HW #4 (Polygon & Polynomial)

Part A

- A *Polygon* is characterized by m points (or sides). Each of the points is n-dimensional. For example, for point i, its coordinates is given by $(x_{i,\,0}, x_{i,1}, \ldots, x_{i,\,n-1})$.
- You are to implement, using C++, or Java if you prefer, a class for *Polygon* using an internal 2D array *point*. *point[i]* is the ith point of the polygon ($0 \leqq i \leqq m$), which stores all the n coordinates. In other words, *_point[i][j]* is the jth coordinate of the ith point, where $0 \leqq i \leqq m$-1 and $0 \leqq j \leqq n$-1.
- You are given the polygon class defined as below:

HW #4 (2)

```
class Polygon {
public:

    // default constructor
    // m is the number of points
    // n is the dimension of the points
    Polygon (unsigned int m = 0, unsigned int n = 0) {
        cout << “default constructor” << endl;
        // …
}
```

HW #4 (3)

```
Polygon (const Polygon& mt); // copy constructor
~Polygon (); // destructor
double* FindCentroid() const; // return the centroid of the
                                                   // polygon

const int getNumOfPoints() const { return _m; }
const int getDimension() const { return _n; }
```

HW #4 (4)

```
// Accessor: Get the value stored at the m-th point and n-th
// dimension
const double getValue (unsigned int m, unsigned int n)
    const  {
    if (m<0 || m>=_m || n<0 || n>=_n) {
          cerr << “ERROR: Index our of range” << endl;
          exit(-1);
    }
    else return _point[m][n];
}
```

HW #4 (5)

```
// Mutator: Set the value at the m-th point and n-th
// coordinate
bool setValue (double & value, unsigned int m, unsiged int
   n) {
   if (m>=0 && m<_m && n>=0 && n<_n) {
        _point[m][n] = value;
        return true;
   }
   else
        return false;
}
```

HW #4 (6)

```
private:
    unsigned int _m; // the number of points
    unsigned int _n; // the dimension of every point
    double** _point; // point[i]: the i-th point
                     // point[i][j]: the j-th coordinate of i-point
};
```

HW #4 (7)

A1:

- Implement the copy constructor. Note that the input parameter mt may be a polygon which has no points (i.e., its _point is NULL and _m = _n = 0.)

```
// copy constructor
Polygon::Polygon (const Polygon& mt) {
    cout << "copy constructor" << endl;
    // implement your code below
```

HW #4 (8)

A2:

- Implement the destructor.

```
// destructor
Polygon::~Polygon() {
    cout << “destructor” << endl;
    // implement your code below
```

HW #4 (9)

A3:

- Implement the member function FindCentroid(), which returns the centroid of the polygon. Given a polygon of m points labeled as x_0, x_1, x_2, …, x_{m-1} and the coordinates of point i given by ($x_{i,0}$, $x_{i,1}$, …, $x_{i,n-1}$), then the coordinate of the centroid is given by (c_0, c_1, …, c_{n-1}) where

  $$c_j = \frac{1}{m}\sum_{i=0}^{m-1} x_{i,j},$$

  for $0 \leqq j \leqq n-1$.
- FindCentroid returns a pointer to an array which holds the centroid of the polygon.

  // Returns the centroid of the polygon as an array

  double* Polygon::FindCentroid() const {

  // implement your code below

HW #4 (10)

A4:

- Show the output of the following function:

```
void PrintPoints (Polygon p) {
    for (unsigned int i=0; i<p.getNumOfPoints(); i++) {
        for (unsigned int j=0; j<p.getDimension(); j++)
                coud << p.getValue(i, j) << ‘ ‘;
        cout << endl;
    }
}
```

HW #4 (11)

```
int main() {
    Polygon p(3, 4);
    double k = 0;
    for (int i=0; i<p.getNumOfPoints(); i++)
        for (int j=0; j<p.getDimension(); j++) {
            k += 1;
            p.setValue(k, i, j);
}
PrintPoints(p);
double* centroid = p.FindCentroid();
```

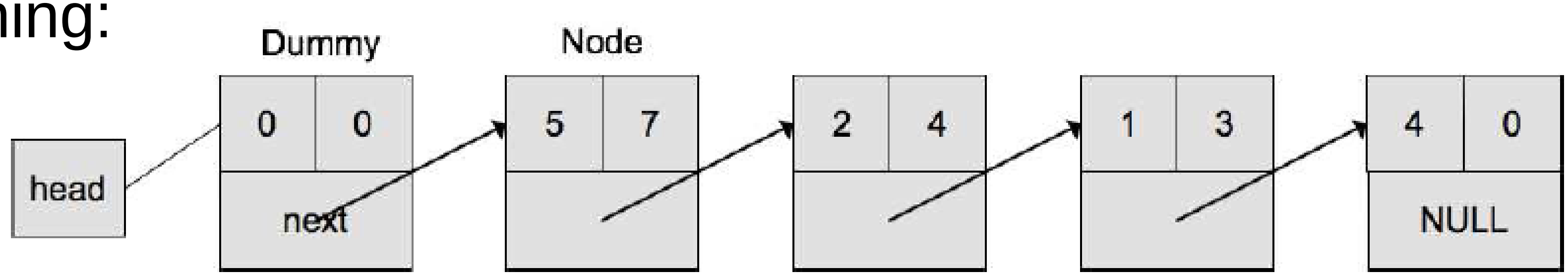
HW #4 (12)

```
    for (int n=0; n<p.getDimension(); n++)
        cout << “centroid[n] << “ “;
    cout << endl;
    return 0;
}
```

HW #4 (13)

Part B

- A polynomial *f(x)* of degree *n* is written as
  $f(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_i x^i + \ldots + c_1 x^1 + c_0 x^0$, (1)
  where *i* is called the exponent of x^i, c_i is the coefficient of x^i, and $c_i x^i$ is called the *i*th term, $0 \leqq i \leqq n$.
- In this problem, you will implement a polynomial ADT using a linked list with a dummy header (for sentinel) node. For example, we represent the polynomial $f(x) = 5x^7 + 2x^4 + 1x^3 + 4x^0$ by the following linked list, where the first node is a dummy whose term field has no meaning:



HW #4 (14)

- In the implementation, each term is stored as a Term object consisting of the coefficient and exponent. Each node in the linked list consists of a Term object and a next pointer pointing to the next node. The definition of all the classes are shown below:

```
#define NodePointer Node*
typedef int CoefType;
// term in a polynomial
class Term {
public:
    CoeffType coef;
    int expo;
};
```

HW #4 (15)

```
// node in a linked list
class Node {
private:
    Term data;
    NodePointer next;
public:
// node constructor
    Node (CoefType co = 0, int ex = 0, Node * ptr = NULL) {
        data.coef = co;
        data.expo = ex;
        next = ptr; }
    friend class Polynomial; };
```

HW #4 (16)

- The polynomial ADT with some of its supporting functions is given below:

```
// polynomial ADT
class Polynomial {
private:
    NodePointer head; // pointing to the first dummy node
public:
    // constructor
    Polynomial (CoefType* c = NULL, int* e = NULL, int num
    = 0);
```

HW #4 (17)

```
// destructor
~Polynomial();
// add a term into the polynomial
void add (CoefType c, int e);
// differentiation
void differentiate();
};
```

HW #4 (18)

B1:

- Implement the *add* member function in the polynomial class, which adds a term with coefficient *c* and exponent *e* to a polynomial object. Note that the term should be added into the polynomial object in such as way that the exponents are in *decreasing* order. If the exponent has already existed, their coefficients will be *added*.

```
// adding a term to the polynomial with decreasing
// exponents
// c: the coefficient
// e: the exponent
void Polynomial::add(CoefType c, int e) {
```

HW #4 (19)

B2:

- Given the above, implement the constructor, which takes in an array of coefficients and their corresponding exponents to form a polynomial with decreasing exponents. Note that the exponents that *e* points to may not be sorted and may be repeated. If the exponents are repeated, their coefficients should be added together.

```
// constructor
// c: pointer to an array of coefficients
// e: pointer to the corresponding exponents
// num: the number of elements in c (or e); num >= 0
Polynomial::Polynomial (CoefType* c, int* e, int num) {
```

HW #4 (20)

B3:

- Implement the destructor:

// destructor

Polynomial::~Polynomial() {

B4:

- Implement the differentiation (or derivative) function which self-differentiates the polynomial. The derivative of $f(x)$ given by Equation (1) is defined as

  $f'(x) = nc_n x^{n-1} + (n-1)c_{n-1}x^{n-2} + \ldots + 2c_2 x + c_1 x^0$.

  void Polynomial::differentiate() {