return 0; } </pre>		

```

• int fa(int n){
    if (n <= 1)
        return 1;
    else
        return n*fb(n-1);
}
int fb(int n){
    if (n <= 1)
        return 1;
    else
        return n*fa(n-1);
}
int main(){
    cout << fa(5);
    return 0;
}

```

3. ** For each of the following recursion functions, indicate what output is produced for every call.

```

• void mystery(int x, int y) {
    if (x > y) {
        cout << '*';
    }
    else if (x == y) {
        cout << '=' << y << '=';
    }
    else {
        cout << y << ' ';
        mystery(x + 1, y - 1);
        cout << ' ' << x;
    }
}

```

Call	Output
mystery(3, 3);	
mystery(5, 1);	
mystery(1, 5);	
mystery(2, 7);	
mystery(1, 8);	

```

• int mystery(int n) {
    if (n < 0)
        return -mystery(-n);
    else if (n < 10)
        return (n + 1) % 10;
    else
        return 10 * mystery(n / 10) + (n + 1) % 10;
}

```

Call	Output
mystery(7);	
mystery(42);	
mystery(385);	
mystery(-790);	
mystery(89294);	

- ```

void mystery(int n) {
 if (n < 0) {
 cout << '-';
 mystery(-n);
 }
 else if (n < 10) {
 cout << n << endl;
 }
 else {
 int two = n % 100;
 cout << (two / 10);
 cout << (two % 10);
 mystery(n / 100);
 }
}

```

| Call                | Output |
|---------------------|--------|
| mystery(7);         |        |
| mystery(825);       |        |
| mystery(38947);     |        |
| mystery(612305);    |        |
| mystery(-12345678); |        |

4. \*\* For each of the following functions, identify the task performed by the given function and the returned value when calling this function with specific argument(s).

- ```

int func(int a, int b) {
    if (a % b == 0)
        return b;
    else
        return foo(b, a % b);
}

```

The task performed is _____.

The returned value of `func(17,3)` is _____.

The returned value of `func(3,9)` is _____.

- ```
int func(int a){
 if (a == 1 || a == 2)
 return 1;
 else
 return func(a-1) + func(a-2);
}
```

The task performed is \_\_\_\_\_.

The returned value of `func(10)` is \_\_\_\_\_.

- ```
int func(int a, int b) {
    if (a < b)
        return 0;
    else
        return (1 + test(a-b, b));
}
```

The task performed is _____.

The returned value of `func(15,4)` is _____.

- ```
int func(int a, int b) {
 if (b == 0)
 return a;
 else
 return func(b, a % b);
}
```

The task performed is \_\_\_\_\_.

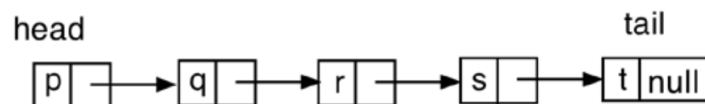
The returned value of `func(98,56)` is \_\_\_\_\_.

5. \*\* The following function takes the first element of the singly linked list, `head`, as argument. For each function, identify the task performed by the given function and the returned value when calling this function with a specific linked list.

- ```
void what(Node* head) {
    if (head == nullptr)
        return;
    else {
        mystery(head->next);
        cout << head->key;
    }
}
```

The task performed is _____.

The returned value of `what(head)` for the following linked list is _____.



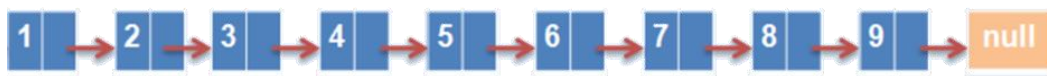
```

• int what(Node *head){
    if (head == nullptr)
        return 0;
    else{
        Node* pos = head->next;
        int tmp = what(pos);
        if (!(tmp % 2 == 0))
            cout << "temp = " << tmp << endl << pos->key << endl;
        return tmp + 1;
    }
}

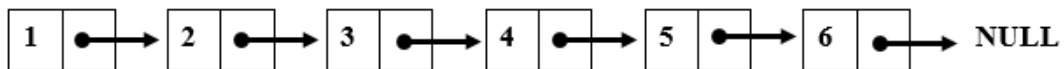
```

The task performed is _____.

The returned value of `what(head)` for the following linked list is _____.



6. ** Given the following singly linked list, in which the pointer head points to the first node.



What are the results of the following functions?

```

void func1(Node* start) {
    if (start == nullptr)
        return;
    func1(start->next);
    cout << start->key;
}

```

```

void func2(Node* start) {
    if (start == nullptr)
        return;
    cout << start->key;
    if (start->next != nullptr)
        func2(start->next->next);
    cout << start->key;
}

```

`func1(head);` _____

`func2(head);` _____

7. ** Complete the following recursive function to return a list of negative numbers obtained from a given list. The given list and its size are stored in the arguments, `in` and `nin`, respectively. The resulting list and its size are stored in the arguments, `out` and `nout`, respectively.

```

void negatives(int in[], int nin, int out[], int &nout){
    if (_____) {

    }
    else {

    }
}

```

For example, let `in[] = { 6, 3, -4, 9, -11, -2, 5 }` and `nin = 7`, after calling `negatives(in, nin, out, nout)`, `out` will be `{-4, -11, -2}` and `nout` be 3.

8. ** Which formula is calculated by each of the following recursive functions?

```

• int mystery(int a){
    if (a == 0)
        return 0;
    else
        return mystery(a - 1) + 2 * a - 1;
}

```

The formula is _____

```

• double mystery(int n){
    if (n == 1)
        return 1;
    else
        return (pow(n, 2) + mystery(n - 1));
}

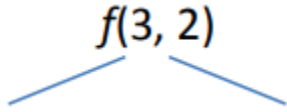
```

The formula is _____

9. *** The function f is defined for two non-negative integers, a and b , recursively as follows

$$f(a, b) = \begin{cases} 0 & \text{if } (a = 0 \text{ or } b = 0) \\ f(a - 1, b - 1) + 2a - 1 & \text{if } (a = b) \\ f(a - b, b) + f(b, b) & \text{if } (a > b) \\ f(a, a) + f(b - a, a) & \text{if } (a < b) \end{cases}$$

Compute $f(3, 2)$ by drawing a recursion tree showing all computations required and then use your tree to compute the answer.

	$f(3, 2) =$ _____
---	-------------------

State the common name for f or write a very compact non-recursive definition of f :

10. *** Consider the following functions

<pre> bool is_even(unsigned int n){ if (n == 0) return true; else return is_odd(n - 1); } </pre>	<pre> bool is_odd(unsigned int n) { if (n == 0) return false; else return is_even(n - 1); } </pre>
--	--

What are the results of the following functions?

$\text{is_even}(11);$ _____ $\text{is_odd}(23);$ _____

True or False

Choose T (True) or F (False) for each of the following statements and then briefly explain in one or two sentences.

1. T F Indirect recursion can involve only two functions.
2. T F Recursion uses more memory than loop.
3. T F Recursive algorithms are certainly less efficient than iterative algorithms.
4. T F Traversing a linked list can be implemented by either a loop or recursion.
5. T F The below function is a bad recursive function.

```
int example(unsigned int a){
    if (a == 0)
        return 0;
    else
        return example(a+1);
}
```

6. T F All recursive functions can be implemented iteratively, and vice versa.
7. T F If a recursive function does not have a base case, the compiler will detect this and display a compiler error.
8. T F It is possible to have more than one recursive call within a function.
9. T F A function can be considered recursive if it has a direct or an indirect call to itself.
10. T F The following iterative function and recursive function are not equivalent.

```
int mystery(int n) {
    int sum = 0;
    while (n > 0) {
        sum = sum + n;
        n--;
    }
    return sum;
}

int mystery(int n){
    if(n > 0)
        return (n + mystery(n - 1));
    return 0;
}
```

Algorithm Workbench

1. * For each of the following requirements, write a recursive function to accept a positive integer, x , and then calculate
 - a) The factorial $x!$
 - b) The $(x + 2)^{\text{th}}$ Fibonacci number, given that the first and second elements are 0 and 1.
 - c) The sum of series $1 + 2 + \dots + x$.
2. * For each of the following questions, write a recursive function to accept two integers, x and y , and then calculate
 - a) The value of x times y : $x*y$
 - b) The value of x powers y : x^y
 - c) Their greatest common divisor, $\text{gcd}(x, y)$
 - d) Their least common multiple, $\text{lcm}(x, y)$
3. ** Write a function to receive an array of integers and its size, and then recursively calculate the sum of all the numbers in the array.
4. ** Write a function to read in a positive integer and recursively build up the corresponding binary representation of that integer.
5. ** For each of the following requirements, write a recursive function to receive the head of a singly linked list, head , and then
 - a) Print out the number of nodes in the linked list.
 - b) Print out alternate nodes of the linked list.
 - c) Return the value of the maximum key in the linked list or -1 if the list is empty.
 - d) Determine whether a given data key k exists in the linked list.
 - e) Insert a given node x to the end of the linked list
 - f) Delete the whole linked list
6. ** Write a function to receive a string and recursively determine whether the string is a palindrome. Here are some examples of palindrome.

Able was I, ere I saw Elba	Desserts, I stressed
A man, a plan, a canal, Panama	Kayak
7. ** Write the recursive function, isReverse , to accept two strings and return true if the two strings contain the same sequence of characters as each other but in the opposite order, and false otherwise. Note that the function is case-insensitive; the empty string, as well as any one-letter string, is considered to be its own reverse; the string could contain characters other than letters, such as numbers, spaces, or other punctuation, which should be treated like any other character. For example,

Call	Value Returned
isReverse("CSE143", "341esc")	true
isReverse("Q", "Q")	true
isReverse("", "")	true
isReverse("Go! Go", "OG !OG")	true
isReverse("Obama", "McCain")	false
isReverse("hello!!", "olleh")	false
isReverse("", "x")	false
isReverse("madam I", "i m adam")	false

8. ** Write the recursive function, **repeat**, to accept a string **s** and a non-negative integer **n** and return a string consisting of **n** copies of **s**.

For example

Call	Value Returned
repeat("hello", 3)	"hellohellohello"
repeat("this is fun", 1)	"this is fun"
repeat("wow", 0)	""
repeat("hi ho! ", 5)	"hi ho! hi ho! hi ho! hi ho! hi ho!"

String concatenation is an expensive operation, so it is best to optimize your string manipulation.

Refer to: <https://stackoverflow.com/questions/611263/efficient-string-concatenation-in-c>.

9. *** Write the recursive function, **digitMatch**, to accept two non-negative integers and return the number of digits that match between them. Two digits match if they are equal and have the same position relative to the end of the number (i.e., starting with the ones digit). In other words, the method should compare the last digits of each number, the second-to-last digits of each number, the third-to-last digits of each number, and so forth, counting how many pairs match. For example, for **digitMatch(1072503891, 62530841)**, the method would compare as follows and return 4 in this case because 4 of these pairs match (2-2, 5-5, 8-8, and 1-1)

```

1 0 7 2 5 0 3 8 9 1
    | | | | | | |
6 2 5 3 0 8 4 1

```

Call	Value Returned
digitMatch(38, 34)	1
digitMatch(5, 5552)	0
digitMatch(892, 892)	3
digitMatch(298892, 7892)	3
digitMatch(380, 0)	1
digitMatch(123456, 654321)	0
digitMatch(1234567, 67)	2

10. *** Implement selection sort using recursion rather than loops.

Repeat the question for insertion sort and bubble sort.