

**University of Toronto  
Faculty of Applied Science and Engineering**

**ECE 244F**

**PROGRAMMING FUNDAMENTALS**

**Fall 2017**

**Final Examination**

**Examiners: T.S. Abdelrahman and D. Yuan**

**Duration: Two and a Half Hours**

**This exam is OPEN Textbook and CLOSED notes. The use of computing and/or communicating devices is NOT permitted.**

**Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted.**

**Work independently. The value of each part of each question is indicated. The total value of all questions is 100.**

**Write your name and student number in the space below. Do the same on the top of each sheet of this exam book.**

**Name:** \_\_\_\_\_

**Student Number:** \_\_\_\_\_

**Q1.** \_\_\_\_\_

**Q11.** \_\_\_\_\_

**Q2.** \_\_\_\_\_

**Q12.** \_\_\_\_\_

**Q3.** \_\_\_\_\_

**Q13.** \_\_\_\_\_

**Q4.** \_\_\_\_\_

**Q14.** \_\_\_\_\_

**Q5.** \_\_\_\_\_

**Q15.** \_\_\_\_\_

**Q6.** \_\_\_\_\_

**Q16.** \_\_\_\_\_

**Q7.** \_\_\_\_\_

**Q17.** \_\_\_\_\_

**Q8.** \_\_\_\_\_

**Q18.** \_\_\_\_\_

**Q9.** \_\_\_\_\_

**Q19.** \_\_\_\_\_

**Q10.** \_\_\_\_\_

**Total**

--

**Question 1. (9 marks).** *General.*

Answer the following questions as indicated. No justification of your answer is required.

**(a) (1 mark).** What is the name of the programming language used in this course?

Write your answer here:

**(b) (1 mark).** Stream variables (e.g. types `istream`, `ostream`, and `sstream`) must be passed by reference in function calls.

Circle one answer:    Yes            No

**(c) (1 mark).** An abstract class is one that has no data member.

Circle one answer:    Yes            No

**(d) (1 mark).** A derived class can remove data member variables from the base class.

Circle one answer:    Yes            No

**(e) (1 mark). Circle one answer.** Algorithm A has an *average* time complexity of  $O(n^2)$ . Algorithm B has an *average* time complexity of  $O(n \log n)$ .

1. Algorithm B is always faster than algorithm A.
2. Algorithm A is always faster than algorithm B.
3. Algorithm A can be faster than algorithm B.
4. None of the above.

**(f) (1 mark). True or False?** The worst case complexity for quicksort is  $O(n^2)$ .

Circle one answer:    Yes            No

**(g) (1 mark).** The complexity of inserting an element into an unsorted linked list can be  $O(1)$ .

Circle one answer:    Yes            No

**(h) (2 mark).** What is the output generated by the program listed below? Write the output in the box.

```
#include <iostream>
using namespace std;

class One {
public:
    One() {cout << "created\n";}
    One(const One& rhs) {cout << "copied\n";}
    ~One() {cout << "destroyed\n";}
};

One first;

int main () {
    One second(first);
    cout << "starting\n";
    One third[2];
    cout << "finishing\n";
    return (0);
}
```

**Question 2. (5 marks).** *Objects with pointers.*

The following partial definition and implementation are for a class `webAddress`, which is intended to store the sequence of characters representing the address of a web site (e.g., `www.google.com` or `www.utoronto.ca`).

```
// Definition
class webAddress {
private:
    char* _address;
    int   _length;
public:
    webAddress(const char* addr);
    ~webAddress();
    webAddress& operator=(webAddress& rhs);
    : // Other members not relevant to the question
};

// Implementation
webAddress::webAddress(const char* addr) {
    _length = strlen(addr);
    _address = new char[_length];
    strcpy(_address, addr);
}

webAddress::~~webAddress() {
    _address = NULL;
}

webAddress& webAddress::operator=(webAddress& rhs) {
    _length = rhs._length;
    _address = rhs._address;
}
```

However, the implementation part of this class has problems. Identify these problems and re-write the implementation of the three member functions above to eliminate these problems. If a method needs no modification, write “the same” inside the box.

```
webAddress::webAddress(const char* addr) {
```

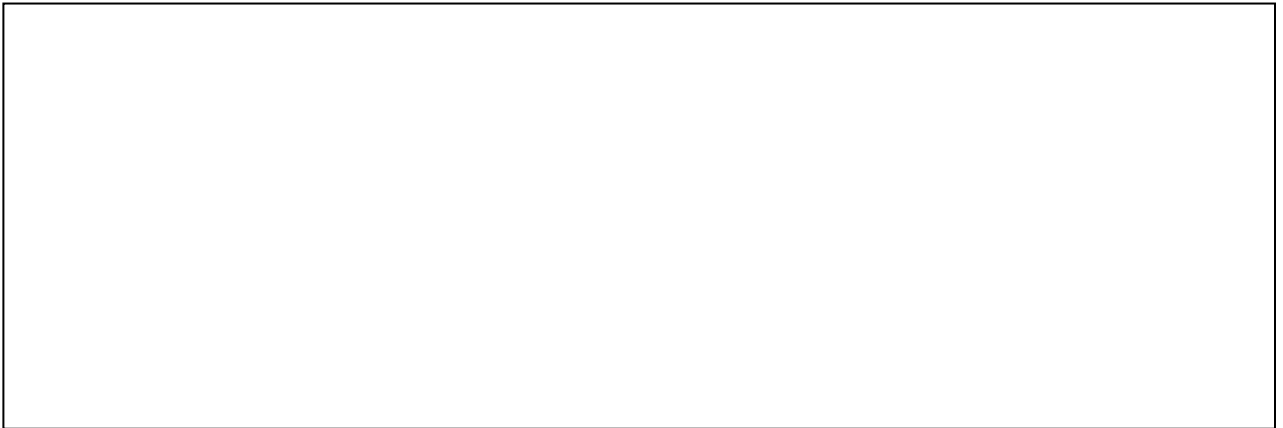
```
}
```

```
webAddress::~~webAddress() {
```



```
}
```

```
webAddress& webAddress::operator=(webAddress& rhs) {
```



```
}
```

**Question 3. (3 marks).** *Memory Management.*

Consider the three classes Strange, Weird, and Bizarre. The data members of each class are shown.

```
class Strange {
    int a;
    int b;
    int c[10];
    Weird* w;
    :
    :
};

class Weird {
    int x;
    Bizzare b;
    :
    :
};

class Bizzare {
    float y;
    char * name;//char array
    :
    :
};
```

A program constructs an object of type Strange on the heap with the following statement:

```
Strange* p = new Strange(); // Assume the default constructor exists
```

Later in the program, the object is deleted with the following statement:

```
delete p;
```

Write the destructor for each of the three classes so that the delete statement causes no memory leaks. If a destructor has no code in it, write “// No Code” in the box.

```
Strange::~~Strange() {

}

}
```

```
Bizzare::~~Bizzare() {

}

}
```

```
Weird::~~Weird() {

}

}
```

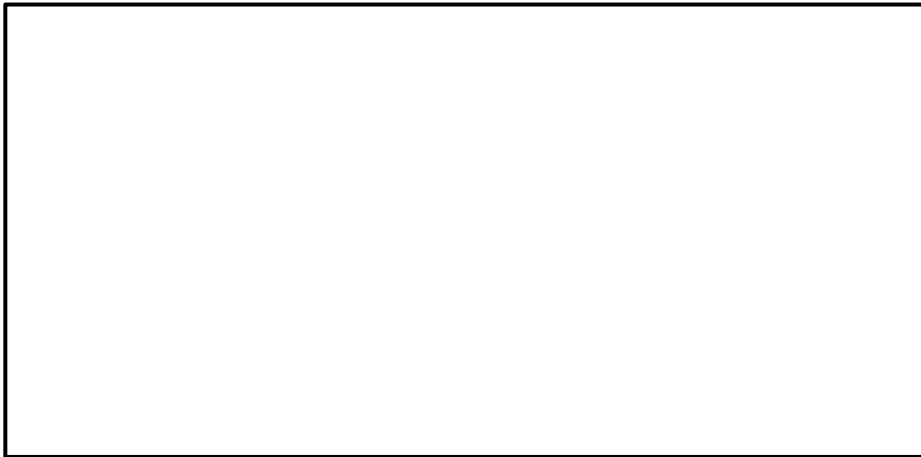
**Question 4. (4 marks). Overloaded Operators.**

Consider the following class definition:

```
#include "Wheel.h"    // Contains the declaration of class Wheel

class Car {
    private:
        int    v;
        Wheel* p;
    public:
        :
}
```

Assume that the class `Wheel` is properly implemented. Write an overloaded assignment operator as a member of the class `Car`. This operator **must** do a deep assignment. Write both the operator's header and its body below.



**Question 5. (4 marks). I/O.**

We wish to write a program that prompts the user to enter her first and last names at the standard input and then print her initials to the standard output.

```
#include <iostream>
using namespace std;

int main () {
    char firstInitial;
    char lastInitial;

    cout << "Enter your first name followed by your last name: ";
    
    cout << "Your initials are: " << firstInitial
         << lastInitial << endl;

    return (0);
}
```

Complete the program by writing code in the box shown above. You are not to declare/use any other variables than the ones shown in the program. However, you may use any of the functions of `iostream` (e.g., `cin.peek()`, `cin.ignore()` or `cin.fail()`). You may also assume that the user will always enter her first name followed by her last name.

Your answer should be **at most 3 lines** of code. You will lose marks for additional lines.

Here are example inputs and outputs for the program (user input is shown in italics):

Enter your first name followed by your last name: *Patricia Williams*  
Your initials are PW

Enter your first name followed by your last name: *Sandy Smith*  
Your initials are SS

Enter your first name followed by your last name: *Rachel McDonalds*  
Your initials are RM



**Question 6. (3 marks).** *Recursion.*

Write a **recursive** function called `printReverse(int* array, int start, int end)` that prints the elements of an integer array in reverse. For example, given the following 4-element array `a`:

	0	1	2	3
a	0	9	1	2

The function invoked as `printReverse(a, 0, 3)` prints to the standard output (using `cout`):

2 1 9 0

Write your answer below.

```
void printReverse(int* array, int start, int, end){
```

```
}
```

**Your code should be no more than 3-4 lines. Longer answers will be penalized.**

**Question 7. (4 marks).** *Recursion.*

Consider the following recursive function, `int mystery(int n, int depth):`

```
#include <iostream>
using namespace std;

int mystery(int n, int depth) {
    for (int i=0; i < n-depth+1; ++i) cout << "+";
    cout << endl;
    if (n <= 1) {cout << "*" << endl; return 0;}
    return (mystery(n-2, depth+1) + mystery(n-3, depth+1));
}

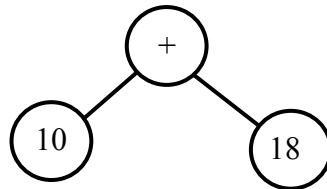
int main() {
    mystery(4,0);
    return (0);
}
```

What is the output produced when `mystery(4,0)` is called?

**Write the output below.** Enter one line of the output per row in the order in which the output is produced. You may or may not need all rows in the table.


**Question 8. (4 marks).** *Binary Trees.*

An *expression tree* is a binary tree whose leaf nodes represent integer values and whose non-leaf nodes represent arithmetic operators. An example of a simple expression tree that represents the expression  $10 + 18$  is shown below.



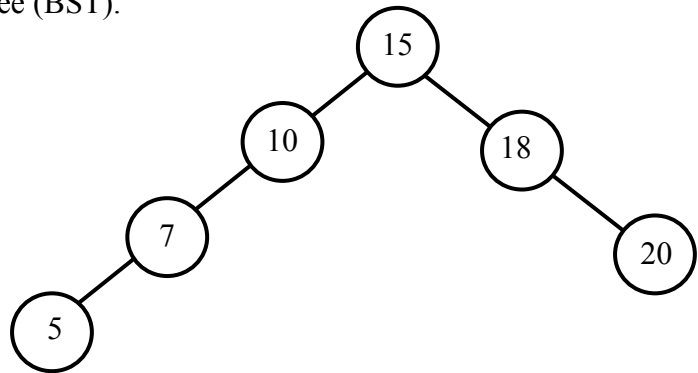
Draw the expression tree whose pre-order traversal is:  $+ \ - \ * \ 6 \ / \ 8 \ 4 \ + \ 3 \ 5 \ 10$

All operators are binary operators, i.e., they take two operands.

Draw the tree here. It should represent a valid expression.

**Question 9. (6 marks).** *Binary Search Trees.*

The tree T shown here is a Binary Search Tree (BST).



**(a) (2 marks).** Insert a node with the key “6” onto the tree T and show the resulting tree. You must use the insertion algorithm developed in the lectures.

**(b) (2 marks).** Starting with the tree in part (a) after the insertion of “6”, delete the node with the key “10” and show the resulting tree.

**(c) (2 marks).** Starting with the tree in part (b) after the removal of “10”, re-insert the node with they key “10” onto the tree and show the resulting tree. Again, use the insertion algorithm developed in the lectures.

**Question 10. (5 marks).** *Tree Traversals.*

Write a **recursive** function to perform the *odd-even-order* traversal of a binary tree, meaning that for each node of the tree, the function always visits the node first. Then if the key of the node is odd, the function traverses the left subtree (in odd-even-order) followed by the right subtree (also in odd-even-order). However, if the key of the node is even, the function traverses the right subtree (in odd-even-order) followed by the left subtree (also in odd-even-order).

**Hint:** The % operator can be used to determine if the integer key is odd or even.

Assume that visiting a node simply prints its key to `cout`. **Your code should be less than 10 lines of code! Long code will be penalized!**

You may assume the following declarations:

```
class treeNode {
    public:
        int data;
        treeNode *left;
        treeNode *right;
};

treeNode *Root; // root of the tree

void oddevenorder (treeNode *rt) {
```

```
// This is how oddevenorder is called
oddevenorder(Root);
```

**Question 11. (5 marks).** *3-Way Trees (Caution: Difficult).*

A *3-way tree* is a tree in which each node has up to 2 keys, named `key0` and `key1`, and up to 3 children, named `child0`, `child1` and `child2`. A 3-way tree is *3-way search-tree* (TWST) if for every node in the tree, the following properties hold:

- The keys in each node are in increasing order, i.e.,  $\text{key0} < \text{key1}$
- The keys in the subtree rooted at `child0` are smaller than `key0`
- `key0` is smaller than the keys in the subtrees rooted at `child1` and `child2`
- The keys in the subtrees rooted at `child0` and `child1` are smaller than `key1`
- `key1` is smaller than the keys in the subtree rooted at `child2`

The code below shows one possible definition of a node in a TWST tree.

```
class TWSTnode {
    public:
        int key0;    // Initialized to -1 if no key value
        int key1;    // Initialized to -1 if no key value
        TWSTnode* child0; // Initialized to NULL if no child
        TWSTnode* child1; // Initialized to NULL if no child
        TWSTnode* child2; // Initialized to NULL if no child
};

// The root of a TWST
TWSTnode* root;
```

Write a recursive non-member function that searches a TWST for a node that has a key `k` as one of its two keys and returns a pointer to it. It returns `NULL` if such a node does not exist.

```
TWSTnode* TWSTsearch(TWSTnode* r, int k) {
```

```
}
// Example use
TWSTsearch(root, 10);
```

**Your code should be really short. Long answers will be penalized.**

**Question 12. (7 marks). Inheritance.**

Consider the following definitions of base and derived classes.

```
class person {
    private:
        int ID;
    public:
        person();
        ~person();
        int getID();
        virtual void print() const = 0;
};

class employee : public person {
    private:
        string company;
    public:
        employee();
        ~employee();
        string getCompany() const;
        void setCompany(string comp);
        virtual void print() const;
};

class developer : public employee {
    private:
        string prog_language;
    public:
        developer();
        ~developer();
        string getLanguage() const;
        void setLanguage(string lang);
        virtual void print() const;
};
```

Assume the implementation of these classes is correct even though the implementation code is not shown here.

In the table below, indicate whether each group of statements (i.e., a row in the table) generate a compiler error or not by placing an X in the appropriate column. All statements appear in the main function of a program. Consider each group of statements (i.e., row in the table) by itself.

Statement	Error	Not Error
person p;		
employee e; developer d; d = e;		
employee e; developer d; e = d;		
person* pp;		
employee* pe; pe = new employee();		
person* pp = new employee();		
developer* pd; employee* pe = new employee(); pd = pe;		
developer* pd = new developer(); employee* pe; pe = pd; pe->getLanguage();		



**Question 13. (4 marks).** *Inheritance.*

Consider the code shown below. The implementation of the various functions is included in the class definition.

```
#include <iostream>
using namespace std;

class Base {
public:
    Base(int b) {cout << "Construct Base: " << b << endl;}
    ~Base() {cout << "Destruct Base" << endl;}
    void print() {cout << "Print Base" << endl;}
};

class D1 : public Base {
public:
    D1(int b, int d1) : Base(b) {
        cout << "Construct D1: " << d1 << endl;
    }
    ~D1() {cout << "Destruct D1" << endl;}
};

class D2 : public D1 {
public:
    D2(int b, int d1, int d2) : D1(b, d1) {
        cout << "Construct D2: " << d2 << endl;
    }
    ~D2() {cout << "Destruct D2" << endl;}
    void print() {cout << "Print D2" << endl;}
};

int main() {

    Base *ptr = new Base(0);
    Base *ptr1 = new D1(1, 2);
    Base *ptr2 = new D2(2, 3, 4);

    ptr->print();
    ptr1->print();
    ptr2->print();

    delete ptr;
    return 0;
}
```

What does the code in `main()` print to the output when the code is executed?

**Write the output below**

**Question 14. (6 marks).** *Inheritance.*

Consider the following code segment that defines a base class `Base` and a derived class `Derived`. The implementation of the various member methods is given inside the class definition.

```
#include <iostream>
using namespace std;

// This is the base class
class Base {
protected:
    int* pa;
public:
    Base() { pa = NULL; }

    Base (const Base & original) {
        pa = new int;
        *pa = *(original.pa);
    }

    Base (const int v) {
        pa = new int;
        *pa = v;
    }

    ~Base() { delete pa; }

    Base & operator= (Base & rhs) {
        cout << "Base operator=" << endl;
        if (pa == NULL) pa = new int;
        *pa = *(rhs.pa);
        return (*this);
    }
};
```

```

// This is the derived class
class Derived : public Base{
private:
    int* pb;
public:
    Derived() { pb = NULL; }

    Derived (const int _base, const int _derived) : Base(_base) {
        pb = new int;
        *pb = _derived;
    }

    Derived (const Derived & original) {
        pb = new int;
        *pb = *(original.pb);
    }

    Derived (const int _derived) {
        pb = new int;
        *pb = _derived;
    }

    ~Derived () { delete pb; }

    Derived & operator= (Derived & rhs) {
        cout << "Derived operator=" << endl;
        if (pb == NULL) pb = new int;
        *pb = *(rhs.pb);
        return (*this);
    }

    void print () {
        if (pa != NULL) cout << "Base::pa -> " << *pa << endl;
        if (pb != NULL) cout << "Derived::pb -> " << *pb << endl;
    }
};

```

**(a) (3 marks).** What is the output produced by the following `main` function?

```
int main() {  
    Derived p(100,101);  
    Derived q(p);  
    p.print();  
    q.print();  
    return 0;  
}
```

Write the output here.

**(a) (3 marks).** What is the output produced by the following `main` function?

```
int main() {  
    Derived p(100,101);  
    Derived q;  
    q = p;  
    p.print();  
    q.print();  
    return 0;  
}
```

Write the output here.

**Question 15. (7 marks).** *Complexity Analysis.*

Determine the **worst-case** time complexity (expressed in big-O notation) for each of the program segments below as a function of the size of the input  $n$ . Show the details of your analysis and clearly indicate your final result.

**(a) (2 marks)** The size of the input is  $n$ .

```
w=0;
for (int i=0; i < n; ++i) {
    for (int j=0; j < 10; ++j) {
        for (int k=0; k < j; ++k) {
            w = w + 1;
        }
    }
}
```

$T(n) = O( \quad )$

**(b) (2 marks).** The size of the input is  $n$ .

```
for (int i=0; i < n; ++i) {
    for (int j=0; j < 5*n ; ++j) {
        o(1)
    }
}
```

$T(n) = O( \quad )$

**(c) (3 marks).** Assume for simplicity that  $n$  is a power of two.

```
int mystery (int k, int n) {
    if (n <= 1) return(0);
    return (mystery (n/2)+ 2*mystery (n/2)); // integer division
}
```

Write the recurrence equation for  $T(n)$ .

Solve the recurrence equation to obtain an expression of  $T(n)$  in terms of  $n$ .

Express  $T(n)$  using the big-O notation.

$$T(n) = O( \quad )$$

**Question 16. (4 marks).** *Complexity Analysis.*

Consider the recursive code below.

```
void split (int a[], int begin, int end) {
    if (begin >= end) return;
    int middle = (begin + end)/2;
    split(a, begin, middle);
    split(a, middle + 1, end);
    scan(a, begin, middle, end);
}

void scan(int a[], int begin, int middle, int end) {
    int l1 = begin;
    int l2 = middle + 1;
    int i;
    int b[MAX]; // MAX is a large enough number

    for (i = begin; l1 <= middle && l2 <= end; i++) {
        if (a[l1] <= a[l2]) {
            b[i] = a[l1];
            l1 = l1 + 1;
        }
        else {
            b[i] = a[l2];
            l2 = l2 + 1;
        }
    }
    while (l1 <= middle) {
        b[i] = a[l1];
        i = i + 1;
        l1 = l1 + 1;
    }
    while (l2 <= end) {
        b[i] = a[l2];
        i = i + 1;
        l2 = l2 + 1;
    }
    for (i = begin, i <= end; i++) a[i] = b[i];
}
```

Determine the **worst-case** time complexity of code, given the size of the input array,  $n$ . You can assume the function `split` to be invoked as `split(a, 0, n-1)`.



Write the recurrence equation for  $T(n)$ , the execution time of the code.

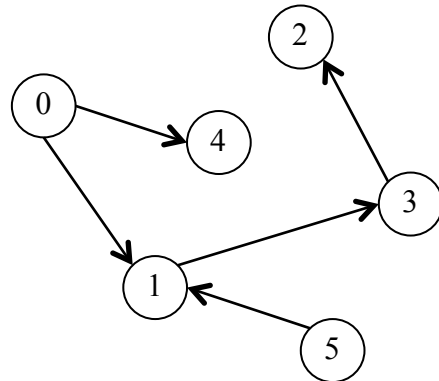
Solve the recurrence equation to obtain an expression of  $T(n)$  in terms of  $n$ .

Express  $T(n)$  using the big-O notation.

$$T(n) = O( \quad )$$

**Question 17. (4 marks).** *Graphs.*

Consider the graph shown below.



**(a) (2 marks).** Show the *adjacency matrix* representation of the graph.

**(b) (2 marks).** Give a *depth-first* traversal of the graph. Assume that visiting a node means printing its number to the standard output.

Give the traversal here:

--	--	--	--	--	--

**Question 18. (8 marks).** *Binary Search Trees (Caution: Difficult).*

The following is a definition of a node in a binary search tree. The member function `secondLargest()` returns the key of the ***second*** largest key in the binary search tree rooted at the node.

```
class TreeNode {
private:
    int value;
    TreeNode *left;
    TreeNode *right;
public:
    int secondLargest();
};

// Pointer to root of the tree
TreeNode* root;
```

Write the function body of `TreeNode::secondLargest()` in the box below. The function is invoked as `root->secondLargest()` and returns the second largest element of the binary search tree rooted at `root`. You may assume that you are always given a tree with at least two nodes.

```
int TreeNode::secondLargest() {

}

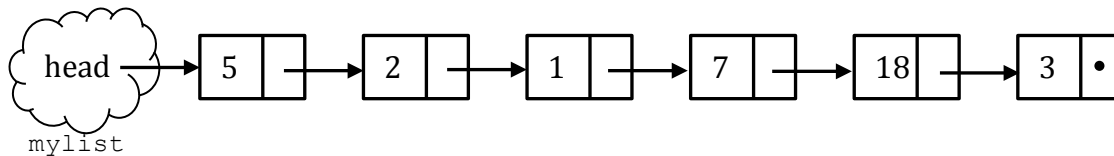
}
```

Your program will be marked by correctness and *simplicity*.

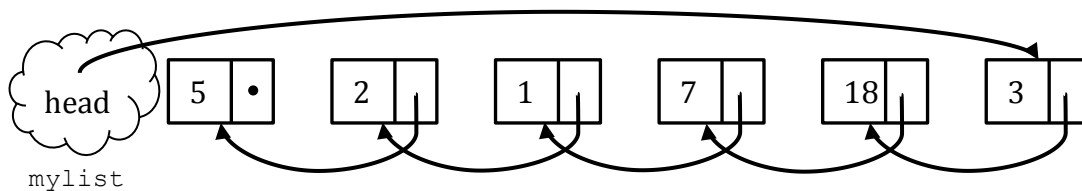
**Question 19. (8 marks).** *Linked Lists (Caution: Difficult).*

Consider the following class definitions of a node in a linked list and a linked list. The function `List::reverse()` reverses the linked list pointed to be `List::head`. Thus, in the example below, the invocation of the function as `mylist.reverse()` reverses the list as shown.

Before invocation:



After invocation:



```
class Node {
private:
    int data;
    Node *next;
public:
    int getData() const { return data; }
    Node *getNext() const { return next; }
};

class List {
private:
    Node *head;
public:
    void reverse(); // You write this function
};
```

Here are the requirements for your `List::reverse()` function:

- After the invocation, the entire linked list is reversed with the head properly pointing to the new head (i.e., the last node in the old linked list).
- The linked list can be traversed only once.
- No variables, in addition to what is already provided in the code skeleton below, can be used. Similarly, no additional nodes may be allocated.

Write the function body of `List::reverse()` in the box below. Your program will be marked by correctness and *simplicity*.

```
void List::reverse() {
    Node *p = NULL;

```