Université Claude Bernard Lyon 1

LIFAPSD – Algorithmique, Programmation et Structures de données

Nicolas Pronost

LIRIS

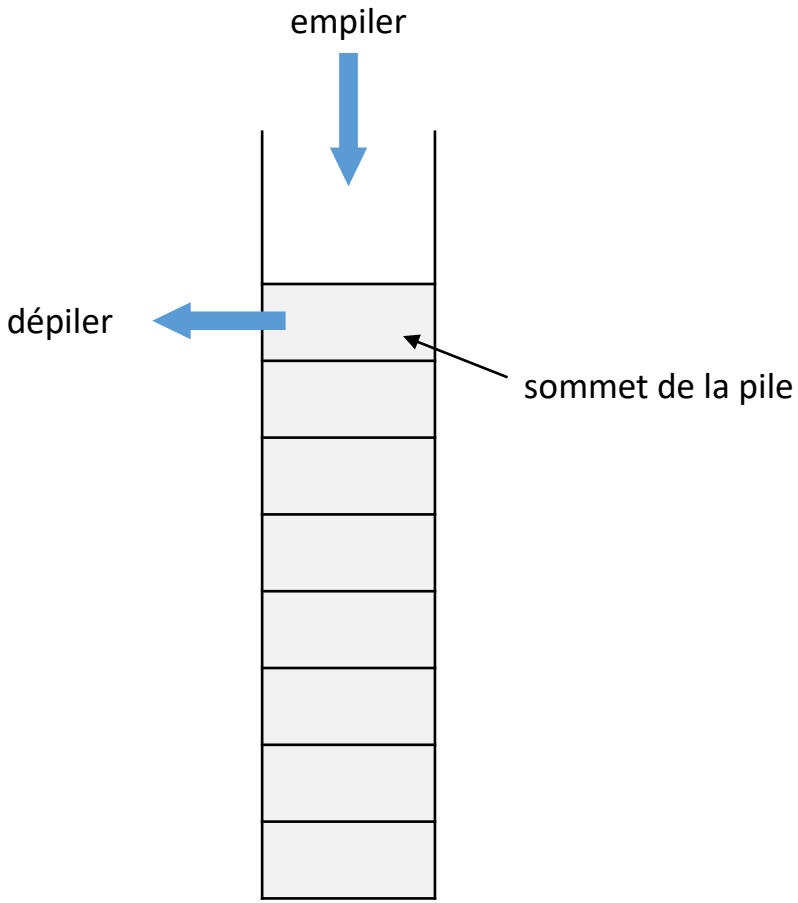
Chapitre 8
Pile et file

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



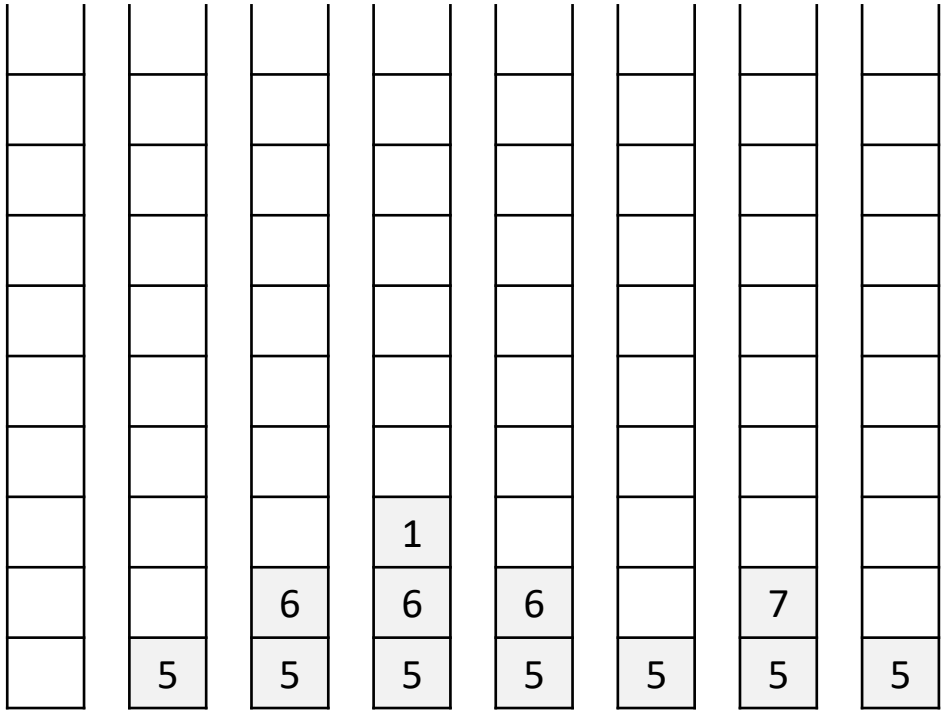
Module Pile

Module Pile

- **Importer:**
  - Module ElementP
- **Exporter:**
  - Type Pile
    - Constructeur Pile()
      - Postconditions : la pile est une pile vide
    - Destructeur ~Pile()
      - Postconditions : libération de la mémoire utilisée sur le tas, la pile est une pile vide
    - Procédure empiler (e: ElementP)
      - Postcondition : une copie de e est ajoutée en sommet de la pile
      - Paramètre en mode donnée : e
    - 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 : vrai si la pile est vide, faux sinon
    - Fonction consulterSommet () : ElementP
      - Précondition : la pile n'est pas vide
      - Résultat : 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.consulterSommet()
   p.dépiler()
   p.empiler(7)
   s ← p.consulterSommet()
   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 : sommet de pile = dernière case
  - empiler = ajouter en dernière position
  - dépiler = supprimer l'élément en dernière position
  - ➢ coût constant $O(1)$

- En TP, on choisira l'implémentation par tableau dynamique

Mise en œuvre d'une pile en C++

- En utilisant un tableau dynamique

Pile.h

```
class Pile {
public:
   TableauDynamique t;

   Pile ();
   ~Pile ();

   void empiler (ElementP e);
   void depiler ();
   ElementP consulterSommet () const;
   bool estVide () const;
};
```

Exemple 1 : gestion de la récursivité

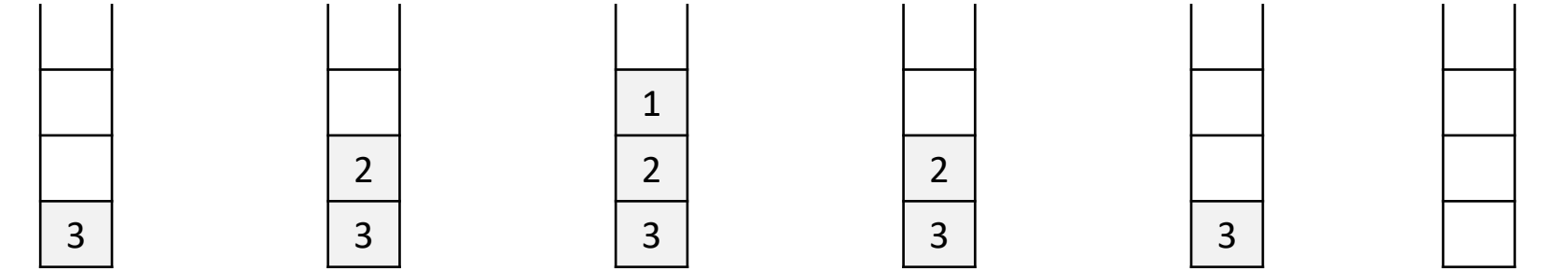
- Prenons l'exemple du calcul récursif de la factorielle d'un entier positif : $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$

```
unsigned int factoriel (unsigned int n) {
   if (n > 0) {
      unsigned int x = factoriel(n-1);
      return n*x;
   }
   else return 1;
}

appel : unsigned int z = factoriel(3);
```

Exemple 1 : gestion de la récursivité

- Le calcul peut aussi être fait facilement avec une pile où on empile les valeurs de n de manière décroissante (dans un for) et où on les dépile en les multipliant ensemble



Exemple 2 : 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 2 : 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)

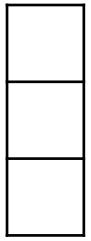
  variable objet 1 – x tmp * 8 / + 25 / 12 - =

Exemple 2 : 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

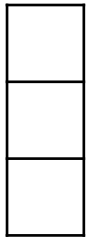
Exemple 2 : passage infixe à postfixe

- 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



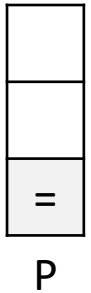
Exemple 2 : passage infixe à postfixe

- 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



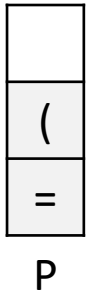
Exemple 2 : passage infixe à postfixe

- 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



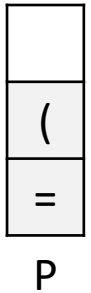
Exemple 2 : passage infixe à postfixe

- 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



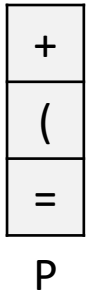
Exemple 2 : passage infixe à postfixe

- 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



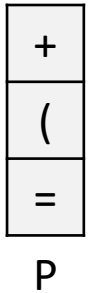
Exemple 2 : passage infixe à postfixe

- 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



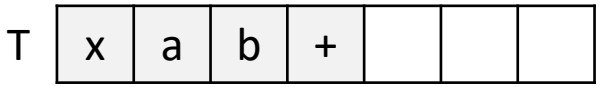
Exemple 2 : passage infixe à postfixe

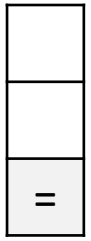
- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - **Etape 6 : b**
  - Etape 7 :)
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale



Exemple 2 : passage infixe à postfixe

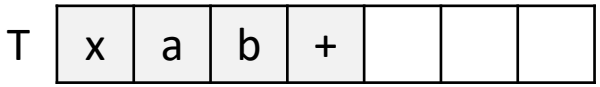
- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - **Etape 7 :)**
  - Etape 8 : *
  - Etape 9 : 5
  - Etape finale

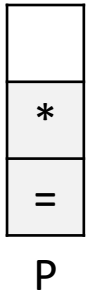




Exemple 2 : passage infixe à postfixe

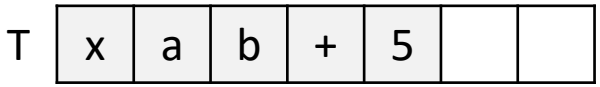
- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - **Etape 8 : ***
  - Etape 9 : 5
  - Etape finale

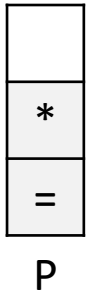




Exemple 2 : passage infixe à postfixe

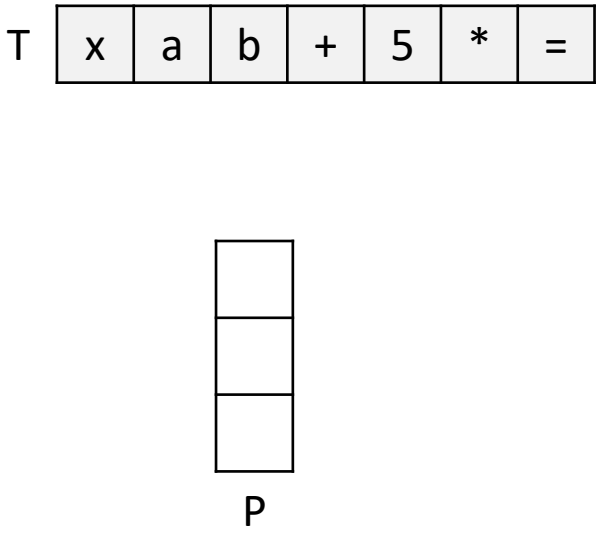
- T contient alors la représentation postfixe de l'expression infixe
- Démonstration avec l'expression : x = (a + b) * 5
  - Etape 1 : x
  - Etape 2 : =
  - Etape 3 : (
  - Etape 4 : a
  - Etape 5 : +
  - Etape 6 : b
  - Etape 7 :)
  - Etape 8 : *
  - **Etape 9 : 5**
  - Etape finale





Exemple 2 : 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**

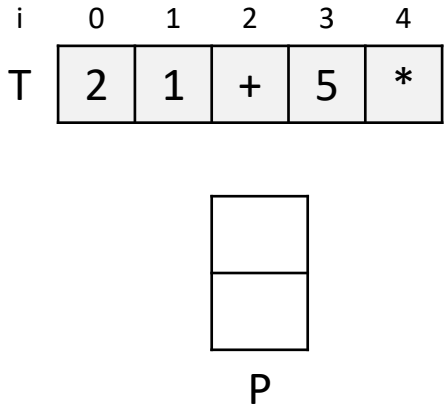


Exemple 2 : évaluation de l'expression

- 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
- Le code de l'algorithme sera vu en TD

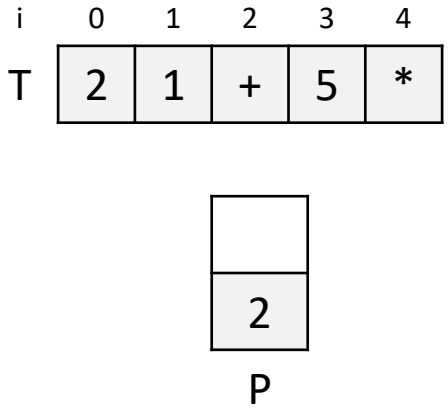
Exemple 2 : évaluation de l'expression

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



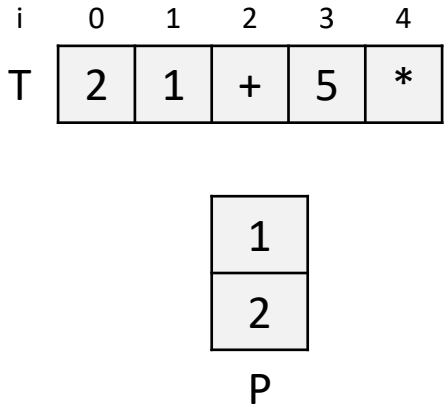
Exemple 2 : évaluation de l'expression

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



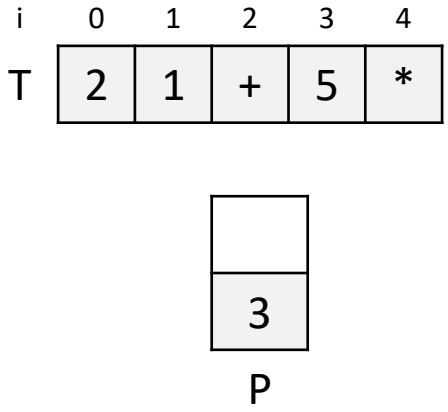
Exemple 2 : évaluation de l'expression

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



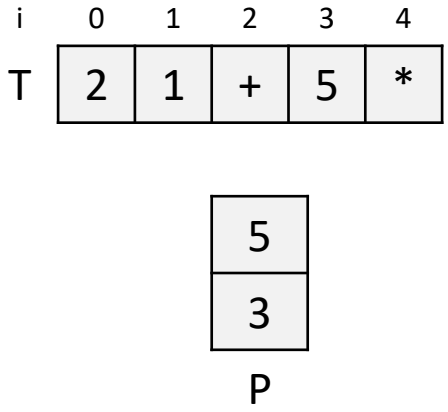
Exemple 2 : évaluation de l'expression

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



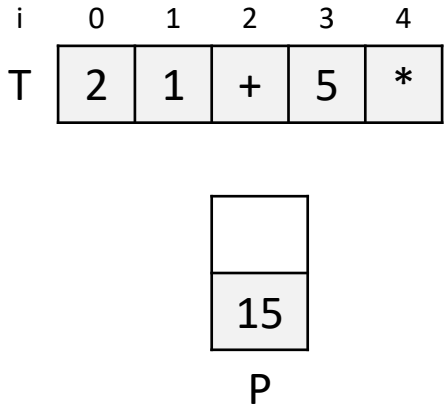
Exemple 2 : évaluation de l'expression

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



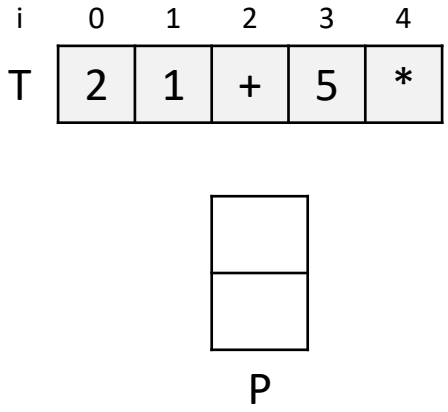
Exemple 2 : évaluation de l'expression

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



Exemple 2 : évaluation de l'expression

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

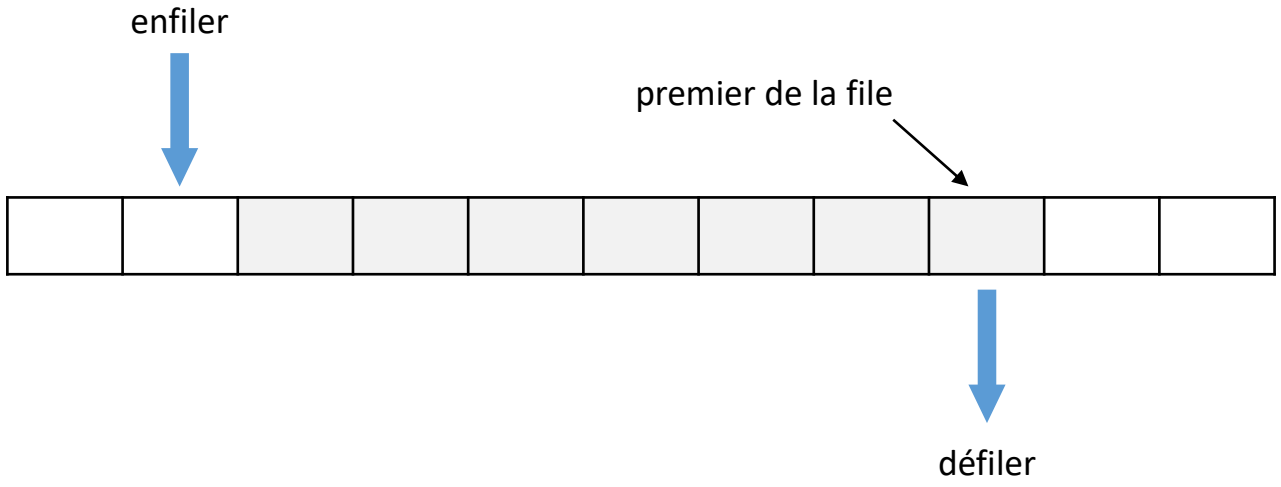


Principe d'une file

- 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



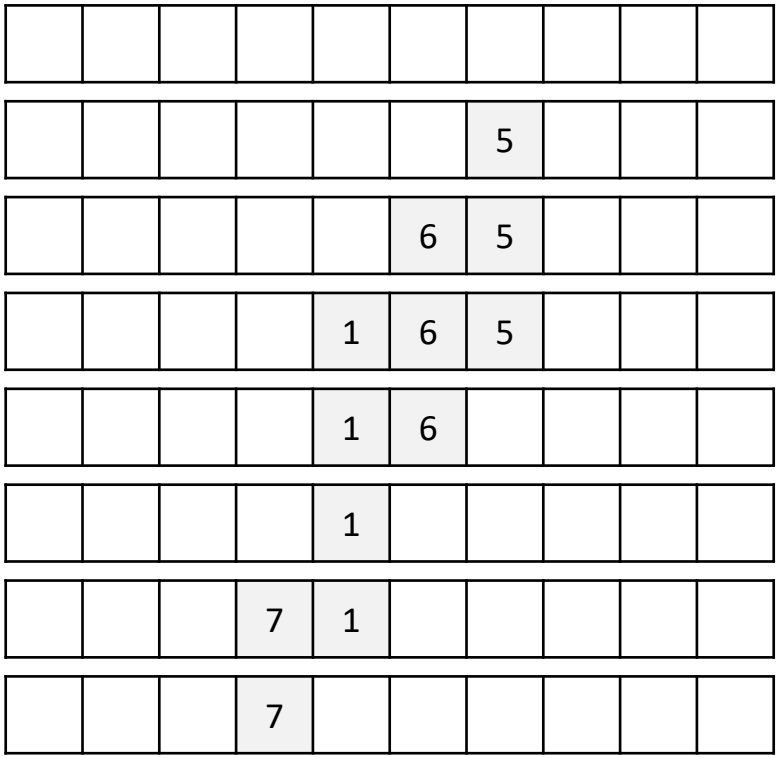
Module File

Module File

- **Importer:**
  - Module ElementF
- **Exporter:**
  - Type File
    - Constructeur File()
      - Postconditions : la file est une file vide
    - Destructeur ~File()
      - Postconditions : libération de la mémoire utilisée sur le tas, la file est une file vide
    - Procédure enfiler (e: ElementF)
      - Postcondition : une copie de e est ajoutée à la file
      - Paramètre en mode donnée : e
    - 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 : vrai si la file est vide, faux sinon
    - Fonction premierDeLaFile () : ElementF
      - Précondition : la file n'est pas vide
      - Résultat : 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 de l'un constant $O(1)$ et l'autre linéaire $O(n)$
- En TP, on choisira l'implémentation par liste doublement chaînée où le premier de la file est en tête de liste

Mise en œuvre d'une file en C++

- En utilisant une liste

File.h

```
class File {
public:
   Liste l;

   File ();
   ~File ();

   void enfiler (ElementF e);
   void defiler ();
   ElementF premierDeLaFile () const;
   bool estVide () const;
};
```

Exemples

- Beaucoup de systèmes informatiques reposent sur le principe de file 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

Traitement d'une file d'attente

- 1 programme principal et 2 processus parallèles

main.cpp

```
File f;
Thread t1 (&f);
Thread t2 (&f);
t1.run();
t2.run();
bool quit = false;

while (!quit) {
   while (!f.estVide()) {
      ElementF e = f.premierDeLaFile();
      f.defiler();
      traitementElement(e);
   }
   ...
}
```

Thread.cpp

```
Thread::Thread(File * pf) { tf = pf; }

Thread::run() {
   if (...) {
      ElementF e = ...
      tf->enfiler(e);
   }
}
```