Structures de Données

Séance 1 – Pile et file

INFRES 12 – janvier 2020

Structure de données

Type abstrait de données : définit les opérations que l'on peut réaliser sur les données de ce type.

Structure de données : réalisation concrète d'un type abstrait

Pile

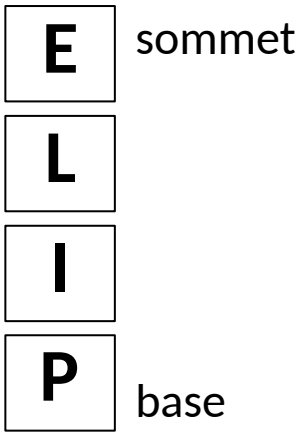
Pile - définition

- Contient un ensemble d'éléments
- Le dernier rentré sera le premier à sortir (DRPS) :

Last In First Out (**LIFO**)

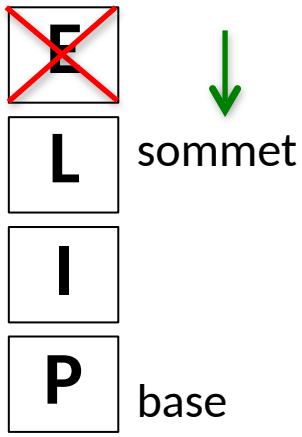
- Primitives :
  - empiler : ajout au sommet
  - dépiler : retrait de l'élément au sommet
  - estVide
  - [sommet : lecture du sommet]

- Exemple :
  - Ajout successif de P, I, L puis E =>



Pile - définition

- Exemple :
  - Appel de dépiler => renvoie ‘E’ et l’enlève de la pile



File

File - définition

- Contient un ensemble d'éléments
- Le premier rentré sera le premier à sortir (PRPS) :

First In First Out (FIFO).

- Primitives :
  - enfiler : ajout en dernier
  - défiler : retrait du premier élément
  - estVide
  - [premier / dernier]

- Exemple :
  - Ajout successif de F, I, L puis E =>

File - définition

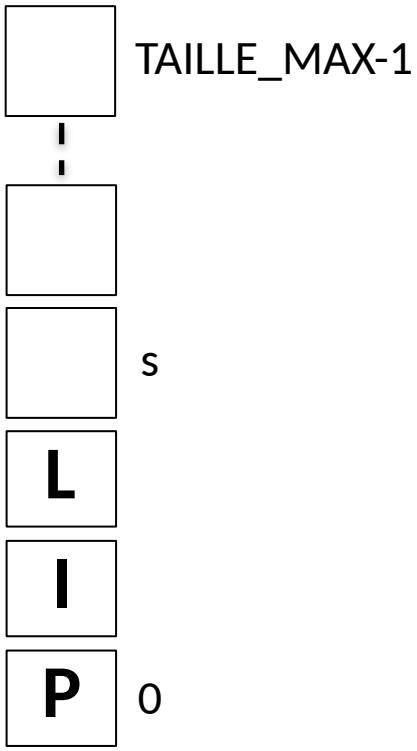
- Exemple :
  - Appel de défiler => renvoie ‘F’ et l’enlève de la file

Pile Implantation

Pile
Implantation par tableau

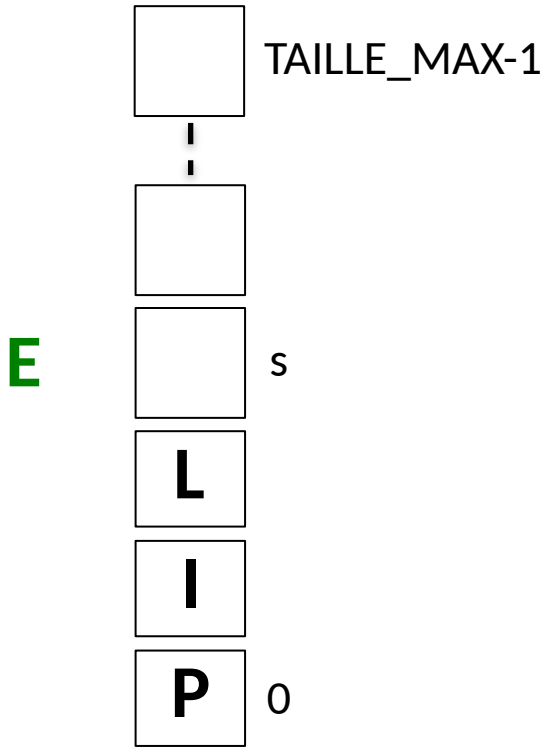
- Si l'on connaît la taille maximale : implantation par tableau avec gestion de l'indice du sommet.

```
struct pile {
      int s; // indice du sommet+1
      char valeurs[TAILLE_MAX];
}; // pile vide : s = 0
```



Pile
Implantation par tableau

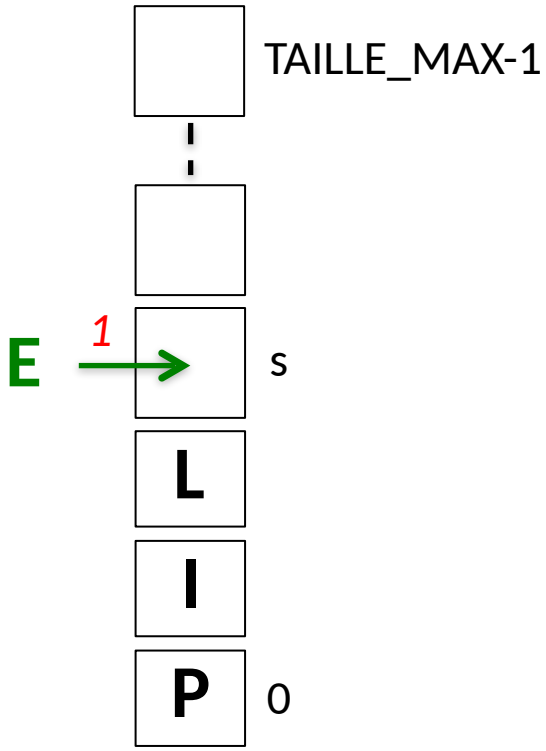
```
void empiler(struct pile *p, char val){
      ???
}
```



Pile
Implantation par tableau

```
void empiler(struct pile *p, char val){
     // pile pleine si p->s = TAILLE_MAX-1
     p->valeurs[p->s   ] = val;
}
```

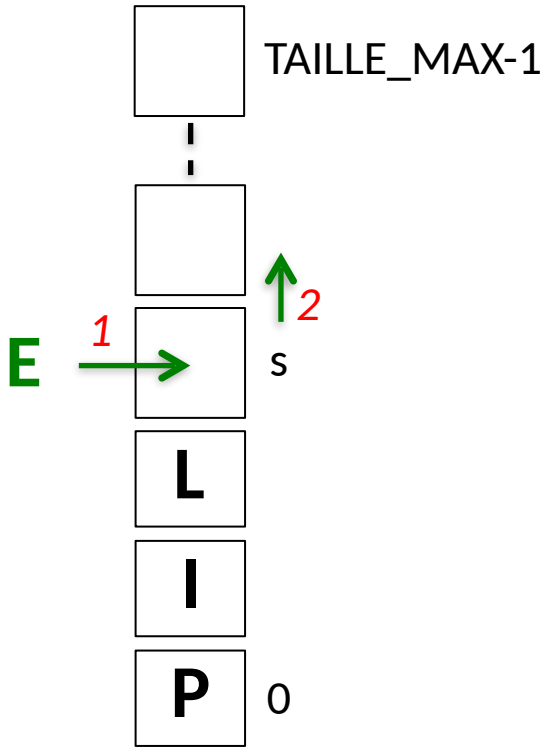
1



Pile
Implantation par tableau

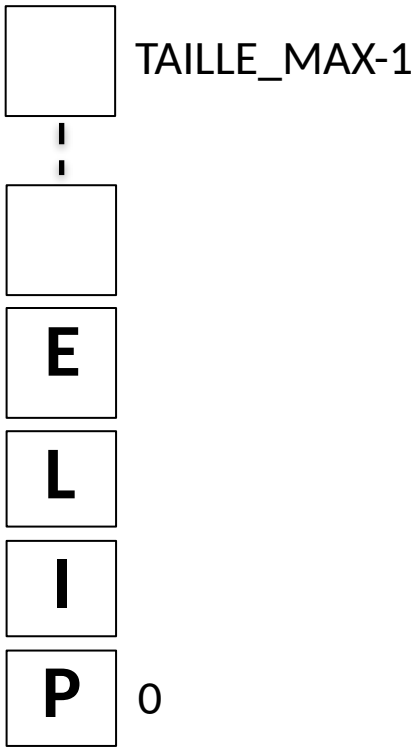
```
void empiler(struct pile *p, char val){
     // pile pleine si p->s = TAILLE_MAX
     p->valeurs[p->s++] = val;
}
```

2 *1*



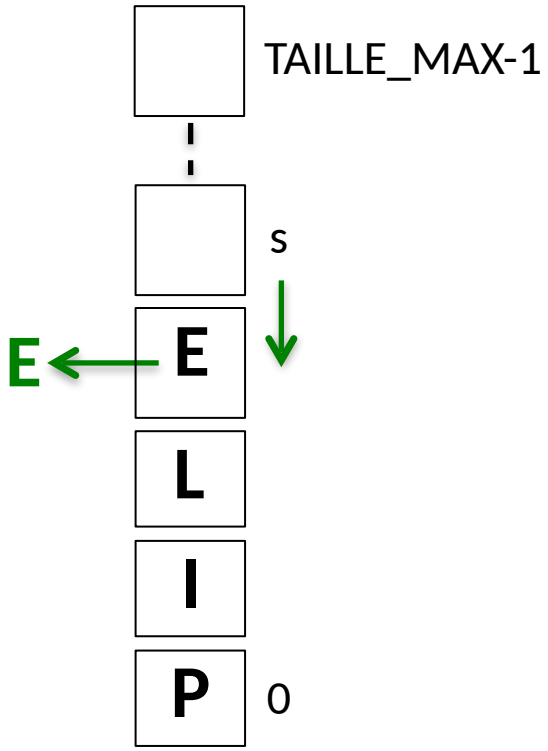
Pile
Implantation par tableau (2)

```
char depiler(struct pile *p){
      ???
}
```



Pile
Implantation par tableau (2)

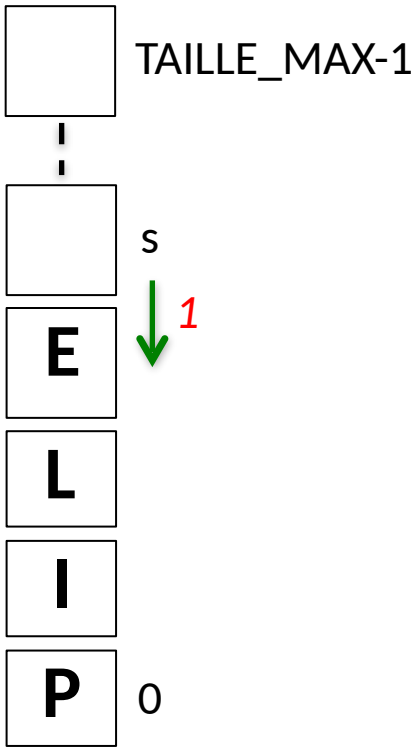
```
char depiler(struct pile *p){
      ???
}
```



Pile
Implantation par tableau (2)

```
char depiler(struct pile *p){
                              --p->s
}
```

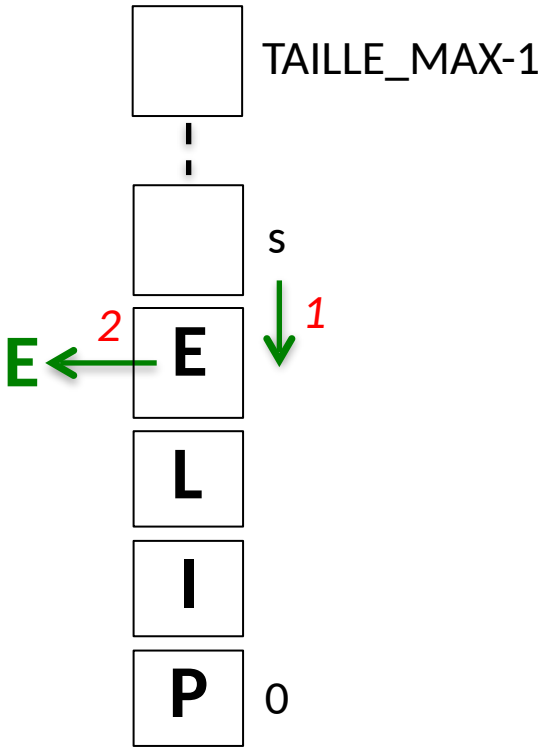
1



Pile
Implantation par tableau (2)

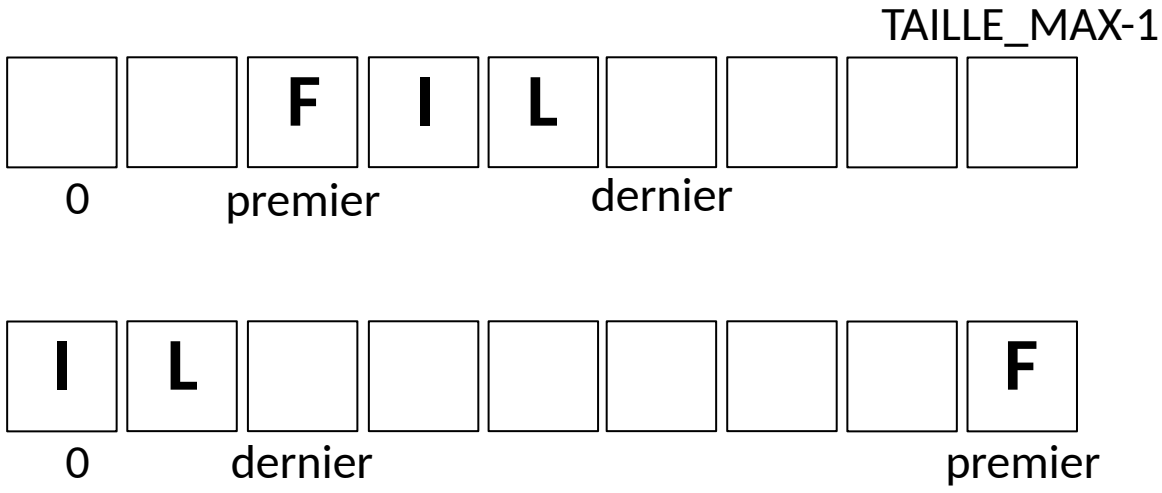
```
char depiler(struct pile *p){
      return p->valeurs[--p->s];
}
```

2 *1*



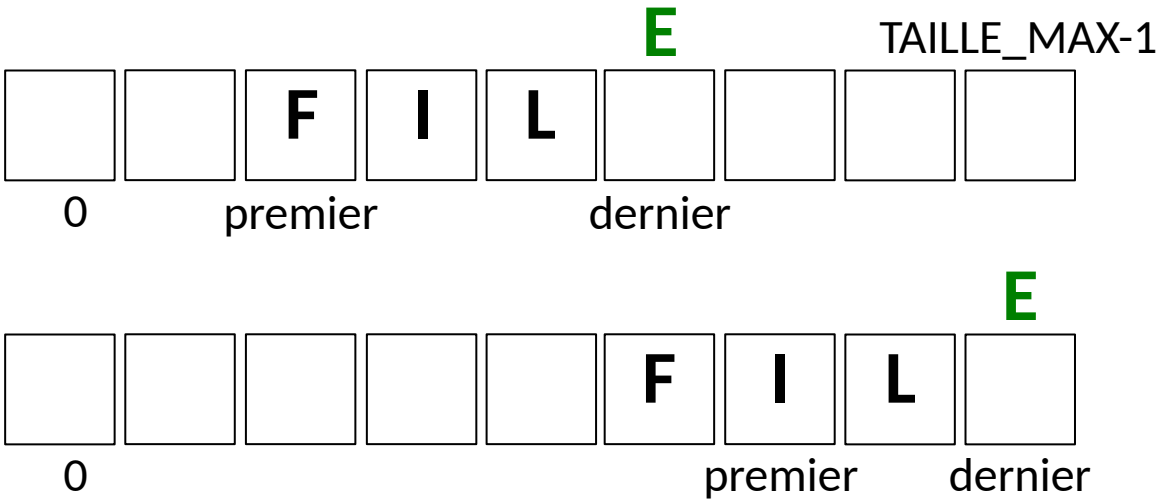
File
Implantation par tableau circulaire

```
struct file {
      int premier; // indice du premier
      int dernier; // indice du dernier + 1
      char valeurs[TAILLE_MAX];
}; // file vide : premier = dernier
```



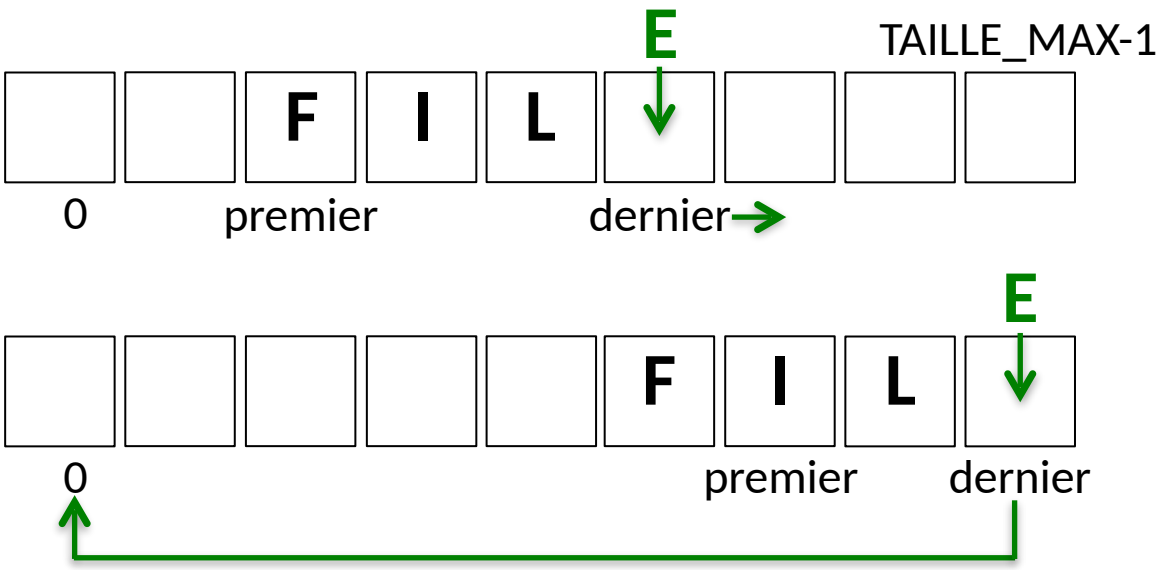
File
Implantation par tableau circulaire

```
void enfiler(struct file *f, char val) {

      ???

}
```



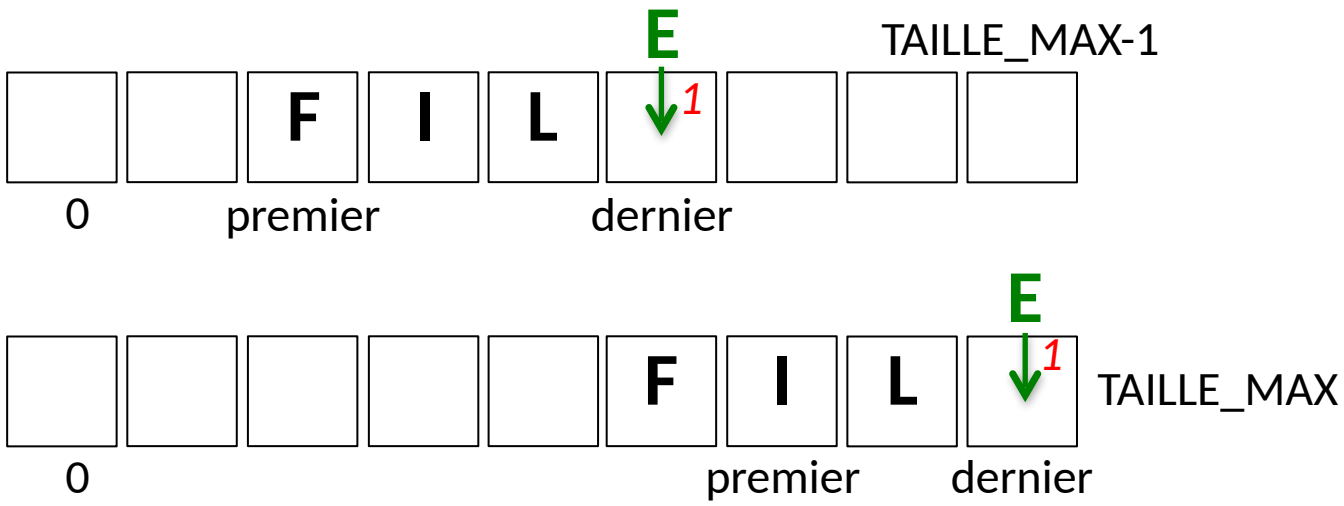
File
Implantation par tableau circulaire

```
void enfiler(struct file *f, char val) {

      ???

}
```



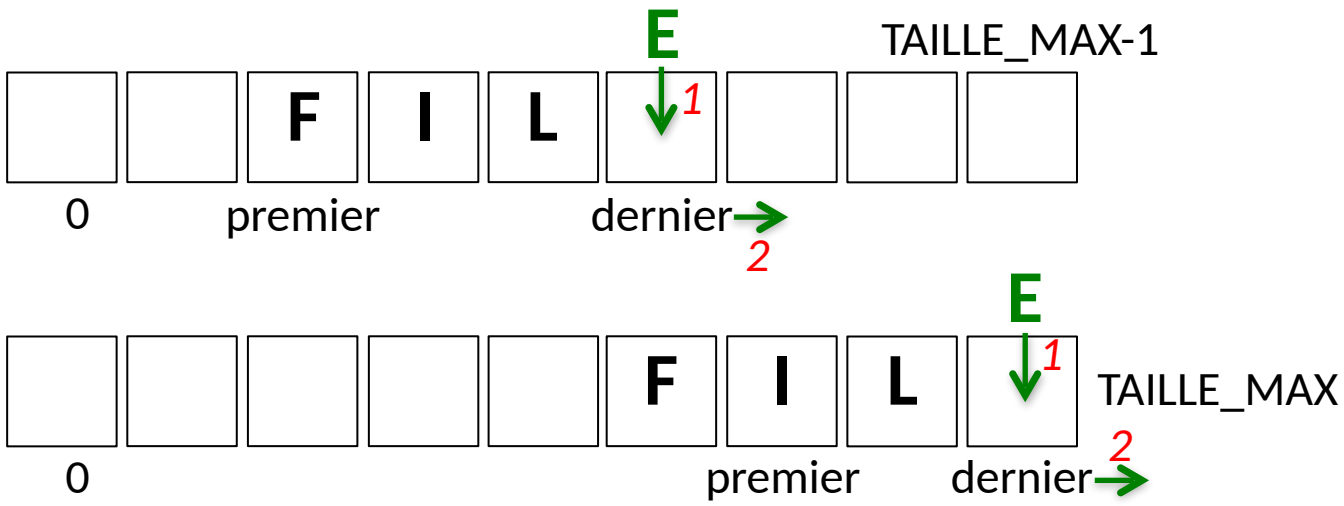
File
Implantation par tableau circulaire

```
void enfiler(struct file *f, char val) {
    f->valeurs[f->dernier  ] = val;
                            1


}
```



File
Implantation par tableau circulaire

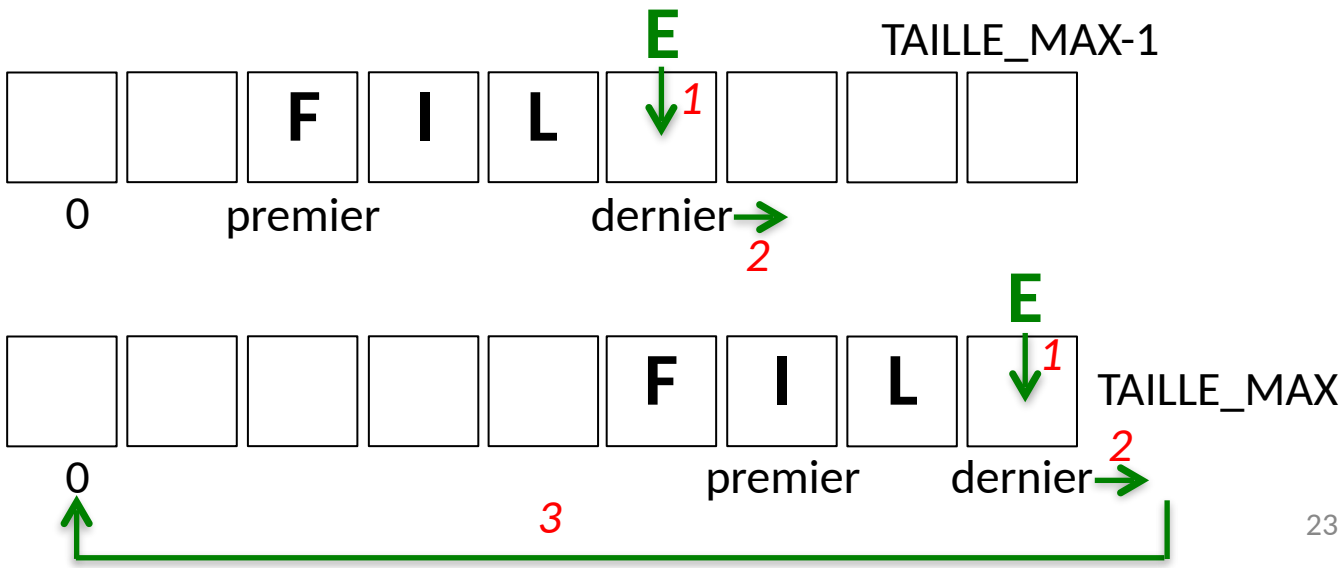
```
void enfiler(struct file *f, char val) {
     f->valeurs[f->dernier++] = val;
                              2     1


}
```



File
Implantation par tableau circulaire

```
void enfiler(struct file *f, char val) {
     f->valeurs[f->dernier++] = val;
     if (f->dernier == TAILLE_MAX)
        f->dernier = 0;
}
```

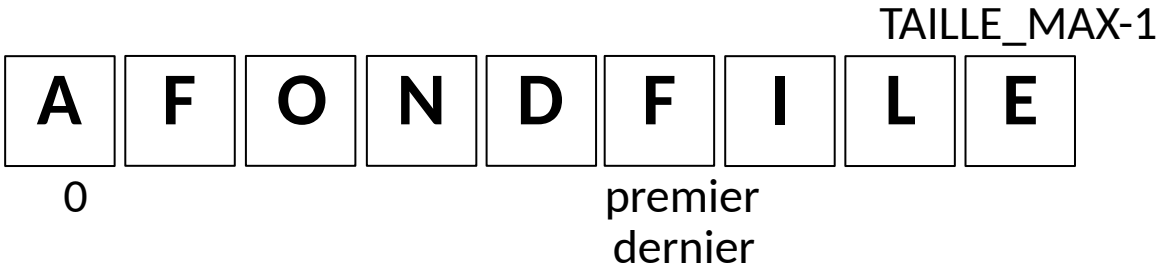
2 1 3



File
Implantation par tableau circulaire

File pleine ?

Si l'on utilise tout le tableau : premier = dernier, comme pour une file vide !

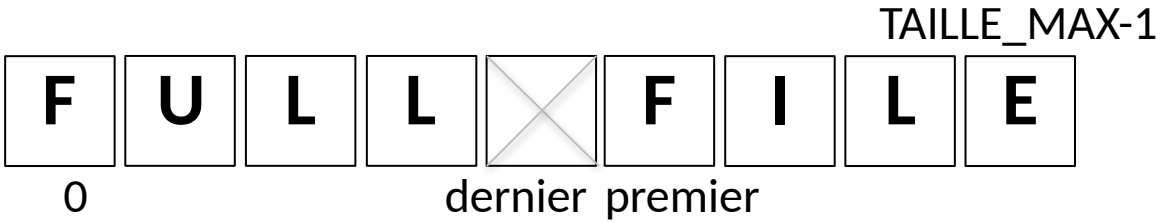


File
Implantation par tableau circulaire

File pleine ?

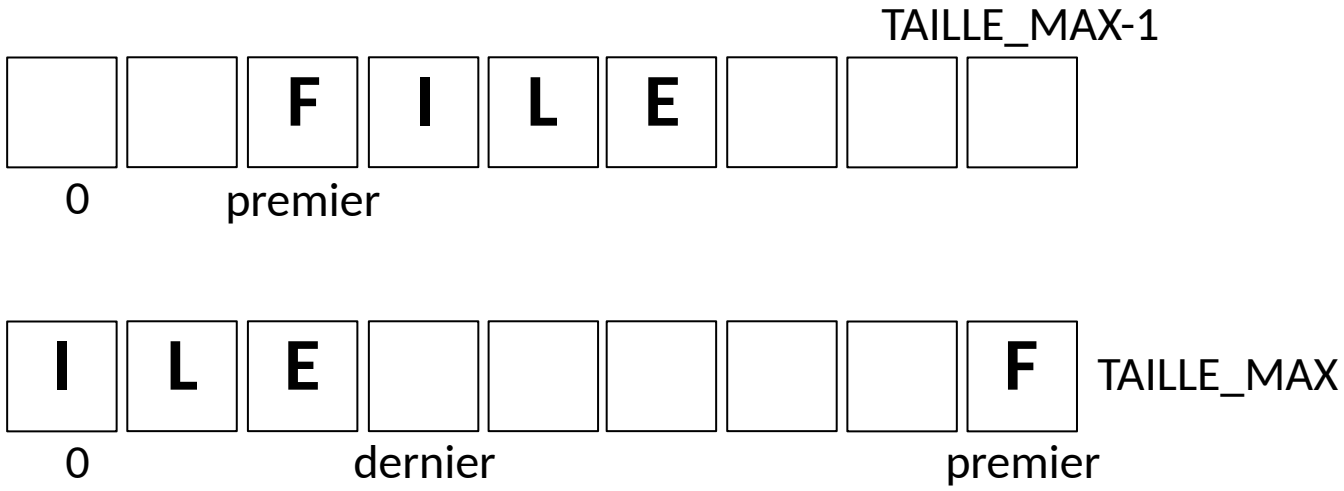
=> On conserve une case vide entre dernier et premier : la file ne contiendra donc pas plus de TAILLE_MAX - 1 éléments.

```
dernier == TAILLE_MAX - 1 && premier == 0
|| (dernier + 1) == premier
```



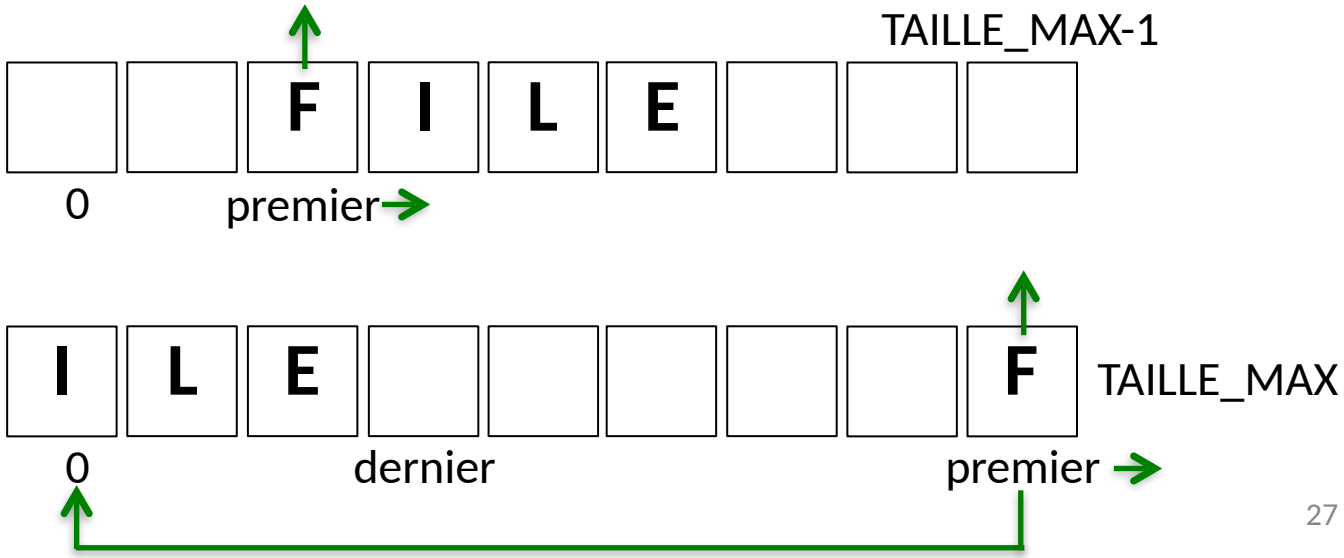
File
Implantation par tableau circulaire

```
char defiler(struct file *f) {

      ???


}
```



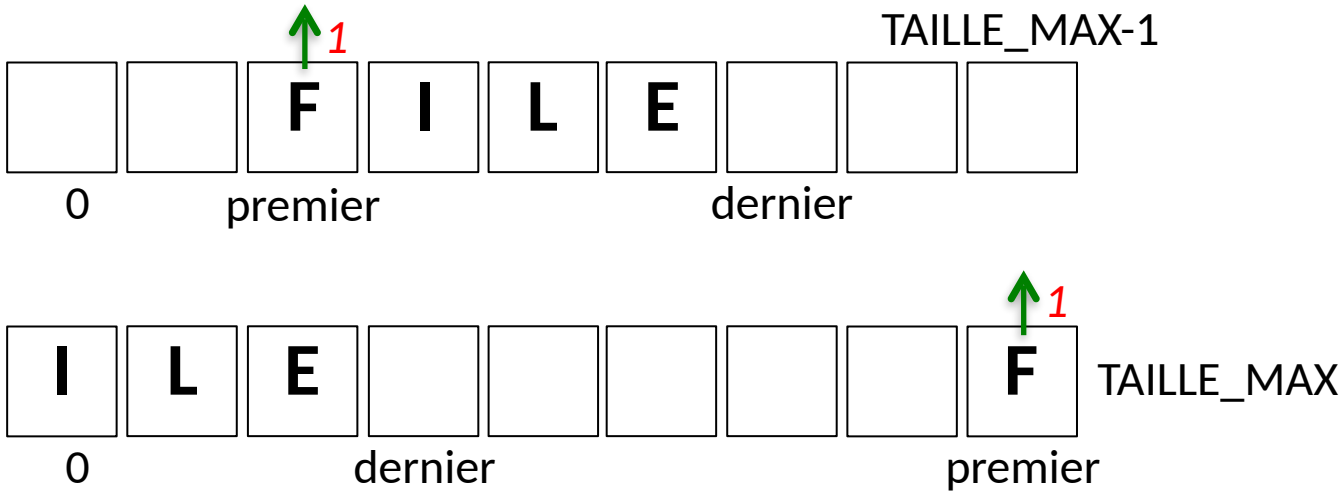
File
Implantation par tableau circulaire

```
char defiler(struct file *f) {

      ???


}
```



File
Implantation par tableau circulaire

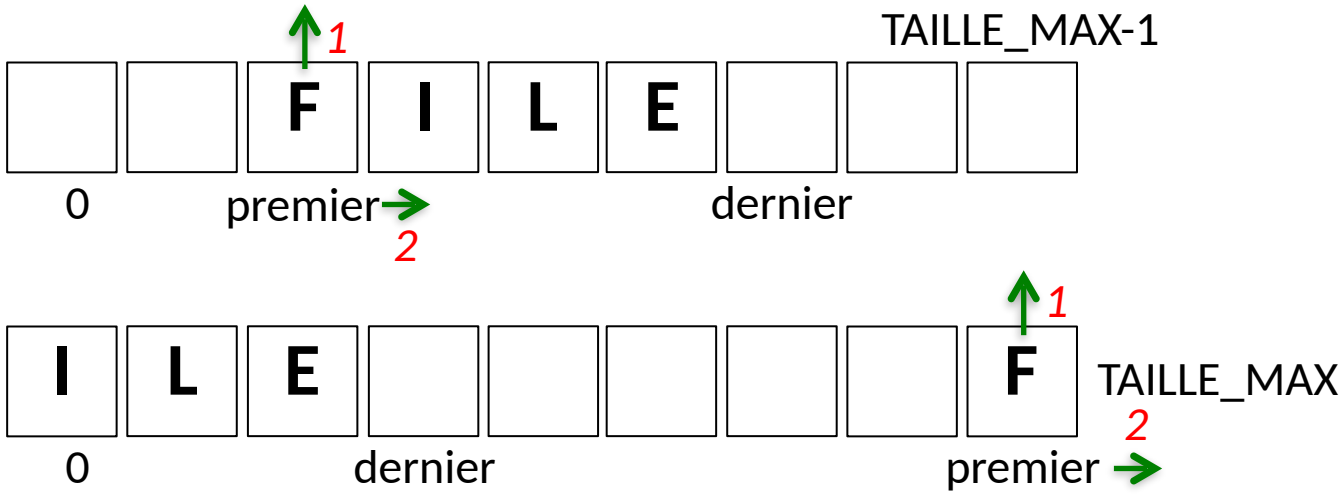
```
char defiler(struct file *f) {
      char v = f->valeurs[f->premier  ];
                1
}
```



File
Implantation par tableau circulaire

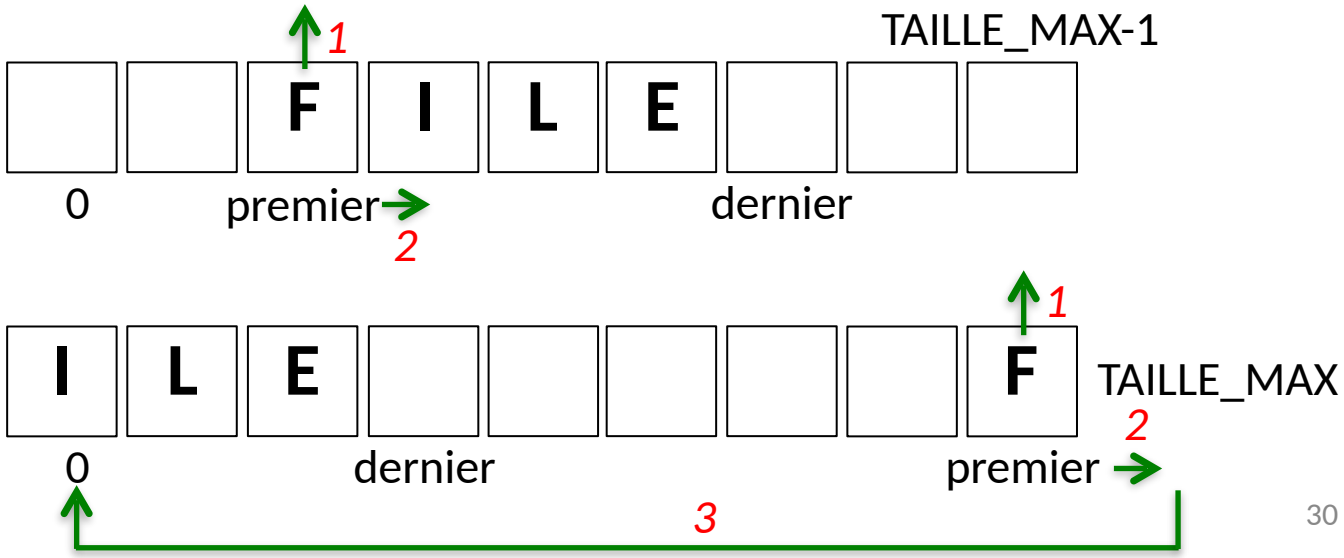
```
char defiler(struct file *f) {
      char v = f->valeurs[f->premier++];
}
```

1 2



File
Implantation par tableau circulaire

```
char defiler(struct file *f) {
      char v = f->valeurs[f->premier++];
      if (f->premier == TAILLE_MAX)
            f->premier = 0;
      return v;
}
```



Exemples d’utilisation

Notation polonaise inversée

Notation standard : 2 * (3 * 4 * 5 + 6 * 7)

Dite infixée (opérateur au milieu des opérandes)

Notation polonaise inversée : 2 3 4 5 * * 6 7 * + *

Dite postfixée (opérateur après les opérandes)

Notation polonaise inversée
Evaluation

Notation standard : 2 * (3 * 4 * 5 + 6 * 7) = 204

Notation polonaise inversée : 2 3 4 5 * * 6 7 * + *

2 3 20 * 6 7 * + *

2 3 20 * 6 7 * + *

2 60 6 7 * + *

2 60 6 7 * + *

2 60 42 + *

2 60 42 + *

2 102 *

2 102 *

204

=> utilisation d'une pile

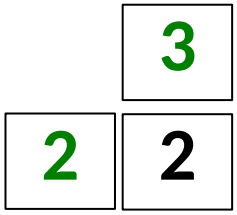
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *

2

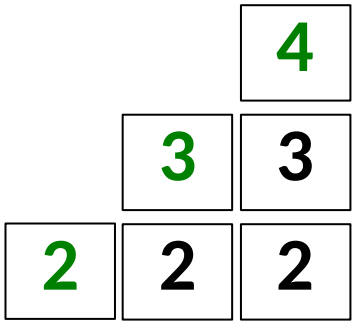
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



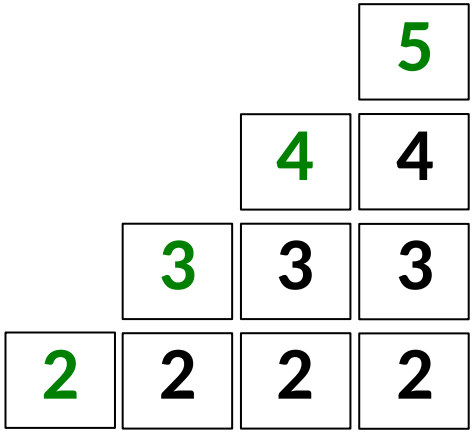
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



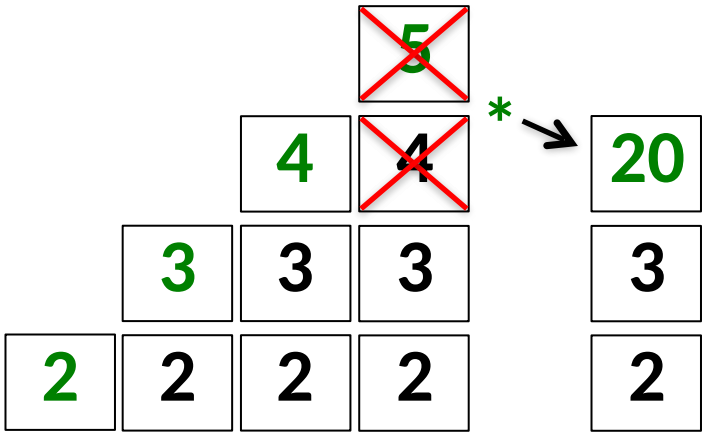
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



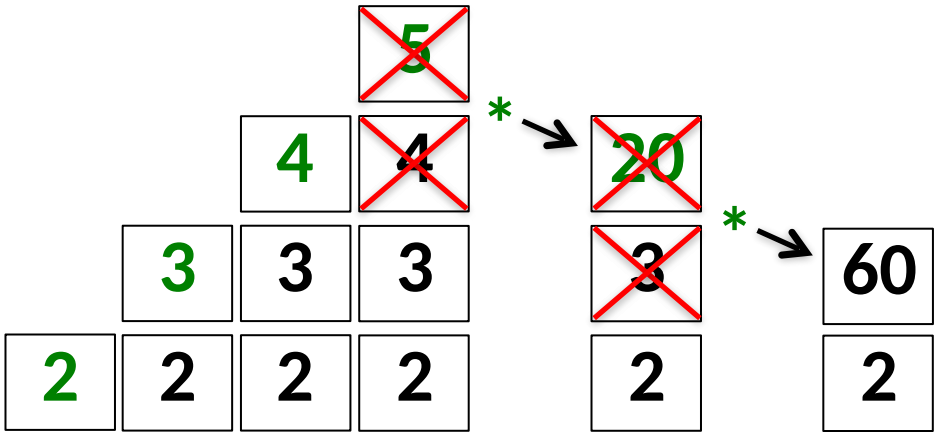
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



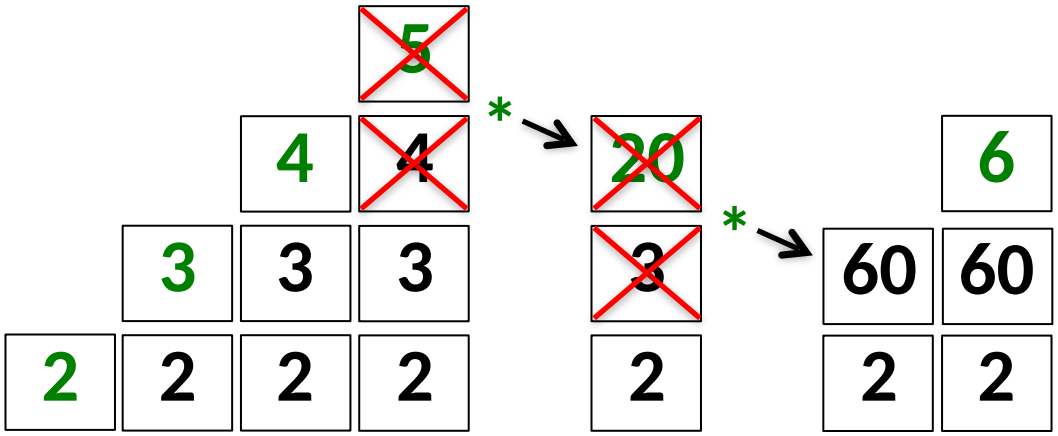
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



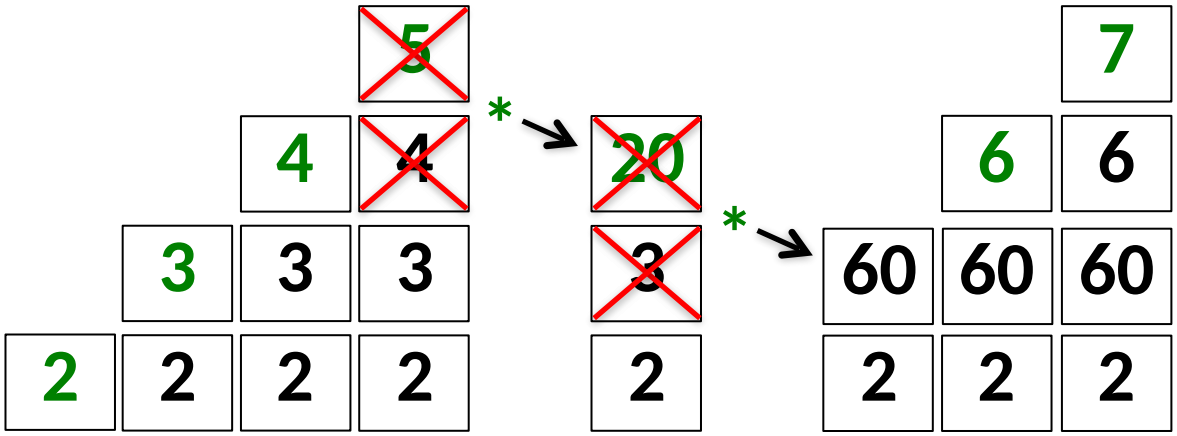
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



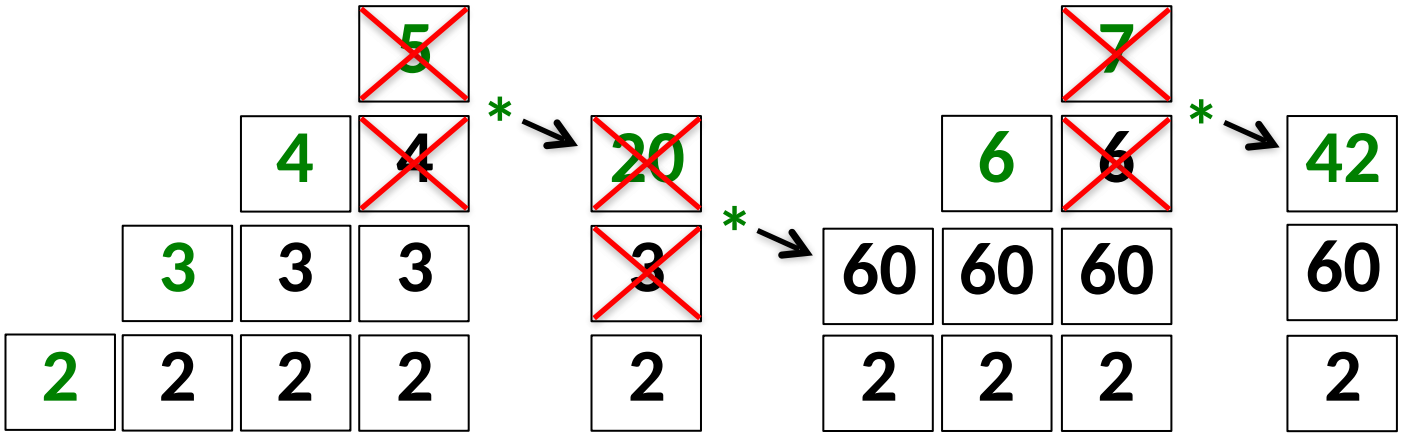
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



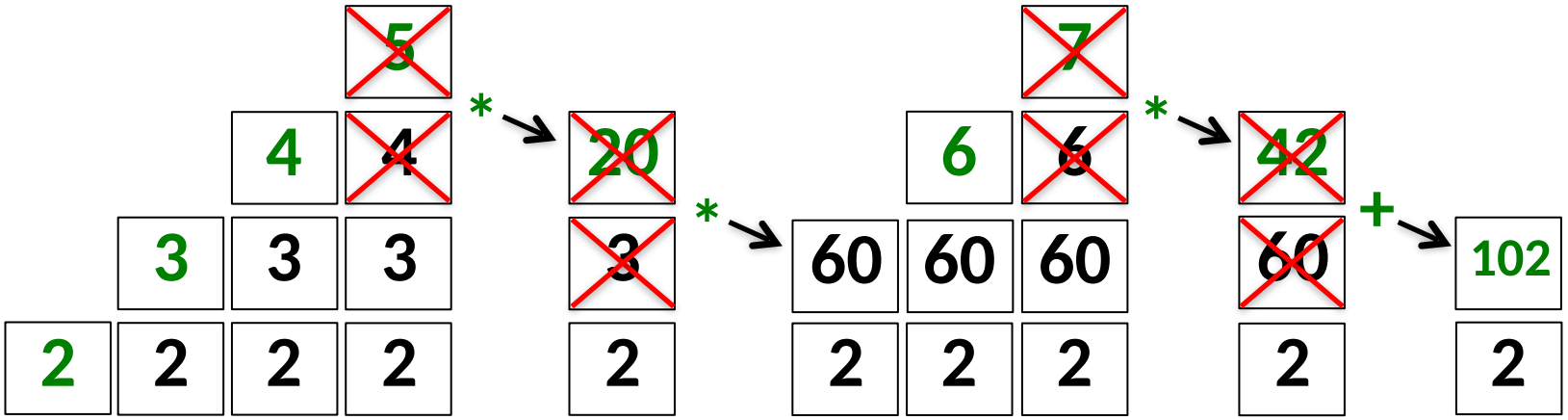
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



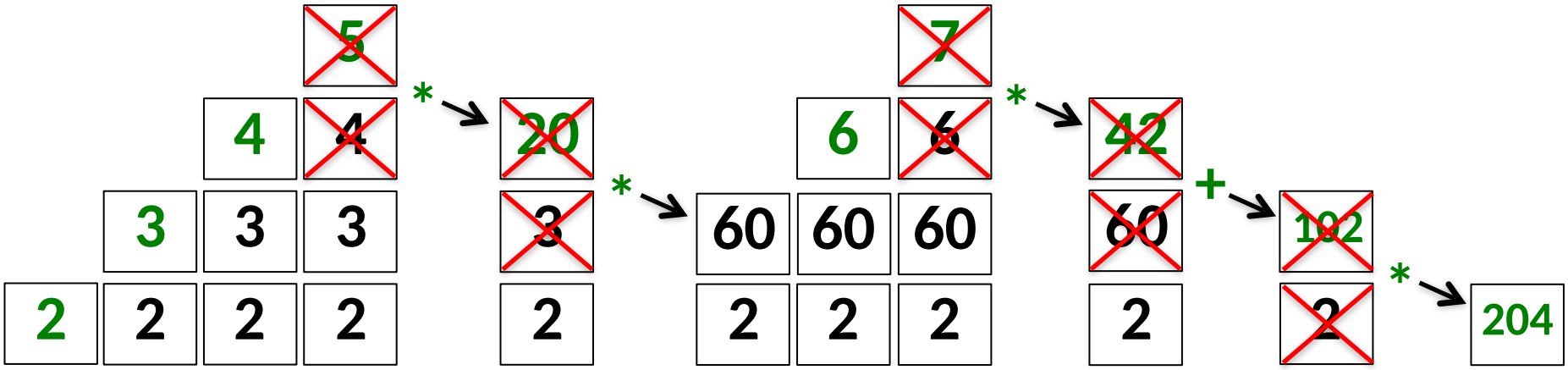
Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



Notation polonaise inversée
Evaluation

2 3 4 5 * * 6 7 * + *



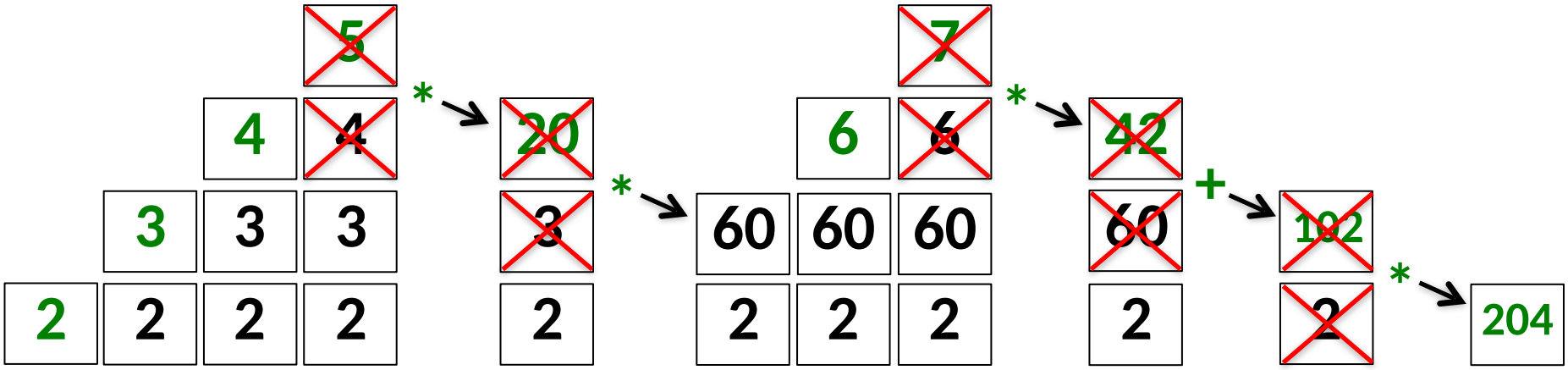
Notation polonaise inversée
Evaluation

Pour chaque caractère c

si c est un nombre empiler (c)

si c vaut '+' empiler(depiler() + depiler())

si c vaut '' empiler(depiler() * depiler())*



Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)

3

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)

2

Un nombre : on enfile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)

2

*

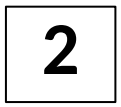
Un opérateur et pile vide : on empile

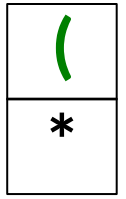
Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)





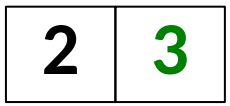
Une parenthèse ouvrante : on empile

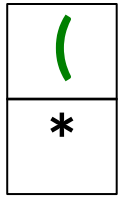
Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)





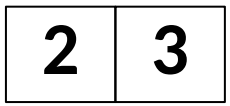
Un nombre : on enfile

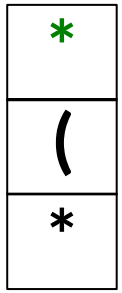
Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)





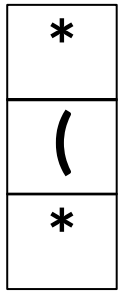
Un opérateur sans opérateur au sommet de la pile : on empile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



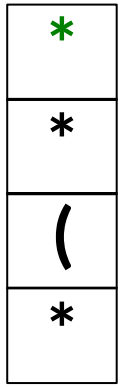
Un nombre : on enfile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



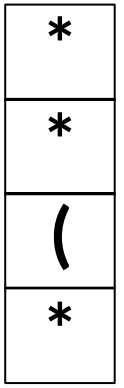
Un opérateur de priorité supérieure ou égale à celle du sommet : on empile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



2	3	4	5

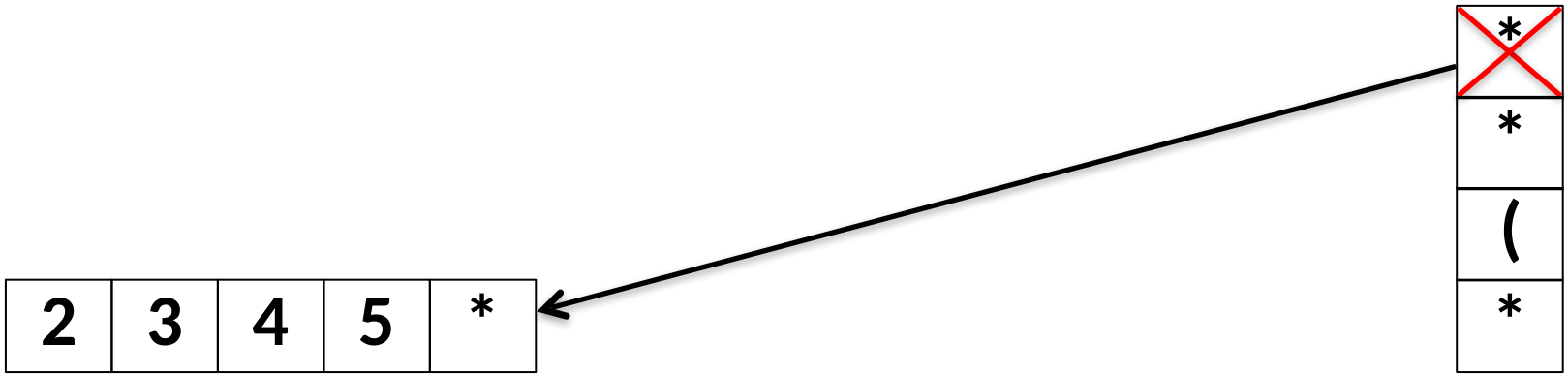
Un nombre : on enfile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



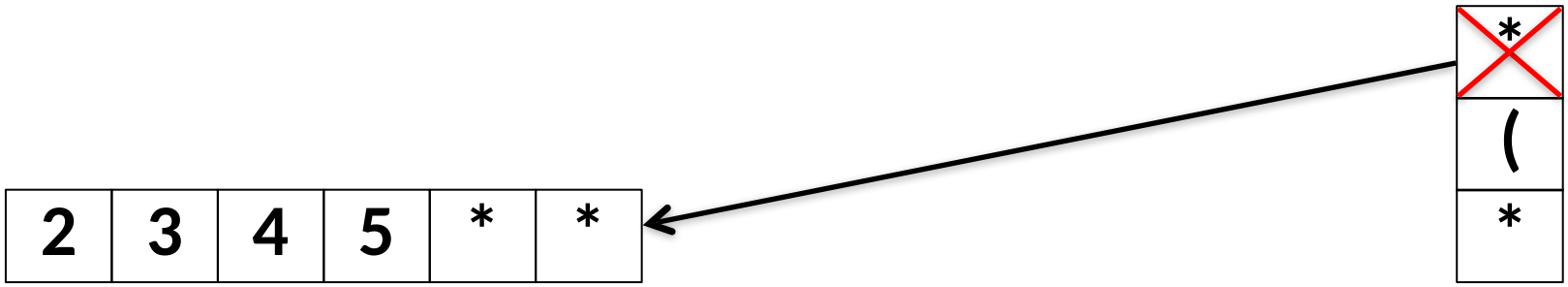
Un opérateur de priorité inférieure à celle du sommet : on dépile dans la file

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



Un opérateur de priorité inférieure à celle du sommet : on dépile dans la file

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)

2	3	4	5	*	*

+
(
*

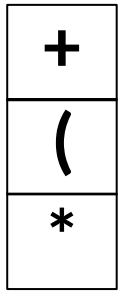
Un opérateur avec le sommet qui n'est pas un opérateur : on empile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



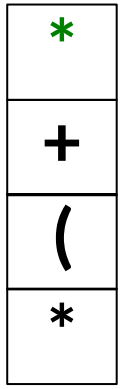
Un nombre : on enfile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



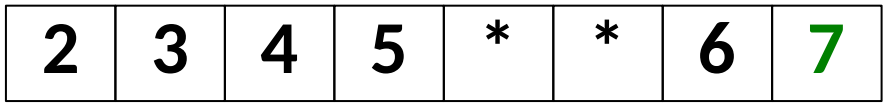
Un opérateur de priorité supérieure ou égale à celle du sommet : on empile

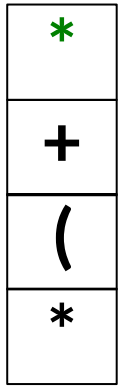
Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)





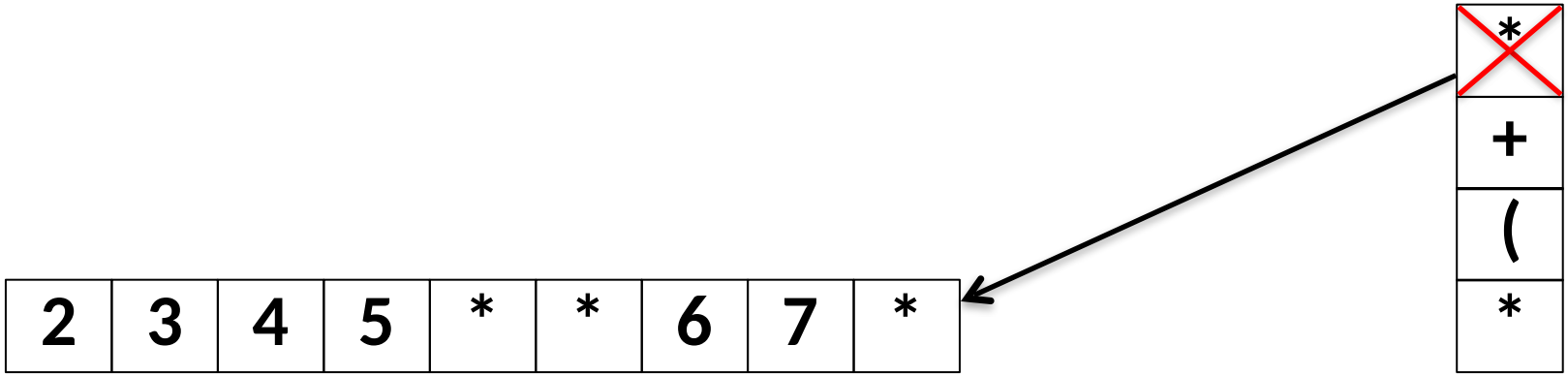
Un nombre : on enfile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



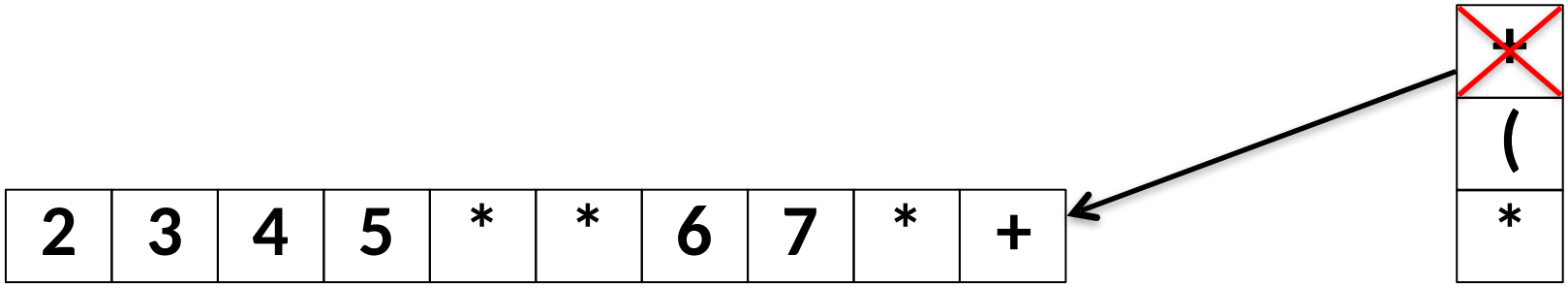
Une parenthèse fermante : on dépile pour enfiler jusqu'à la prochaine parenthèse ouvrante que l'on dépile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)



Une parenthèse fermante : on dépile pour enfiler jusqu'à la prochaine parenthèse ouvrante que l'on dépile

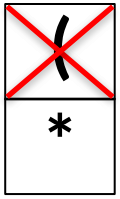
Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7)

2	3	4	5	*	*	6	7	*	+



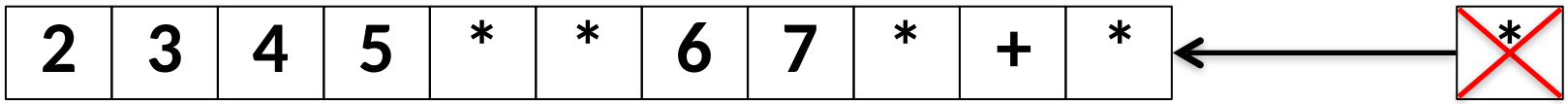
Une parenthèse fermante : on dépile pour enfiler jusqu'à la prochaine parenthèse ouvrante que l'on dépile

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

2 * (3 * 4 * 5 + 6 * 7) fin



Fin de l'expression : on dépile tout dans la file

Notation polonaise inversée
Traduction

Traduction notation standard en notation polonaise inversée.

Pile intermédiaire, file pour le résultat

Pour chaque caractère c

si c est un nombre alors enfiler(c)

si c est une parenthèse ouvrante alors empiler(c)

si c est un opérateur alors

Tant que !pileVide() et sommet est un opérateur prioritaire sur c faire

enfiler(depiler())

empiler(c)

si c est une parenthèse fermante alors

Tant que sommet() != parenthèse ouvrante faire enfiler(depiler())

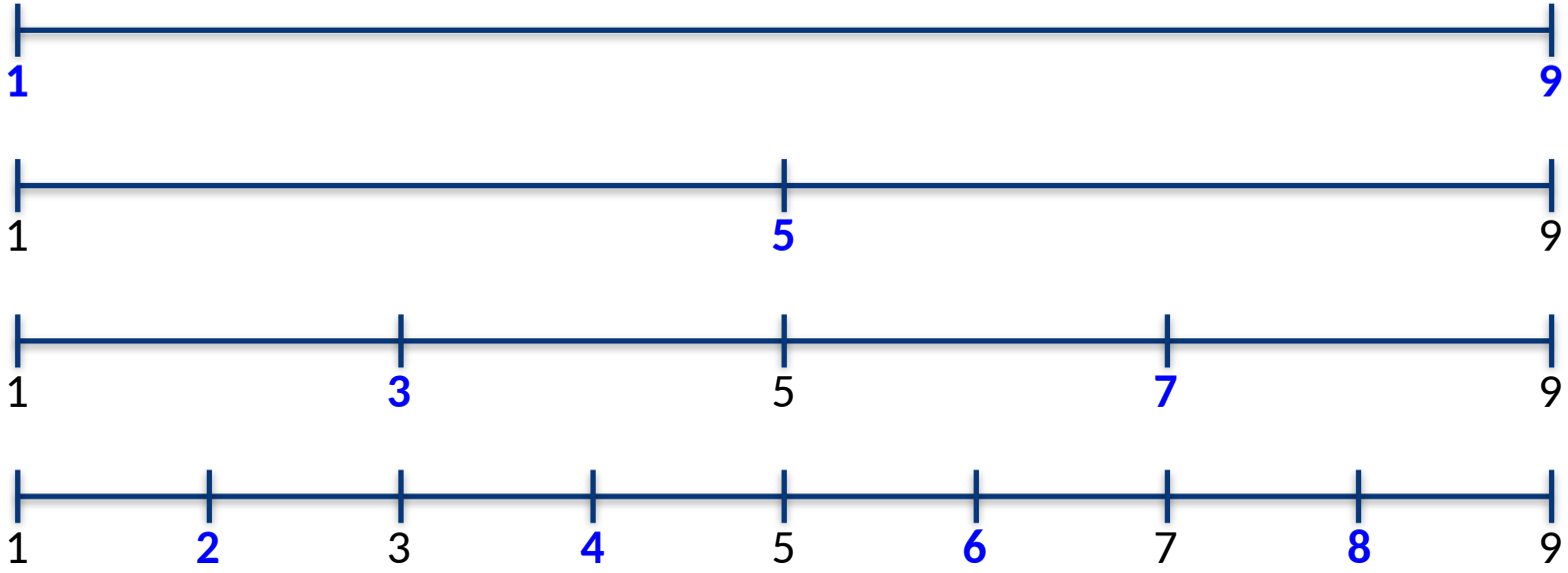
depiler()

Fin pour

Tant que !pileVide() faire enfiler(depiler())

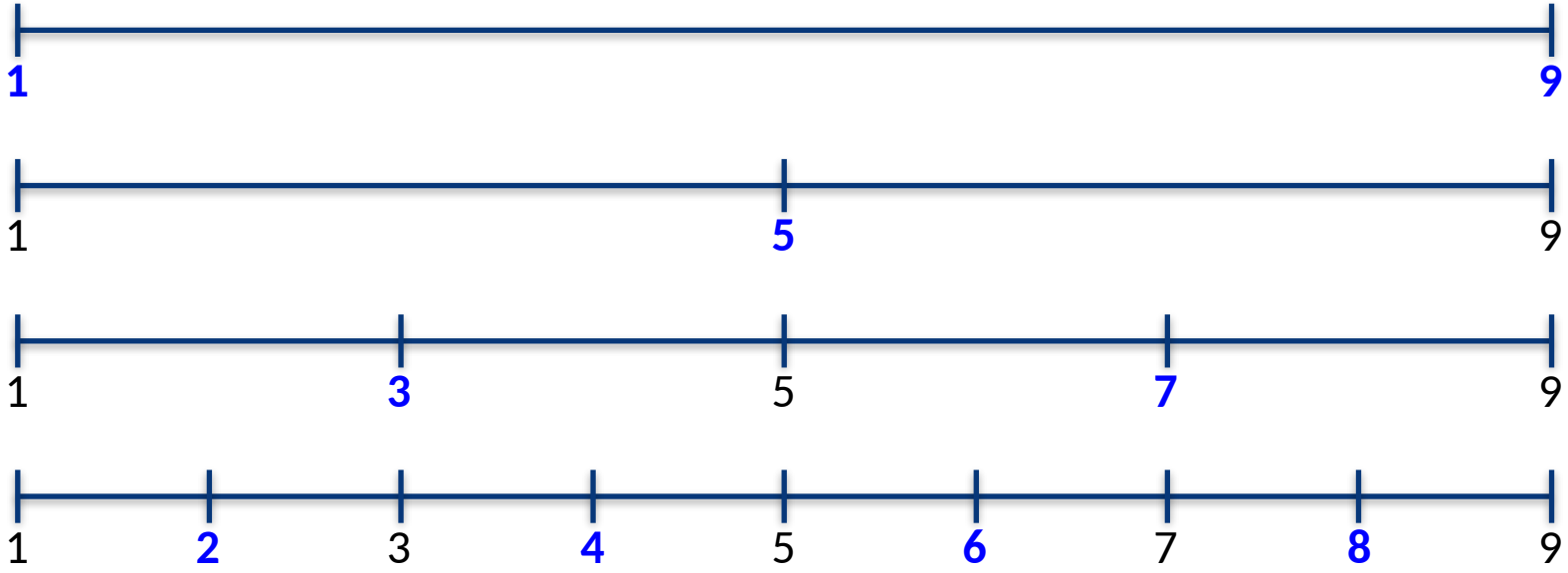
Parcours dichotomique

- Exemple : afficher des graduations selon la place disponible, les textes de deux graduations consécutives devant être séparées d'une distance minimale.
- Pour des graduation allant de 1 à 9 inclus



Parcours dichotomique

- Exemple : afficher des graduations selon la place disponible, les textes de deux graduations consécutives devant être séparées d'une distance minimale.
- Pour des graduation allant de 1 à 9 inclus : 1 9 5 3 7 2 4 6 8

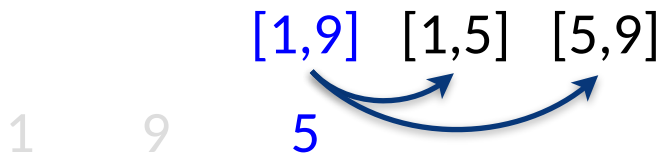


Parcours dichotomique

- Algorithme ?
-

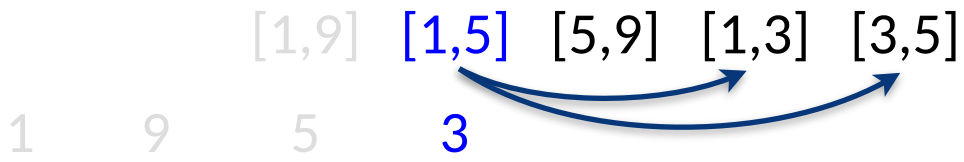
Parcours dichotomique

- Algorithme ?
- On part avec [1, 9] : on prend le milieu **5**
- On traite ensuite dans l'ordre [1, 5] puis [5, 9]



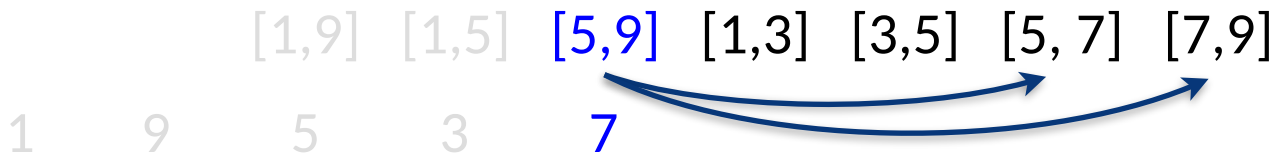
Parcours dichotomique

- Algorithme ?
- On part avec [1, 9] : on prend le milieu **5**
- On traite ensuite dans l'ordre [1, 5] puis [5, 9]
- Pour [1, 5] on prend le milieu **3** et on devra ensuite traiter [1, 3] et [3, 5] mais après [5, 9]



Parcours dichotomique

- Algorithme ?
- On part avec [1, 9] : on prend le milieu **5**
- On traite ensuite dans l'ordre [1, 5] puis [5, 9]
- Pour [1, 5] on prend le milieu **3** et on devra ensuite traiter [1, 3] et [3, 5] mais après [5, 9]
- Pour [5, 9] on prend le milieu **7** et on devra ensuite traiter [5, 7] et [7, 9], après tout le reste



Parcours dichotomique

- Algorithme ?
- On part avec [1, 9] : on prend le milieu **5**
- On traite ensuite dans l'ordre [1, 5] puis [5, 9]
- Pour [1, 5] on prend le milieu **3** et on devra ensuite traiter [1, 3] et [3, 5] mais après [5, 9]
- Pour [5, 9] on prend le milieu **7** et on devra ensuite traiter [5, 7] et [7, 9], après tout le reste
- On traite les suivants

[1,9] [1,5] [5,9] [1,3] [3,5] [5,7] [7,9]

1 9 5 3 7 2

Parcours dichotomique

- Algorithme ?
- On part avec [1, 9] : on prend le milieu **5**
- On traite ensuite dans l'ordre [1, 5] puis [5, 9]
- Pour [1, 5] on prend le milieu **3** et on devra ensuite traiter [1, 3] et [3, 5] mais après [5, 9]
- Pour [5, 9] on prend le milieu **7** et on devra ensuite traiter [5, 7] et [7, 9], après tout le reste
- On traite les suivants

[1,9] [1,5] [5,9] [1,3] [3,5] [5,7] [7,9]

1 9 5 3 7 2 4

Parcours dichotomique

- Algorithme ?
- On part avec [1, 9] : on prend le milieu **5**
- On traite ensuite dans l'ordre [1, 5] puis [5, 9]
- Pour [1, 5] on prend le milieu **3** et on devra ensuite traiter [1, 3] et [3, 5] mais après [5, 9]
- Pour [5, 9] on prend le milieu **7** et on devra ensuite traiter [5, 7] et [7, 9], après tout le reste
- On traite les suivants

[1,9] [1,5] [5,9] [1,3] [3,5] [5,7] [7,9]

1 9 5 3 7 2 4 6

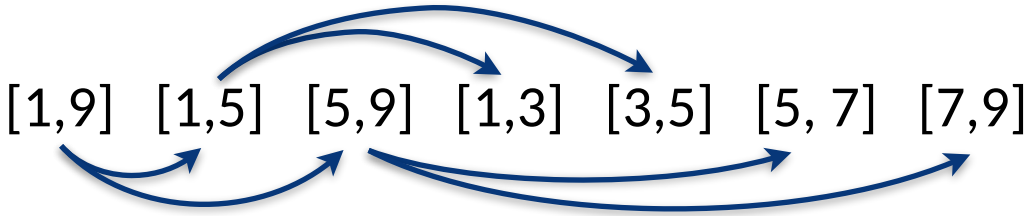
Parcours dichotomique

- Algorithme ?
- On part avec [1, 9] : on prend le milieu **5**
- On traite ensuite dans l'ordre [1, 5] puis [5, 9]
- Pour [1, 5] on prend le milieu **3** et on devra ensuite traiter [1, 3] et [3, 5] mais après [5, 9]
- Pour [5, 9] on prend le milieu **7** et on devra ensuite traiter [5, 7] et [7, 9], après tout le reste
- On traite les suivants

[1,9]	[1,5]	[5,9]	[1,3]	[3,5]	[5,7]	[7,9]
1	9	5	3	7	2	4	6	8

Parcours dichotomique

- Algorithme ?
- On part avec [1, 9] : on prend le milieu **5**
- On traite ensuite dans l'ordre [1, 5] puis [5, 9]
- Pour [1, 5] on prend le milieu **3** et on devra ensuite traiter [1, 3] et [3, 5] mais après [5, 9]
- Pour [5, 9] on prend le milieu **7** et on devra ensuite traiter [5, 7] et [7, 9], après tout le reste
- On traite donc les intervalles dans l'ordre suivant :



Parcours dichotomique

- Utilisation d’une file

Parcours dichotomique - File

```
void parcoursFile(int d, int f) {
  struct file *fi = creerFile();
  enfiler(fi, d); enfiler(fi, f);
  int m;
  while (!estVide(fi)) {
    m = ((d = defiler(fi)) + (f = defiler(fi))) / 2;
    printf("%d ", m);
    if (m – d > 1) {enfiler(fi, d); enfiler(fi, m);}
    if (f – m > 1) {enfiler(fi, m); enfiler(fi, f);}
  }
}
```

1 9

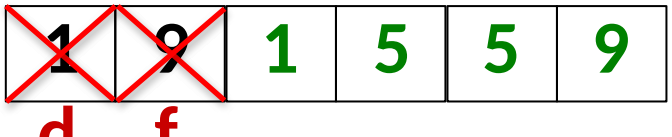
1	9

Parcours dichotomique - File

```
void parcoursFile(int d, int f) {
  struct file *fi = creerFile();
  enfiler(fi, d); enfiler(fi, f);
  int m;
  while (!estVideFile(fi)) {
    m = ((d = defiler(fi)) + (f = defiler(fi))) / 2;
    printf("%d ", m);
    if (m – d > 1) {enfiler(fi, d); enfiler(fi, m);}
    if (f – m > 1) {enfiler(fi, m); enfiler(fi, f);}
  }
}
```

m

1 9 5

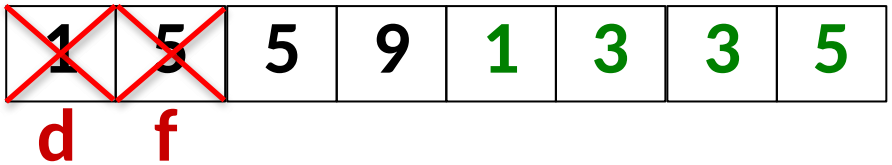


Parcours dichotomique - File

```
void parcoursFile(int d, int f) {
  struct file *fi = creerFile();
  enfiler(fi, d); enfiler(fi, f);
  int m;
  while (!estVideFile(fi)) {
    m = ((d = defiler(fi)) + (f = defiler(fi))) / 2;
    printf("%d ", m);
    if (m – d > 1) {enfiler(fi, d); enfiler(fi, m);}
    if (f – m > 1) {enfiler(fi, m); enfiler(fi, f);}
  }
}
```

m

1 9 5 3

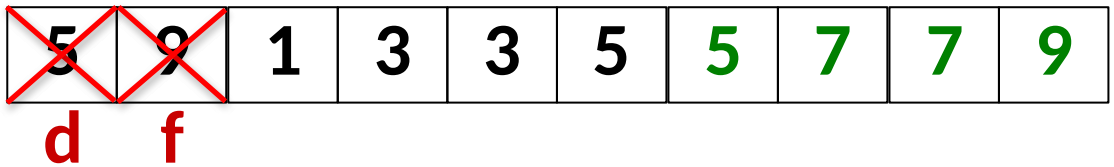


Parcours dichotomique - File

```
void parcoursFile(int d, int f) {
  struct file *fi = creerFile();
  enfiler(fi, d); enfiler(fi, f);
  int m;
  while (!estVideFile(fi)) {
    m = ((d = defiler(fi)) + (f = defiler(fi))) / 2;
    printf("%d ", m);
    if (m – d > 1) {enfiler(fi, d); enfiler(fi, m);}
    if (f – m > 1) {enfiler(fi, m); enfiler(fi, f);}
  }
}
```

m

1 9 5 3 7

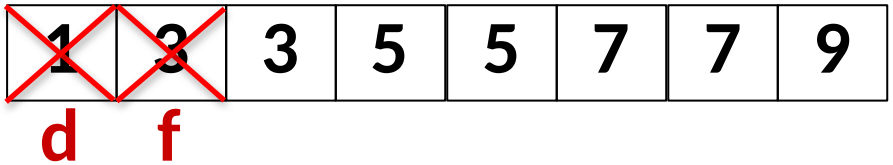


Parcours dichotomique - File

```
void parcoursFile(int d, int f) {
  struct file *fi = creerFile();
  enfiler(fi, d); enfiler(fi, f);
  int m;
  while (!estVideFile(fi)) {
    m = ((d = defiler(fi)) + (f = defiler(fi))) / 2;
    printf("%d ", m);
    if (m – d > 1) {enfiler(fi, d); enfiler(fi, m);}
    if (f – m > 1) {enfiler(fi, m); enfiler(fi, f);}
  }
}
```

m

1 9 5 3 7 2

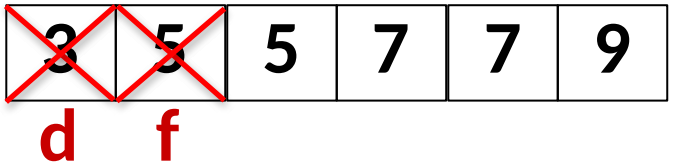


Parcours dichotomique - File

```
void parcoursFile(int d, int f) {
  struct file *fi = creerFile();
  enfiler(fi, d); enfiler(fi, f);
  int m;
  while (!estVideFile(fi)) {
    m = ((d = defiler(fi)) + (f = defiler(fi))) / 2;
    printf("%d ", m);
    if (m – d > 1) {enfiler(fi, d); enfiler(fi, m);}
    if (f – m > 1) {enfiler(fi, m); enfiler(fi, f);}
  }
}
```

m

1 9 5 3 7 2 4

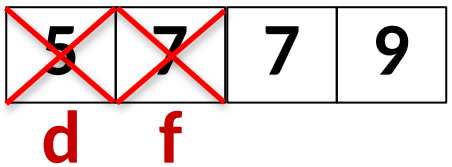


Parcours dichotomique - File

```
void parcoursFile(int d, int f) {
  struct file *fi = creerFile();
  enfiler(fi, d); enfiler(fi, f);
  int m;
  while (!estVideFile(fi)) {
    m = ((d = defiler(fi)) + (f = defiler(fi))) / 2;
    printf("%d ", m);
    if (m – d > 1) {enfiler(fi, d); enfiler(fi, m);}
    if (f – m > 1) {enfiler(fi, m); enfiler(fi, f);}
  }
}
```

m

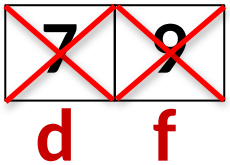
1 9 5 3 7 2 4 6



```
void parcoursFile(int d, int f) {
  struct file *fi = creerFile();
  enfiler(fi, d); enfiler(fi, f);
  int m;
  while (!estVideFile(fi)) {
    m = ((d = defiler(fi)) + (f = defiler(fi))) / 2;
    printf("%d ", m);
    if (m – d > 1) {enfiler(fi, d); enfiler(fi, m);}
    if (f – m > 1) {enfiler(fi, m); enfiler(fi, f);}
  }
}
```

m

1 9 5 3 7 2 4 6 8



Parcours dichotomique

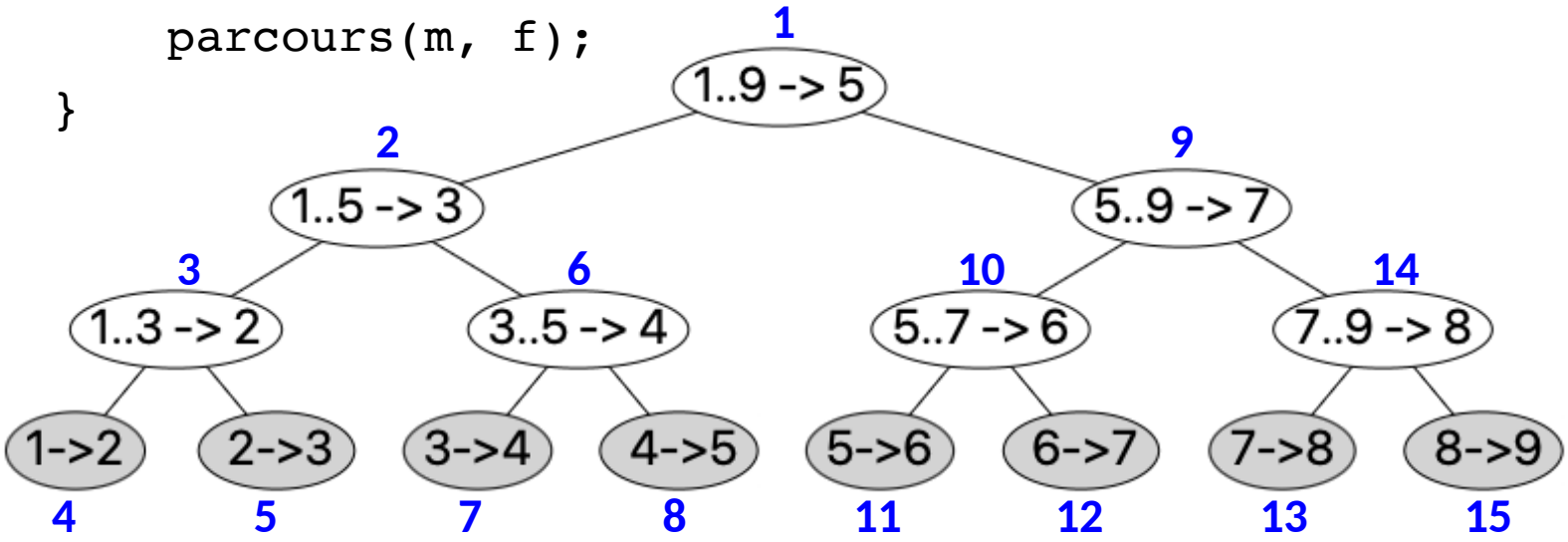
- Approche récursive

```
void parcours (int d, int f) {
    if (f – d > 1) {
        int m = (d + f) / 2;
        printf("%d ", m);
        parcours(d, m);
        parcours(m, f);
    }
}
…
printf("1 9 "); parcours(1, 9); ?
```

Parcours dichotomique

- Approche récursive

```
void parcours (int d, int f) {
    if (f – d > 1) {
        int m = (d + f) / 2;
        printf("%d ", m);
        parcours(d, m);
        parcours(m, f);
    }
}
```



Parcours dichotomique

- Suppression de la récursion ?

Parcours dichotomique

- Suppression de la récursion

  => utilisation d'une pile

Parcours dichotomique - Pile

Approche récursive

```
void parcours(int d, int f){


  if (f – d > 1) {
    int m = (d + f) / 2;
    printf("%d ", m);
    parcours(d, m);
    parcours(m, f);
  }
}
```

Approche itérative

```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

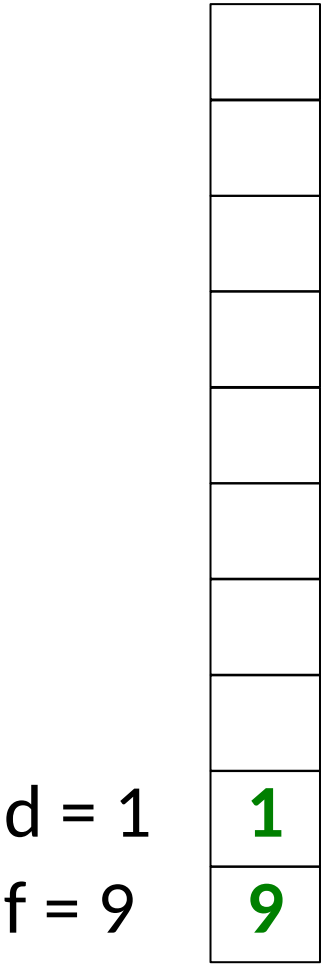
```
printf("1 9 "); parcoursPile(1, 9); ?
```

Parcours dichotomique - Pile

Approche itérative

```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

1 9



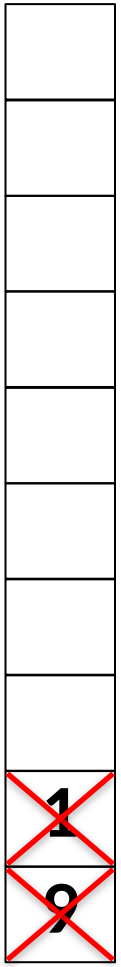
Parcours dichotomique - Pile

Approche itérative

```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5



d = 1

f = 9

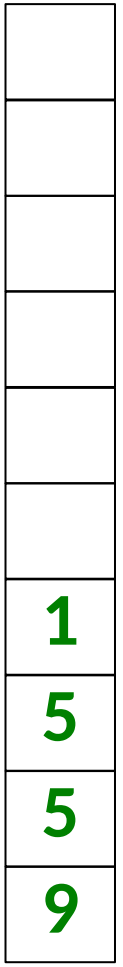
Parcours dichotomique - Pile

Approche itérative

```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5



d = 1

f = 9

Parcours dichotomique - Pile

Approche itérative

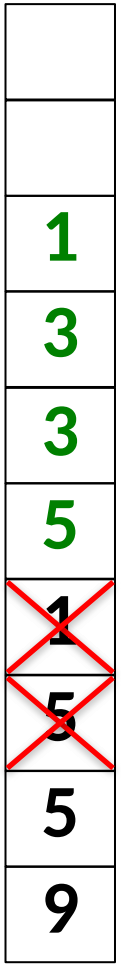
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3

d = 1

f = 5



Parcours dichotomique - Pile

Approche itérative

```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2

d = 1

f = 3

Parcours dichotomique - Pile

Approche itérative

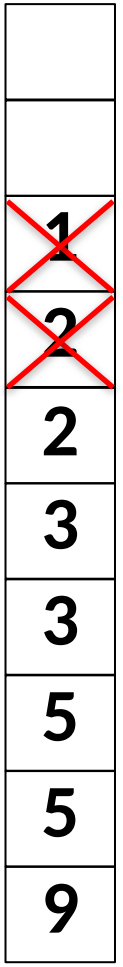
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2

d = 1

f = 2



Parcours dichotomique - Pile

Approche itérative

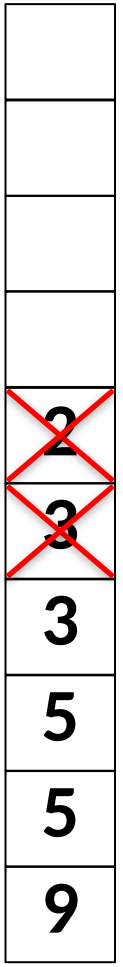
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2

d = 2

f = 3



Parcours dichotomique - Pile

Approche itérative

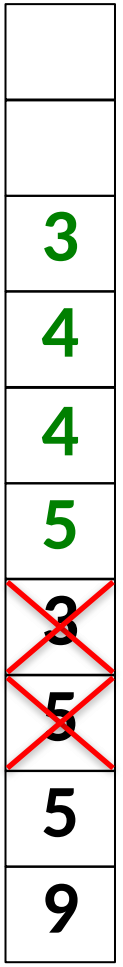
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4

d = 3

f = 5



Parcours dichotomique - Pile

Approche itérative

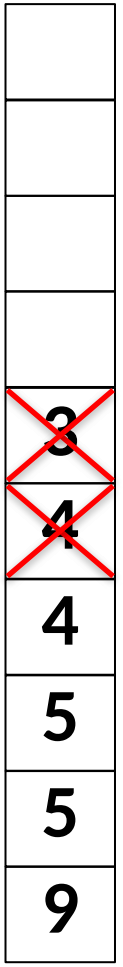
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4

d = 3

f = 4



Parcours dichotomique - Pile

Approche itérative

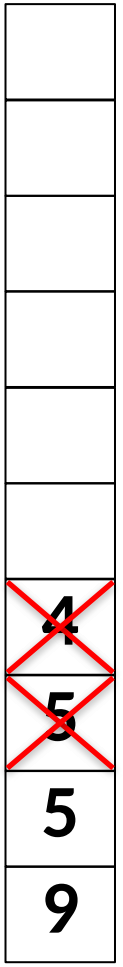
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4

d = 4

f = 5



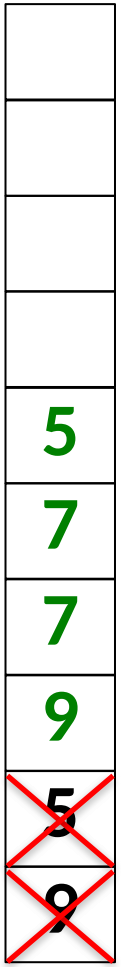
Parcours dichotomique - Pile

Approche itérative

```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4 7



d = 5

f = 9

Parcours dichotomique - Pile

Approche itérative

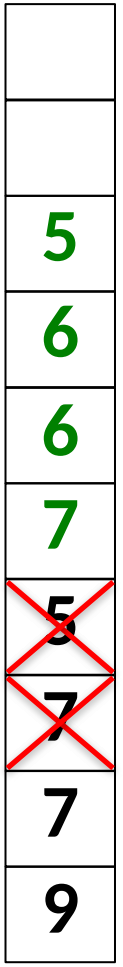
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4 7 6

d = 5

f = 7



Parcours dichotomique - Pile

Approche itérative

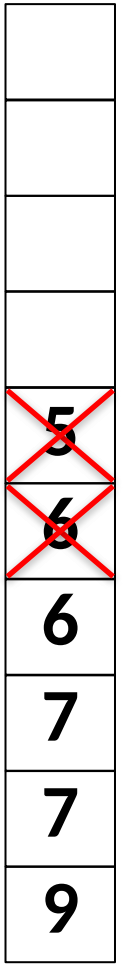
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4 7 6

d = 5

f = 6



Parcours dichotomique - Pile

Approche itérative

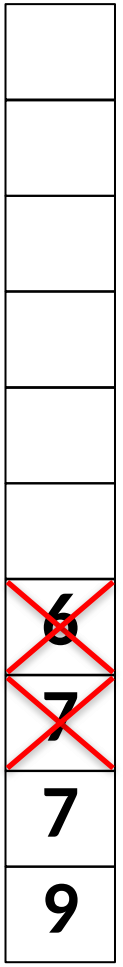
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4 7 6

d = 6

f = 7



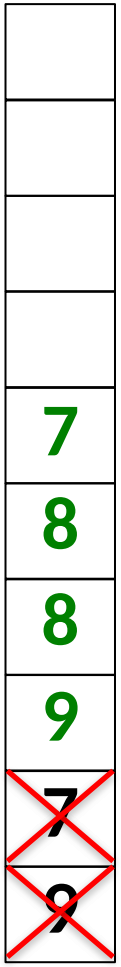
Parcours dichotomique - Pile

Approche itérative

```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4 7 6 8



d = 7

f = 9

Parcours dichotomique - Pile

Approche itérative

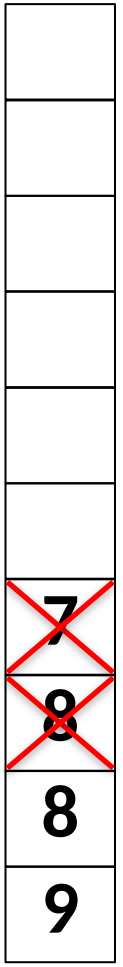
```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4 7 6 8

d = 7

f = 8



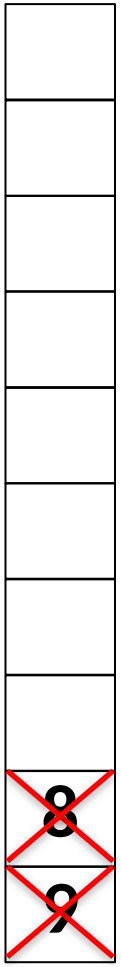
Parcours dichotomique - Pile

Approche itérative

```
void parcoursPile(int d, int f) {
  struct pile *p = creerPile();
  empiler(p, f); empiler(p, d);
  while (!estVide(p)) {
    d = depiler(p); f = depiler(p);
    if (f - d > 1) {
      int m = (d + f) / 2;
      printf("%d ", m);
      empiler(p, f); empiler(p, m);
      empiler(p, m); empiler(p, d);
    }
  }
}
```

m

1 9 5 3 2 4 7 6 8



d = 8

f = 9

Exercices

- Implantation d'une pile d'**unsigned char** par tableau
- Implantation d'une file d'**unsigned char** par tableau circulaire
- Notation polonaise inversée pour +, * et des nombres à un chiffre (de 0 à 9) :
  - Traduction depuis la notation standard
  - Evaluation de la file obtenue