Université Claude Bernard Lyon 1

DIU EIL – UE 4
Pile et file

Nicolas Pronost

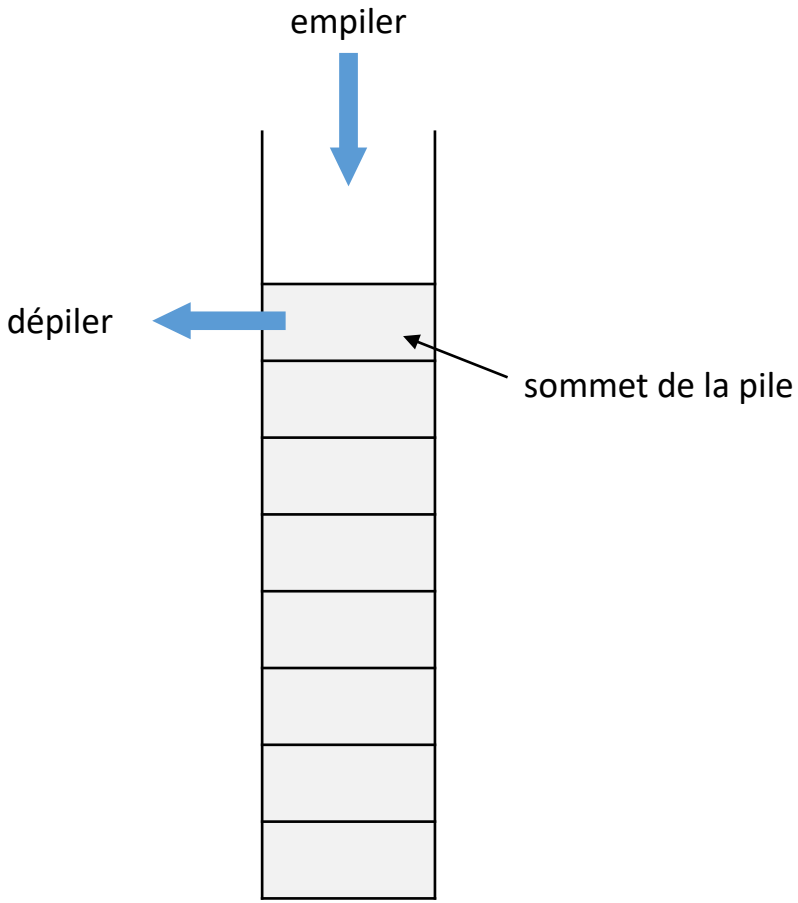
LIRIS

Principe d'une pile

- Une pile est une structure de donnée où seul un élément est accessible à la fois : le sommet de la pile
- Les lectures et écritures utilisent le principe du dernier arrivé, premier utilisé
  - LIFO en anglais (Last In First Out)
- Les opérations possibles sont
  - création et destruction de la pile
  - empiler un élément dans la pile
  - dépiler le sommet de la pile
  - consulter le sommet de la pile
  - tester si la pile est vide

Représentation d'une pile

- On représente le contenu d'une pile comme un tableau dont on ne pourrait manipuler que la dernière case

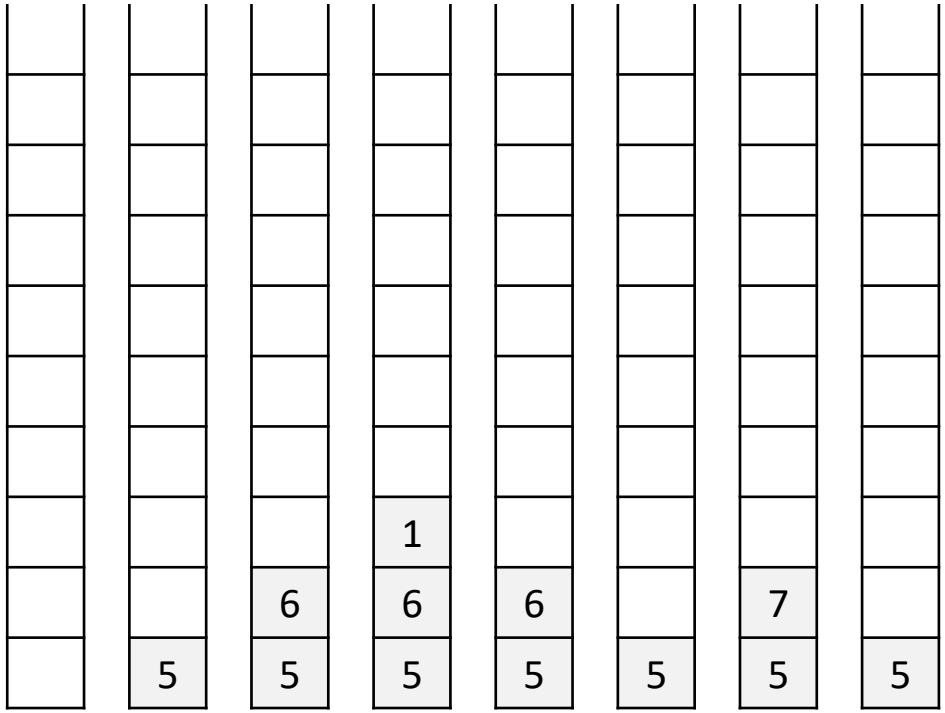


Fonctionnalités de Pile

- Constructeur Pile()
  - Postconditions : la pile est une pile vide
- Destructeur ~Pile()
  - Postconditions : libération de la mémoire utilisée, la pile est une pile vide
- Procédure empiler (e)
  - Postcondition : e est ajouté en sommet de la pile
- Procédure dépiler ()
  - Précondition : la pile n'est pas vide
  - Postcondition : le sommet de la pile est dépilé
- Procédure vider ()
  - Postcondition : la pile ne contient plus aucun élément
- Fonction estVide () : booléen
  - Résultat : retourne vrai si la pile est vide, faux sinon
- Fonction sommet () : tout type
  - Précondition : la pile n'est pas vide
  - Résultat : retourne le sommet de la pile
- Fonction traiter () : tout type
  - Précondition : la pile n'est pas vide
  - Postcondition : le sommet de la pile est dépilé
  - Résultat : retourne le sommet de la pile

Exemple d'utilisation

```
Variables locales :
   p : Pile, s : entier, v : booléen
Début
   v ← p.estVide()
   p.empiler(5)
   p.empiler(6)
   p.empiler(1)
   p.dépiler()
   s ← p.sommet()
   p.dépiler()
   p.empiler(7)
   s ← p.sommet()
   p.dépiler()
   v ← p.estVide()
Fin
```



Implémentation d'une pile

- Par une liste chaînée : sommet de pile = tête de liste
  - empiler = ajouter en tête
  - dépiler = supprimer la tête
  - ➢ coût constant $O(1)$
- Par un tableau dynamique (liste Python) : sommet de pile = dernière case
  - empiler = ajouter en dernière position
  - dépiler = supprimer l'élément en dernière position
  - ➢ coût amorti constant $O(1)$

Exemple : évaluation d'expression

- On souhaite évaluer les expressions du type :

```
identificateur = expression arithmétique
```

- Exemple: variable = ((objet – 1) + x * tmp / 8) / 25 – 12
- Rappel sur les priorités des opérateurs arithmétiques
  - * et / sont plus prioritaires que + et –
  - = est le moins prioritaire
- Une solution consiste à d'abord transformer cette expression en une représentation postfixée qui est facile à évaluer
  - Notation infixé : a + 1
  - Notation préfixée : + a 1
  - Notation postfixée : a 1 +

Exemple : évaluation d'expression

- L'expression de l'exemple

  variable = ((objet – 1) + x * tmp / 8) / 25 – 12

- Doit donc devenir

  variable (((objet 1 –) ((x tmp *) 8 /) +) 25 /) 12 -) =

- Mais les parenthèses deviennent inutiles (pas d'ambigüité)

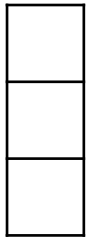
  variable objet 1 – x tmp * 8 / + 25 / 12 - =

Exemple : passage infixe à postfixe

- Méthode : soit P une pile et T un tableau, on lit les éléments de l'expression de gauche à droite
  - Si l'élément courant est
    - un identificateur (ex. variable, objet, x, tmp) : on le recopie dans T
    - un nombre (ex. 1, 8, 25, 12) : on le recopie dans T
    - un opérateur mathématique (ex. +, -, *, /) alors :
      - on dépile de P les opérateurs de priorité supérieure ou égale
      - on les recopie dans T
      - on empile dans P l'opérateur courant
    - une parenthèse fermante) alors :
      - on dépile de P les éléments jusqu'à une parenthèse ouvrante (
      - on recopie ces éléments dans T (parenthèse ouvrante exclue)
    - l'opérateur d'affectation = ou une parenthèse ouvrante (, on l'empile dans P
  - Si on est à la fin de l'expression, on dépile tous les éléments de P et on les recopie dans T

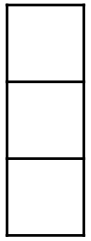
Exemple : passage infixe à postfixe

- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale



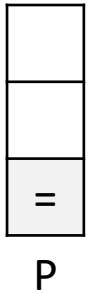
Exemple : passage infixe à postfixe

- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - **Etape 1 : x**
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale



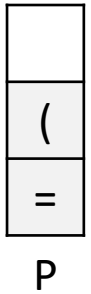
Exemple : passage infixe à postfixe

- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - **Etape 2 : =**
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale



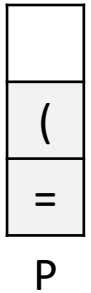
Exemple : passage infixe à postfixe

- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - **Etape 3 : (**
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale



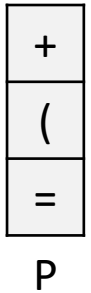
Exemple : passage infixe à postfixe

- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - **Etape 4 : a**
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale



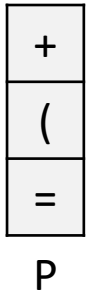
Exemple : passage infixe à postfixe

- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - **Etape 5 : +**
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale



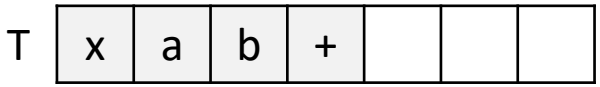
Exemple : passage infixe à postfixe

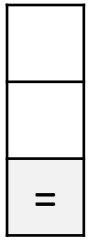
- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - **Etape 6 : b**
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale



Exemple : passage infixe à postfixe

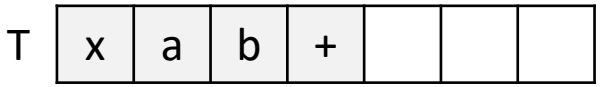
- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - **Etape 7 :)**
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale

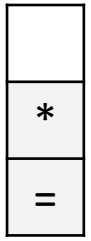




Exemple : passage infixe à postfixe

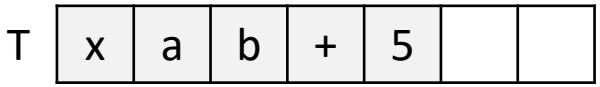
- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - **Etape 8 : ***
  - Etape 9 : 5
  - Etape finale

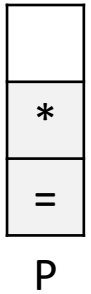




Exemple : passage infixe à postfixe

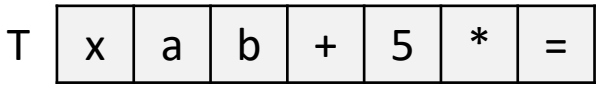
- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - **Etape 9 : 5**
  - Etape finale

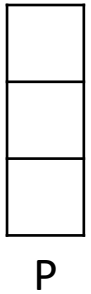




Exemple : passage infixe à postfixe

- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - **Etape finale**



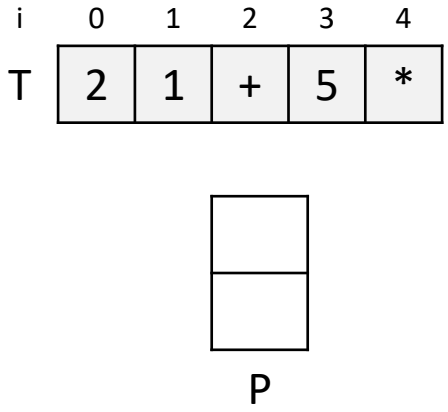


Exemple : évaluation de l'expression

- Méthode d'évaluation d'une expression postfixe :
  - on itère sur les éléments de T (itération i) en utilisant une pile P
    - si T[i] est un nombre alors on l'empile dans P
    - si T[i] est un opérateur alors on évalue l'opération entre les deux premiers éléments de la pile (que l'on dépile), et on empile le résultat
  - à la fin de l'itération le résultat est au sommet de la pile

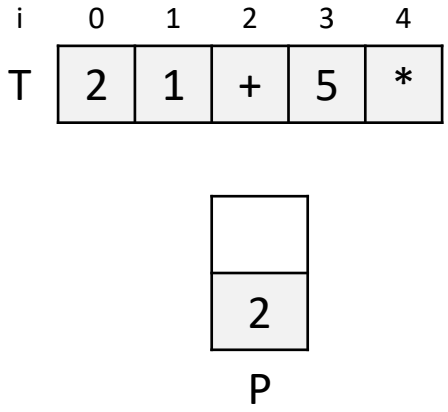
Exemple : évaluation de l'expression

- Sur l'exemple (2 + 1) * 5
  - i = 0
  - i = 1
  - i = 2
  - i = 3
  - i = 4
  - résultat



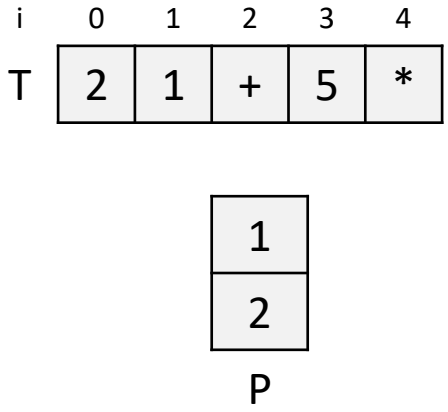
Exemple : évaluation de l'expression

- Sur l'exemple (2 + 1) * 5
  - **i = 0**
  - i = 1
  - i = 2
  - i = 3
  - i = 4
  - résultat



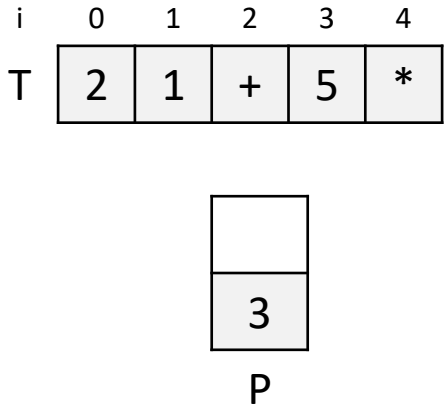
Exemple : évaluation de l'expression

- Sur l'exemple (2 + 1) * 5
  - i = 0
  - **i = 1**
  - i = 2
  - i = 3
  - i = 4
  - résultat



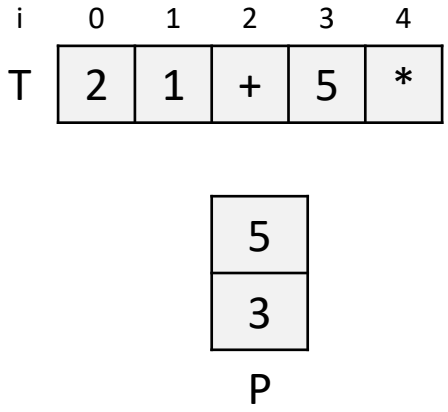
Exemple : évaluation de l'expression

- Sur l'exemple (2 + 1) * 5
  - i = 0
  - i = 1
  - **i = 2**
  - i = 3
  - i = 4
  - résultat



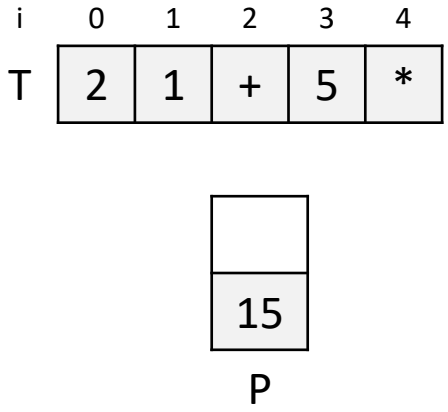
Exemple : évaluation de l'expression

- Sur l'exemple (2 + 1) * 5
  - i = 0
  - i = 1
  - i = 2
  - **i = 3**
  - i = 4
  - résultat



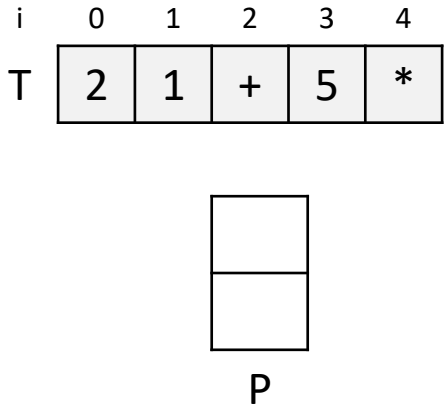
Exemple : évaluation de l'expression

- Sur l'exemple (2 + 1) * 5
  - i = 0
  - i = 1
  - i = 2
  - i = 3
  - **i = 4**
  - résultat



Exemple : évaluation de l'expression

- Sur l'exemple (2 + 1) * 5
  - i = 0
  - i = 1
  - i = 2
  - i = 3
  - i = 4
  - **résultat = 15**

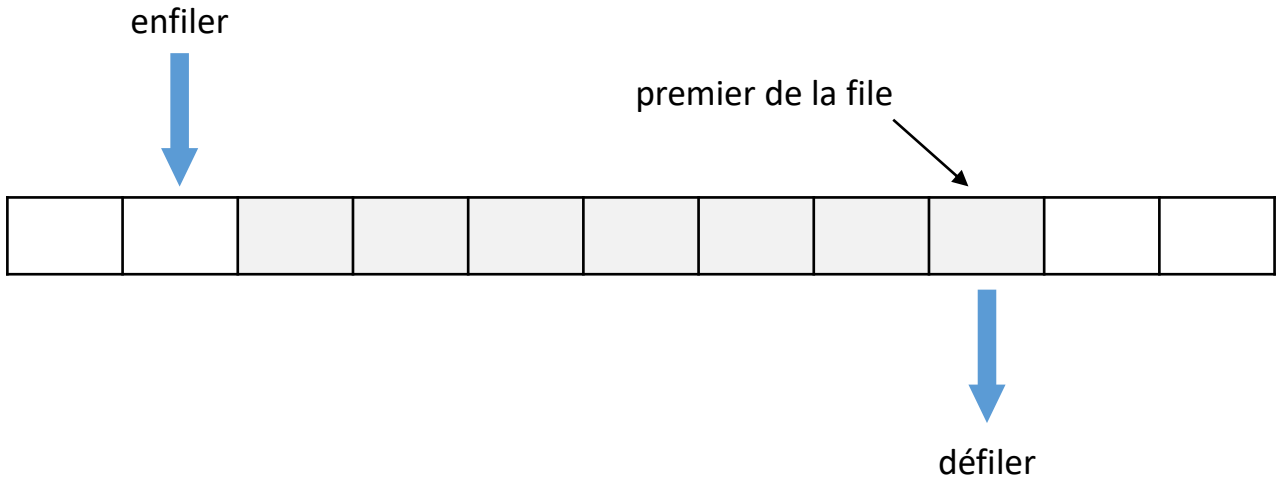


Principe d'une file

- Une file est une structure de donnée où seul un élément est accessible à la fois : le premier de la file
- Les lectures et écritures utilisent le principe du premier arrivé, premier utilisé
  - FIFO en anglais (First In First Out)
- Les opérations possibles sont
  - création et destruction de la file
  - enfiler un élément dans la file
  - défiler le premier de la file
  - consulter le premier de la file
  - tester si la file est vide

Représentation d'une file

- On représente le contenu d'une file comme une liste dont on ne pourrait insérer qu'à un bout et supprimer que à l'autre

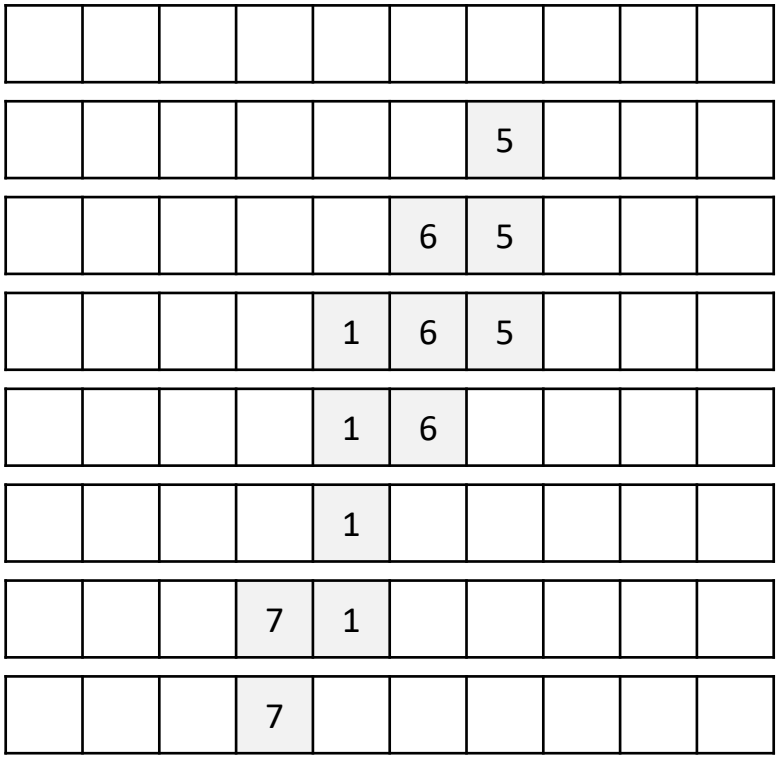


Module File

- Constructeur File()
  - Postconditions : la file est une file vide
- Destructeur ~File()
  - Postconditions : libération de la mémoire utilisée, la file est une file vide
- Procédure enfiler (e)
  - Postcondition : e est ajouté à la file
- Procédure défiler ()
  - Précondition : la file n'est pas vide
  - Postcondition : le premier de la file est supprimé
- Procédure vider ()
  - Postcondition : la file ne contient plus aucun élément
- Fonction estVide () : booléen
  - Résultat : retourne vrai si la file est vide, faux sinon
- Fonction premierDeLaFile () : tout type
  - Précondition : la file n'est pas vide
  - Résultat : retourne le premier de la file
- Fonction traiter() : tout type
  - Précondition : la file n'est pas vide
  - Postcondition : le premier de la file est supprimé
  - Résultat : retourne le premier de la file

Exemple d'utilisation

```
Variables locales :
   f : File, s : entier, v : booléen
Début
   v ← f.estVide()
   f.enfiler(5)
   f.enfiler(6)
   f.enfiler(1)
   f.défiler()
   s ← f.premierDeLaFile()
   f.défiler()
   f.enfiler(7)
   s ← f.premierDeLaFile()
   f.défiler()
   v ← f.estVide()
Fin
```



Implémentation d'une file

- Par une liste chaînée : premier de la file = tête (ou queue) de liste
  - enfiler = ajouter en queue (ou tête)
  - défiler = supprimer la tête (ou queue)
  - ➢ coût dépend de l'implémentation de la liste (simplement chaînée ou doublement chaînée, constant $O(1)$ ou linéaire $O(n)$)
- Par un tableau dynamique : premier de la file = première case du tableau (ou dernière)
  - enfiler = ajouter en dernière position (ou première)
  - défiler = supprimer l'élément en première position (ou dernière)
  - ➢ coût amorti de l'un constant $O(1)$ et l'autre linéaire $O(n)$

Exemples

- Beaucoup de systèmes informatiques reposent sur le principe de pile implémenté dans des buffers
  - mémorisation de transactions
  - serveur d'impression (buffer de requêtes)
  - moteur multitâche pour l'allocation du temps processeur
  - gestion des évènements (ex. frappe clavier)
- Là où il y a un ensemble de données en attente de traitement, il y a souvent une file