University of Toronto
Faculty of Applied Science and Engineering

ECE 244F

PROGRAMMING FUNDAMENTALS

Fall 2019

Final Exam

Examiners: T.S. Abdelrahman and M. Shaghaghi

Duration: 2.5 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 exam book. Answer all questions in the space provided. No additional sheets are permitted.

Work independently. The value 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.

In answering the questions, you must assume the C++ 11 standard and the use of the `g++` compiler available on the Linux machines in the ECF labs.

Name: _____

Student Number: _____

Q1. _____/6	**Q10. _____/4**
Q2. _____/5	**Q11. _____/3**
Q3. _____/4	**Q12. _____/7**
Q4. _____/4	**Q13. _____/8**
Q5. _____/9	**Q14. _____/6**
Q6. _____/10	**Q15. _____/2**
Q7. _____/4	**Q16. _____/12**
Q8. _____/3	**Q17. _____/4**
Q9. _____/4	**Q18. _____/5**

Total

Question 1. (6 marks). *Pointers and Parameter Passing*.

(a) (2 marks). Assume the definition and implementation of a class `Mystery` exist. Write the prototype of a non-member function called `doSomething` that takes a pointer to an object of type `Mystery`, called `p`, passed by reference and returns an object of type `Mystery` by value. Write your answer in the box below.

(b) (1 marks). Given the following declarations, what is the value of each of the following expressions? Write your answer where indicated

```
int i=10;
int *pi=&i;
double d=12.5;
double *pd=&d;

(*pi) + 1;        value:

(*pd) + 1;        value:
```

.

(c) (3 marks). What is the output of the following program? Write your answer in the box below.

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

int main() {

    int i=10;
    int j=20;
    int* p;
    int* q;
    int** r;
    p = &i;
    q = &j;
    *q = 5;
    if (*p < j) r = &p;
    else r = &q;
    **r = (**r) + (*p) + 1;
    cout << i << " " << j << endl;

    return (0);
}
```

Question 2. (5 marks). *Pointers and Memory Allocation*.

Consider the following classes whose instances respectively represent points in two-dimensional space and the line between two points, `a` and `b`.

```
class Point {
    public:
       int* x;
       int* y;
};
```

```
class Line {
    public:
       Point* a;
       Point* b;
};
```

A non-member function, called `create_line`, dynamically allocates the memory space needed to represent the two end points of a line as well as the line itself. The function takes four parameters, all integers. The first two represent the x and y coordinates of point `a` of a line to be created. The second two represent the x and y coordinates of point `b` of the line. The dynamically allocated memory is initialized from the function parameters. The dynamically allocated memory is then de-allocated.

The skeleton of the code is given to you below. The code uses two pointers, `line_ptr` and `pptr`, as well as the classes defined above. You need to complete the implementation of the function, with the following restrictions. You are not allowed to declare any additional variables other than `line_ptr` and `pptr`. Further, you cannot use `line_ptr` in the code you write, i.e., you can only use `pptr` in your code.

You need not worry about any error handling.

```
void create_line (int x_a, int y_a, int x_b, int y_b) {
  Line*  line_ptr;
  Line** pptr = &line_ptr;
  /* Write your code here. You cannot use line_ptr!! */
```

```
}
```

Question 3. (4 marks). *Arrays and Objects.*

Consider the following modified and simplified definition/implementation of the class `Shape`, used in your lab assignment 3. You may assume the class is correctly defined/implemented.

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

class Shape {
   private:
       string name;
       string type;
   public:
              Shape() { }
       string getName() const {return name;}
       string getType() const {return type;}
       void   setName(string n) {name = n;}
       void   setType(string t) {type = t;}
};
```

A `main` function dynamically allocates then de-allocates `n Shape` objects, with other dynamically allocated variables, where `n` is an integer value read from `cin`. The code to de-allocate the objects and the other variables (so that no memory leak exists) is shown below at the end of `main`.

Give the code to allocate the `n` objects and to set the type of each of the `n Shape` objects to the string `circle`. Assume `iostream` has been included and that the `std` namespace is used.

```
int main () {
   int n;
   cin >> n;

   // Write code to allocate objects and other variables here









   // Write code to set type of each Shape object to circle here




// De-allocate all dynamically allocated variables
   for (int i=0; i < n; ++i) {
       delete *(p[i]); delete p[i];
   }
   delete [] p;
   return (0);
}
```

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

A programmer writes two classes, `X` and `Y`, defined and implemented correctly. The programmer now uses them together, along with integers and booleans, to perform the operations shown in `main` below.

```
int main() {
   X a(1);
   Y b(2);
   bool isIt = false;

   isIt = (a == b);
   isIt = (b == a);

   isIt = (a == 3);
   isIt = (3 == b);

   return (0);
}
```

The programmer **must** have written a set of overloaded operator== functions to make the above code work in `main`.

(a) (1 mark). How many of these operator== functions **must** be non-member functions? Write your answer in the box across.

(b) (1 mark). How many of these operator== functions **may** be member methods of class `X`? Write your answer in the box across.

(c) (1 mark). How many of these operator== functions **may** be member methods of class `Y`? Write your answer in the box across.

(d) (1 mark). Is it possible to have none of the operator== functions as members of classes `X` or `Y`? Write **Yes** or **No** in the box across.

Question 5. (9 marks). *Objects with Pointers.*

Consider the following class definition and implementation.

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

class Duo {
   private:
      int* p;
      int* q;
   public:
      Duo(int a, int b) {
         p = new int;
         *p = a;
         q = new int;
         *q = b;
      }

      int get_a() {return *p;}
      int get_b() {return *q;}

      void set_a(int a) {*p = a;}
      void set_b(int b) {*q = b;}

      Duo funnyMultiply(Duo & rhs) {
          Duo temp(0,0);
          *(temp.p) = (*p) * *(rhs.p);
          *(temp.q) = (*q) * *(rhs.q);
          *(rhs.p) = *(rhs.p) - 1;
          *(rhs.q) = *(rhs.q) - 1;
          return (temp);
      }

      Duo print() {
          cout << *p << " " << *q << endl;
          return (*this);
      }
};
```

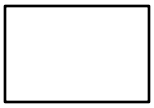
The following `main` program uses class `Duo`.

```
int main() {
    Duo X(3,5);
    Duo Y(8,9);
    Duo Z(2,4);

    Z = X.funnyMultiply(Y);
    Z.print();                                  // Statement 1
    Z.set_a(1);
    Z.set_b(2);
    Z.print();                                  // Statement 2
    X.print();                                  // Statement 3
    Y.print();                                  // Statement 4

    Duo W(6,12);
    Duo V(2,3);
    W.print().funnyMultiply(V).print();         // Statement 5
    W.print();                                  // Statement 6

    // Point A

    cout << "Program is done" << endl;

    return (0);
}
```

(a) (7 marks). Write the output produced by each of the labeled statement (`Statement 1` to `Statement 6`) in `main`. Write your answer in the table below.

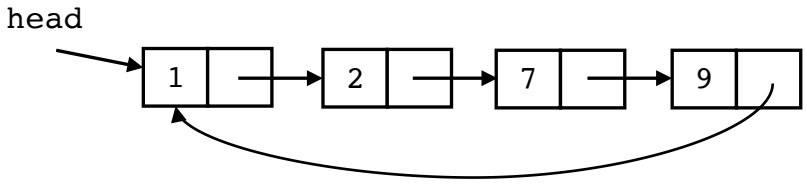
`Statement 1`	
`Statement 2`	
`Statement 3`	
`Statement 4`	
`Statement 5`	
`Statement 6`	

(b) (2 marks). How many integers exist in memory in the form of a memory leak when execution reaches Point A in the main function above? Write your answer in the box below.



Question 6. (10 marks). *Linked Lists.*

Circular linked lists are a variation on linked lists described in class. In a circular linked list, the `next` field in the last node in the list is not set to `NULL` (or `nullptr`). Rather, the field is made to point to the first node in the list, hence the name "circular". An example of a circular linked list is shown below.



Consider the class `ListNode` shown below. It represents a node in a circular linked list. It is similar to the one described in class and that you implemented in the labs, but all members are public for simplicity. The declaration of a `head` pointer, which points to the head of the list, is also shown.

```
class ListNode {
   public:
     int id;
     ListNode* next;
};

ListNode* head;
```

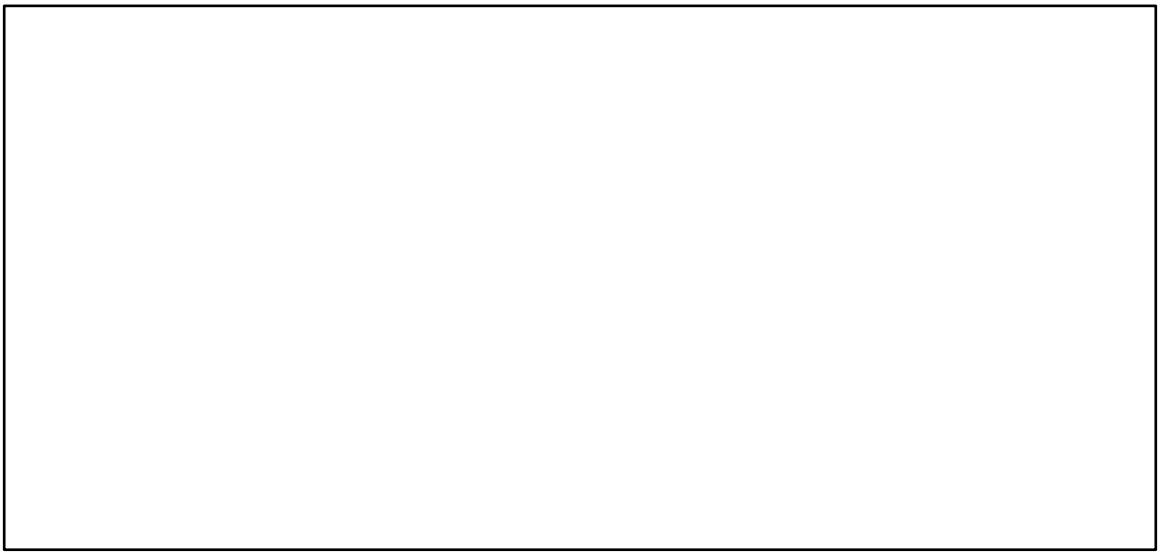
(a) (3 marks). Write a non-member function `traverse (ListNode* h)` that traverses the linked list. The function is invoked as `traverse (head)` to start the traversal at the head of the list. In the traversal, visiting a node is simply printing its `id` field to `cout`.

```
void traverse(ListNode* h) {
```

```
}
```

(b) (4 marks). Write a non-member function `delete (ListNode*& h, ListNode* p)` that deletes the node after the one pointed to by `p` from the circular list pointed to by `h`. The function is invoked as `delete (head, ptr)`, where `ptr` is guaranteed to point to one of the nodes on the list.

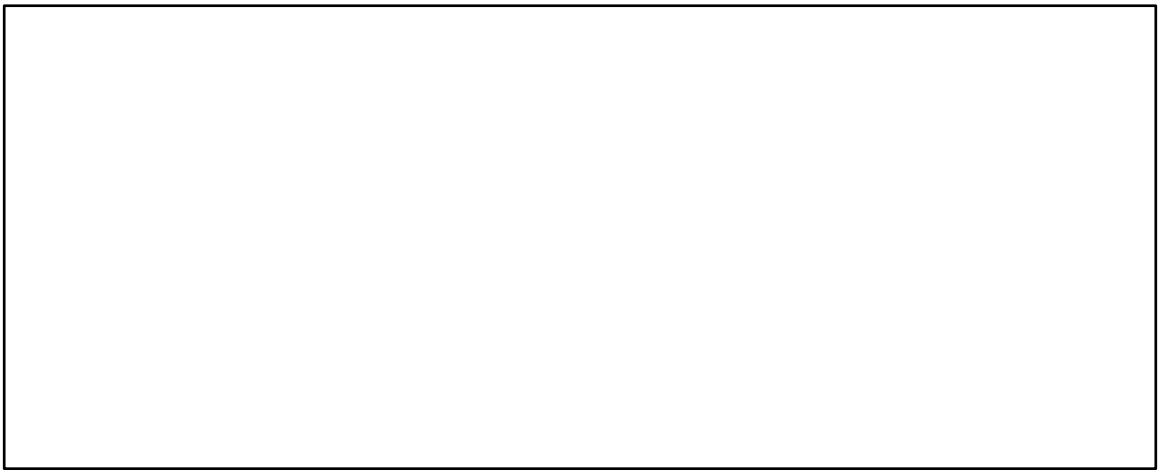
```
void delete(ListNode*& h, ListNode* p) {
```



```
}
```

(c) (3 marks). It is sometimes not known if the linked list pointed to by `head` is circular or just a regular linked list with the next field in the last node set to `NULL`. Write a non-member function `isCircular(ListNode* h)` that returns true if the list is circular and false otherwise. The function is invoked as `isCircular(head)`.

```
bool isCircular(ListNode* h) {
```



```
}
```

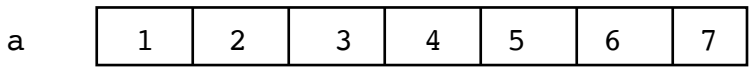
Question 7. (4 marks). *Recursion.*

Write the output produced by the program below. Write your answer in the box below.

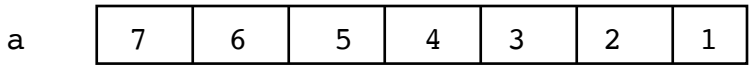
```
#include <iostream>
using namespace std;

void printSequence(int n, int m, bool flag) {
    cout << m << " ";
    if (flag == false && n == m) return;
    if (flag) {
      if (m-5 > 0) printSequence (n, m-5, true);
      else printSequence (n, m-5, false);
    }
    else printSequence (n, m+5, false);
}

int main() {
   printSequence(16, 16, true);
   cout << endl;
   return (0);
}
```

Question 8. (3 marks). *Recursion.*

Write a ***recursive*** function called `reverseArray` that reverses an `n`-element array *in-place*, i.e., without using an additional array. For example, given the array `a` below:



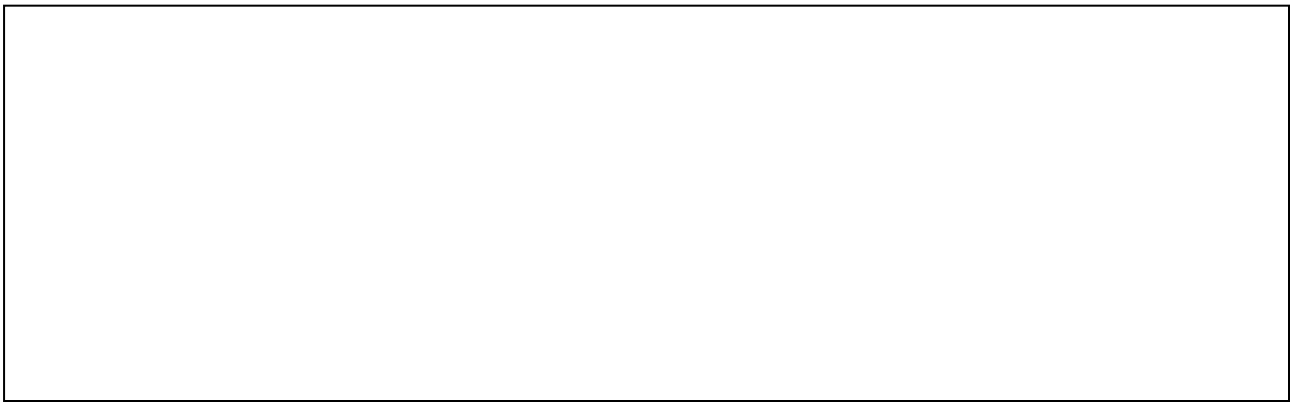
the function changes the array to:



That is, the `reverseArray` function swaps elements `a[i]` with element `a[n-i-1]`.

Write your answer below. <u>You are **not** allowed to use any loops in your solution</u>.

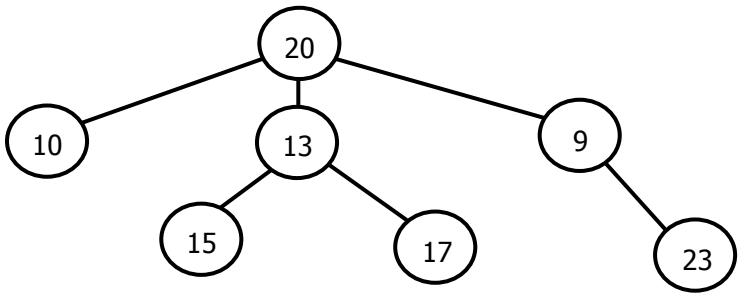
```
void reverseArray ( int* array, int left, int right ) {
```



```
}

// Here is how the function is called for an n-element a
reverseArray(a, 0, n-1);
```

Question 9. (4 marks). *Trees.*

A *k-ary tree* (also known as a *k-way tree*) is a structure that is either empty or has one node that is connected to *k* disjoint structures, each of which is a *k-ary* tree. Thus, a binary tree is a special case of a *k-ary* tree for which $k = 2$. An example of a 3-ary tree is shown below.



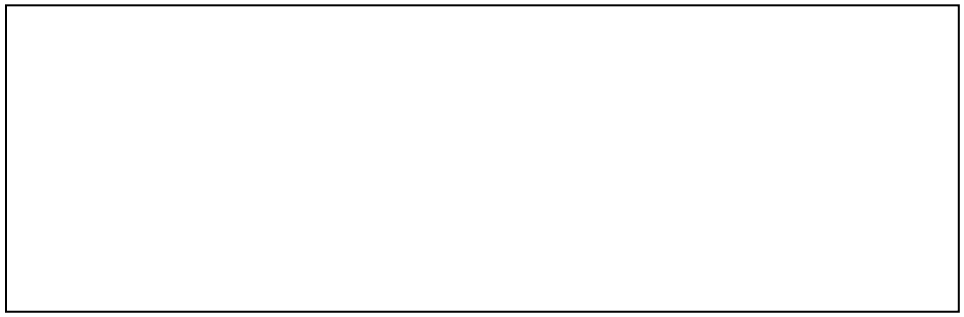
The class below gives one possible representation of a *k-ary* tree, where *k* is no more than 5. The array `child` is an array of pointers, where element *i* of the array points to sub-tree *i* of a node. The root of the tree is also declared.

```
#define K 5
class kAryTree {
   public:
      int key;
      kAryTree* child[K];
};

kAryTree* root;
```

Write a recursive non-member function `preorder` that performs the preorder traversal of a *k-ary* tree. The function should print the key of a node, then print the keys of its children starting with the left-most child (i.e., the one pointed to by `child[0]`) to the right-most child. As an example, the preorder traversal of the 3-ary tree shown above is: `20 10 13 15 17 9 23`.

Your code will be also marked for simplicity.

```
void preorder (kAryTree* myroot) {
```



```
}
// This is how preorder is called
preorder(root);
```

Question 10. (4 marks). *Tree Traversals.*

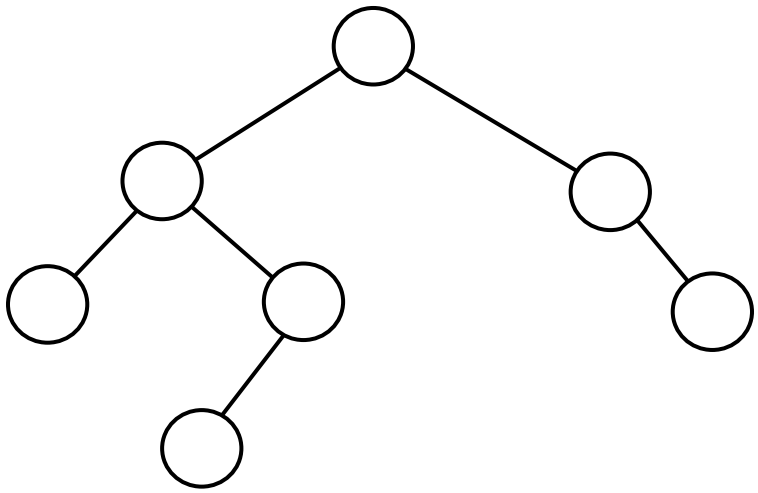
A binary tree T has the following traversals:

Pre-order traversal: 10 16 17 6 14 12 18
In-order traversal: 17 16 10 14 12 6 18
Post-order traversal: 17 16 12 14 18 6 10

Draw the binary tree T below. Keep in mind that there is only a single binary tree that has the three traversals shown above. You are to draw <u>one tree</u>.

Question 11. (3 marks). *Binary Search Trees.*

A binary search tree (BST) T has 7 nodes. The nodes have the following keys: 6, 9, 22, 14, 8, 1 and 13. The tree is shown below.



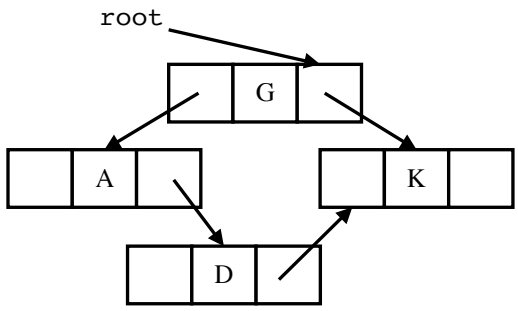
Label each node in the tree above with its key (by writing the key inside the node), keeping in mind that the tree is a binary search tree.

Question 12. (7 marks). *Binary Trees.*

Consider the class `TreeNode` defined below. The root of the tree is pointed to by `root`.

```
class TreeNode {
   public:
     int key;
     TreeNode* left;
     TreeNode* right;
};

TreeNode* root;
```

A mistake that occurs when coding binary trees is to forget to make the left or right pointers NULL in leaf nodes in the tree. Should one of these pointers end up pointing to another node in the tree, a cycle is effectively created. This is shown in the example below.



You wish to write a non-member function `bool cycle(TreeNode* root)` that returns true if the tree has cycles, otherwise returns false.

Your solution requires an additional private data member in `TreeNode`. Show the declaration of this variable and indicate how it should be initialized in the constructors.

(a) (1 mark). Declare the additional private member you wish to add to `TreeNode` here:

Answer:

(b) (1 mark). Show how these members should be initialized in the constructors of `TreeNode`:

Answer:

(c) (5 marks). Write the body of your function below.

```
bool cycle(TreeNode* root) {
    
}
```

Question 13. (8 marks). *Binary Search Trees. [Caution: Difficult]*

Consider the class `TreeNode` defined below. It represents a node in a binary search tree. The root of the tree is pointed to by `root`.

```
class TreeNode {
   public:
     int key;
     TreeNode* left;
     TreeNode* right;
};

TreeNode* root;
```

Write a ***recursive*** non-member function `secondSmallest` that prints to `cout` the second smallest `key` value in a given binary search tree. Your code must not use any variables other than the function parameters and the members of `TreeNode` objects. If you use any other variables, you will receive a mark of 0.

The function takes two parameters. The first is a pointer to the root of a tree. The second is an integer variable, which you may use as you please. Your code will be marked for simplicity and efficiency.

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

void secondSmallest(TreeNode* myroot, int& c) {
```

```
}

// The function is called as follows, where root points
// to the root of the tree
int c = 0;
secondSmallest(root, c);
```

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

Consider the following class definitions and implementations of classes, `Shape` and `Circle`. They are followed by a `main` function that utilizes these classes.

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

class Shape {
   protected:
    int shapeID;

   public:
     int getID() {return shapeID;}
     void setID(int k) {shapeID = k;}
     virtual void draw()=0;
     virtual void print()=0;
};

class Circle : public Shape {
   protected:
     float radius;

   public:
      float getRadius() {return radius;}
      void setRadius(float r) {radius = r;}
      virtual void draw() {
          // code to draw
      }
};

class Rectangle : public Shape {
   protected:
     float length;
     float width;

   public:
      float getLength() {return length;}
      void setLength(float x) {length = x;}
      float getWidth() {return width;}
      void setWidth(float x) {width = x;}
      virtual void draw(){
          // code to draw
      }
      virtual void print(){
          // Code to print
      }
};

int main() {
   // Statements in the main function
   return 0;
}
```

Indicate which of the following statements <u>that appear in the `main` function</u> compile with no errors or produce a compile time error. Indicate your answer in the table below.

Statement	Answer
`Shape s;`	Compile time error? Yes No (Circle one answer) If your answer above is Yes, give the reason in one sentence:
`Circle c;`	Compile time error? Yes No (Circle one answer) If your answer above is Yes, give the reason in one sentence:
`Rectangle r;` `r.setID(9);`	Compile time error? Yes No (Circle one answer) If your answer above is Yes, give the reason in one sentence:
`Rectangle d;` `d.length=3.0;`	Compile time error? Yes No (Circle one answer) If your answer above is Yes, give the reason in one sentence:

Question 15. (2 marks). *Inheritance.*

Consider the following class definitions.

```
class BaseC {
    private:
         int a;
         void help();
    protected:
         int b;
    public:
         int c;
         int d;
         void print();
};
```

```
class DerivedC : public BaseC {
    private:
         int x;
         void nohelp();
    public:
         int w;
         int z;
         void noprint();
};
```

The following declaration appears in the `nohelp` member function of the class `DerivedC`.

```
DerivedC derived;
```

Indicate by placing an **X** in the appropriate column whether each of the following statements is ***correct*** code (i.e., compiles with no errors), or ***incorrect*** code (i.e., generates a compile time error). Assume the statements in the rows of the table also appear in the `nohelp` member function of the class `DerivedC`.

	Correct?	**Incorrect?**
`derived.a = 8;`		
`derived.b = 10;`		
`derived.x = 12;`		
`derived.w = 4;`		

Question 16. (12 marks). *Inheritance.*

Consider the following base and derived classes. They compile with no errors.

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

class CircuitElement {
   protected:
     int code;
   public:
     CircuitElement () {cout << "circuit element\n";}
     CircuitElement (int c) {code = c;
                             cout << "circuit element with code\n";}
     ~CircuitElement () {cout << "no circuit element\n";}
     float getPower() {return 0.0;}
     virtual void print() {cout << "error\n";}
};

class Resistor : public CircuitElement {
   protected:
      int resistance;
   public:
      Resistor(int r) {
           code = 1;
           resistance = r;
           cout << "resistor\n";
      }
      ~Resistor() {cout << "no resistor\n";}
      float getPower() {return 0.0;}
      void print() {
           cout << "Resistor: " << resistance << endl;
      }
};

class Capacitor : public CircuitElement {
   protected:
      int capacitance;
   public:
      Capacitor(int c):CircuitElement(2) {
           capacitance = c;
           cout << "capacitor\n";
      }
      float getPower() {return 0.0;}
      ~Capacitor() {cout << "no capacitor\n";}
      void print() {
           cout << "Capacitor: " << capacitance << endl;
      }
};


// Code continued on next page
```

```
class PowerResistor : public Resistor {
    protected:
       float power;
    public:
       PowerResistor(int r, float p):Resistor(r) {
             power = p;
             cout << "power resistor\n";
       }
       ~PowerResistor() {cout << "no power resistor\n";}
       float getPower() {return power;}
       virtual float powerResistance() {return (power*resistance);}
       void print() {
             Resistor::print();
             cout << "Power resistor: " << power << endl;
       }
};

class PowerCapacitor : public Capacitor {
    protected:
       float power;
    public:
       PowerCapacitor(int c, float p):Capacitor(c) {
             power = p;
             cout << "power capacitor\n";
       }
       ~PowerCapacitor() {cout << "no power capacitor\n";}
       float getPower() {return power;}
       virtual float powerCapacitance() {return (power*capacitance);}
       void print() {
             Capacitor::print();
             cout << "Power capacitor: " << power << endl;
       }
};
```

Consider each of the following `main` functions that use the above classes. You may assume that each `main` function includes the code of the classes above, including the `#include`'s and the `using namespace std;`.

For each `main` function, if the function compiles with no errors (ignore warnings), then write the output produced by the function. If the function compiles with errors, then write: "compile errors".

(a)

```
int main() {
    Capacitor c(150);
    PowerResistor p(180,350);
    return 0;
}
```

If `main` compiles with no errors, write the output produced by its execution? Otherwise, write: "compile errors".

(b)

```
int main() {
    CircuitElement* pr;
    pr = new PowerResistor(200, 250);
    delete pr;
    return 0;
}
```

If `main` compiles with no errors, write the output produced by its execution? Otherwise, write: "compile errors".

(c)

```
int main() {
    Capacitor* c;
    c = new CircuitElement(5);
    delete c;
    return 0;
}
```

If `main` compiles with no errors, write the output produced by its execution? Otherwise, write: "compile errors".

(d)

```
int main() {
    PowerResistor* pr;
    Resistor* r;
    r = new Resistor(450);
    pr = r;
    delete pr;
    return 0;
}
```

If `main` compiles with no errors, write the output produced by its execution? Otherwise, write: "compile errors".

(e)

```
int main() {
    PowerResistor r(250, 1000);
    CircuitElement e;
    e = r;
    return 0;
}
```

If `main` compiles with no errors, write the output produced by its execution? Otherwise, write: "compile errors".

(f)

```
int main() {
    CircuitElement* e = new PowerCapacitor(500, 1000);
    e->print();
    return 0;
}
```

If `main` compiles with no errors, write the output produced by its execution? Otherwise, write: "compile errors".

(g)

```
int main() {
    Capacitor* c = new PowerCapacitor(300, 100);
    cout << c->getPower() << endl;
    return 0;
}
```

If `main` compiles with no errors, write the output produced by its execution? Otherwise, write: "compile errors".

(h)

```
int main() {
    PowerResistor pr(20,100);
    Resistor* r = &pr;
    cout << r->powerResistance() << endl;
    return 0;
}
```

If `main` compiles with no errors, write the output produced by its execution? Otherwise, write: "compile errors".

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

Consider the graph described by the adjacency matrix below. The vertices of the graph are numbered 0 to 4.

```
0 1 1 0 0
0 0 0 1 0
0 0 0 0 0
0 0 0 0 0
0 0 0 1 0
```

(a) (2 marks). Give a *depth-first* traversal of the graph, starting at vertex 0. Assume that visiting a vertex means printing its number to the standard output.

Give the traversal here: | | | | | |

(b) (2 marks). Give a *breadth-first* traversal of the graph, starting at vertex 0. Assume that visiting a vertex means printing its number to the standard output.

Give the traversal here: | | | | | |

Question 18. (5 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`.

```
for (int i=0; i < n; ++i) {
    for (int j=0; j*j < n; ++j) {
         // Some code with O(1)
    }
}
```

T(n) = 

(b) (3 marks). The size of the input is `n`. Write the recurrence equation and then solve it to obtain an expression of the execution time.

```
int recursive(int n) {
   int x,y,w;

   if (n <= 1) return (0);
   x = recursive (n/3);
   y = recursive (n/3);
   w = recursive (n/3);
   return (x+y+w);
}
```

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

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

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

`T(n) =` 

This Page is Intentionally Left Blank – Use for Rough Work