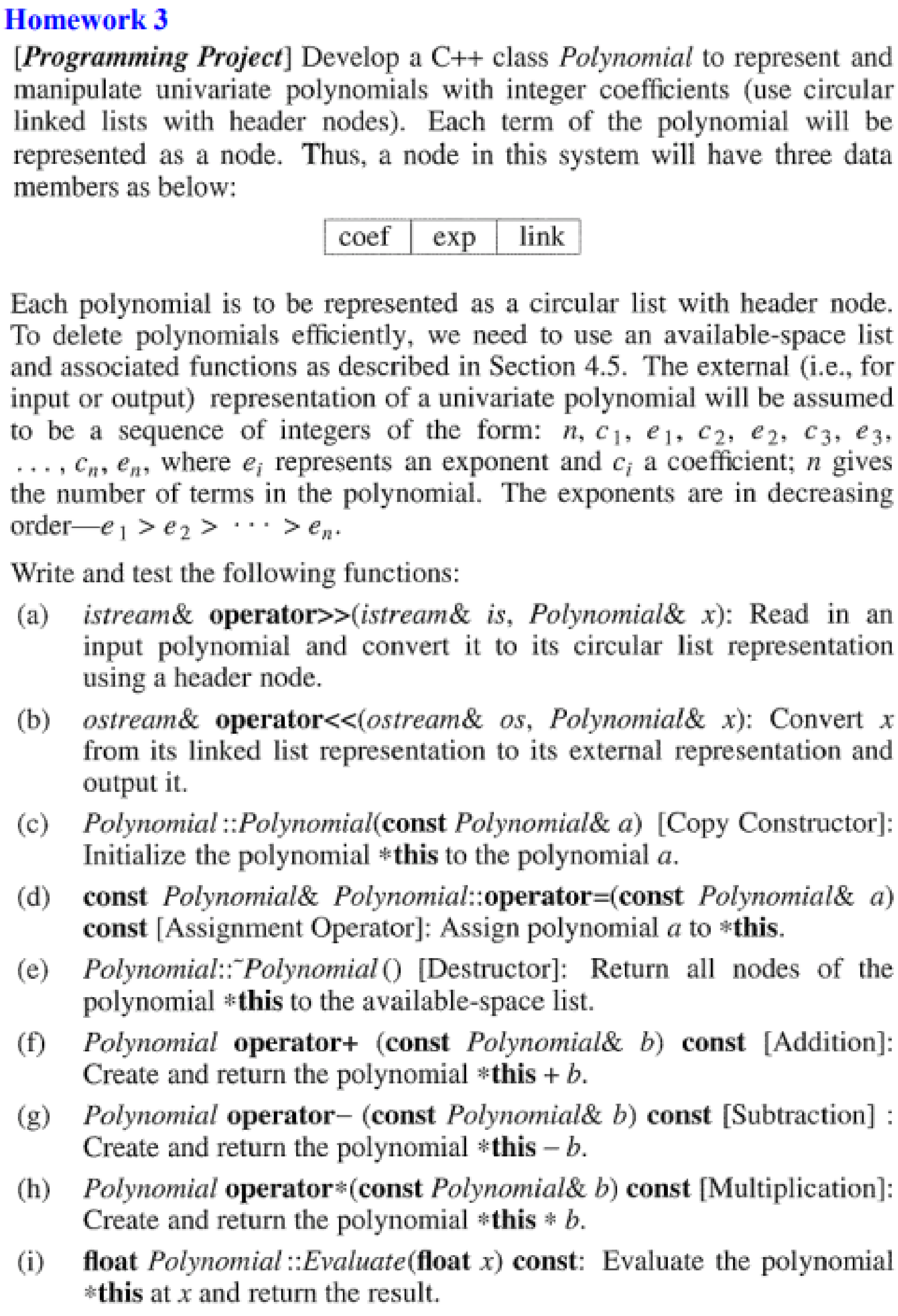
HW3



Homework 3

[***Programming Project***] Develop a C++ class *Polynomial* to represent and manipulate univariate polynomials with integer coefficients (use circular linked lists with header nodes). Each term of the polynomial will be represented as a node. Thus, a node in this system will have three data members as below:

coef	exp	link

Each polynomial is to be represented as a circular list with header node. To delete polynomials efficiently, we need to use an available-space list and associated functions as described in Section 4.5. The external (i.e., for input or output) representation of a univariate polynomial will be assumed to be a sequence of integers of the form: n, c_1, e_1, c_2, e_2, c_3, e_3, $\ldots$, c_n, e_n, where e_i represents an exponent and c_i a coefficient; n gives the number of terms in the polynomial. The exponents are in decreasing order—$e_1 > e_2 > \cdots > e_n$.

Write and test the following functions:

(a) *istream*& **operator>>**(*istream*& *is*, *Polynomial*& *x*): Read in an input polynomial and convert it to its circular list representation using a header node.

(b) *ostream*& **operator<<**(*ostream*& *os*, *Polynomial*& *x*): Convert *x* from its linked list representation to its external representation and output it.

(c) *Polynomial*::*Polynomial*(**const** *Polynomial*& *a*) [Copy Constructor]: Initialize the polynomial ***this** to the polynomial *a*.

(d) **const** *Polynomial*& *Polynomial*::**operator=**(**const** *Polynomial*& *a*) **const** [Assignment Operator]: Assign polynomial *a* to ***this**.

(e) *Polynomial*::~*Polynomial*() [Destructor]: Return all nodes of the polynomial ***this** to the available-space list.

(f) *Polynomial* **operator+** (**const** *Polynomial*& *b*) **const** [Addition]: Create and return the polynomial ***this** + *b*.

(g) *Polynomial* **operator−** (**const** *Polynomial*& *b*) **const** [Subtraction] : Create and return the polynomial ***this** − *b*.

(h) *Polynomial* **operator***(**const** *Polynomial*& *b*) **const** [Multiplication]: Create and return the polynomial ***this** * *b*.

(i) **float** *Polynomial*::*Evaluate*(**float** *x*) **const**: Evaluate the polynomial ***this** at *x* and return the result.

```
#include <iostream>
#include <math.h>
using namespace std;

struct Term{
    float coef;
    int exp;
    struct Term* next;
};

class Available{
private:
    struct Term *av;
public:
    void getBack(Term* node);
    Term* get();
    Available(){av=NULL;}
};
Available* list = new Available();
void Available::getBack(Term* node){
    Term* p=node;
    while(p->next) p=p->next;
    p->next=av;
    av=node;
}
Term* Available::get(){
    if(av!=NULL){
        Term* ret=av;
        av=av->next;
        ret->next=NULL;
        return ret;
    }
    return (struct Term*)malloc(sizeof(struct Term));
}

class Polynomial{
friend ostream& operator<<(ostream& os, Polynomial& p);
friend istream& operator>>(istream& is, Polynomial& p);
```

```
private:
    struct Term *first;
public:
    void newTerm(float c, int e);
    Polynomial(){ first=NULL; }
    Polynomial(const Polynomial& B);
    ~Polynomial();
    Polynomial& operator=(const Polynomial& a);
    Polynomial operator+(const Polynomial& a);
    Polynomial operator-(const Polynomial& a);
    Polynomial operator*(const Polynomial& a);
    float Evaluate(const float x)const;
};
Polynomial::Polynomial(const Polynomial& B){
    first=NULL;
    Term* p=B.first;
    while(p!=NULL){
        this->newTerm(p->coef, p->exp);
        p=p->next;
    }
}
Polynomial::~Polynomial(){
    list->getBack(first);
}
void Polynomial::newTerm(float c, int e){
    if(first==NULL){
        first = list->get();
        first->coef=c;
        first->exp=e;
        first->next=NULL;
    }else{
        struct Term* p=first;
        while(p->next!=NULL) p=p->next;
        p->next = list->get();
        p=p->next;
        p->coef=c;
        p->exp=e;
        p->next=NULL;
    }
}
ostream& operator<<(ostream& os, Polynomial& p){
    Term* current=p.first; bool flag=true;
    while(current!=NULL){
        if(current->coef!=0){
            flag=false;
            if(current->coef>0) os<<"+";
            else os<<"-";
            os<<abs(current->coef)<<"X^"<<current->exp;
        }
        current = current->next;
    }
    if(flag) os<<0;
    return os;
}
istream& operator>>(istream& is, Polynomial& p){
    float c, e;
    while(is>>c>>e, c||e){
        p.newTerm(c, e);
    }
    return is;
}
Polynomial& Polynomial::operator=(const Polynomial& a){
    list->getBack(first);
    first=NULL;
    Term* current = a.first;
    while(current!=NULL){
        newTerm(current->coef, current->exp);
        current = current->next;
    }
    return *this;
}
Polynomial Polynomial::operator+(const Polynomial& a){
    Term *i=this->first, *j=a.first;
    Polynomial C;
    while(i!=NULL&&j!=NULL){
        if(i->exp==j->exp){
            C.newTerm(i->coef+j->coef, i->exp);
            i=i->next; j=j->next;
        }else if(i->exp>j->exp){
            C.newTerm(i->coef, i->exp);
            i = i->next;
        }else{
            C.newTerm(j->coef, j->exp);
            j = j->next;
        }
    }
    while(i!=NULL){ C.newTerm(i->coef, i->exp);i=i->next; }
    while(j!=NULL){ C.newTerm(j->coef, j->exp);j=j->next; }
    return C;
}
Polynomial Polynomial::operator-(const Polynomial& a){
    Term *i=this->first, *j=a.first;
    Polynomial C;
    while(i!=NULL&&j!=NULL){
        if(i->exp==j->exp){
            C.newTerm(i->coef-j->coef, i->exp);
            i=i->next; j=j->next;
        }else if(i->exp>j->exp){
            C.newTerm(i->coef, i->exp);
            i = i->next;
        }else{
            C.newTerm(-1*j->coef, j->exp);
            j = j->next;
        }
    }
    while(i!=NULL){ C.newTerm(i->coef, i->exp);i=i->next; }
    while(j!=NULL){ C.newTerm(-1*j->coef, j->exp);j=j->next;
    return C;
}
Polynomial Polynomial::operator*(const Polynomial& a){
    Polynomial C;
    for(Term* i=this->first; i!=NULL; i=i->next){
        for(Term* j=a.first; j!=NULL; j=j->next){
            int e=i->exp+j->exp;bool flag=true;
            for(Term* k=C.first; k!=NULL; k=k->next){
                if(k->exp==e){
                    k->coef+=i->coef*j->coef;
                    flag=false;
                    break;
                }
            }
            if(flag) C.newTerm(i->coef*j->coef, e);
        }
    }
    return C;
}
float Polynomial::Evaluate(const float x)const{
    float ans=0; Term* current=first;
    while(current!=NULL){
        ans+=current->coef*pow(x, current->exp);
        current=current->next;
    }
    return ans;
}

int main(){
    Polynomial test;
    cin>>test;
    test.~Polynomial();
    Polynomial test2; cin>>test2;
    // cout<<test;
    // Polynomial* copyt = new Polynomial(test);
    // cout<<*copyt<<"\n";
    // Polynomial opt=test;
    // cout<<opt;
    Polynomial a; cin>>a;
    Polynomial b = test2+a;
    Polynomial b2 = test2-a;
    Polynomial b3 = test2*a;
    cout<<b<<"\n";
    cout<<b2<<"\n";
    cout<<b3<<"\n";
    cout<<b3.Evaluate(2);
    return 0;
}
```

想法說明: 將作業二改成linked-list形式

輸入亦是輸入直到輸入兩個零(coef==0&&exp==0)

```
istream& operator>>(istream& is, Polynomial& p){
    float c, e;
    while(is>>c>>e, c||e){
        p.newTerm(c, e);
    }
    return is;
}
```

運算邏輯也與作業二時相同，但效能會比較差，不過記憶體會用比較少且分散(理論上)。

加法: (減法類似)

```
Polynomial Polynomial::operator+(const Polynomial& a){
    Term *i=this->first, *j=a.first;
    Polynomial C;
    while(i!=NULL&&j!=NULL){
        if(i->exp==j->exp){
            C.newTerm(i->coef+j->coef, i->exp);
            i=i->next; j=j->next;
        }else if(i->exp>j->exp){
            C.newTerm(i->coef, i->exp);
            i = i->next;
        }else{
            C.newTerm(j->coef, j->exp);
            j = j->next;
        }
    }
    while(i!=NULL){ C.newTerm(i->coef, i->exp);i=i->next; }
    while(j!=NULL){ C.newTerm(j->coef, j->exp);j=j->next; }
    return C;
}
```

乘法: (解法和作業二時一樣，都是加新項時，先檢查有沒有加過了，再分別處理)

```
Polynomial Polynomial::operator*(const Polynomial& a){
    Polynomial C;
    for(Term* i=this->first; i!=NULL; i=i->next){
        for(Term* j=a.first; j!=NULL; j=j->next){
            int e=i->exp+j->exp;bool flag=true;
            for(Term* k=C.first; k!=NULL; k=k->next){
                if(k->exp==e){
                    k->coef+=i->coef*j->coef;
                    flag=false;
                    break;
                }
            }
            if(flag) C.newTerm(i->coef*j->coef, e);
        }
    }
    return C;
}
```

可用串列: 這次新的東西，有刪除及新增node時可節省時間。以全域變數(list)儲存。

```
class Available{
private:
    struct Term *av;
public:
    void getBack(Term* node);
    Term* get();
    Available(){av=NULL;}
};
Available* list = new Available();
void Available::getBack(Term* node){
    Term* p=node;
    while(p->next) p=p->next;
    p->next=av;
    av=node;
}
Term* Available::get(){
    if(av!=NULL){
        Term* ret=av;
        av=av->next;
        ret->next=NULL;
        return ret;
    }
    return (struct Term*)malloc(sizeof(struct Term));
}
```

polynomial:

```
class Polynomial{
friend ostream& operator<<(ostream& os, Polynomial& p);
friend istream& operator>>(istream& is, Polynomial& p);
private:
    struct Term *first;
public:
    void newTerm(float c, int e);
    Polynomial(){ first=NULL; }
    Polynomial(const Polynomial& B);
    ~Polynomial();
    Polynomial& operator=(const Polynomial& a);
    Polynomial operator+(const Polynomial& a);
    Polynomial operator-(const Polynomial& a);
    Polynomial operator*(const Polynomial& a);
    float Evaluate(const float x)const;
};
```

---

ex. 1

測試abcd項目(輸入、輸出、以其它polynomial參數建構子和operator=附值)

```
int main(){
    Polynomial test;
    cin>>test;
    Polynomial* copyt = new Polynomial(test);
    cout<<*copyt<<"\n";
    Polynomial opt=test;
    cout<<opt;
    return 0;
}
```

輸入:

2 2

1 1

0 0

輸出:

+2X^2+1X^1
+2X^2+1X^1

說明: 輸入給test, 以test建構copyt, 以等號附值給opt, 最後輸出確認

ex. 2

測試efghi項(可用串列運作、加減乘與計算)

```
int main(){
    Polynomial test;
    cin>>test;
    test.~Polynomial();
    Polynomial test2; cin>>test2;
    Polynomial a; cin>>a;
    Polynomial b = test2+a;
    Polynomial b2 = test2-a;
    Polynomial b3 = test2*a;
    cout<<b<<"\n";
    cout<<b2<<"\n";
    cout<<b3<<"\n";
    cout<<b3.Evaluate(2);
    return 0;
}
```

輸入:

2 2

1 1

0 0

2 2

1 1

0 0

2 2

1 1

0 0

輸出:

```
+4X^2+2X^1
0
+4X^4+4X^3+1X^2
100
```

說明: 第一筆輸入2x^2+1x^1給test, 而後刪掉還給可用串列，第二筆輸入給test2，第三筆給a。接著分別算出加減乘，最後算相乘後的式子帶入2為多少。

第一行—加法, 第二行—減法, 第三行—乘法, 第四行—計算第三行帶入2為多少