# Rezultate lab4

Enunt

Obiectiv:

* Intelegerea/aprofundarea sablonului “producator-consumator”
* Intelegerea/aprofundarea sincronizarii conditionale
* Intelegerea/aprofundarea excluderii mutuale (granularitatea sectiunilor critice)

Se considera **n** polinoame reprezentate prin lista de monoame.

Se cere adunarea polinoamelor folosind o implementare multithreading (p threaduri).

Consideratii generale:

- reprezentarea unui polinom in memorie: lista inlantuita (1 nod=1monom) ordonata dupa exponentii monoamelor cu urmatorul INVARIANT (predicat adevarat la orice moment al executiei) de reprezentare:

-monoamele sunt ordonate dupa exponenti

-nu se pasteaza in lista monoame cu coeficient 0;

- nu exista doua noduri (monoame) cu acelasi exponent

- polinoamele se citesc din fisiere – cate un fisier pentru fiecare polinom;

- un fisier contine informatii de tip (coeficient, exponent) pentru fiecare monom al

unui polinom,

- fisierele input se creeaza prin generare de numere aleatoare.

(Conditie: fisierele nu contin monoame cu coeficient egal cu 0 dar nu sunt ordonate dupa exponent!)

Rezolvare:

Se porneste prin crearea unei liste inlantuita - L corespunzatoare unui polinom nul.   
In final aceasta lista va continue polinomul rezultat.

Metoda A) Implementare secventiala

* Se citeste pe rand din fiecare fisier cate un monom si se adauga in lista rezultat -L (atentie – invariantul trebuie sa ramana adevarat dupa fiecare adaugare de monom).

Metoda B) Implementare paralela – p threaduri

1. Primul thread citeste cate un monom si il adauga intr-o structura de date de tip coada.

(conditie – pentru structura de tip coada NU se admite folosirea unei structuri de date pentru care partea de sincronizare este deja implementata!!!)

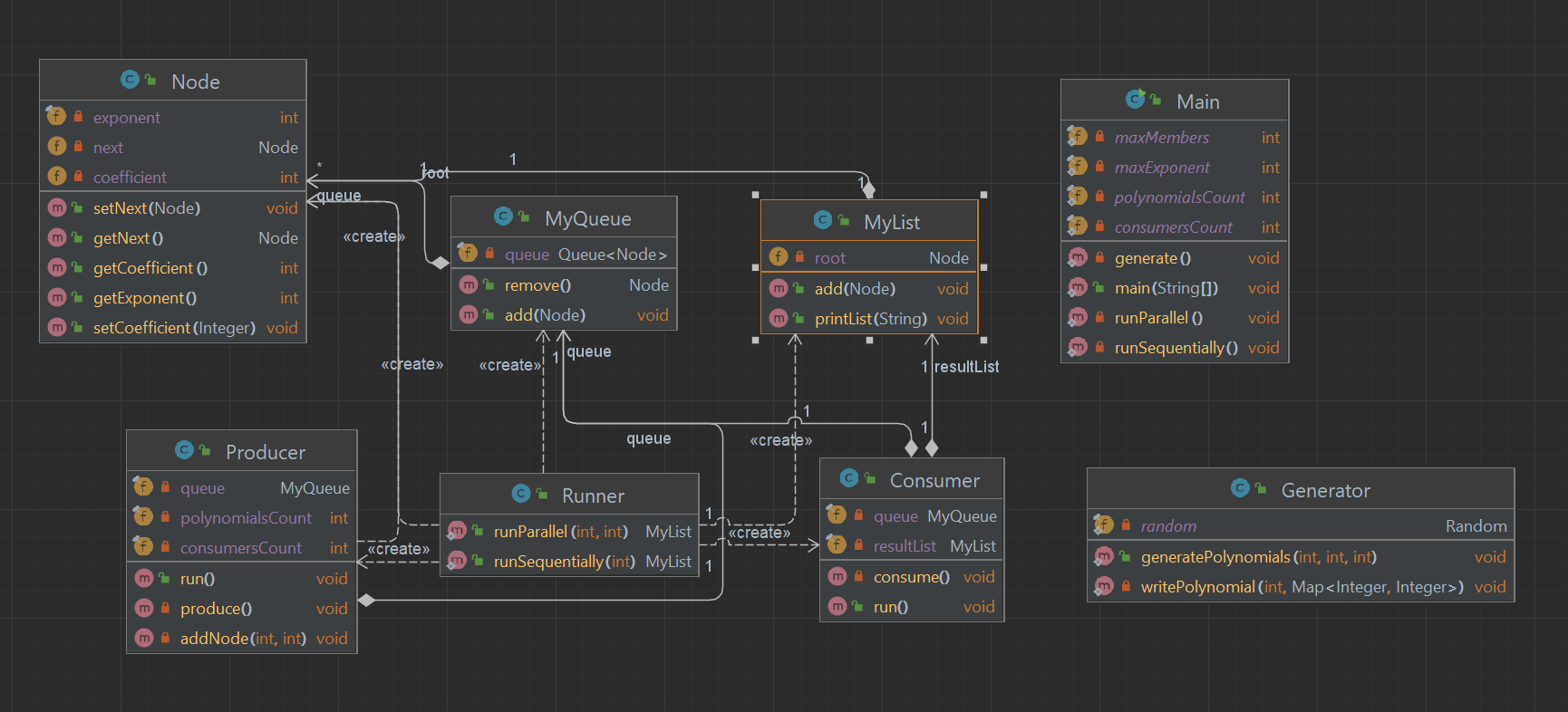
1. Celelalte threaduri preiau cate un monom din coada si il aduna la polinomul reprezentat in lista L.
   * Se continua operatiile 1., 2. pana cand toate monoamele, din toate fisierele, sunt adunate la lista L.
2. Primul thread scrie rezultatul obtinut in lista L intr-un fisier rezultat

(conditie: fisierul nu contine monoame cu coefficient egal cu 0)

**Productor-consumator**

**Sincronizare la nivel de lista!!!**

Diagrama de clase



Partitionarea datelor

* Cazul secvential
  + In acest caz, exista un singur thread(main) care citeste polinoamele din fisier, adauga monom cu monom in coada si la sfarsit null
  + Tot acest thread, ia element cu element din coada si il adauga in lista rezultat, pana cand intalneste null
* Cazul parallel
  + Avem doua categorii de thread-uri: producer(unul) si consumer(mai multe)
  + Producer-ul citeste polinoamele din fisiere si adauga monoamele in coada, iar la sfarsit adauga null pentru dfiecare consumer
  + Consumer-ii iau din coada element cu element, fiecare cum prinde; in momentul in care un consumer are ca element null, executia acestuia se incheie; actiunea propriu-zisa de “consum” consta in adaugarea monomului in lista rezultat, cu calculele aferente

Cazuri de testare

* Datele au fost validate, comparand rezultatele cazului secvential cu cel paralel, accesand site-ul <https://text-compare.com/>

|  |  |  |
| --- | --- | --- |
| **Caz** | **Nr Threads** | **Timp(milis)** |
| 10 polinoame  maxExponent = 1000  maxMembers = 50 | 1 | 51 |
| 4 | 49 |
| 6 | 53 |
| 8 | 56 |
| 5 polinoame  maxExponent = 10000  maxMembers = 100 | 1 | 57 |
| 4 | 53 |
| 6 | 55 |
| 8 | 58 |

Concluzii

* Timpii sunt foarte asemanatori, deci nu ma pot pronunta asupra carui caz este optim, dar pare ca in 4 consumatori sunt ideali
* Cazurile de testare contin prea putine date si astfel nu exista o concurenta sufficient de pronuntata