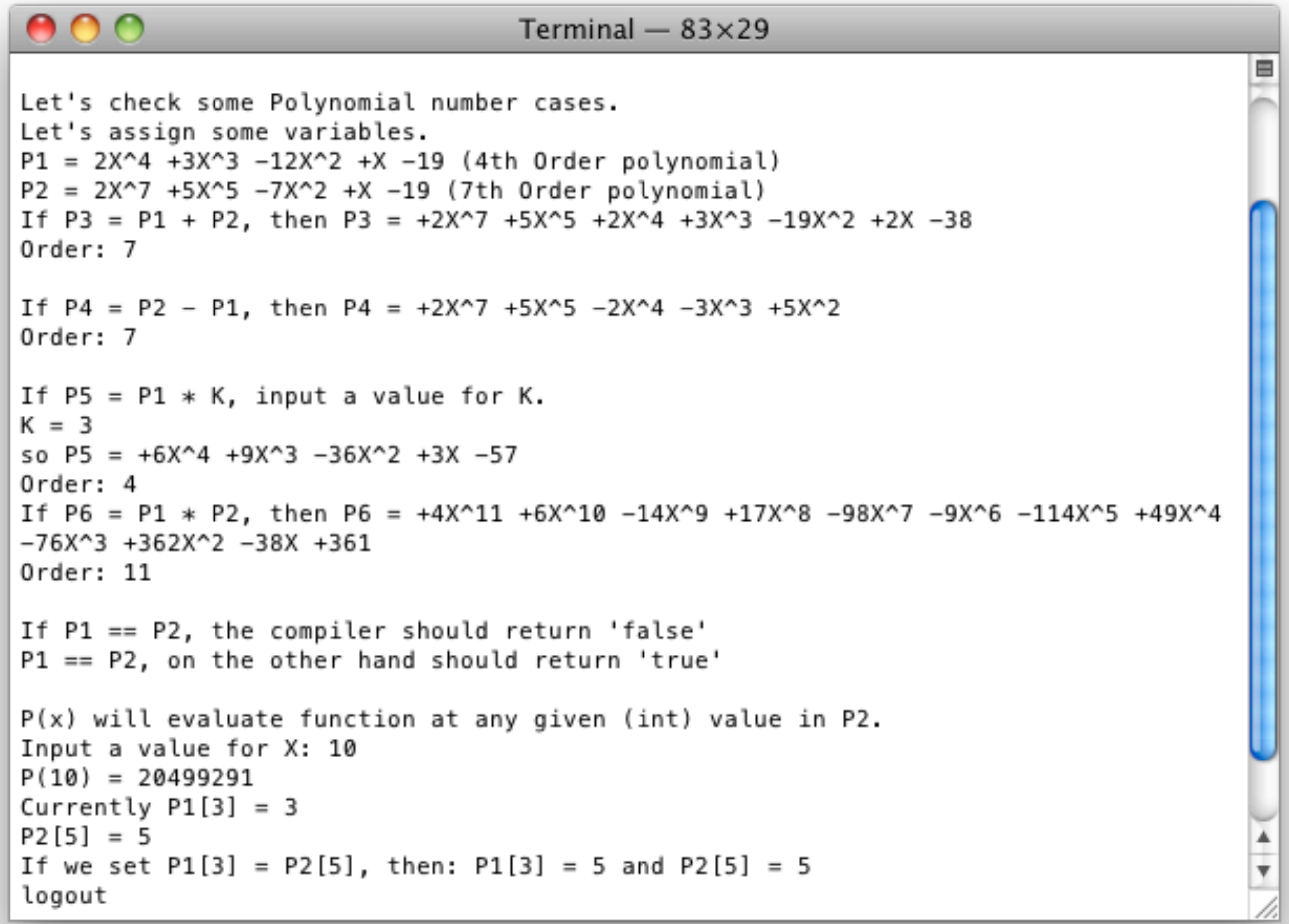
```cpp
/*
 Program: Poly
 Author: Michael Campbell
 Date: 5/11/2011
 Synopsis: Final Q#1
 */

#include <iostream>
#include "Poly.h"

int main ()
{

    Poly P3, P4, P5, P6;
    int nums[] = {-19, 1, -12, 3, 2};
    int nums2[] = {-19, 1, -7, 0, 0, 5, 0, 2};
    int K, X;
    bool isEqual, isEqual1;
    Poly P1(4, nums), P2;
    P2.set(nums2, 7);

    cout << "\nLet's check some Polynomial number cases.\n"
    << "Let's assign some variables.\n"
    << "P1 = 2X^4 +3X^3 -12X^2 +X -19 (4th Order polynomial)\n"
    << "P2 = 2X^7 +5X^5 -7X^2 +X -19 (7th Order polynomial)" << endl;

    cout << "If P3 = P1 + P2, then P3 = ";
    P3 = P1 + P2;
    cout << P3 << endl;

    cout << "If P4 = P2 - P1, then P4 = ";
    P4 = P2 - P1;
    cout << P4 << endl;

    cout << "If P5 = P1 * K, input a value for K.\n"
    << "K = ";
    cin >> K;
    cout << "so P5 = ";
    P5 = P1 * K;
    cout << P5;

    cout << "If P6 = P1 * P2, then P6 = ";
    P6 = P1 * P2;
    cout << P6 << endl;

    isEqual = P1 == P2;
    isEqual1 = P1 == P1;
    cout << "If P1 == P2, the compiler should return \'" << boolalpha <<
        isEqual << "\'" << endl;
    cout << "P1 == P2, on the other hand should return \'" << isEqual1 << "\'"
        << endl;

    cout << "\nP(x) will evaluate function at any given (int) value in P2.\n"
        << "Input a value for X: ";
    cin >> X;
    cout << "P(" << X << ") = " << P2(X) << endl;
    cout << "Currently P1[3] = " << P1[3] << endl;
    cout << "P2[5] = " << P2[5] << endl;
cout << "If we set P1[3] = P2[5], then: ";
    P1[3] = P2[5];
    cout << "P1[3] = " << P1[3] << " and "
        << "P2[5] = " << P2[5] << endl;
    return EXIT_SUCCESS;
}
```

```
//Poly.cpp
#include <iomanip>
#include "Poly.h"

Poly::Poly():order(1){              //default constructor - order = 1 & coeff[0
    ] = 1
    coeff = new int[order + 1];
    coeff[0] = 1;
}
Poly::Poly(int Order, int Default):order(Order){    //creates Nth order poly
    coeff = new int[order + 1];
    for (int i = 0; i < order + 1; i++) {
        coeff[i] = Default;
    }
}
                                                    //and inits all coeffs
Poly::Poly(const Poly& PolyNew){            //copy constructor
    order = PolyNew.order;
    coeff = new int[order + 1];
    for (int i = 0; i < order + 1; i++) {
        coeff[i] = PolyNew.coeff[i];
    }
}

//mutators & accessors

void Poly::set(){
    int UserOrder;
    cout << "Please enter the order of the polynomial" << endl;
    cin >> UserOrder;
    order = UserOrder;
    coeff = new int[UserOrder + 1];
    cout << "Please enter the coefficients in Ascending Order" << endl;
    for (int i = 0; i < UserOrder + 1; i++) {
        cin >> coeff[i];
    }

}
void Poly::set(int Coeff[], int size){
    order = size;
    coeff = new int[order + 1];
    for (int i = 0; i < size + 1; i++) {
        coeff[i] = Coeff[i];
    }
}
int Poly::getOrder(){                               //get order of polynomial
    return order;
}
int * Poly::get(){                          //returns pointer to coeff array
    return coeff;
}

//overloaded operators
Poly Poly::operator +(const Poly &rhs){          //add two polynomials
    int BigOrder = 0, SmallerOrder = 0;
    int * holder;
    if (order == rhs.order) {
        SmallerOrder = order;
        BigOrder = order;
    }else if (order < rhs.order){
        SmallerOrder = order;
        BigOrder = rhs.order;
    } else{
        SmallerOrder = rhs.order;
        BigOrder = order;
    }
    holder = new int[BigOrder + 1];

    for (int i = 0; i < SmallerOrder + 1; i++) {
        holder[i] = coeff[i] + rhs.coeff[i];
    }
    if (BigOrder == order) {
        for (int i = SmallerOrder + 1; i < BigOrder + 1; i++) {
            holder[i] = coeff[i];
        }
    }else
        for (int i = SmallerOrder + 1; i < BigOrder + 1; i++) {
            holder[i] = rhs.coeff[i];
        }
    return Poly(BigOrder, holder);
}
Poly Poly::operator -(const Poly &rhs){           //subt two polynomials
    int BigOrder = 0, SmallerOrder = 0;
    int * holder;

    if (order == rhs.order) {
        SmallerOrder = order;
        BigOrder = order;
    }else if (order < rhs.order){
        SmallerOrder = order;
        BigOrder = rhs.order;
    } else{
        SmallerOrder = rhs.order;
        BigOrder = order;
    }
    holder = new int[BigOrder + 1];

    for (int i = 0; i < SmallerOrder + 1; i++) {
        holder[i] = coeff[i] - rhs.coeff[i];
    }
    if (BigOrder == order) {
        for (int i = SmallerOrder + 1; i < BigOrder + 1; i++) {
            holder[i] = coeff[i];
        }
    }else
        for (int i = SmallerOrder + 1; i < BigOrder + 1; i++) {
            holder[i] = rhs.coeff[i];
        }
    return Poly(BigOrder, holder);
}
Poly Poly::operator *(const int scale){     //scale a polynomial
    int * holder;
    holder = new int[order + 1];
    for (int i = 0; i < order + 1; i++) {
        holder[i] = coeff[i] * scale;
    }

    return Poly(order, holder);
}
Poly Poly::operator *(const Poly &rhs){     //mult two polynomials
    int NewOrder = order + rhs.order;
    int * holder;

    //initialize to 0
    holder = new int[NewOrder + 2];
    for (int i = 0; i < NewOrder + 2; i++) {
        holder[i] = 0;
    }

    for (int i = 0; i < order + 1; i++) {
        for (int j = 0; j < rhs.order + 1; j++) {
            holder[i + j] += (coeff[i] * rhs.coeff[j]);
        }
    }

    return Poly(NewOrder, holder);
}
bool Poly::operator ==(const Poly &rhs){    //equality operator
    if ((order == rhs.order) && (coeff == rhs.coeff)) {
        return true;
    }else
        return false;
}
Poly& Poly::operator =(const Poly& rhs){    //assignment operator
    Poly *temp;

    order = rhs.order;
    coeff = new int[order + 1];
    for (int i = 0; i < order + 1; i++) {
        coeff[i] = rhs.coeff[i];
    }
    temp = new Poly(order, coeff);
    return *temp;
}
const int & Poly::operator[](int I)const{   // return the Ith coefficient
    return coeff[I];
}
int & Poly::operator[](int I){          //return the Ith coefficient
    return coeff[I];
}
int Poly::operator()(int X){            //evaluate P(x) according
    int result = coeff[order];

    for (int i = order; i >0; --i) {
        result = result * X + coeff[i-1];
    }

//    For example, the fourth-degree polynomial
//
```

```
//    P(x) = a4 * x^4 + a3 * x^3 + a2 * x^2 + a1 *x + a0
//
//    can be written in the 'nested multiplication' form
//
//    P(x) = ( ( ( ( a4 * x + a3 ) * x ) + a2 ) * x + a1 ) * x + a0
    return result;
}

ostream& operator<<(ostream& Out, const Poly& rhs){
    for (int i = rhs.order; i >= 0; i--) {
        if (rhs.coeff[i]==0) {          //if coeff[i] = 0, no point in
            including the 0X
            continue;
        }else
            if (rhs.coeff[i] >= 0) {    //add addition sign
                cout << "+";
            }

            if (i == 0) {
                cout << rhs.coeff[i] << " ";
            }else if (rhs.coeff[i] == 1 && i == 1){ //no 1X, displays X
                cout << "X"<< " ";
            }else if (rhs.coeff[i] == -1 && i == 1){ //no -1x, displays -X
                cout << "-X"<< " ";
            }else if (i == 1){
                cout << rhs.coeff[i] << "X"<< " ";
            }else
                cout << rhs.coeff[i] << "X^" << i << " ";
    }
    cout << "\nOrder: " << rhs.order << endl;
    return Out;
}
istream& operator>>(istream& In, Poly& rhs){
    int UserOrder, *coeffs;

    cout << "Please enter the order of the polynomial" << endl;
    In >> UserOrder;

    coeffs = new int[UserOrder + 1];
    cout << "Please enter the coefficients in Ascending Order" << endl;
    for (int i = 0; i < UserOrder + 1; i++) {
        In >> coeffs[i];
    }
    rhs.order = UserOrder;
    rhs.coeff = coeffs;

    return In;
}
```

```
//Poly.h
#ifndef POLY_H
#define POLY_H

#include <iostream>
using namespace std;

class Poly {

    int order;          //order of the polynomial
    int *coeff;         //point to array of coeff on the heap
                        //size of coeff array predicated on (order + 1)
public:
    Poly();                             //default constructor - order = 1 &
        coeff[0] = 1
    Poly(int Order, int Default = 0);   //creates Nth order poly
                                        //and inits all coeffs
    Poly(int Order, int *Coeff):order(Order), coeff(Coeff){}       //creates an
        Nth                 polynomial & inits
    ~Poly(){}           //destructor
    Poly(const Poly& PolyNew);          //copy constructor

    //mutators & accessors
    void set();
    void set(int Coeff[], int size);
    int getOrder();                     //get order of polynomial
    int * get();                        //returns pointer to coeff array

    //overloaded operators
    Poly operator +(const Poly &rhs);     //add two polynomials
    Poly operator -(const Poly &rhs);     //subt two polynomials
    Poly operator *(const int scale);            //scale a polynomial
    Poly operator *(const Poly &rhs);     //mult two polynomials
    bool operator ==(const Poly &rhs);           //equality operator
    Poly& operator =(const Poly& rhs);           //assignment operator

    const int & operator[](int I)const; // return the Ith coefficient
    int & operator[](int I);            //return the Ith coefficient

    int operator()(int X);              //evaluate P(x) according

    friend ostream& operator<<(ostream& Out, const Poly& rhs);
    friend istream& operator>>(istream& In, Poly& rhs);
};

#endif
```


Terminal — 83×29

```
Let's check some Polynomial number cases.
Let's assign some variables.
P1 = 2X^4 +3X^3 -12X^2 +X -19 (4th Order polynomial)
P2 = 2X^7 +5X^5 -7X^2 +X -19 (7th Order polynomial)
If P3 = P1 + P2, then P3 = +2X^7 +5X^5 +2X^4 +3X^3 -19X^2 +2X -38
Order: 7

If P4 = P2 - P1, then P4 = +2X^7 +5X^5 -2X^4 -3X^3 +5X^2
Order: 7

If P5 = P1 * K, input a value for K.
K = 3
so P5 = +6X^4 +9X^3 -36X^2 +3X -57
Order: 4
If P6 = P1 * P2, then P6 = +4X^11 +6X^10 -14X^9 +17X^8 -98X^7 -9X^6 -114X^5 +49X^4
-76X^3 +362X^2 -38X +361
Order: 11

If P1 == P2, the compiler should return 'false'
P1 == P2, on the other hand should return 'true'

P(x) will evaluate function at any given (int) value in P2.
Input a value for X: 10
P(10) = 20499291
Currently P1[3] = 3
P2[5] = 5
If we set P1[3] = P2[5], then: P1[3] = 5 and P2[5] = 5
logout
```