COMP 151 – Lectures 1, 2, 3
Midterm April 4, 2001
Exam Time – 90 minutes

Name: ______________________ E-mail: ______________________
Student No: ______________________ Lecture/Lab Section: ______________

Answer all of the following questions. Do your work on the exam paper.
If necessary, you may write on the back of the sheets.

1. (4 points) Consider the following class definitions

```
class Length
{ public:
     Length(int f, int i);
     Length(int f);
     int GetLength();
private:
     int feet; int inches;
};

Length::Length(int f, int i)
{feet = f; inches = i;}

Length::Length(int f)
{feet = f; inches = 0;}

int Length::GetLength()
{ return 12*feet + inches;}
```

For each of the following declarations underline whether it is a legal or illegal declaration of an object A in the class *Length*. *Each line worth one point*

Legal	<u>Illegal</u>	Length A;
Legal	<u>Illegal</u>	Length A();
<u>Legal</u>	Illegal	Length A(1,2);
Legal	<u>Illegal</u>	Length A(1,2,3);

2. (4 points) Consider the following function implementation:

```
void Do (int *p1, int &p2)
{*p1 = *p1 - p2;
  p2 = *p1 + p2;
}
```

Suppose **int i=5, j=7;**.
What values are stored in **i** and **j** after the call **Do(&i, j)**? *1 point each.*

```
i == -2; j == 5;
```

Now suppose **int i = 5**.
What is the value stored in **i** after the call **Do(&i, i)**? *2 points*

```
i == 0;
```

3. (10 points) You are given the function definition

```
void f(const int& x,  int& y) {cout<<"x: "<<x<<" "<<"y: "<<y<<endl;}
```

and the following variable definitions/initializations.

```
int x=5;
const int y=6;
int* const I1ptr=&x;
int const* I2ptr=&x;
```

Assuming the above definitions and declarations, decide, for each of the following statements, whether it is legal or illegal, and underline the appropriate answer:
2 pts each for first and seventh lines. 1 pt for each of the others.

Legal	$\underline{\text{Illegal}}$	f(x,y);
$\underline{\text{Legal}}$	Illegal	f(y,x);
$\underline{\text{Legal}}$	Illegal	*I1ptr=6;
Legal	$\underline{\text{Illegal}}$	*I2ptr=5;
$\underline{\text{Legal}}$	Illegal	x=6;
Legal	$\underline{\text{Illegal}}$	int *ptr=&y;
Legal	$\underline{\text{Illegal}}$	I1ptr = I2ptr;
$\underline{\text{Legal}}$	Illegal	I2ptr = I1ptr;

4. (14 points) What is the output of the following program? If the program contains an error that would cause it to fail (memory leakage is not considered such a fault) (i) first print out the output up to the failure point, (ii) then point out the error (underline its exact location in the program) and (iii) finally explain why it is an error. If there is more than one error you only have to identify the first one.

```
#include <iostream.h>
main()
{int X = 1;
int *P = new int;
int *Q = new int;
int *R;

R = P;          *P = 5;          P = &X;       *Q = 6;
cout << X << " " << *P << " " << *Q << " " << *R << endl;

X=2;            *R = *P;         *P = 7;
cout << X << " " << *P << " " << *Q << " " << *R << endl;

Q=P;             P = new int;    *P = 8;        R = P;
cout << X << " " << *P << " " << *Q << " " << *R << endl;

delete P;       *Q = 4;          P = Q;
cout << X << " " << *P << " " << *Q << " " << *R << endl;
}
```

Output

```
1 1 6 5
7 7 6 2
7 8 7 8
4 4 4
```

error is in the very last line when running

```
cout << " " << *P << " " << *Q << " " << *R << endl;
```

when trying to output ***R**. *The error is a* dangling pointer. **R** *had been pointing to the same thing as* **P** *but after* **delete P**, **R** *had been left dangling ponting to nothing. Accessing* ***R** *may therefore cause a failure.*

Marking Guidlines:

- 1/2 pt per correct answer in lines 1 and 2
- 1 pt per correct answer in lines 3 and 4
- 1 pt for correctly circling the error (1/2 pt if "delete P" is circled)
- 2 pt for correct explanation of why it is an error

5. (16 points) This page contains the definition of a class **thing** and some functions. The next page contains a program that uses **thing.** Write down the output of the program.

```
#include <iostream.h>
class thing
{ public:
  thing::thing()
    {x=0;      cout << "Default constructor: x = " << x << endl;}

  thing::thing(int v)
    {x=v;      cout << "Extra constructor: x = " << x << endl;}

  thing::thing(const thing & T)
    {x = T.x;  cout << "Copy constructor: x = " << x << endl;}

  thing::~thing()
    {cout << "Destroying thing with x = " << x << endl;}

  void thing::set(int v)
   {x=v;}

  private:
    int x;
};

void Do1(thing T, int v)
{T.set(v);}

void Do2(const thing & T, int v)
{thing TT;
 TT = T;
 TT.set(v);
}

thing Do3(thing T, int v)
{T.set(v);
 return T;
}
```

```
void main()
{ thing X;     X.set(3);
  cout << "----1----" << endl;

  thing Y=X;   Y.set(1);
  cout << "----2----" << endl;

  thing Z;     Z = X;    Z.set(3);
  cout << "----3----" << endl;

  thing  W(Z); W.set(2);
  cout << "----4----" << endl;

  Do1(X, 6);
  cout << "----5----" << endl;

  Do2(Y, 5);
  cout << "----6----" << endl;

  Do3(Z,7);
  cout << "----7----" << endl;
}
```

Output

```
Default constructor: x = 0
----1----
Copy constructor: x = 3
----2----
Default constructor: x = 0
----3----
Copy constructor: x = 3
----4----
Copy constructor: x = 3
Destroying thing with x = 6
----5----
Default constructor: x = 0
Destroying thing with x = 5
----6----
Copy constructor: x = 3
Copy constructor: x = 7
Destroying thing with x = 7
Destroying thing with x = 7
----7----
Destroying thing with x = 2
Destroying thing with x = 3
Destroying thing with x = 1
Destroying thing with x = 3
```

General Marking Guidelines

- *-1 for every missing or incorrect statement*
- *-1 for extra statement added*
- *-1 when items in a section are in wrong order*
- *Regardless of the above; in each section, the maximimum number of points that can be deducted is no more than 1 + number of statements that should correctly appear in the section.*

6. (6 points) What is the output of the following program?

```
#include <iostream.h>
class H
{ public:
     H() {cout << "Constructing H\n";}
    ~H() {cout << "Destructing  H\n";}
  private:
  int x;
};

class B
{ public:
     B() {hptr = new H; cout << "Constructing B\n"; }
    ~B() {cout << "Destructing  B\n"; delete hptr;}
  private:
  H * hptr;
};

class D : public B
{ public:
     D() {cout << "Constructing D\n";}
    ~D() {cout << "Destructing  D\n";}
};

main()
{ D d;}
```

<u>Output:</u>

```
Constructing H
Constructing B
Constructing D
Destructing  D
Destructing  B
Destructing  H
```

Marking Guidelines

CB = "Constructing B", DB = Destructing B", etc..

- *-5 if there is some destructor statement before any constructor one*
- *-2 if missing DB or DC; -3 if missing both*
- *-2 if there is an extra constructor/destructor.*
- *-2 if missing CH or DH; -3 if missing both.*
- *-2 if CD before CB or DB before DD; -3 if the B and D statements are in the order CD, CB, DB, DD.*

7. (16 points) In this question you are given the following two class definitions

```
class Team{
 public:
  Team(const string& pers, int yr):founder(pers), year(yr) {};
  void print()
     {cout<<"founder:  "<<founder<<"," <<" year: "<<year<<endl;}
  void set_year(int x) {year=x; cout<<"year: "<<year<<endl;}
 private:
  string founder;
  int year;
};

class Swim:public Team {
 public:
  Swim(const string& pers, int yr, string nm):Team(pers, yr), name(nm)
     {MembersNum=0;};
  void print()
     {Team::print();
     cout<<"team name: "<<name<<"," << " members: "<<MembersNum<<endl;}
  void set_members(int y)
     {MembersNum=y; cout<<" members: "<<MembersNum<<endl;}
 private:
  string name;
  int MembersNum;
};
```

and the following declarations.

```
 Team team1("Ng", 1995);
 Swim swim1("Smith",1999,"USTSwim"), swim2("Cheng",1969,"HKUSwim");
 Team  team2 = swim2;
 Team     *a = &swim1;
 Swim     *s = &swim1;
```

For each of the following statements identify whether or not the statement is legal based on the definitions/declarations given above. If it is illegal, state so. If it is legal, write down the output generated by the statment. *Each question worth 2 points*

(a) `team1.print();` *Legal*
founder: Ng, year: 1995

(b) `team2.print();` *Legal*
founder: Cheng, year: 1969

(c) `swim1.print();` *Legal*
founder: Smith, year: 1999
team name: USTSwim, members: 0

(d) `s=a;` *Illegal*

(e) `a->print();` *Legal*
founder: Smith, year: 1999

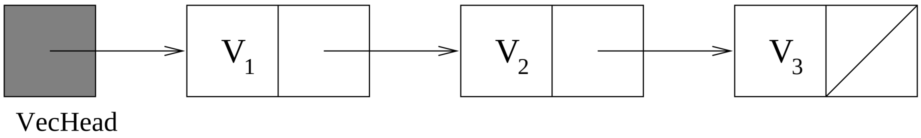
(f) `a->set_members(10);` *Illegal*

(g) `s->print();` *Legal*
founder: Smith, year: 1999
team name: USTSwim, members: 0

(h) `s->set_year(1998);` *Legal*
year: 1998

8. (30 points)

In this problem we define a new class named Vector that stores a linked list of three items v_1, v_2 and v_3 as shown in the diagram.



Vector assumes the existence of a *class* and *typedef*

```
class node
{ public:
  int       Data;
  node*     Next;
};
typedef node* ptrType;
```

A *Vector* can be thought of as a linked list of three nodes.

Two vectors $X = (x_1, x_2, x_3)$ and $Y = (y_1, y_2, y_3)$ can be added together componentwise to give $Z = X + Y = (x_1 + y_1,\ x_2 + y_2,\ x_3 + y_3)$.

In this problem you will have to implement some of the vector functions and answer questions about them.

Here is a small example program that uses $Vector$ along with its output. It is included so that you can see how $Vector$ works.

```
main()
{ Vector V;
  cout << "V = "; V.Display(); cout << endl;

  V.SetVector(1,1);  V.SetVector(2,2);   V.SetVector(3,3);
  cout << "V = "; V.Display(); cout << endl;

  Vector Z;
  Z = W + X;
  cout << "Z = "; Z.Display(); cout << endl;
}

OUTPUT
------
V = (0, 0, 0)
V = (1, 2, 3)
Z = (5, 10, 15)
```

Here is the class definition of Vector: `vi` refers to item v_i in Vector $V = (v_1, v_2, v_3)$.

```
class Vector
{ public:

  //default constructor.
  //Constructs list and sets all values = 0
  Vector();

  //Copy Constructor
  Vector(const Vector& V);

  //Destructor
  ~Vector();

  //Sets vi to be equal to Item
  void SetVector(int i, int Item);

  //Returns vi
  int Get(int i);

  //Displays the vector
  void Display();

  //Overloaded deep assignment operator
  void operator=(const Vector &V);

  //Line needed for +
  friend  Vector operator+(Vector V1, Vector V2);

  private:
      ptrType VectorHead;
}; //End Definition of Vector
```

The copy constructor and assignment operator (*operator=*) both make deep copies.

In what follows you may assume that the functions *SetVector, Display, Get* and *operator=* have all already been implemented and that you may use these functions in any code you write.

(A) (10 points) Write an implementation of the copy constructor. Remember that it should make a deep copy.

There are many possibilities. All of them need to dynamically allocate memory for the calling-object's nodes. Here is one possibility

```
Vector::Vector(const Vector &V)
{ptrtype Node2, Node3;
 VectorHead = new node; VectorHead->Data = V.Get(1);
 Node2      = new node;      Node2->Data = V.Get(2);
 Node3      = new node;      Node3->Data = V.Get(3);

 Vectorhead->Next = Node2;
      Node2->Next = Node3;
      Node3->Next = NULL;
}
```

(B) (9 points) Write the implementation of **Vector operator+(Vector V1, Vector V2);** *There are many possibilities. Here are two*

```
Vector operator+ (Vector V1, Vector V2)
{Vector Temp;
 for(int i=1; i <=3; i++)
   Temp.SetVector(i,V1.Get(i) + V2.Get(i));
 return Temp;
}

Vector operator+ (Vector V1, Vector V2)
{ for(int i=1; i <=3; i++)
   V1.SetVector(i, V1.Get(i) + V2.Get(i));
 return V1;
}
```

(C) (11 points) This question has two parts. In the first part you should write the output of the following program underneath the line Output:

```
main()
{ Vector V;
  V.SetVector(1,1);
  V.SetVector(2,1);
  V.SetVector(3,1);

  Vector X=V;
  Vector Y;
  Y=V;

  X.SetVector(1,2);
  Y.SetVector(1,3);

  cout << "V = "; V.Display(); cout << endl;
  cout << "X = "; X.Display(); cout << endl;
  cout << "Y = "; Y.Display(); cout << endl;
 }
Output
------
V = (1,1,1)
X = (2,1,1)
Y = (3,1,1)
```

In the second part of this section you should imagine the scenario fulfilling the following two assumptions:

(a) we have **not written** our own copy constructor and have used the system provided default one instead.

(b) we still write our own assignment operator (*operator=*) though, and this assignment operator constructs a deep copy.

Given assumptions (a) and (b) what is the output of the program in part (C) now? Explain how you derived your answer.

```
Output
------
V = (2,1,1)
X = (2,1,1)
Y = (3,1,1)
```

Since we have not written our own copy constructor the statement **Vector X=V** *will use the system copy constructor and make X a shallow copy of V. That is,* **X.VectorHead** *will point to the same linked list as* **V.VectorHead**. *Thus,* **X** *and* **V** *both share the same list; any change made to* **X** *is made to* **V** *and vice-versa.*

Since **Y=V** *calls our deep assignment operator* **Y** *will be constructed as a (deep) copy of* **V** *and will have its own list. Thus changes made to* **Y** *will not be made to* **V** *and vice-versa.*