Instytut Informatyki i Matematyki Komputerowej UJ
Programowanie 2

Zadanie G - Wielomiany

Punktów do uzyskania: **10**

1. Ogólne warunki rozwiązania

- Celem zadania jest zaprogramowanie klasy o koniecznej nazwie `POLYNOMIAL`, będącej szczególną implementacją wielomianów.
- Współczynniki wielomianów są zawsze typu **`int`**.
- Stopień wielomianów może być dowolnie duży.
- Wielomianem pustym (zerowym) jest wielomian stopnia zerowego o współczynniku równym 0, co oznacza, że nie może być wielomianu bez żadnych współczynników.
- **Współczynniki wielomianów są pamiętane przy spełnianiu ich wzajemnego względnego pierwszeństwa implikującego przed zapamiętaniem podzielenie przez największy wspólny dzielnik wszystkich współczynników. Dla przykładu, wielomian $4 - 6x + 8x^2$ wczytany z wejścia, wykreowany konstruktorem lub uzyskany w wyniku działań zostanie zapamiętany poprzez wartości współczynników wynoszące:**

  2, –3, 4.

- Dobór pól klasy `POLYNOMIAL` jest dowolny, ale wymogiem jest istnienie pola statycznego typu **`int`** o nazwie `overloaded` o opisanym dalej znaczeniu.
- Klasa może zawierać dowolną ilość metod (w tym konstruktorów), ale z racji opisanych dalej warunków konieczna będzie implementacja jednego szczególnego konstruktora w konsekwencji oznaczająca konieczność implementacji własnego destruktora.

2. Wymagane podprogramy pozaklasowe

- Przeładowany operator wyjścia strumieniowego `<<` dla wielomianu stopnia n wypisujący $n + 1$ współczynników oddzielonych przecinkiem z następującą po nim pojedynczą spacją. Wypisane współczynniki są poprzedzone znakiem otwierającego okrągłego nawiasu z następującą spacją oraz zakończone pojedynczą spacją z następującym okrągłym zamykającym nawiasem. Dla przykładu, wielomian postaci $1 - 5x^2 + 3x^4$ będzie wypisany w postaci:

```
( 1, 0, -5, 0, 3 )
```

- Przeładowany operator wejścia strumieniowego `>>` najpierw wczytujący stopień wielomianu, a następnie odpowiednią liczbę współczynników począwszy od współczynnika zerowego. Należy założyć, że dane zawsze są poprawne (mieszczące się w typie **`int`**) oraz że ostatni podawany współczynnik jest niezerowy. Podane z wejścia przykładowe liczby

```
4   1 2 3 4 5
```

oznaczają wczytanie wielomianu:

$$1 + 2x + 3x^2 + 4x^3 + 5x^4$$

3. Wymagane metody klasy `POLYNOMIAL`

- Konstruktor o zmiennej liczbie argumentów typu **`int`**, z pierwszym argumentem typu **`int`** określającym stopień wielomianu oraz kolejnymi argumentami określającymi wartości współczynników począwszy od współczynnika zerowego. Należy założyć, że dla wielomianów stopnia niezerowego argument określający współczynnik potęgi równej stopniowi jest różny od zera. Dla przykładu konstruktor wywołany w postaci:

```
POLYNOMIAL p ( 4, 0, 1, 2, 3, 4 );
```

oznacza kreację wielomianu $x + 2x^2 + 3x^3 + 4x^4$.

- Destruktor odpowiadający za działania powyższego własnego konstruktora.
- Przeładowany operator przypisania `=`, wymagający jedynie poprawnego działania całości programu.
- Przeładowane operatory dodawania +, odejmowania (– binarny), przeciwieństwa (– unarny) i mnożenia * oznaczające tradycyjnie rozumiane operacje na wielomianach.
- Przeładowania operatorów dzielenia / oraz reszty % zwracające odpowiednio iloraz oraz resztę dzielenia wielomianów argumentów. Początkowy warunek pamiętania wielomianów o wzajemnej pierwszości współczynników zostaje utrzymany, a ponadto obowiązuje uogólniona definicja największego wspólnego podzielnika oznaczającą w przypadku wartości ułamkowych największą możliwą wartość całkowitą dającą po podzieleniu zbiór liczb względnie pierwszych. Mówiąc inaczej, w przypadku, gdy matematyczne wyniki działań dają współczynniki ułamkowe, to dla zapamiętania muszą być przemnożone przez najmniejszą wartość dającą współczynniki całkowite. Dla przykładu, przy deklaracji:

```
POLYNOMIAL p ( 3, 1, 3, 4, -1 );
POLYNOMIAL q ( 2, 6, 8, -7 );
```

kreującej wielomiany:

$$1 + 3x + 4x^2 - x^3$$
$$6 + 8x - 7x^2$$

tradycyjne dzielenie wielomianów oznacza:

$$(1+3x+4x^2-x^3)=\left(\frac{-20}{49}+\frac{x}{7}\right)\cdot(6+8x-7x^2)+\left(\frac{169}{49}+\frac{265x}{49}\right)$$

Dlatego fragment kodu:

```
cout << p / q << endl;
cout << p % q << endl;
```

powoduje wypisanie:

```
( -20, 7 )
( 169, 265 )
```

- Przeładowania operatorów bitowych przesunięć `<<` oraz `>>` mających ograniczenia oryginałów, zaś dla wielomianów powodujące:
  - Dla operatora `<<` przy jednostkowym prawym argumencie przesunięcie współczynników o jeden w kierunku zmniejszania stopnia. Formalnie efekt odpowiadający podzieleniu lewego argumentu przez x, czyli zmniejszenie stopnia wielomianu o jeden z utrzymaniem wartości współczynników, ale z odrzuceniem dotychczasowego współczynnika zerowego.
  - Dla operatora `>>` przy jednostkowym prawym argumencie przesunięcie współczynników o jeden w kierunku wzrastania stopnia. Formalnie efekt odpowiadający pomnożeniu lewego argumentu przez x, czyli zwiększenie stopnia wielomianu o jeden z utrzymaniem wartości współczynników oraz dodaniem współczynnika zerowego o wartości zerowej.
  - Dla obu powyższych przeładowań wielokrotność prawego argumentu oznacza odpowiednie zwielokrotnienie opisanych działań.
  - Dla przykładu, przy wcześniejszej deklaracji wielomianów `p` oraz `q` działanie:

```
cout << ( p << 1 ) << endl;
cout << ( q >> 2 ) << endl;
```

powoduje wypisanie:

```
( 3, 4, -1 )
( 0, 0, 6, 8, -7 )
```

- Przeładowania wersji kompozycyjnych przeładowanych operatorów `+`, `-`, `*`, `/`, `%`, `<<`, `>>` , oznaczające przeładowanie operatorów `+=`, `-=`, `*=`, `/=`, `%=`, `<<=`, `>>=`.
- Przeładownie operatorów inkrementacyjnych `++` i dekrementacyjnych `--` w wersjach zarówno prefiksowej jak i postfiksowej. Operatory inkrementacyjne zwiększają każdy współczynnik wielomianu o wartość jednostkową, zaś dekrementacyjne analogicznie zmniejszają. Dla przykładowych wcześniejszych deklaracji działanie:

```
cout << p << endl;
cout << q << endl;
q = ++p;
cout << p << endl;
cout << q << endl;
p = q--;
cout << p << endl;
cout << q << endl;
```

oznacza wypisanie:

```
( 1, 3, 4, -1 )
( 6, 8, -7 )
( 2, 4, 5 )
( 2, 4, 5 )
( 2, 4, 5 )
( 1, 3, 4 )
```

- Przeładowanie operatorów pamięci dynamicznej **`new`** oraz **`delete`**. Operatory odpowiadają swoim nieprzeładowanym odpowiednikom, ale modyfikują wspomniane wcześniej pole `overloaded`. Każdorazowe użycie przeładowanego operatora **`new`** oznacza inkrementację zmiennej, zaś analogicznie użycie przeładowanego operatora **`delete`** dekrementację. Tym samym fragment kodu:

```
int main () {
    cout << POLYNOMIAL::overloaded << endl;
    POLYNOMIAL* polyPtr = new POLYNOMIAL;
    cout << polyPtr->overloaded << endl;
    delete polyPtr;
    cout << POLYNOMIAL::overloaded << endl;
```

oznacza wypisanie kolejno liczb `0 1 0`.

4. Przeładowania globalne

Konieczna jest globalna implementacja przeładowania operatorów relacyjnych `<`, `<=`, `==`, `>=`, `>` oraz `!=`. Relację porządkującą wielomiany opisują reguły:

- Wielomian o większym stopniu jest większy od każdego wielomianu o mniejszym stopniu.
- Przy równym stopniu wielomianów porządek rosnący wyznacza leksykograficzny porządek wielkości współczynników począwszy od współczynnika przy najwyższej potędze.

5. Uwarunkowania implementacyjne

- Rozwiązanie musi być zawarte w pliku o nazwie `POLYNOMIAL.cpp` i wysyłane w wersji spakowanej programem `zip`.
- Przesłany plik musi zawierać definicję klasy `POLYNOMIAL` oraz konieczne funkcje przeładowujące. Może zawierać własne podprogramy (zarówno globalne jak i metody klasy `POLYNOMIAL`), ale nie może zawierać żadnych zmiennych globalnych.
- Rozwiązanie jest włączane do pliku głównego programu po włączeniu niezbędnych plików nagłówkowych, a przed ciałem głównego programu.
- Słowo `include` jest zabronione w całym pliku rozwiązania, ale należy założyć wcześniejsze włączenie plików `iostream` oraz `cstdarg`.
- Należy założyć, że przed włączeniem pliku z rozwiązaniem w głównym pliku źródłowym użyto deklaracji:

```
using namespace std;
```