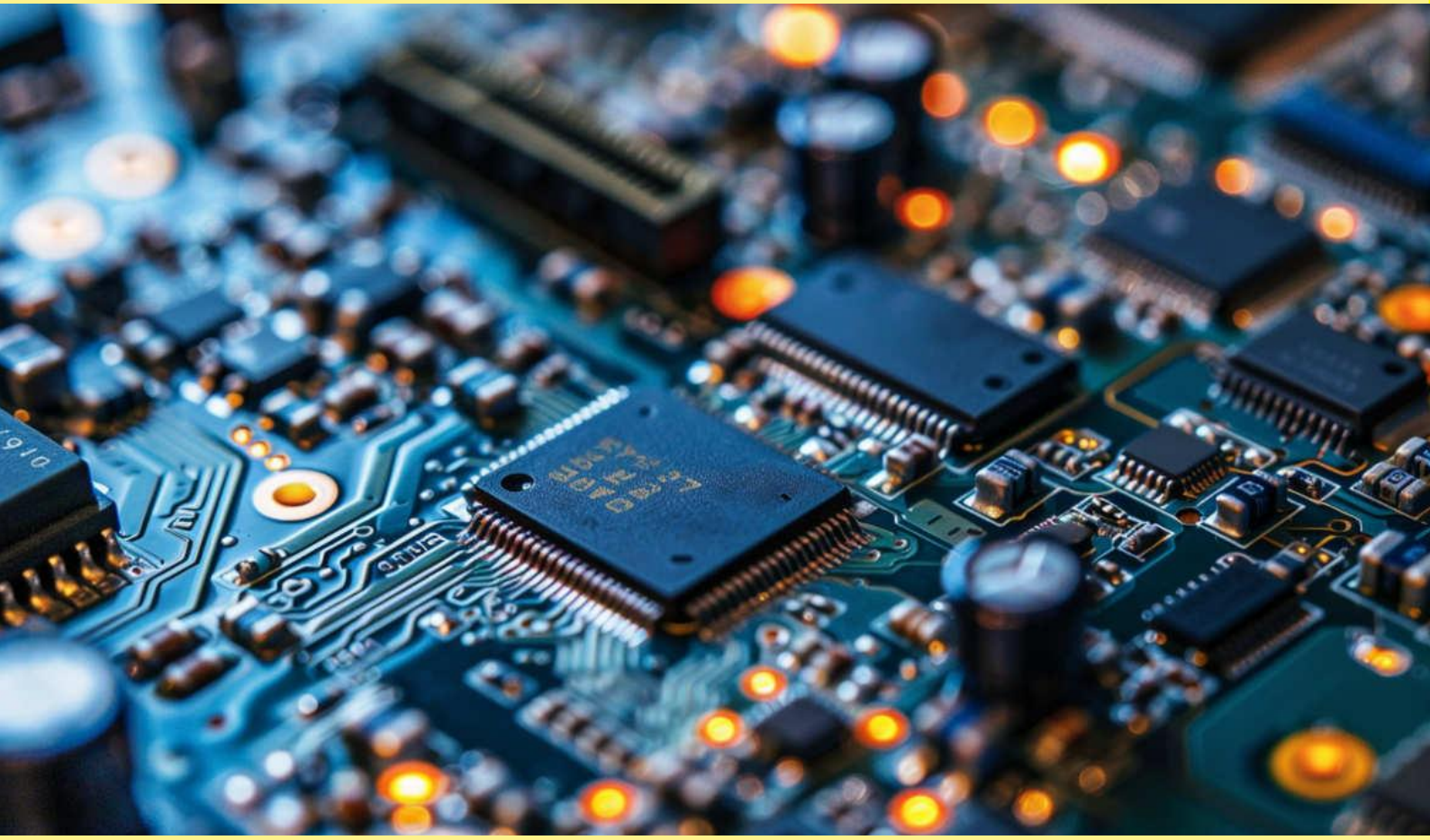
CPI A2 – PROJET POO

PROSIT 6 – UNE VRAIE CALCULATRICE



CESI
ÉCOLE D'INGÉNIEURS

CONTEXTE :

Nous sommes en 1985 et 2 ingénieurs de Microsoft, Bill G et Steve B ont un travail pour nous. Un code partiellement rempli codant une calculatrice prenant en compte les priorités opératoire (parenthèses) nous est donné. Notre travail est de la faire fonctionner.

INFORMATIONS IMPORTANTES :

Un exemple du fonctionnement est donné : ? = (((2+3)-10)*2)

1. push(P, ‘(’) => P = <’(‘>
2. push(P, ‘(‘) => P = <’(‘, ‘(‘>
3. push(P, ‘(‘) => P = <’(‘, ‘(‘, ‘(‘>
4. push(P, ‘2‘) => P = <’(‘, ‘(‘, ‘(‘, ‘2’>
5. push(P, ‘+‘) => P = <’(‘, ‘(‘, ‘(‘, ‘2’, ‘+’>
6. push(P, ‘3‘) => P = <’(‘, ‘(‘, ‘(‘, ‘2’, ‘+’, ‘3’>
7. push(P, ‘)‘) => P = <’(‘, ‘(‘, ‘(‘, ‘2’, ‘+’, ‘3’, ‘)’>
8. pop(P) => P = <’(‘, ‘(‘, ‘(‘, ‘2’, ‘+’, ‘3’>
9. b = pop(P) => P = <’(‘, ‘(‘, ‘(‘, ‘2’, ‘+’>
10. op = pop(P) => P = <’(‘, ‘(‘, ‘(‘, ‘2’>
11. a = pop(P) => P = <’(‘, ‘(‘, ‘(‘>
12. pop(P) => P = <’(‘, ‘(‘>
13. push(P, ‘5‘) => P = <’(‘, ‘(‘, ‘5‘> 5 = a+b
14. push(P, ‘-‘) => P = <’(‘, ‘(‘, ‘5‘, ‘-‘>
15. push(P, ‘10‘) => P = <’(‘, ‘(‘, ‘5‘, ‘-‘, ‘10’>
16. push(P, ‘)‘) => P = <’(‘, ‘(‘, ‘5‘, ‘-‘, ‘10’, ')'>
17. ...

MOTS INCONNUS :

Test unitaires : Un test unitaire est une pratique de test dans laquelle des sections de code, appelées “unités”, sont testées individuellement pour vérifier leur bon fonctionnement. Ces unités peuvent être des fonctions, des méthodes ou même des classes.

push (agrandis la sélection sur le caractère de droite)

pop (retire un caractère à droite)

« a », « b » mémoire des nombre

« op » mémoire des opérateurs

Unary : opérateurs

Binary : opérandes

PROBLEMATIQUE :

Comment pouvons-nous compléter le code afin que les méthodes du code de la calculatrice soit fonctionnel ? (Prendre en compte les priorités opératoires)

PLAN D'ACTION :

- Récupérer le code sur GitHub et réussir à le faire fonctionner [20min]
- Saisir le fonctionnement du code à l'aide du diagramme de classe donné [40min]
- Ajouter les méthodes manquantes et faire fonctionner le code [360min]

REALISATION :

Dans ce diagramme de séquence, l'utilisateur lance le programme avec un fichier d'entrée, ce qui déclenche l'initialisation de la grille via le GestionnaireFichiers, qui charge les données du fichier. Ensuite, une boucle s'exécute à chaque itération : `main` met à jour la grille, calcule l'état futur des cellules avec Cellule (basé sur les voisins vivants), et applique ces changements dans Grille. Après chaque itération, l'état de la grille est affiché dans la console via `afficherConsole()` et sauvegardé dans un fichier par GestionnaireFichiers. Le programme répète ces étapes jusqu'à la fin des itérations. Réaliser ce programme permet de structurer le projet, de diviser les tâches entre différents composants, et d'améliorer la gestion du code.