Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Programmation impérative avec Ada

Joëlle Cohen

6 avril 2016

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Plan

- Rappel sur les passages de paramètres
  - les procédures
  - les fonctions
- Paquetages
  - généralités
  - syntaxe
  - Le paquetage tableau1
  - Utilisation
  - Recherche dans un tableau
  - Recherche dichotomique
  - Tris simples
  - Complexité des tris simples
  - nombre minimal nécessaire
  - utilisation des tableaux : les piles
- Généricité
  - un paquetage pile générique
  - les files
- Exceptions
  - Exemple
  - Exceptions prédéfinies

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Rappel sur les passages de paramètres

Une procédure est un sous-programme qui a pour but d'être appelé par le programme.
Elle se déclare de la façon suivante :

```
prodedure P(liste_parametres) is
-- partie declarative
begin
-- suite d'instructions
end P;
```

P est l'identificateur du sous-programme (son nom).
`liste_parametres` est la liste des paramètres formels (les entités) sur lesquels ce sous-programme agit.
`partie_déclarative` est la liste des variables (variables locales) ou type ou unités nécessaires au fonctionnement du sous-programme.

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Rappel sur les passages de paramètres

Une procédure est un sous-programme qui a pour but d'être appelé par le programme.
Elle se déclare de la façon suivante :

```
prodedure P(liste_parametres) is
-- partie declarative
begin
-- suite d'instructions
end P;
```

P est l'identificateur du sous-programme (son nom).
`liste_parametres` est la liste des paramètres formels (les entités) sur lesquels ce sous-programme agit.
`partie_déclarative` est la liste des variables (variables locales) ou type ou unités nécessaires au fonctionnement du sous-programme.

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Rappel sur les passages de paramètres

Une procédure est un sous-programme qui a pour but d'être appelé par le programme.
Elle se déclare de la façon suivante :

```
prodedure P(liste_parametres) is
-- partie declarative
begin
-- suite d'instructions
end P;
```

P est l'identificateur du sous-programme (son nom).
`liste_parametres` est la liste des paramètres formels (les entités) sur lesquels ce sous-programme agit.
`partie_déclarative` est la liste des variables (variables locales) ou type ou unités nécessaires au fonctionnement du sous-programme.

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Une procédure effectue une action et non pas un calcul. Elle doit être contrôlée par le programme.
En effet, lorsque le programme utilisera la procédure, il fournira des paramètres effectifs qui seront alors associées aux paramètres formels.
Ces paramètres effectifs peuvent être des valeurs constantes ou bien les valeurs de variables du programme.
Dans ce dernier cas, il est possible que l'action exécutée par la procédure **modifie** les valeurs de ces variables du programme.
Cette possible modification ne doit se faire que si c'est voulu par le programme.
On a donc besoin d'un contrôle sur le comportement de la procédure vis-à-vis de ses paramètres.
Ce contrôle est précisé par les *modes* IN, OUT, IN OUT que l'on associe à chaque paramètre formel dans *liste_parametres*.

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Une procédure effectue une action et non pas un calcul. Elle doit être contrôlée par le programme.
En effet, lorsque le programme utilisera la procédure, il fournira des paramètres effectifs qui seront alors associées aux paramètres formels.
Ces paramètres effectifs peuvent être des valeurs constantes ou bien les valeurs de variables du programme.
Dans ce dernier cas, il est possible que l'action exécutée par la procédure **modifie** les valeurs de ces variables du programme.
Cette possible modification ne doit se faire que si c'est voulu par le programme.
On a donc besoin d'un contrôle sur le comportement de la procédure vis-à-vis de ses paramètres.
Ce contrôle est précisé par les *modes* IN, OUT, IN OUT que l'on associe à chaque paramètre formel dans *liste_parametres*.

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Voici leur signification :

- `in` le paramètre formel est alors comme une constante et sa valeur est celle du paramètre effectif qui lui est associé au moment de l'appel.
  Un paramètre formel en mode `in` ne peut pas figurer à gauche d'une affectation dans le corps de la procédure.
- `out` le paramètre formel est alors comme une variable. ce paramètre doit être initialisé dans le crops de la procédure et sa valeur à la fin de l'exécution du sous-programme est transmise au paramètre effectif associé.
- `in out` le paramètre formel est comme une variable dont la valeur initiale est celle du paramètre effectif associé au moment de l'appel ; enfin, sa valeur à la fin du sous-programme est transmise au paramètre effectif associé.

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Voici leur signification :

- `in` le paramètre formel est alors comme une constante et sa valeur est celle du paramètre effectif qui lui est associé au moment de l'appel.
  Un paramètre formel en mode `in` ne peut pas figurer à gauche d'une affectation dans le corps de la procédure.
- `out` le paramètre formel est alors comme une variable. ce paramètre doit être initialisé dans le crops de la procédure et sa valeur à la fin de l'exécution du sous-programme est transmise au paramètre effectif associé.
- `in out` le paramètre formel est comme une variable dont la valeur initiale est celle du paramètre effectif associé au moment de l'appel ; enfin, sa valeur à la fin du sous-programme est transmise au paramètre effectif associé.

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Voici leur signification :

- `in` le paramètre formel est alors comme une constante et sa valeur est celle du paramètre effectif qui lui est associé au moment de l'appel.
  Un paramètre formel en mode `in` ne peut pas figurer à gauche d'une affectation dans le corps de la procédure.
- `out` le paramètre formel est alors comme une variable. ce paramètre doit être initialisé dans le crops de la procédure et sa valeur à la fin de l'exécution du sous-programme est transmise au paramètre effectif associé.
- `in out` le paramètre formel est comme une variable dont la valeur initiale est celle du paramètre effectif associé au moment de l'appel ; enfin, sa valeur à la fin du sous-programme est transmise au paramètre effectif associé.

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

A propos des fonctions

Tous les paramètres d'une fonction sont par défaut en mode in

remarque : dans une fonction doit **toujours** figurer l'instruction `return`

L'exécution de `return` arrête l'exécution de la fonction, y compris au milieu d'une itération `for` .

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

A propos des fonctions

Tous les paramètres d'une fonction sont par défaut en mode in

remarque : dans une fonction doit **toujours** figurer l'instruction `return`

L'exécution de `return` arrête l'exécution de la fonction, y compris au milieu d'une itération `for` .

Rappel sur les passages de paramètres
les procédures
les fonctions
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

A propos des fonctions

Tous les paramètres d'une fonction sont par défaut en mode in

remarque : dans une fonction doit **toujours** figurer l'instruction `return`

L'exécution de `return` arrête l'exécution de la fonction, y compris au milieu d'une itération `for` .

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Paquetages

Un paquetage permet de regrouper plusieurs sous-programmes, des constantes ... de façon à pouvoir les utiliser dans un autre programme sans avoir à les réécrire.

Exemple : on utilise déjà la notion de paquetage pour les entrées - sorties de textes, d'entiers ou de flottants.

```
with ada.integer_text_io;
use ada.integer_text_io;
```

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Un paquetage devra être une unité cohérente pour réaliser une action précise, pour définir un type (avec sa définition et les opérations sur ce type). L'ensemble des paquetages constituent une bibliothèque.
Cela va dans le sens de la modularité des programmes qui feront appel à d'autres programmes appartenant à des unités de la bibliothèque.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Ada possède de nombreux paquetages prêts à l'emploi. Par exemple :

- `Ada.Standard` : ce paquetage toujours visible contient toutes les définitions et opérations standard sur les booléens, entiers , flottants ... que l'on utilise couramment.
- `Ada.Text_IO` : ce paquetage contient les fonctionnalités d'entrées-sorties de texte
- `Ada.Numerics.Discrete_Random` : ce paquetage contient des fonctions qui permettent d'engendrer aléatoirement des flottants ou des entiers.
- `Ada.Numerics.Elementary_Functions` : ce paquetage contient les fonctions mathématiques élémentaires
- `Ada.Calendar` : ce paquetage contient les fonctions gérant le temps (date, horloge ...)

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Ada possède de nombreux paquetages prêts à l'emploi. Par exemple :

- `Ada.Standard` : ce paquetage toujours visible contient toutes les définitions et opérations standard sur les booléens, entiers , flottants ... que l'on utilise couramment.
- `Ada.Text_IO` : ce paquetage contient les fonctionnalités d'entrées-sorties de texte
- `Ada.Numerics.Discrete_Random` : ce paquetage contient des fonctions qui permettent d'engendrer aléatoirement des flottants ou des entiers.
- `Ada.Numerics.Elementary_Functions` : ce paquetage contient les fonctions mathématiques élémentaires
- `Ada.Calendar` : ce paquetage contient les fonctions gérant le temps (date, horloge ...)

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Ada possède de nombreux paquetages prêts à l'emploi. Par exemple :

- `Ada.Standard` : ce paquetage toujours visible contient toutes les définitions et opérations standard sur les booléens, entiers , flottants ... que l'on utilise couramment.
- `Ada.Text_IO` : ce paquetage contient les fonctionnalités d'entrées-sorties de texte
- `Ada.Numerics.Discrete_Random` : ce paquetage contient des fonctions qui permettent d'engendrer aléatoirement des flottants ou des entiers.
- `Ada.Numerics.Elementary_Functions` : ce paquetage contient les fonctions mathématiques élémentaires
- `Ada.Calendar` : ce paquetage contient les fonctions gérant le temps (date, horloge ...)

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Ada possède de nombreux paquetages prêts à l'emploi. Par exemple :

- `Ada.Standard` : ce paquetage toujours visible contient toutes les définitions et opérations standard sur les booléens, entiers , flottants ... que l'on utilise couramment.
- `Ada.Text_IO` : ce paquetage contient les fonctionnalités d'entrées-sorties de texte
- `Ada.Numerics.Discrete_Random` : ce paquetage contient des fonctions qui permettent d'engendrer aléatoirement des flottants ou des entiers.
- `Ada.Numerics.Elementary_Functions` : ce paquetage contient les fonctions mathématiques élémentaires
- `Ada.Calendar` : ce paquetage contient les fonctions gérant le temps (date, horloge ...)

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Ada possède de nombreux paquetages prêts à l'emploi. Par exemple :

- `Ada.Standard` : ce paquetage toujours visible contient toutes les définitions et opérations standard sur les booléens, entiers , flottants ... que l'on utilise couramment.
- `Ada.Text_IO` : ce paquetage contient les fonctionnalités d'entrées-sorties de texte
- `Ada.Numerics.Discrete_Random` : ce paquetage contient des fonctions qui permettent d'engendrer aléatoirement des flottants ou des entiers.
- `Ada.Numerics.Elementary_Functions` : ce paquetage contient les fonctions mathématiques élémentaires
- `Ada.Calendar` : ce paquetage contient les fonctions gérant le temps (date, horloge ...)

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Syntaxe

Un paquetage nommé par exemple `Pack` est constitué de deux parties chacune dans un fichier

- une partie *spécification* dans le fichier `Pack.ads` : elle contient les définitions des types, constantes, exceptions et les prototypes de procédures et fonctions.
- une partie *corps* dans le fichier `Pack.adb` : elle contient le corps de toutes les fonctions et procédures déclarées dans la partie spécification.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Pack.ads

```
package pack is
-- déclarations D
private
-- déclarations DP
end pack;
```

D sont les déclarations de ce que rend accessible le paquetage `pack` depuis l'extérieur.
DP sont des déclarations privées connues du paquetage seul.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Pack.adb

```
package body pack is
-- corps des fonctions et procédures
end pack;
```

les déclarations D et DP sont visibles et utilisables dans le corps du paquetage pack.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On va construire un paquetage pour manipuler des tableaux d'entiers unidimensionnels.

Tout d'abord on va exprimer nos besoins puis les structurer

- ⇝ définir un type tableau le plus général possible
- ⇝ remplir un tableau
- ⇝ afficher un tableau
- ⇝ .... à venir

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On va construire un paquetage pour manipuler des tableaux d'entiers unidimensionnels.
Tout d'abord on va exprimer nos besoins puis les structurer

- ⇝ définir un type tableau le plus général possible
- ⇝ remplir un tableau
- ⇝ afficher un tableau
- ⇝ .... à venir

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On va construire un paquetage pour manipuler des tableaux d'entiers unidimensionnels.
Tout d'abord on va exprimer nos besoins puis les structurer

- $\rightsquigarrow$ définir un type tableau le plus général possible
- $\rightsquigarrow$ remplir un tableau
- $\rightsquigarrow$ afficher un tableau
- $\rightsquigarrow$ .... à venir

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On va construire un paquetage pour manipuler des tableaux d'entiers unidimensionnels.
Tout d'abord on va exprimer nos besoins puis les structurer

- $\rightsquigarrow$ définir un type tableau le plus général possible
- $\rightsquigarrow$ remplir un tableau
- $\rightsquigarrow$ afficher un tableau
- $\rightsquigarrow$ .... à venir

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On va construire un paquetage pour manipuler des tableaux d'entiers unidimensionnels.
Tout d'abord on va exprimer nos besoins puis les structurer

- $\rightsquigarrow$ définir un type tableau le plus général possible
- $\rightsquigarrow$ remplir un tableau
- $\rightsquigarrow$ afficher un tableau
- $\rightsquigarrow$ .... à venir

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On va construire un paquetage pour manipuler des tableaux d'entiers unidimensionnels.
Tout d'abord on va exprimer nos besoins puis les structurer

- $\rightsquigarrow$ définir un type tableau le plus général possible
- $\rightsquigarrow$ remplir un tableau
- $\rightsquigarrow$ afficher un tableau
- $\rightsquigarrow$ .... à venir

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

tableau1.ads

```
package tableau1 is
nb : constant integer := 49 ;
subtype intervalle is integer range 0..nb;
type tableau is array(intervalle) of integer;
procedure init( t : out tableau);
procedure afficher ( t : in tableau);
end tableau1;
```

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

tableau1.adb

```
with ada.text_io; use ada.text_io;
with ada.integer_text_io;
use ada.integer_text_io;
with Ada.Numerics.Discrete_Random;
package body tableau1 is
-- init donne aux éléments du tableau t
-- des valeurs aléatoires entre -99 et 99
procedure init ( t : out tableau) is
subtype valeur is integer range -99 .. 99 ;
package hasard_valeur is
new Ada.Numerics.Discrete_Random (valeur) ;
use hasard_valeur;
G : hasard_valeur.Generator ;
begin
reset(G);
for i in t'range loop t(i) := random(G); end loop;
end init;
-- afficher écrit 1 par 1 les éléments de t
-- avec par élément 5 espaces
-- à raison de 10 éléments par ligne
procedure afficher (t : in tableau) is
begin
for i in t'range loop
   put(t(i), 5);
   if (i+1) rem 10 =0 then new_line; end if;
end afficher;
end tableau1;
```

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

utilisation

Pour utiliser dans un programme un paquetage `pack` dans un programme, il faut le rendre accessible par la clause `with pack`.
`use pack` est une clause facultative qui permet d'invoquer les fonctionnalités du paquetage sans les préfixer par le nom du paquetage.

```
with tableau1;
use tableau1;
procedure essai1 is
t : tableau; k : integer; temp : integer;
begin
init(t);
afficher(t);
k:= t'first;
for i in t'range loop
 if t(i)>t(k) then k := i; end if;
end loop;
temp := t(t'last) ; t(t'last) := t(k) ; t(k) := temp;
afficher(t);
end essai1;
```

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dans un tableau

On va enrichir le paquetage d'une fonction de recherche séquentielle d'un entier dans un tableau.
On peut demander plusieurs résultats possibles :

- une réponse booléenne
- l'indice où on a trouvé l'entier recherché
  - le premier
  - le dernier
  - que renvoyer s'il ne s'y trouve pas ?
- le nombre de fois que cet entier se trouve dans le tableau

Dans tous ces cas on parle de recherche séquentielle lorsque l'on parcourt les éléments du tableau un par un consécutivement.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Recherche dans un tableau

On va enrichir le paquetage d'une fonction de recherche séquentielle d'un entier dans un tableau.
On peut demander plusieurs résultats possibles :

- une réponse booléenne
- l'indice où on a trouvé l'entier recherché
  - le premier
  - le dernier
  - que renvoyer s'il ne s'y trouve pas ?
- le nombre de fois que cet entier se trouve dans le tableau

Dans tous ces cas on parle de recherche séquentielle lorsque l'on parcourt les éléments du tableau un par un consécutivement.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Recherche dans un tableau

On va enrichir le paquetage d'une fonction de recherche séquentielle d'un entier dans un tableau.
On peut demander plusieurs résultats possibles :

- une réponse booléenne
- l'indice où on a trouvé l'entier recherché
  - le premier
  - le dernier
  - que renvoyer s'il ne s'y trouve pas ?
- le nombre de fois que cet entier se trouve dans le tableau

Dans tous ces cas on parle de recherche séquentielle lorsque l'on parcourt les éléments du tableau un par un consécutivement.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Recherche dans un tableau

On va enrichir le paquetage d'une fonction de recherche séquentielle d'un entier dans un tableau.
On peut demander plusieurs résultats possibles :

- une réponse booléenne
- l'indice où on a trouvé l'entier recherché
  - le premier
  - le dernier
  - que renvoyer s'il ne s'y trouve pas ?
- le nombre de fois que cet entier se trouve dans le tableau

Dans tous ces cas on parle de recherche séquentielle lorsque l'on parcourt les éléments du tableau un par un consécutivement.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Recherche dans un tableau

On va enrichir le paquetage d'une fonction de recherche séquentielle d'un entier dans un tableau.
On peut demander plusieurs résultats possibles :

- une réponse booléenne
- l'indice où on a trouvé l'entier recherché
  - le premier
  - le dernier
  - que renvoyer s'il ne s'y trouve pas ?
- le nombre de fois que cet entier se trouve dans le tableau

Dans tous ces cas on parle de recherche séquentielle lorsque l'on parcourt les éléments du tableau un par un consécutivement.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dans un tableau

On va enrichir le paquetage d'une fonction de recherche séquentielle d'un entier dans un tableau.
On peut demander plusieurs résultats possibles :

- une réponse booléenne
- l'indice où on a trouvé l'entier recherché
  - le premier
  - le dernier
  - que renvoyer s'il ne s'y trouve pas ?
- le nombre de fois que cet entier se trouve dans le tableau

Dans tous ces cas on parle de recherche séquentielle lorsque l'on parcourt les éléments du tableau un par un consécutivement.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

appartenance

```
function appartenance(t : in tableau; x : in integer)
      return boolean is
b : boolean := false;
begin
for i in t'range loop
if t(i)=x then b=true; enf if ;
end loop;
return b;
end appartenance;
```

inconvénient : la boucle se poursuit même si on a trouvé x dans t .

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

appartenance

```
function appartenance(t : in tableau; x : in integer)
      return boolean is
b : boolean := false;
begin
for i in t'range loop
if t(i)=x then b=true; enf if ;
end loop;
return b;
end appartenance;
```

inconvénient : la boucle se poursuit même si on a trouvé x dans t .

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

appartenance

```
function appartenance(t : in tableau; x : in integer)
        return boolean is
b : boolean := false; i : integer := t'first;
begin
loop
if t(i)=x then b:=true; exit ; end if ;
i := i+1;
if i>t'last then exit; end if;
end loop;
return b;
end appartenance;
```

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

rech_premier_indice

```
function rech_premier_indice(t : in tableau; x : in integer)
        return integer is
 i : integer := t'first; k : integer :=  -1;
begin
loop
if t(i)=x then k:=i; exit ; end if ;
i := i+1;
if i>t'last then exit; end if;
end loop;
return k;
end rech_premier_indice;
```

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

rech_dernier_indice

```
function rech_dernier_indice(t : in tableau; x : in integer)
        return integer is
 i : integer := t'last; k : integer :=  -1;
begin
loop
if t(i)=x then k:=i; exit ; end if ;
i := i-1;
if i<t'first then exit; end if;
end loop;
return k;
end rech_dernier_indice;
```

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

nbre_occurrence

```
function nbre_occurrence(t : in tableau; x : in integer)
      return integer is
k : integer := 0;
begin
for i in t'range loop
if t(i)=x then k:=k+1; enf if ;
end loop;
return k;
end nbre_occurrence;
```

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 54 dans le tableau suivant

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 54 dans le tableau suivant

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 54 dans le tableau suivant

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 54 dans le tableau suivant

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 54 dans le tableau suivant

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 54 dans le tableau suivant

i	1	2	3	4	5	6	7	**8**	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	**54**	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Recherche dichotomique dans un tableau trié

le recherche dichotomique dans un tableau trié tient compte du fait que le tableau est trié par ordre croissant. Par exemple on recherche la valeur 5 dans le même tableau.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-23	-11	-3	8	31	32	54	77	92

Le curseur de gauche a dépassé le curseur de droite : la recherche est négative.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

rech_dichotomique

```
function rech_dichotomique(t : tableau; x :  integer)
      return boolean is
b:boolean := false ;
g : integer := t'first; d: integer :=t'last; m: integer;
begin
loop
m := (g+d) / 2;
if t(m)=x then b:= true; exit ; end if ;
if t(m) < x then g := m+1;
else d := m-1; end if;
if g>d then exit; end if;
end loop;
return b;
end rech_dichotomique;
```

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

rech_dichotomique_rec

```
function rech_dichotomique_rec
(t:tableau; x:integer; g,d:integer)
return boolean is
begin
if g>d then
    return false;
elsif t((g+d)/2) = x then
    return true
elsif t((g+d)/2) < x then
    return rech_dichotomique_rec(t,x,1+(g+d)/2,d) ;
else
    return rech_dichotomique_rec(t,x,g,(g+d)/2-1);
endif;
end rech_dichotomique_rec;
.....
begin -- programme principal
init(t);
if rech_dichotomique_rec(t, 92, t'first , t'last)
 then put("oui");
 else put("non");
end if; end;
```

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Tris simples

On va étudier trois méthodes de tris simples :

- tri par sélection
- tri à bulle
- tri par insertion

Rappel sur les passages de paramètres

Paquetages
- généralités
- syntaxe
- Le paquetage tableau1
- Utilisation
- Recherche dans un tableau
- Recherche dichotomique
- Tris simples
- Complexité des tris simples
- nombre minimal nécessaire
- utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	23	-43	98	-31	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
- généralités
- syntaxe
- Le paquetage tableau1
- Utilisation
- Recherche dans un tableau
- Recherche dichotomique
- Tris simples
- Complexité des tris simples
- nombre minimal nécessaire
- utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	23	-43	98	-31	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	98	-31	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	98	-31	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	-31	98	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	-31	98	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple de tri à bulle

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	-31	98	-54	102	77	-32

Rappel sur les passages de paramètres

Paquetages
- généralités
- syntaxe
- Le paquetage tableau1
- Utilisation
- Recherche dans un tableau
- Recherche dichotomique
- Tris simples
- Complexité des tris simples
- nombre minimal nécessaire
- utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	-31	98	-54	102	77	-32

Rappel sur les passages de paramètres

Paquetages
- généralités
- syntaxe
- Le paquetage tableau1
- Utilisation
- Recherche dans un tableau
- Recherche dichotomique
- Tris simples
- Complexité des tris simples
- nombre minimal nécessaire
- utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	-31	98	-54	77	102	-32

Rappel sur les passages de paramètres

Paquetages
- généralités
- syntaxe
- Le paquetage tableau1
- Utilisation
- Recherche dans un tableau
- Recherche dichotomique
- Tris simples
- Complexité des tris simples
- nombre minimal nécessaire
- utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	-31	98	-54	77	102	-32

Rappel sur les passages de paramètres

Paquetages
- généralités
- syntaxe
- Le paquetage tableau1
- Utilisation
- Recherche dans un tableau
- Recherche dichotomique
- Tris simples
- Complexité des tris simples
- nombre minimal nécessaire
- utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-75	-43	23	-31	98	-54	77	-32	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

i	1	2	3	4	5	6	7	8	9	**10**
$t(i)$	-11	-75	-43	23	-31	98	-54	77	-32	**102**

Il faut recommencer jusqu'à ce que le tableau soit trié.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Principe du tri par selection

pour k variant de t'last à t'first $+1$ **faire**

- trouver le plus grand élément parmi $t(t'first) \ldots t(k)$
- le placer en position k

fin pour

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Principe du tri par selection

pour k variant de t'last à t'first $+1$ **faire**

- trouver le plus grand élément parmi $t(t'first) \ldots t(k)$
- le placer en position k

fin pour

REMARQUE k varie en *décroissant*.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Principe du tri par selection

pour k variant de t'last à t'first $+1$ **faire**

- trouver le plus grand élément parmi $t(t'first) \ldots t(k)$
- le placer en position k

fin pour

REMARQUE k varie en *décroissant*.
ATTENTION « le placer en position k » doit être fait sans perte de valeur du tableau.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

≪ trouver le plus grand élément parmi $t(t'first) \ldots t(k)$ ≫ va consister en la recherche de *l'emplacement* de ce plus grand élément.

≪ le placer en position k ≫ va consister en un échange entre la valeur actuellement en position k et la valeur du plus grand élément dont la position vient d'être déterminée.

EXEMPLE

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	102	-54	77	-32

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	-32	-54	77	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

« trouver le plus grand élément parmi $t(t'first) \dots t(k)$ » va consister en la recherche de *l'emplacement* de ce plus grand élément.
« le placer en position k » va consister en un échange entre la valeur actuellement en position k et la valeur du plus grand élément dont la position vient d'être déterminée.

EXEMPLE

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	102	-54	77	-32

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	-32	-54	77	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

≪ trouver le plus grand élément parmi $t(t'first) \ldots t(k)$ ≫ va consister en la recherche de *l'emplacement* de ce plus grand élément.
≪ le placer en position k ≫ va consister en un échange entre la valeur actuellement en position k et la valeur du plus grand élément dont la position vient d'être déterminée.

EXEMPLE

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	102	-54	77	-32

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	-32	-54	77	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

≪ trouver le plus grand élément parmi $t(t'first) \ldots t(k)$ ≫ va consister en la recherche de *l'emplacement* de ce plus grand élément.
≪ le placer en position k ≫ va consister en un échange entre la valeur actuellement en position k et la valeur du plus grand élément dont la position vient d'être déterminée.

EXEMPLE

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	102	-54	77	-32

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	-32	-54	77	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

1^{er} passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	102	-54	77	-32

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

1^{er} passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	-32	-54	77	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$2^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	98	-31	-32	-54	77	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$2^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	77	-31	-32	-54	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$3^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	77	-31	-32	-54	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$3^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	-54	-31	-32	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$4^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	23	-75	-43	-54	-31	-32	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$4^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-32	-75	-43	-54	-31	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$5^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-11	-32	-75	-43	-54	-31	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$5^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-31	-32	-75	-43	-54	-11	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$6^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-31	-32	-75	-43	-54	-11	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$6^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-54	-32	-75	-43	-31	-11	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$7^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-54	-32	-75	-43	-31	-11	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$7^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-54	-43	-75	-32	-31	-11	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$8^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-54	-43	-75	-32	-31	-11	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$8^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-54	-75	-43	-32	-31	-11	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$9^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-54	-75	-43	-32	-31	-11	23	77	98	102

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exemple complet

$9^{ème}$ passage :

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	-43	-32	-31	-11	23	77	98	102

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

deux fonctions utiles

```
-- max renvoie l'indice du plus grand élément
-- du tableau t paramètre entre les indices paramètres
-- g et d avec g < d

function max(t :  tableau; g,d : integer)
return integer is
k: integer := g;
begin
for i in g..d loop
if t(k) < t(i) then k := i; end if;
end loop;
return k;
end max;

-- echanger échange les deux éléments du tableau paramètre
-- d'indices respectifs les paramètres i et j

procédure echanger(t:in out tableau; i, j: in integer) is
temp : integer;
begin
temp := t(i); t(i) := t(j); t(j) := temp;
end echanger ;
```

Rappel sur les passages de paramètres

Paquetages
- généralités
- syntaxe
- Le paquetage tableau1
- Utilisation
- Recherche dans un tableau
- Recherche dichotomique
- Tris simples
- Complexité des tris simples
- nombre minimal nécessaire
- utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

tri_selection

```
procedure tri_selection(t : in out tableau) is
begin
for k in reverse t'first+1 .. t'last loop
echanger(t, k, max(t,t'first,k)); end loop;
end tri_selection;
```

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Principe du tri par insertion

C'est le principe du joueur de carte :
on trie au fur et à mesure que l'on découvre une nouvelle valeur en l'insérant parmi celles déjà triées.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Principe du tri par insertion

C'est le principe du joueur de carte :
on trie au fur et à mesure que l'on découvre une nouvelle valeur en l'insérant parmi celles déjà triées.
Par exemple : on a déjà

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74				

et on donne la valeur 25 à insérer

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74				

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Principe du tri par insertion

C'est le principe du joueur de carte :
on trie au fur et à mesure que l'on découvre une nouvelle valeur en l'insérant parmi celles déjà triées.
Par exemple : on a déjà

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74				

et on donne la valeur 25 à insérer

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74				

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	25	34	56	74			

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

POUR k variant de t'first $+1$ à t'last **faire**

- trouver la place p de la valeur $t(k)$ parmi les valeurs $t(1) \cdots t(k-1)$
- le placer en position p

FIN POUR

Le problème est que le tableau est donné entièrement rempli et que les valeurs à insérer sont déjà dans le tableau.
L'action de placer $t(k)$ en position p oblige à déplacer la valeur $t(p)$ pour ne pas la perdre.
De plus il ne faut pas perturber le classement
$t(1) < \cdots < t(k-1)$.
On va donc *insérer* la valeur $t(k)$ en position p en *décalant* d'un rang vers la droite toutes les valeurs depuis la position p jusqu'à la position $k-1$.
On reproduit en fait le geste du joueur de cartes qui écarte les deux cartes entre lesquelles vient se glisser la nouvelle carte.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

POUR k variant de t'first $+1$ à t'last **faire**

- trouver la place p de la valeur $t(k)$ parmi les valeurs $t(1) \cdots t(k-1)$
- le placer en position p

FIN POUR

Le problème est que le tableau est donné entièrement rempli et que les valeurs à insérer sont déjà dans le tableau.

L'action de placer $t(k)$ en position p oblige à déplacer la valeur $t(p)$ pour ne pas la perdre.

De plus il ne faut pas perturber le classement $t(1) < \cdots < t(k-1)$.

On va donc *insérer* la valeur $t(k)$ en position p en *décalant* d'un rang vers la droite toutes les valeurs depuis la position p jusqu'à la position $k-1$.

On reproduit en fait le geste du joueur de cartes qui écarte les deux cartes entre lesquelles vient se glisser la nouvelle carte.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

POUR k variant de t'first $+1$ à t'last **faire**

- trouver la place p de la valeur $t(k)$ parmi les valeurs $t(1) \cdots t(k-1)$
- le placer en position p

FIN POUR

Le problème est que le tableau est donné entièrement rempli et que les valeurs à insérer sont déjà dans le tableau.
L'action de placer $t(k)$ en position p oblige à déplacer la valeur $t(p)$ pour ne pas la perdre.
De plus il ne faut pas perturber le classement $t(1) < \cdots < t(k-1)$.
On va donc *insérer* la valeur $t(k)$ en position p en *décalant* d'un rang vers la droite toutes les valeurs depuis la position p jusqu'à la position $k-1$.
On reproduit en fait le geste du joueur de cartes qui écarte les deux cartes entre lesquelles vient se glisser la nouvelle carte.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

POUR k variant de t'first $+1$ à t'last **faire**

- trouver la place p de la valeur $t(k)$ parmi les valeurs $t(1) \cdots t(k-1)$
- le placer en position p

FIN POUR

Le problème est que le tableau est donné entièrement rempli et que les valeurs à insérer sont déjà dans le tableau.
L'action de placer $t(k)$ en position p oblige à déplacer la valeur $t(p)$ pour ne pas la perdre.
De plus il ne faut pas perturber le classement
$t(1) < \cdots < t(k-1)$.
On va donc *insérer* la valeur $t(k)$ en position p en *décalant* d'un rang vers la droite toutes les valeurs depuis la position p jusqu'à la position $k-1$.
On reproduit en fait le geste du joueur de cartes qui écarte les deux cartes entre lesquelles vient se glisser la nouvelle carte.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

POUR k variant de t'first $+1$ à t'last **faire**

- trouver la place p de la valeur $t(k)$ parmi les valeurs $t(1) \cdots t(k-1)$
- le placer en position p

FIN POUR

Le problème est que le tableau est donné entièrement rempli et que les valeurs à insérer sont déjà dans le tableau.
L'action de placer $t(k)$ en position p oblige à déplacer la valeur $t(p)$ pour ne pas la perdre.
De plus il ne faut pas perturber le classement
$t(1) < \cdots < t(k-1)$.
On va donc *insérer* la valeur $t(k)$ en position p en *décalant* d'un rang vers la droite toutes les valeurs depuis la position p jusqu'à la position $k-1$.
On reproduit en fait le geste du joueur de cartes qui écarte les deux cartes entre lesquelles vient se glisser la nouvelle carte.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Par exemple, avec $k = 7$ et les valeurs d'indice 1 à 6 triées.

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74	25	-12	17	-5

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	34	56	74	-12	17	-5

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	**25**	34	56	74	-12	17	-5

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

tri_insertion

```
procedure tri_insertion(t : in out tableau) is
p : integer; temp : integer;
begin
for k in t'first+1 .. t'last loop
p:= t'first; temp := t(k);
-- temp est la valeur à placer
while t(p)<temp loop p := p+1;
end loop;
-- p est la position de temp
-- parmi t(t'first) , ... , t(k-1)
for i in reverse (p+1) .. k loop
t(i) := t(i-1); end loop;
-- décalage des valeurs vers la droite
t(p) := temp;
-- placement de la valeur
end loop;
end tri_selection;
```

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

variante du tri_insertion

On va utiliser les échanges pour placer la valeur

```
procedure tri_insertion(t : in out tableau) is
i : integer;
begin
for k in t'first+1 .. t'last loop
-- t(k) est la valeur à placer
i := k-1;
while (i>=t'first) and then (t(i)>t(i+1))
 loop
 echanger(t,i,i+1) ;
 -- t(k) a reculé
 i:= i-1;
end loop;
end loop;
end tri_insertion;
```

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On peut remarquer que les trois tris étudiés sont basés sur les actions élémentaires suivantes : des comparaisons d'entiers et des échanges entre deux variables. Ce sont ces deux actions que l'on va dénombrer pour chacun des tris précédents.
On notera n le nombre d'éléments d'un tableau à trier.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

tri par sélection

il y a de façon claire exactement $n-1$ échanges.
Trouver la position du plus grand élément depuis la position 1 jusqu'à la position k nécessite $k-1$ comparaisons.
Sachant que k varie depuis n jusqu'à 2, on a donc

$$\sum_{k=2}^{k=n}(k-1) = \sum_{k=1}^{k=n-1} k = \frac{n(n-1)}{2} \text{ comparaisons.}$$

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

tri à bulle

Les échanges dépendent du résultat de comparaisons. On comptera donc le nombre maximum d'échanges en vérifiant toutefois qu'il existe au moins un cas pour lequel ce nombre est effectivement atteint.

Il y a k comparaisons, pour chaque valeur de k variant de 1 à $n-1$; le nombre de comparaisons est donc

$$\sum_{k=1}^{k=n-1} k = \frac{n(n-1)}{2}.$$

Pour chacune de ces comparaisons, il peut y avoir un échange.

Donc le nombre maximum d'échanges est aussi $\frac{n(n-1)}{2}$.

Il est facile de constater que si les valeurs sont en ordre initial décroissant, il y a bien un échange pour chaque comparaison.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

tri par insertion

Les échanges sont aussi dépendants du résultat de comparaisons. On va donc de nouveau rechercher une majoration de ces nombres.
Comme précédemment, si les valeurs sont en ordre initial décroissant, le nombre d'échanges et de comparaisons sont maximaux et vaut $k-1$ pour k variant de 2 à n.
On a donc au maximum $\frac{n(n-1)}{2}$ échanges / comparaisons.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Pour les trois tris, pour n très grand, le nombre de comparaisons est de l'ordre de n^2.
Cela ne rend compte que partiellement du comportement de ces algorithmes dont on pourrait chercher un comportement « moyen » .
En effet on n'est pas toujours confronté au pire des cas (c'est-à-dire ordre initial décroissant).
Cett étude en moyenne nécessiterait une modélisation probabiliste du problème.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On cherche à calculer le nombre minimal de comparaisons nécessaires pour trier n valeurs dans le pire des cas.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On va établir ce qu'on appelle un arbre de décision pour le tri de 3 valeurs.

Les noeuds internes seront des comparaisons entre deux valeurs à trier.

Les deux fils d'un noeud correspondent aux deux résultats possibles de la comparaison associée au noeud.

Les feuilles représentent un tri possible des n valeurs à trier.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

On va établir ce qu'on appelle un arbre de décision pour le tri de 3 valeurs.
Les noeuds internes seront des comparaisons entre deux valeurs à trier.
Les deux fils d'un noeud correspondent aux deux résultats possibles de la comparaison associée au noeud.
Les feuilles représentent un tri possible des n valeurs à trier.

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

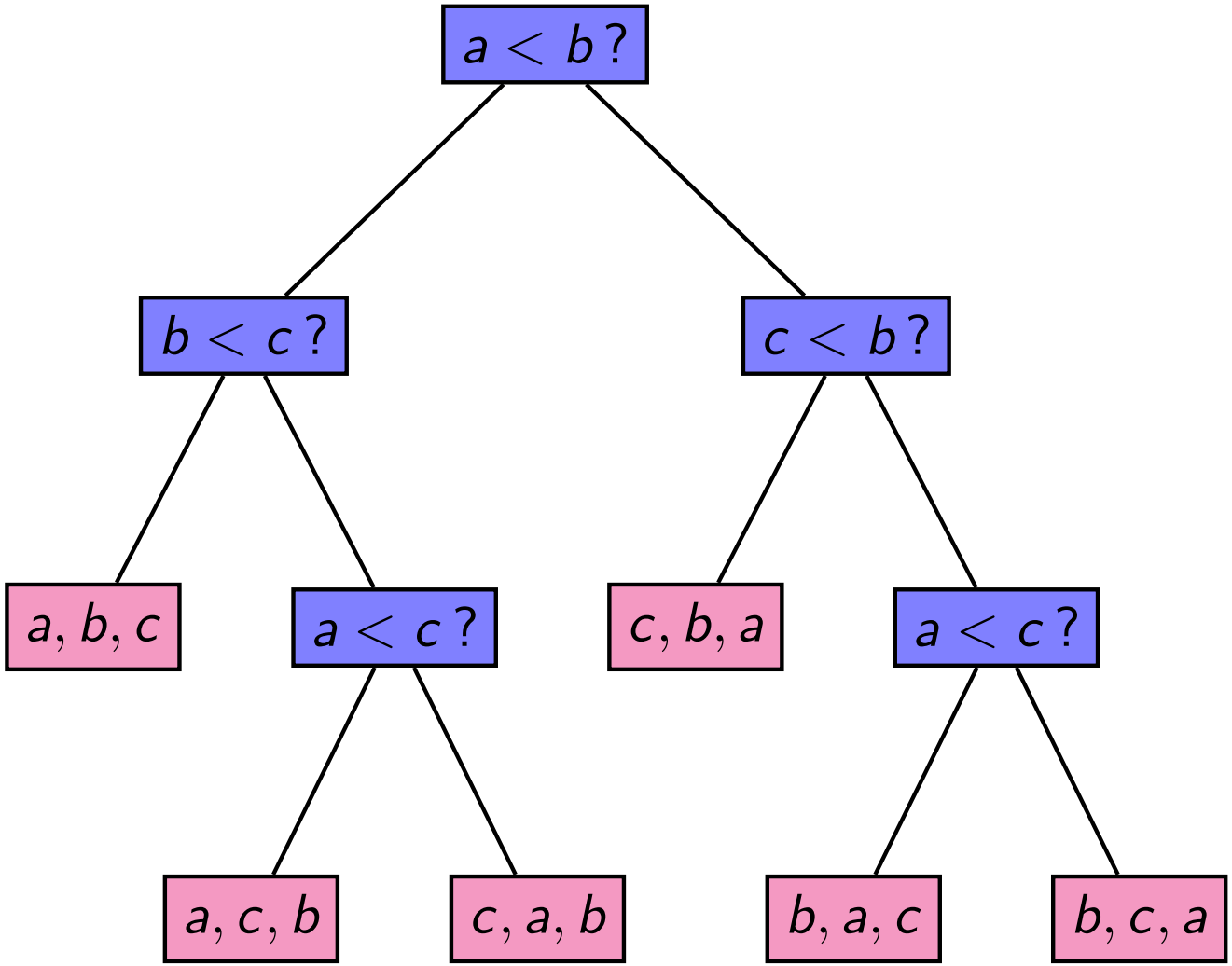
Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking



Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Dans cet arbre il y a 6 feuilles : pourquoi ?

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Dans cet arbre il y a 6 feuilles : pourquoi ?
il y a 6 classements possibles de 3 valeurs : 123, 132, 213, 231, 312, 321

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Dans cet arbre il y a 6 feuilles : pourquoi ?
il y a 6 classements possibles de 3 valeurs : 123, 132, 213, 231, 312, 321
Dans le cas général, combien y a-t-il de feuilles pour n valeurs ?

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Dans cet arbre il y a 6 feuilles : pourquoi ?
il y a 6 classements possibles de 3 valeurs : 123, 132, 213, 231, 312, 321
Dans le cas général, combien y a-t-il de feuilles pour n valeurs ?

$$n!$$

le nombre de permutations de n valeurs

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Dans un arbre binaire de hauteur h il y a au maximum 2^h feuilles

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Dans un arbre binaire de hauteur h il y a au maximum 2^h feuilles
Sachant que l'on a besoin de $n!$ feuilles il faut que h vérifie

$$2^h \geq n!$$

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Dans un arbre binaire de hauteur h il y a au maximum 2^h feuilles
Sachant que l'on a besoin de $n!$ feuilles il faut que h vérifie

$$2^h \geq n!$$

$$h \geq \log_2(n!)$$

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Dans un arbre binaire de hauteur h il y a au maximum 2^h feuilles
Sachant que l'on a besoin de $n!$ feuilles il faut que h vérifie

$$2^h \geq n!$$

$$h \geq \log_2(n!)$$

Or h représente le nombre maximal de comparaisons pour atteindre une feuille donc un classement.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Dans un arbre binaire de hauteur h il y a au maximum 2^h feuilles

Sachant que l'on a besoin de $n!$ feuilles il faut que h vérifie

$$2^h \geq n!$$

$$h \geq \log_2(n!)$$

Or h représente le nombre maximal de comparaisons pour atteindre une feuille donc un classement. Conclusion : il faut au moins $\log_2(n!)$ comparaisons pour trier n valeurs dans le pire des cas

REMARQUE : pour n très grand, $\log_2(n!) \sim n \log_2 n$

On peut donc espérer trouver des algorithmes plus performants avec un nombre de comparaisons de l'ordre de $n \log_2 n$

(que les tris simples qui eux fonctionnent avec un nombre de comparaisons de l'ordre de n^2).

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

n	10	10^2	10^3	10^5	10^6
n^2	100	10^4	10^6	10^{10}	10^{12} (mille millards)
$n \log_2 n$	33,2	332	3320	332000	3,3 millions

Si une comparaison prend 1 nano seconde ($=10^{-9}$s) alors classer un million de valeurs avec nos tris simples prend mille secondes soit 15 minutes environ alors qu'un tri s'effectuant avec $n \log_2 n$ comparaisons prendrait 3 millisecondes.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

pile

Une pile est une structure de données qui permet de se souvenir de la dernière donnée enregistrée.
Ce fonctionnement est utilisé dans de nombreux logiciels :

- fonction `undo` : gedit, emacs, Word, Excel, jeux ...
- fonction `retour à la page précédente` : navigateur FireFox, ...
- calculatrice polonaise inverse : 23 39 + 25 - renvoie 37
- dernier entré dans un ascenceur= premier sorti
- le petit poucet retrouve son chemin grâce aux petits cailloux
- récursivité gérée par un compilateur

une pile est aussi appelée LIFO (Last In, First Out)

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

pile

Une pile est une structure de données qui permet de se souvenir de la dernière donnée enregistrée.
Ce fonctionnement est utilisé dans de nombreux logiciels :

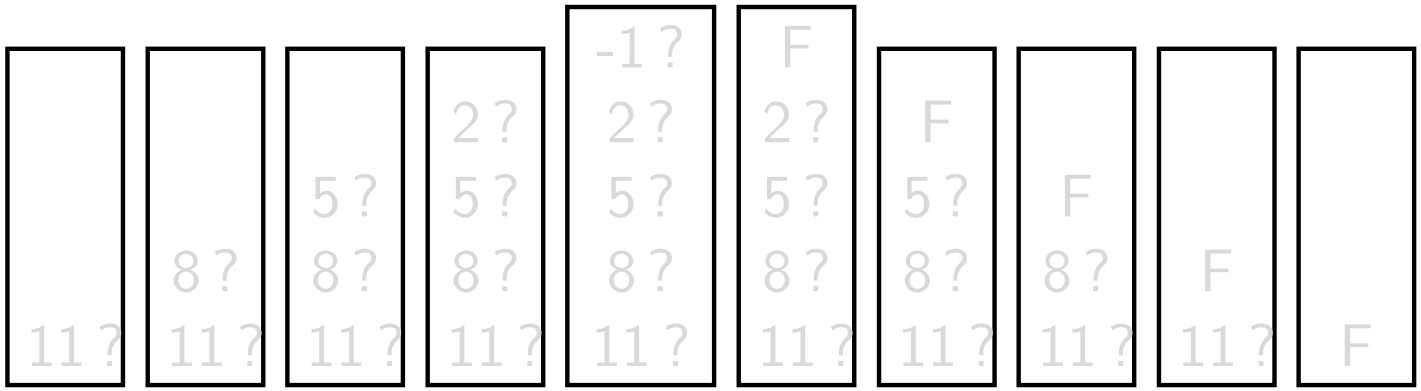
- fonction `undo` : gedit, emacs, Word, Excel, jeux ...
- fonction `retour à la page précédente` : navigateur FireFox, ...
- calculatrice polonaise inverse : 23 39 + 25 - renvoie 37
- dernier entré dans un ascenceur= premier sorti
- le petit poucet retrouve son chemin grâce aux petits cailloux
- récursivité gérée par un compilateur

une pile est aussi appelée LIFO (Last In, First Out)

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

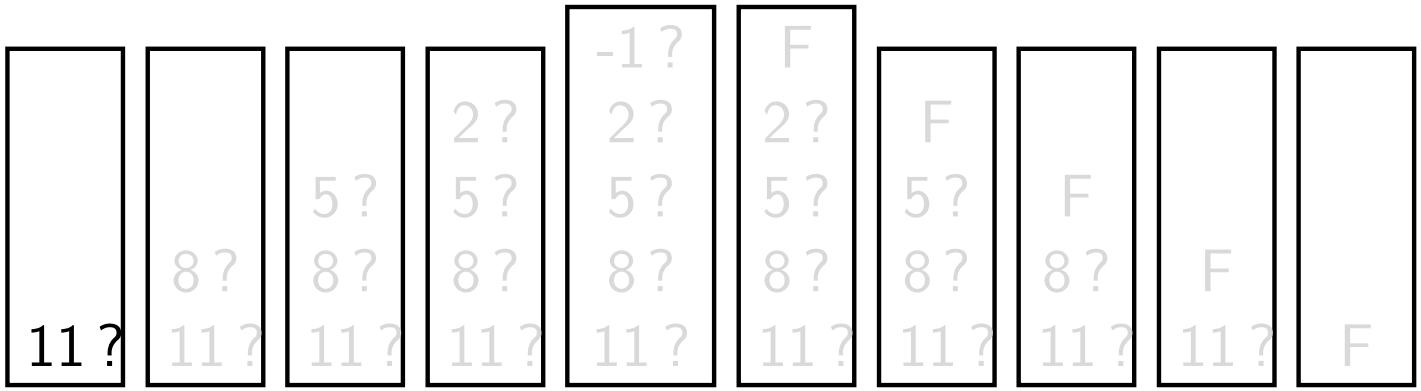
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

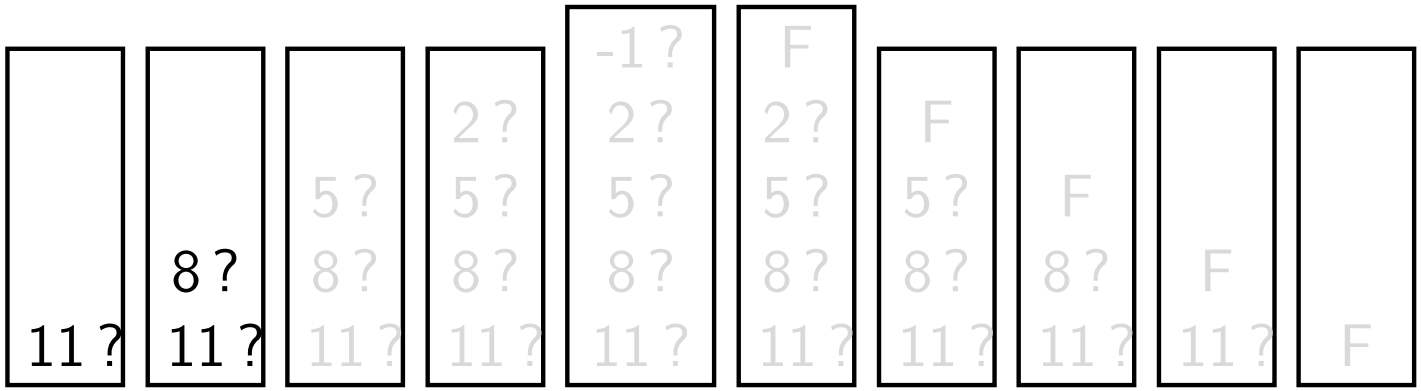
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

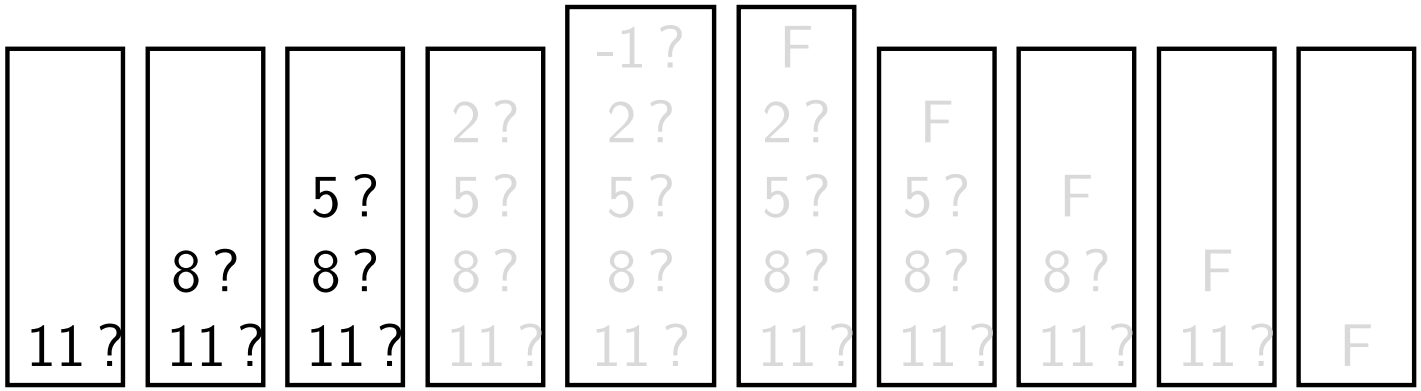
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

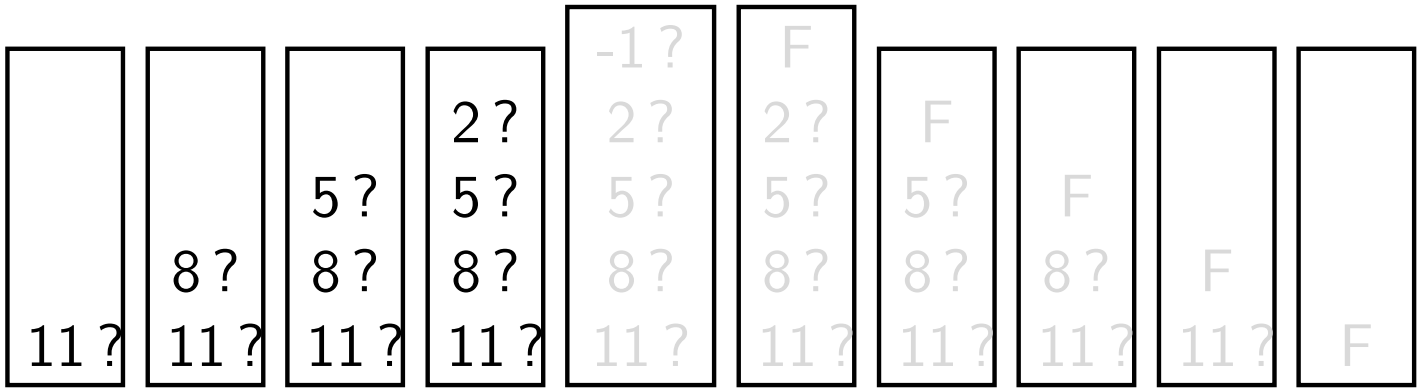
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

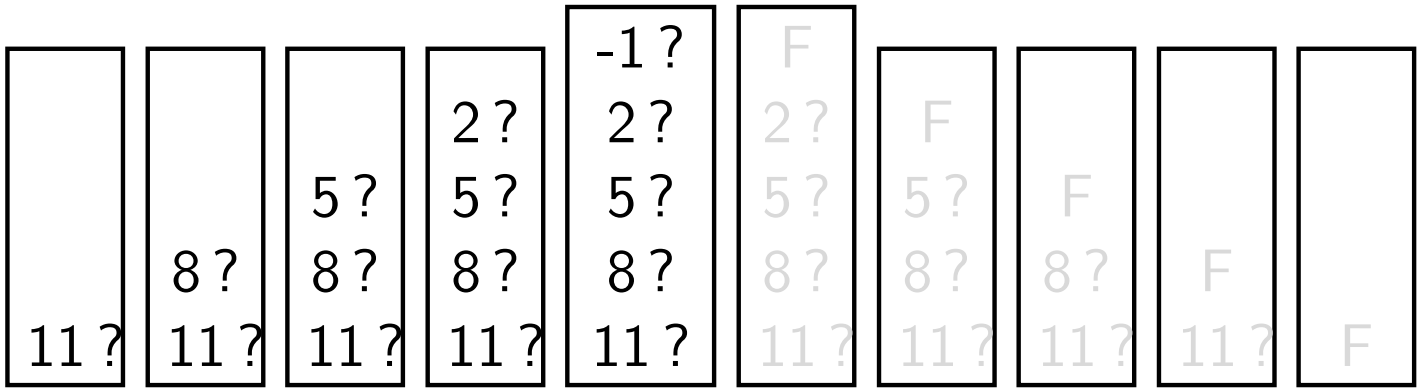
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

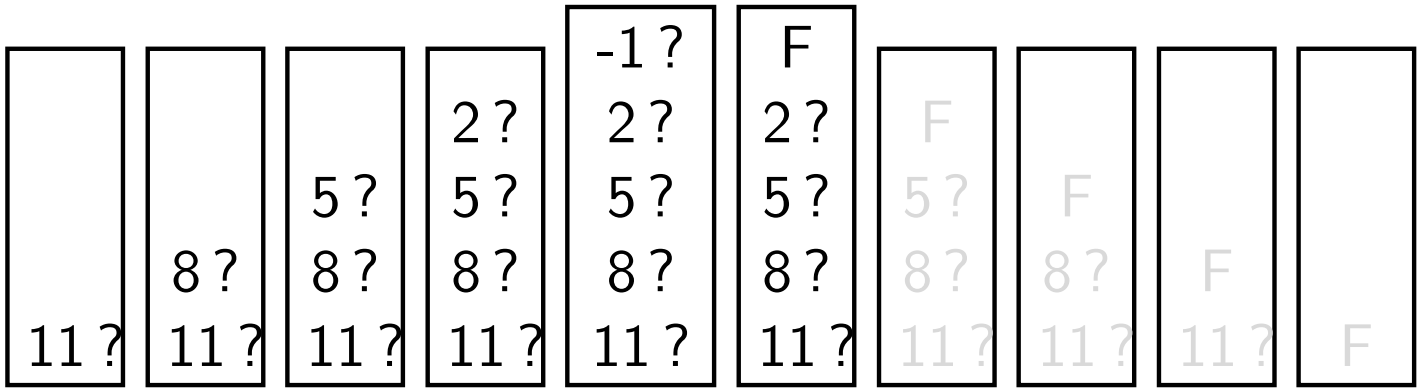
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

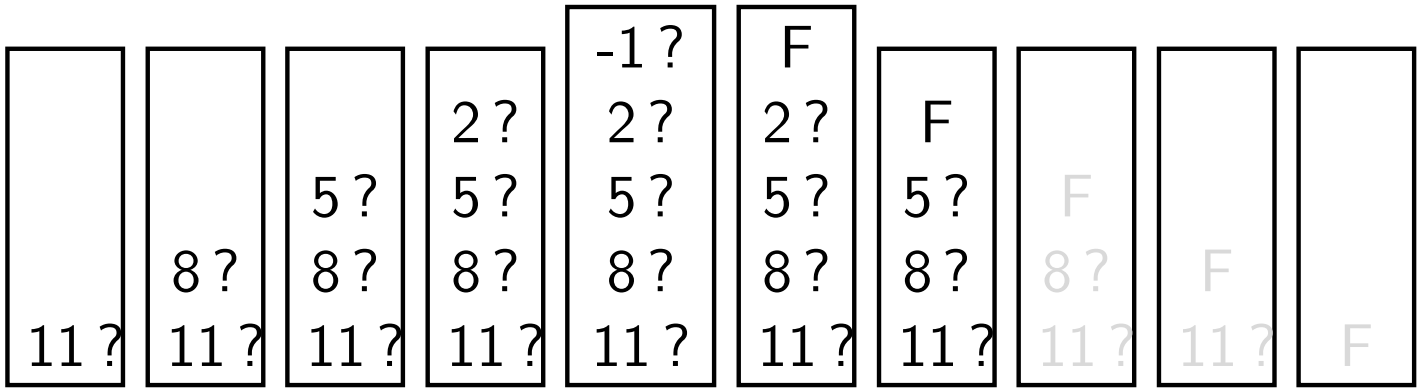
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

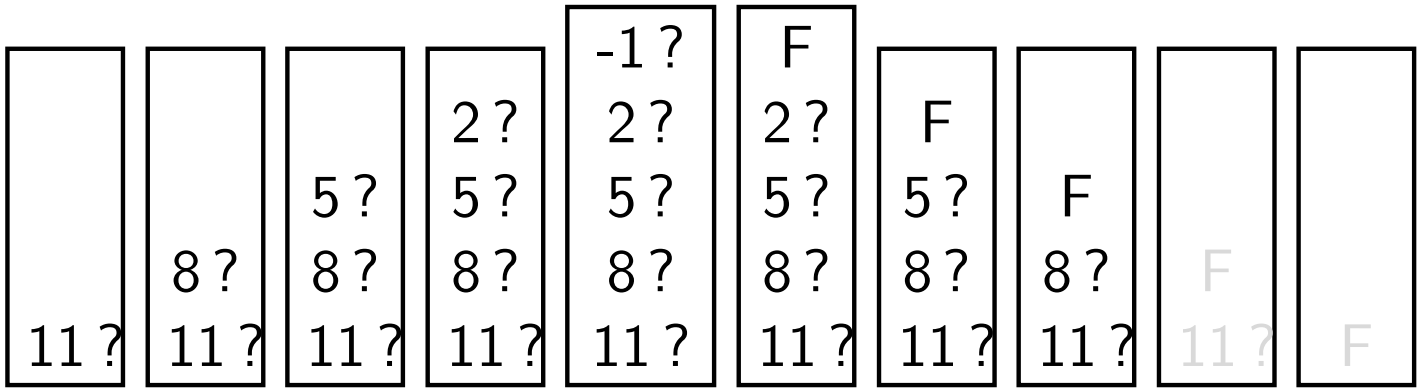
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

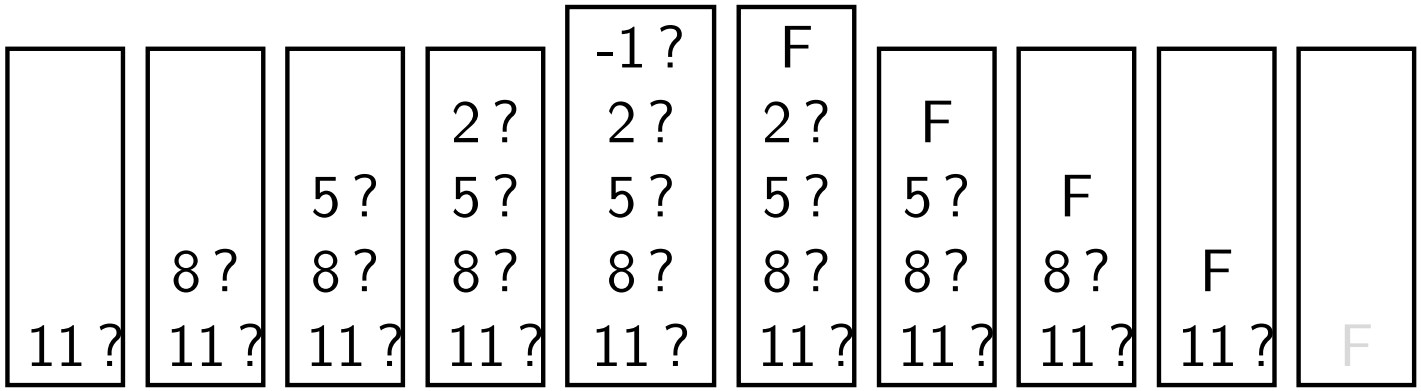
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

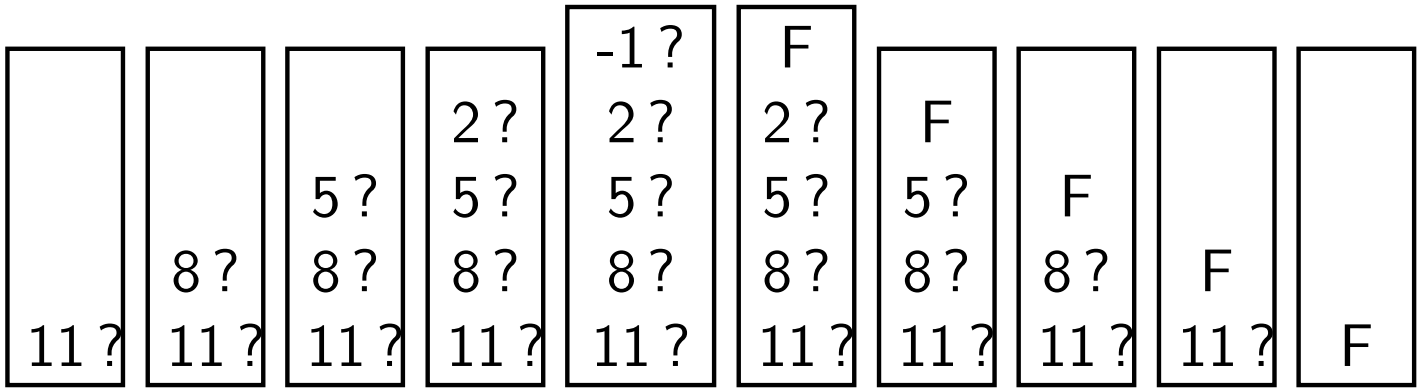
```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

```
function f(n: integer) return boolean is
begin
if n<0 then return false;
elsif n=0 then return true;
else return f(n-3); end if;
end f;
```



Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

pile

Une pile est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font au même bout de la structure : son *sommet*.
On va pouvoir implémenter une pile grâce aux tableaux et écrire un paquetage pour manipuler des piles.
Ce paquetage va

- définir le type `pile`
- fournir les fonctionnalités nécessaires pour manipuler les piles en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de pile vide
  - lecture du sommet

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

pile

Une pile est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font au même bout de la structure : son *sommet*.
On va pouvoir implémenter une pile grâce aux tableaux et écrire un paquetage pour manipuler des piles.
Ce paquetage va

- définir le type `pile`
- fournir les fonctionnalités nécessaires pour manipuler les piles en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de pile vide
  - lecture du sommet

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

pile

Une pile est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font au même bout de la structure : son *sommet*.
On va pouvoir implémenter une pile grâce aux tableaux et écrire un paquetage pour manipuler des piles.
Ce paquetage va

- définir le type `pile`
- fournir les fonctionnalités nécessaires pour manipuler les piles en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de pile vide
  - lecture du sommet

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

pile

Une pile est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font au même bout de la structure : son *sommet*.
On va pouvoir implémenter une pile grâce aux tableaux et écrire un paquetage pour manipuler des piles.
Ce paquetage va

- définir le type `pile`
- fournir les fonctionnalités nécessaires pour manipuler les piles en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de pile vide
  - lecture du sommet

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

pile

Une pile est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font au même bout de la structure : son *sommet*.
On va pouvoir implémenter une pile grâce aux tableaux et écrire un paquetage pour manipuler des piles.
Ce paquetage va

- définir le type `pile`
- fournir les fonctionnalités nécessaires pour manipuler les piles en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de pile vide
  - lecture du sommet

Rappel sur les passages de paramètres

Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

le fichier pileentier.ads

```
package pileentier is
type contenant is array (natural range <>) of integer ;
type pile is record
tabpile : contenant(0..255) ;
top : integer ;
end record ;
procedure init (p : out pile) ;
procedure ajouter (p : in out pile ; e :in integer ) ;
procedure supprimer (p : in out pile) ;
function sommet (p : pile) return integer ;
function vide (p : pile) return boolean ;
end pileentier ;
```

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

le fichier pileentier.adb

```
package body pileentier is
procedure init (p : out pile) is
begin
p.top := -1; end init;
procedure ajouter (p : in out pile ; e :in integer ) is
begin
p.tabpile(p.top+1) := e ;
p.top := p.top+1 ; end ajouter ;
procedure supprimer (p : in out pile) is
begin
p.top := p.top-1 ; end supprimer ;
function sommet (p : pile) return integer is
begin
return (p.tabpile(p.top)) ; end sommet ;
function vide (p : pile) return boolean is
begin
return (p.top=-1) ; end vide ;
end pileentier ;
```

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Cette version pose certains problèmes :

1. elle ne fonctionne que pour des piles d'entiers alors que le mécanisme de la pile est indépendant du type des valeurs qu'elle contient
2. ajouter(p,e) provoque une erreur si p.top $= 255$
3. sommet(p) ou supprimer(p) provoque une erreur si p.top $= -1$
4. la déclaration d'une pile ne l'initialise pas et donc on ne peut appeler aucune procédure ou fonction si l'on n'a pas initialisé préalablement la pile.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Cette version pose certains problèmes :

1. elle ne fonctionne que pour des piles d'entiers alors que le mécanisme de la pile est indépendant du type des valeurs qu'elle contient
2. ajouter(p,e) provoque une erreur si p.top $= 255$
3. sommet(p) ou supprimer(p) provoque une erreur si p.top $= -1$
4. la déclaration d'une pile ne l'initialise pas et donc on ne peut appeler aucune procédure ou fonction si l'on n'a pas initialisé préalablement la pile.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Cette version pose certains problèmes :

1. elle ne fonctionne que pour des piles d'entiers alors que le mécanisme de la pile est indépendant du type des valeurs qu'elle contient
2. ajouter(p,e) provoque une erreur si p.top $= 255$
3. sommet(p) ou supprimer(p) provoque une erreur si p.top $= -1$
4. la déclaration d'une pile ne l'initialise pas et donc on ne peut appeler aucune procédure ou fonction si l'on n'a pas initialisé préalablement la pile.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Cette version pose certains problèmes :

1. elle ne fonctionne que pour des piles d'entiers alors que le mécanisme de la pile est indépendant du type des valeurs qu'elle contient
2. ajouter(p,e) provoque une erreur si p.top $= 255$
3. sommet(p) ou supprimer(p) provoque une erreur si p.top $= -1$
4. la déclaration d'une pile ne l'initialise pas et donc on ne peut appeler aucune procédure ou fonction si l'on n'a pas initialisé préalablement la pile.

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

$\rightsquigarrow$Pour le point 3, l'utilisateur à qui on a fourni les spécifications doit connaître les préconditions et donc un usage impropre de *sommet* ou *supprimer* est de sa responsabilité (d'autant qu'il a le moyen de tester si une pile est vide).

$\rightsquigarrow$Pour le point 4, l'existence de *init* doit s'accompagner d'un commentaire pour faire comprendre que son usage est indispensable pour toute pile déclarée.

On peut aussi changer les spécifications dans la déclaration du type pile : on ajoute `top : integer :=-1;` et dans ce cas `init` deviendra `reset`

$\rightsquigarrow$Pour le point 2 (dual du 3), on peut enrichir les spécifications avec en plus dans la signature

$$\textit{pilepleine} : \text{PILE} \rightarrow \text{BOOLÉEN}$$

$\rightsquigarrow$pour le point 1, on utilisera le principe de généricité

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

$\rightsquigarrow$Pour le point 3, l'utilisateur à qui on a fourni les spécifications doit connaître les préconditions et donc un usage impropre de *sommet* ou *supprimer* est de sa responsabilité (d'autant qu'il a le moyen de tester si une pile est vide).
$\rightsquigarrow$Pour le point 4, l'existence de *init* doit s'accompagner d'un commentaire pour faire comprendre que son usage est indispensable pour toute pile déclarée.
On peut aussi changer les spécifications dans la déclaration du type pile : on ajoute `top : integer :=-1;` et dans ce cas `init` deviendra `reset`
$\rightsquigarrow$Pour le point 2 (dual du 3), on peut enrichir les spécifications avec en plus dans la signature

$$\textit{pilepleine} : \text{PILE} \rightarrow \text{BOOLÉEN}$$

$\rightsquigarrow$pour le point 1, on utilisera le principe de généricité

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

$\rightsquigarrow$Pour le point 3, l'utilisateur à qui on a fourni les spécifications doit connaître les préconditions et donc un usage impropre de *sommet* ou *supprimer* est de sa responsabilité (d'autant qu'il a le moyen de tester si une pile est vide).
$\rightsquigarrow$Pour le point 4, l'existence de *init* doit s'accompagner d'un commentaire pour faire comprendre que son usage est indispensable pour toute pile déclarée.
On peut aussi changer les spécifications dans la déclaration du type pile : on ajoute `top : integer :=-1;` et dans ce cas `init` deviendra `reset`
$\rightsquigarrow$Pour le point 2 (dual du 3), on peut enrichir les spécifications avec en plus dans la signature

$$\textit{pilepleine} : \text{PILE} \rightarrow \text{BOOLÉEN}$$

$\rightsquigarrow$pour le point 1, on utilisera le principe de généricité

Rappel sur les passages de paramètres
Paquetages
généralités
syntaxe
Le paquetage tableau1
Utilisation
Recherche dans un tableau
Recherche dichotomique
Tris simples
Complexité des tris simples
nombre minimal nécessaire
utilisation des tableaux : les piles
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

$\rightsquigarrow$Pour le point 3, l'utilisateur à qui on a fourni les spécifications doit connaître les préconditions et donc un usage impropre de *sommet* ou *supprimer* est de sa responsabilité (d'autant qu'il a le moyen de tester si une pile est vide).
$\rightsquigarrow$Pour le point 4, l'existence de *init* doit s'accompagner d'un commentaire pour faire comprendre que son usage est indispensable pour toute pile déclarée.
On peut aussi changer les spécifications dans la déclaration du type pile : on ajoute `top : integer :=-1;` et dans ce cas `init` deviendra `reset`
$\rightsquigarrow$Pour le point 2 (dual du 3), on peut enrichir les spécifications avec en plus dans la signature

$$\textit{pilepleine} : \text{PILE} \rightarrow \text{BOOLÉEN}$$

$\rightsquigarrow$pour le point 1, on utilisera le principe de généricité

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Les piles fonctionnent de la même façon quelque soit le type des valeurs qu'elles contiennent.
Ada a un mécanisme d'abstraction qui permet de définir des paquetages *génériques* qui seront *paramétrés* par un certain nombre d'informations.
En particulier un type peut être générique et c'est ce qui nous intéresse en ce qui concerne le fonctionnement des piles. On va définir un type *privé*.
Ce paramétrage permet donc une réutilisation du paquetage générique par *instanciation* du type des valeurs que contient la pile.
On va rajouter au début de notre fichier `pilegenerique.ads` les lignes suivantes :

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

```
generic type item is private ;
package pilegenerique is ....
```

- `generic` indique que le paquetage est générique
- le paramètre *item* est générique
- il est *privé* et le paquetage ne le connaît pas
- on réécrit le code en remplaçant `integer` par `item`
- `pilegenerique` est le nom de ce paquetage

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

```
generic type item is private ;
package pilegenerique is ....
```

- `generic` indique que le paquetage est générique
- le paramètre *item* est générique
- il est *privé* et le paquetage ne le connaît pas
- on réécrit le code en remplaçant `integer` par `item`
- `pilegenerique` est le nom de ce paquetage

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

```
generic type item is private ;
package pilegenerique is ....
```

- `generic` indique que le paquetage est générique
- le paramètre *item* est générique
- il est *privé* et le paquetage ne le connaît pas
- on réécrit le code en remplaçant `integer` par `item`
- `pilegenerique` est le nom de ce paquetage

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

```
generic type item is private ;
package pilegenerique is ....
```

- `generic` indique que le paquetage est générique
- le paramètre *item* est générique
- il est *privé* et le paquetage ne le connaît pas
- on réécrit le code en remplaçant `integer` par `item`
- `pilegenerique` est le nom de ce paquetage

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

```
generic type item is private ;
package pilegenerique is ....
```

- `generic` indique que le paquetage est générique
- le paramètre *item* est générique
- il est *privé* et le paquetage ne le connaît pas
- on réécrit le code en remplaçant `integer` par `item`
- `pilegenerique` est le nom de ce paquetage

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Pour utiliser de tels paquetages il faudra *instancier* ce qui est générique – soit ici le type – de la façon suivante :

```
with pilegenerique ;
procedure essaipile_gen is
package mapiledentier is new pilegenerique(integer ) ;
-- on instancie le type générique item par integer
package mapiledeflottant is new pilegenerique(float ) ;
-- on instancie le type générique item par float
use mapiledentier ;
use mapiledeflottant ;
Ipile : mapiledentier.pile ;
-- on doit préfixer le type générique pile
-- par le paquetage voulu
Fpile : mapiledeflottant.pile ;
begin
init(Ipile) ; init(Fpile) ;
-- il est inutile de préfixer les fonctionnalités
-- car le compilateur fera la distinction
-- grâce au type du paramètre
ajouter(Ipile , 3) ;
ajouter(Ipile , 1) ;
supprimer(Ipile) ;
ajouter(Fpile , 2.0) ;
put (sommet(Ipile)+ integer (sommet(Fpile))) ;
end essaipile_gen ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

On a toujours le problème de limitation de la taille de la pile.
De plus le type `item` étant utilisé dans un tableau de type `contenant` doit être contraint ; on ne peut donc pas instancier le type `item` à un type tableau ou chaîne de caractères.
Enfin, dernière contrariété, l'utilisateur connaissant la construction du type `pile` peut contourner les fonctionnalités proposées par le paquetage et créer ses propres sous-programmes – par exemple utiliser le champ `top` de la pile pour connaître le nombre d'éléments de la pile.
Le problème viendra alors lorsque la version du paquetage `pile` sera modifiée. L'utilisateur devra alors changer tout son programme.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

On a toujours le problème de limitation de la taille de la pile.
De plus le type `item` étant utilisé dans un tableau de type `contenant` doit être contraint ; on ne peut donc pas instancier le type `item` à un type tableau ou chaîne de caractères.
Enfin, dernière contrariété, l'utilisateur connaissant la construction du type `pile` peut contourner les fonctionnalités proposées par le paquetage et créer ses propres sous-programmes – par exemple utiliser le champ `top` de la pile pour connaître le nombre d'éléments de la pile.
Le problème viendra alors lorsque la version du paquetage `pile` sera modifiée. L'utilisateur devra alors changer tout son programme.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exercices

Pour les exercices suivants vous envisagerez deux points de vue :

- l'utilisateur qui se sert uniquement des fonctionnalités fournies par le paquetage
- le concepteur qui inclut dans le paquetage une nouvelle fonctionnalité (et donc est en droit d'utiliser la représentation physique de la pile).

1. Ecrire une procédure qui inverse une pile (la pile passée en paramètre doit être dans le même état après exécution de cette procédure)
2. Ecrire une procédure qui copie une pile (la pile passée en paramètre doit être dans le même état après exécution de cette procédure)
3. Ecrire une procédure qui trie une pile d'entiers (le plus petit étant au sommet) à l'aide d'une autre pile.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

file

Une file est une structure de données dans laquelle l'ajout se fait à une extrémité – la fin (ou queue) – et la suppression se fait au début de la structure – la tête.
Pour une file vide, après ajout d'un élément celui-ci est à la fois début et fin ; un deuxième élément viendra derrière lui et il sera la fin de la file.
Puisque la suppression se fait au début de la file, l'élément supprimé est nécessairement le premier élément à avoir été enfilé. Une file est aussi désignée par l'acronyme anglais FIFO (**F**irst **I**n **F**irst **O**ut).
Une file sert à gérer une politique d'accès pour laquelle la priorité est donnée par l'ordre d'arrivée.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

file

Une file est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font aux deux bouts de la structure.
Un paquetage définissant des files devra

- définir le type `file`
- fournir les fonctionnalités nécessaires pour manipuler les files en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de file vide
  - lecture de la tête de file

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

file

Une file est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font aux deux bouts de la structure.
Un paquetage définissant des files devra

- définir le type `file`
- fournir les fonctionnalités nécessaires pour manipuler les files en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de file vide
  - lecture de la tête de file

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

file

Une file est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font aux deux bouts de la structure.
Un paquetage définissant des files devra

- définir le type `file`
- fournir les fonctionnalités nécessaires pour manipuler les files en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de file vide
  - lecture de la tête de file

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

file

Une file est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font aux deux bouts de la structure.
Un paquetage définissant des files devra

- définir le type `file`
- fournir les fonctionnalités nécessaires pour manipuler les files en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de file vide
  - lecture de la tête de file

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

file

Une file est donc un ensemble de données qui a une structure linéaire dans laquelle ajout et suppression se font aux deux bouts de la structure.
Un paquetage définissant des files devra

- définir le type `file`
- fournir les fonctionnalités nécessaires pour manipuler les files en respectant leur fonctionnement :
  - initialisation
  - ajout
  - suppression
  - test de file vide
  - lecture de la tête de file

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

description

Pour représenter une file on utilisera tout d'abord un tableau pour stocker les valeurs de type `item`.
Il nous faut de plus un moyen de repérer le début et la fin de la file ; on utilisera donc deux repères sur le tableau qui contiendront respectivement l'indice de la valeur au début de la file et l'indice de la dernière valeur entrée.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exemple d'usage

0	1	2	3	4	5	...	253	254	255
			x	x	x	...	x		

en rouge le début de la file, en bleu la fin.
Ici il y a eu 254 ajouts et 3 suppressions.
Assez naturellement on peut définir

- idébut comme l'indice de la valeur en début de file. Et à chaque suppression cet indice est incrémenté.
- ifin comme l'indice de la valeur en fin de file. Et à chaque insertion cet indice est incrémenté.

Que se passe-t-il s'il y a 3 ajouts ?
plus de place ?
Une solution ?
on utilise le tableau en anneau pour utiliser les cellules qui ont été libérées par des suppressions.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exemple d'usage

0	1	2	3	4	5	...	253	254	255
			x	x	x	...	x		

en rouge le début de la file, en bleu la fin.
Ici il y a eu 254 ajouts et 3 suppressions.
Assez naturellement on peut définir

- idébut comme l'indice de la valeur en début de file. Et à chaque suppression cet indice est incrémenté.
- ifin comme l'indice de la valeur en fin de file. Et à chaque insertion cet indice est incrémenté.

Que se passe-t-il s'il y a 3 ajouts ?
plus de place ?
Une solution ?
on utilise le tableau en anneau pour utiliser les cellules qui ont été libérées par des suppressions.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exemple d'usage

0	1	2	3	4	5	...	253	254	255
			x	x	x	...	x		

en rouge le début de la file, en bleu la fin.
Ici il y a eu 254 ajouts et 3 suppressions.
Assez naturellement on peut définir

- idébut comme l'indice de la valeur en début de file. Et à chaque suppression cet indice est incrémenté.
- ifin comme l'indice de la valeur en fin de file. Et à chaque insertion cet indice est incrémenté.

Que se passe-t-il s'il y a 3 ajouts ?
plus de place ?
Une solution ?
on utilise le tableau en anneau pour utiliser les cellules qui ont été libérées par des suppressions.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exemple d'usage

0	1	2	3	4	5	...	253	254	255
			x	x	x	...	x		

en rouge le début de la file, en bleu la fin.
Ici il y a eu 254 ajouts et 3 suppressions.
Assez naturellement on peut définir

- idébut comme l'indice de la valeur en début de file. Et à chaque suppression cet indice est incrémenté.
- ifin comme l'indice de la valeur en fin de file. Et à chaque insertion cet indice est incrémenté.

Que se passe-t-il s'il y a 3 ajouts ?
plus de place ?
Une solution ?
on utilise le tableau en anneau pour utiliser les cellules qui ont été libérées par des suppressions.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exemple d'usage

On continue avec 4 ajouts

0	1	2	3	4	5	...	253	254	255
x	x		x	x	x	...	x	x	x

On poursuite avec 254 suppressions.

0	1	2	3	4	5	...	253	254	255
	x					...			

idébut et ifin sont identiques.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exemple d'usage

Maintenant on supprime la dernière valeur.

0	1	2	3	4	5	...	253	254	255
						...			

idébut et ifin se sont croisés et la file est vide.

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exemple d'usage

Maintenant on ajoute 256 valeurs.

0	1	2	3	4	5	...	253	254	255
x	x	x	x	x	x	...	x	x	x

idébut et ifin se sont croisés et la file est pleine.
idébut et ifin ont les mêmes valeurs respectives que lorsqu'elle était vide.
Les choses se compliquent.
solution : on utilise

- *soit un champ supplémentaire pour la longueur*
- *soit une case tampon qui ne pourra pas contenir de valeur*

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

le fichier filegenerique.ads

```
generic type item is private ;
package filegenerique is
type contenant is array (natural range <>) of item ;
NB_MAX: constant positive := 256 ;
type file is record
tabfile : contenant(0..NB_MAX-1) ;
top : natural:=0 ;
bottom : natural:=0;
nbitem : natural :=0;
end record ;
-- la déclaration initialise la file
procedure reinit (f : in out file) ;
procedure ajouter (f : in out file ; e :in item) ;
procedure supprimer (f : in out file) ;
function premier (f : file) return item ;
function vide (f : file) return boolean ;
function filepleine (f :file) return boolean ;
end filegenerique ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

```
package body filegenerique is
function suivant (f : file ; n : integer ) return natural is
begin
if n= NB_MAX-1 then return 0 ; else return (n+1) ; end if ;
end suivant ; -- fct utilitaire
procedure reinit (f :  in out file) is
begin
f.top := 0; f.bottom := 0;  f.nbitem :=0; end reinit ;
procedure ajouter (f : in out file ; e :in item) is
begin
f.tabfile(f.bottom) := e ;
f.bottom := suivant(f,f.bottom) ;
f.nbitem := f.nbitem +1;
end ajouter ;
procedure supprimer (f : in out file) is
begin f.top := suivant(f,f.top) ;
          f.nbitem := f.nbitem -1;
end supprimer ;
function premier (f : file) return item is
begin return (f.tabfile(f.top)) ; end premier ;
function vide (f : file) return boolean is
begin return (f.nbitem=0) ; end vide ;
function filepleine (f : file) return boolean is
begin return (f.nbitem=NB_MAX) ;
end filepleine ;
end filegenerique ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
un paquetage pile générique
les files
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exercices

1. Ecrire une procédure qui inverse une file.
2. Ecrire une procédure qui inverse une pile avec une file.
3. Réécrire les fonctionnalités sur les files en vous servant de 2 piles pour simuler une file.
4. *Séparer le bon grain de l'ivraie* : on considère un tableau d'entiers t. **Sans trier le tableau**, trouver un moyen de placer toutes les valeurs négatives au début du tableau en un nombre d'opérations « élémentaires » proportionnel à la taille du tableau.

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Exemple

Exceptions prédéfinies

Déclarer une exception

Traitement d'une exception

lever

traiter

exemple

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Exceptions

Lors de la compilation ou de l'exécution d'un programme, certaines instructions ne peuvent être exécutées et la compilation ou l'exécution est stoppée accompagnée d'un message d'erreur, par exemple *static expression raises "constraint_error"*. On dit qu'*une exception est levée*.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Exemple
Exceptions prédéfinies
Déclarer une exception
Traitement d'une exception
lever
traiter
exemple
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

Exemple

```
type arcenciel is (violet, indigo, bleu, vert, jaune,
 orange, rouge) ;
couleur : arcenciel := rouge;
nuance : arcenciel ;
begin
nuance := arcenciel'succ(couleur);
end ;
```

L'appel à `arcenciel'succ(couleur);` provoque une erreur puisque dans le type `arcenciel`, `rouge` n'a pas de successeur. Il y a différentes exceptions selon les règles du langage qui ne sont pas respectées et un programme peut envisager les erreurs possibles et ainsi prendre en compte ces éventuelles exceptions.

```
bloc_couleur: begin
nuance := arcenciel'succ(couleur);
exception when constraint_error => arcenciel'first ;
end bloc_couleur ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Exemple
Exceptions prédéfinies
Déclarer une exception
Traitement d'une exception
lever
traiter
exemple
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exceptions prédéfinies

- constraint_error : lorsqu'on sort des bornes ou bien lorsqu'on exécute une opération arithmétique non valide comme une division par zéro
- program_error : lorsqu'on ne respecte pas une structure de contrôle comme un appel à un sous-programme non encore élaboré
- storage_error : lorsqu'on manque d'espace mémoire
- data_error : lorsqu'on a lu une valeur ne correspondant pas au type déclaré ; elle est contenue dans le paquetage Text_IO
- tasking_error : cela concerne les *tâches* qui sont hors de notre propos

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Exemple
Exceptions prédéfinies
Déclarer une exception
Traitement d'une exception
lever
traiter
exemple
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

déclarer une exception

Pour déclarer une exception on utilise le type `exception`.
On peut déclarer une exception par un identificateur dans un *bloc* et ainsi définir la portée de cette exception limitée au bloc.
La syntaxe est la suivante :

```
monbloc :
declare monexception : exception ;
begin
-- instructions
end monbloc;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Exemple
Exceptions prédéfinies
Déclarer une exception
Traitement d'une exception
lever
traiter
exemple
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

lever une exception

Les exceptions prédéfinies sont levées implicitement à l'exécution.
Pour les exceptions déclarées par le programmeur, il faut les lever explicitement par l'instruction :

```
raise identificateur_exception ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Exemple
Exceptions prédéfinies
Déclarer une exception
Traitement d'une exception
lever
traiter
exemple
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

traiter une exception

traiter une exception c'est prendre en compte et gérer cette exception là où elle peut être levée.
Dès qu'une exception est levée, les instructions de traitement correspondant à l'exception levée prennent la main. Le bloc des instructions qui gèrent l'exception est introduit par l'instruction :

```
exception when identificateur_exception1 => ...;
when identificateur_exception2 => ...;
...
when others => ...;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Exemple
Exceptions prédéfinies
Déclarer une exception
Traitement d'une exception
lever
traiter
exemple
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

traiter une exception (suite)

L'exécution de l'unité de programme est donc abandonnée là où est levée l'exception au profit de son traitement.
Une exception prédéfinie peut être propagée si elle n'est pas correctement traitée dans l'unité où elle a été levée ; elle devra alors être traitée dans une unité appelant la précédente.
Une exception déclarée et levée dans un bloc (ou un sous-programme) devra être traitée dans ce bloc (ou ce sous-programme) : elle ne peut être attrapée en dehors.
Dans tous les cas si une exception levée n'est pas traitée alors l'exécution du programme est arrêtée avec un message d'erreur.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Exemple
Exceptions prédéfinies
Déclarer une exception
Traitement d'une exception
lever
traiter
exemple
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

exemple

```
loop
bloc_saisie :
declare borne : exception ;
begin
put ("donner un entier entre 1 et 10") ;
get (n) ;
new_line ;
if not (1<=n and n<=10) then raise borne ; end if ;
exit ;
exception when data_error =>
skip_line;
put_line ("ce n'est pas un entier ; recommencez") ;
when borne =>
skip_line;
put_line ("entre 1 et 10 ! recommencez !");
end bloc_saisie ;
end loop;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Exemple
Exceptions prédéfinies
Déclarer une exception
Traitement d'une exception
lever
traiter
exemple
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

L'exception data_error est levée à l'exécution de `get` si autre chose qu'un entier est saisi au clavier.
L'exception borne est levée à l'exécution de `if` si l'entier saisi n'est pas compris entre 1 et 10.
L'instruction `skip_line` permet de vider le *buffer* d'entrée c'est-à-dire le fichier dans lequel sont momentanément stockées les valeurs saisies au clavier.
Remarquez bien que l'instruction `exit` n'est pas exécutée si une exception est levée ; par conséquent la boucle `loop` se poursuit jusqu'à ce qu'il n'y ait plus d'exception levée.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Variables dynamiques

Lors de l'exécution d'un programme les variables déclarées du programme sont stockées dans un espace mémoire géré comme une pile et appelé segment de données.
Ce segment de données a une taille limitée selon les versions du compilateur et du système d'exploitation, néanmoins elle peut être modifiée à ce niveau ; mais la place ainsi occupée par une variable s'avérant inutile ne peut pas être utilisée.
Pour cette raison l'utilisation de gros tableaux à dimension fixe peut surcharger le segment de données et rendre impossible l'exécution du programme.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

De plus, la dimension fixe d'un tableau peut induire des problèmes : la taille choisie est trop importante à posteriori et on a gâché de la place ou bien on peut manquer de place dans le cas contraire, situations qui compliquent la mise au point du programme.
En outre la portabilité du programme en question s'en trouve réduite puisqu'il a nécessité une éventuelle modification de la taille du segment de données au niveau du système d'exploitation.
Pour remédier à cet inconvénient, on va utiliser des variables *dynamiques* c'est-à-dire des variables qui pourront être créées ou détruites **en cours d'exécution du programme**.
Il existe un type qui correspond aux adresses mémoire : le type `access` ; il permet de manipuler les adresses de données qui peuvent être créées et détruites au cours de l'exécution d'un programme.
Les adresses mémoire seront codés sur 4 octets et stockés
Ces variables de type `access` seront gérées dans un autre espace mémoire appelé *tas*.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Lorsqu'on déclare une variable avec son identificateur et son type alors dans le segment de données sont fixées

- l'espace mémoire pour l'identificateur
- l'adresse d'un espace mémoire pour la valeur associée à cet identificateur
- l'espace mémoire pour la valeur

ce qui correspond au schéma suivant :

Lorsqu'on manipule `identificateur` on manipule sa valeur : par exemple `n: integer := 0; begin n := n+1 ; ...` change 0 en 1 pour `n`.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Pour le type `access` la valeur manipulée sera une adresse mémoire. Cette adresse mémoire mène à un espace mémoire qui n'est pas dans segment de données mais le *tas*. Cet espace mémoire aura une dimension fixé par la déclaration de la variable de type `access`.
Une variable de type access sera codé
Donc on doit déclarer un type `access` en **précisant le type de la donnée** dont le type `access` sera l'adresse.
Quelle que soit le type de la donnée pointée une variable de type `access` sera codée sur 4 octets.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

syntaxe de la déclaration

Soit `t_quelconque` un type quelconque. Les variables du type `access` suivant auront comme valeur les **adresses** d'objets de type `t_quelconque`.
remarque : une référence sur un type `t_quelconque` **non contraint** est possible.

```
type p_t_quelconque is access t_quelconque;
p1, p2, p3 : p_t_quelconque;
-- p1, p2 et p3 seront les
--adresses d'objets de type t_quelconque
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Création

Les trois variables `p1, p2 et p3` ne permettent pour l'instant l'accès à aucun objet parce qu'aucun espace mémoire n'a été réservé pour stocker des objets de type `t_quelconque` dont `p1, p2, p3` seraient les adresses. Leur valeur est automatiquement initialisée à une valeur `null` qui est l'adresse de *rien*.

On a donc juste déclaré et réservé une adresse (ou un lien ou une référence) qui n'est pas encore « activé » .

A tout moment du programme on peut créer des objets de type `p_t_quelconque` par l'opérateur `new` portant sur le type `t_quelconque`, la valeur retournée par `new` sera l'adresse de l'espace mémoire alloué par le système pour gérer la donnée de type `t_quelconque`.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Création

Les trois variables `p1, p2 et p3` ne permettent pour l'instant l'accès à aucun objet parce qu'aucun espace mémoire n'a été réservé pour stocker des objets de type `t_quelconque` dont `p1, p2, p3` seraient les adresses. Leur valeur est automatiquement initialisée à une valeur `null` qui est l'adresse de *rien*.

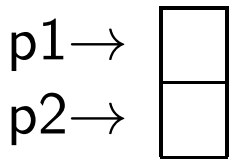
On a donc juste déclaré et réservé une adresse (ou un lien ou une référence) qui n'est pas encore « activé » .

A tout moment du programme on peut créer des objets de type `p_t_quelconque` par l'opérateur `new` portant sur le type `t_quelconque`, la valeur retournée par `new` sera l'adresse de l'espace mémoire alloué par le système pour gérer la donnée de type `t_quelconque`.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
type p_entier is access integer;
p1, p2, p3 : p_entier;
p1 :=  new integer; p2 := new integer;
```

deux espaces mémoire sont réservés pour contenir des entiers ; aucune valeur n'est stockée pour l'instant.
Chaque espace mémoire sera accessible par sa référence `p1` ou p2.

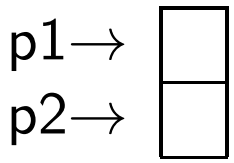


On peut ensuite affecter une valeur à cet espace mémoire par l'opérateur `all`.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
type p_entier is access integer;
p1, p2, p3 : p_entier;
p1 :=  new integer; p2 := new integer;
```

deux espaces mémoire sont réservés pour contenir des entiers ; aucune valeur n'est stockée pour l'instant.
Chaque espace mémoire sera accessible par sa référence `p1` ou p2.

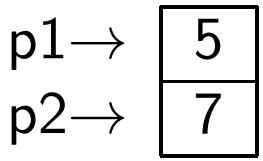


On peut ensuite affecter une valeur à cet espace mémoire par l'opérateur `all`.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Affectation

```
p1.all := 5 ; p2.all := 7;
-- p1 est l'adresse d'un espace mémoire contenant 5
-- p2 est l'adresse d'un espace mémoire contenant  7
```



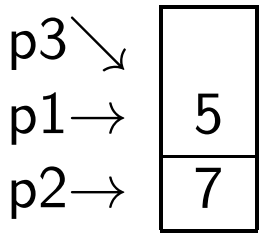
remarque : si `t_quelconque` est **non contraint**, au moment de l'utilisation de `new` il faudra **fixer les contraintes** car le compilateur doit connaître la taille de la zone mémoire à réserver.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
p3 := p1 ;
-- p1 et p3 sont du même type
-- donc cette affectation est valide
```

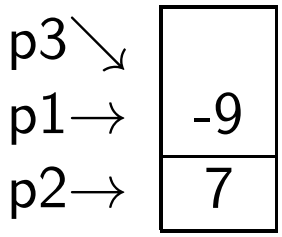
Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
p3 := p1 ;
-- p1 et p3 sont du même type
-- donc cette affectation est valide
```



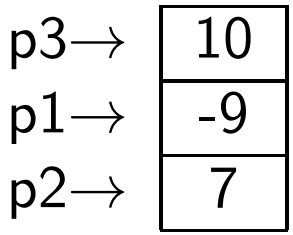
Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
p3.all := -9 ;
-- p3 et p1 pointent sur
-- un espace mémoire contenant -9
```



Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

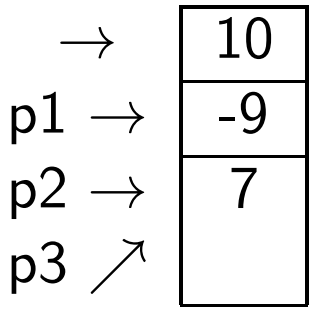
```
p3 := new integer'10 ;
-- création et affectation en une unique instruction
```



Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
p3:= p2 ;
```

l'accès à l'espace mémoire contenant 10 est **perdu.**



Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Récupération de la mémoire

Pour pouvoir, en cours d'exécution, récupérer la place mémoire occupée par un objet devenu inutile, il est nécessaire de *désallouer* spécifiquement cet espace par l'appel d'une procédure. Il faut prévoir

- la clause `with Ada.Unchecked_Deallocation;`
- déclarer une procédure de désallocation spécifique aux types `p_t_quelconque` et `t_quelconque`

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Exemple (suite)

```
procedure dispose is new
        Ada.Unchecked_Deallocation(integer, p_entier);
...
dispose(p2);
```

L'espace mémoire contenant 7 est rendu disponible et pourra être réutilisé ultérieurement.

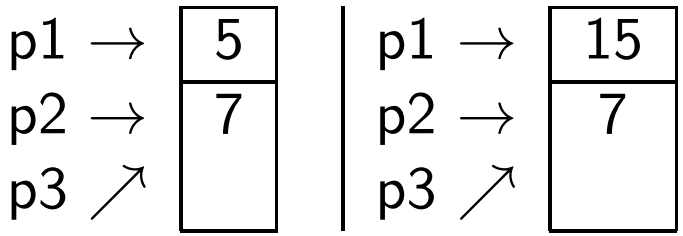
Attention : l'utilisation de p2 ou p3 conduit à une erreur car ils ne référencent plus d'espace mémoire.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Opérations

Les seules opérations permises sur un type access sont les tests d'égalité et de différence.

```
type p_entier is access integer;
p1, p2, p3 : p_entier;
p1 :=  new integer; p2 := new integer;
p1.all := 5 ; p2.all := 7;p3 := p2 ;
if p2=p3 then
p1.all := p1.all + p2.all + 3;end if;
```



Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

paquetage pilegenerique

```
generic type item is private ;
package pilegenerique is
type contenant is array (natural range <>) of item ;
type pile is record
tabpile : contenant(0..255) ;
top : integer:=-1 ;
end record ;
procedure reinit (p : in out pile) ;
procedure ajouter (p : in out pile ; e :in item ) ;
procedure supprimer (p : in out pile) ;
function sommet (p : pile) return item ;
function vide (p : pile) return boolean ;
end pileentier ;
```

Un défaut de ce paquetage qu'on n'a pas encore relevé est le suivant : le type générique `item` étant le type des valeurs d'un tableau il doit être **contraint**. On ne peut pas, par exemple, empiler des tableaux de longueurs variées.
Le type `access` est un type contraint : toute adresse sera codé sur 4 octets. Donc on pourra empiler des tableaux de longueurs varaibles par l'intermédiaire de pointeurs sur ces tableaux.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

paquetage pilegenerique2.ads

```
generic type item(<>) is private ;
package pilegenerique2 is
type itemref is access item;
type contenant is array (natural range <>) of itemref ;
type pile is record
tabpile : contenant(0..255) ;
top : integer:=-1 ;
end record ;
procedure reinit (p : in out pile) ;
procedure ajouter (p : in out pile ; e :in item ) ;
procedure supprimer (p : in out pile) ;
function sommet (p : pile) return item ;
function vide (p : pile) return boolean ;
end pilegenerique2 ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

pilegenerique2.adb

```
with ada.Unchecked_Deallocation;
package body pilegenerique2 is
procedure reinit (p : in out pile)  is
begin t.top:=-1; end reinit;
procedure ajouter (p : in out pile ; e :in item ) is
begin
p.tabpile(p.top+1) := new item'(e) ;
p.top := p.top+1 ;
end ajouter;
procedure supprimer (p : in out pile) is
procedure dispose is new
      ada.Unchecked_Deallocation(item, itemref);
begin
dispose(p.tabpile(p.top));
p.top := p.top-1; end supprimer;
function sommet (p : pile) return item is
begin return (p.tabpile(p.top).all); end sommet;
function vide (p : pile) return boolean is
begin return p.top := -1; end vide;
end pilegenerique2 ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Exemple

Pour utiliser pleinement notre paquetage on va tout d'abord créer un paquetage de tableaux d'entiers non contraints.

```
package tableau2 is
type tableau is array (natural range <>) of integer ;
function init() return tableau;
procedure afficher ( t : in tableau);
end tableau2;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
with ada.integer_text_io;
use ada.integer_text_io;
with Ada.Numerics.Discrete_Random;
package body tableau2 is

function init return tableau is
subtype valeur is integer range -99 .. 99 ;
package hasard_valeur is
new Ada.Numerics.Discrete_Random (valeur) ;
use hasard_valeur;
GV : hasard_valeur.Generator ;
subtype longueur is positive range 9 .. 49 ;
package hasard_longueur is
new Ada.Numerics.Discrete_Random (longueur) ;
use hasard_longueur;
GL : hasard_longueur.Generator;
 l : natural:=random(GL); t : tableau(0 .. l);
begin
reset(GL); reset (GV);
l := random(GL);
for i in t'range loop t(i) := random(GV); end loop;
return t;
end init;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
procedure afficher (t : in tableau) is
begin
for i in t'range loop
   put(t(i), 5);
   if (i+1) rem 10 =0 then new_line; end if;
end loop;
new_line;
end afficher;
end tableau2;
```

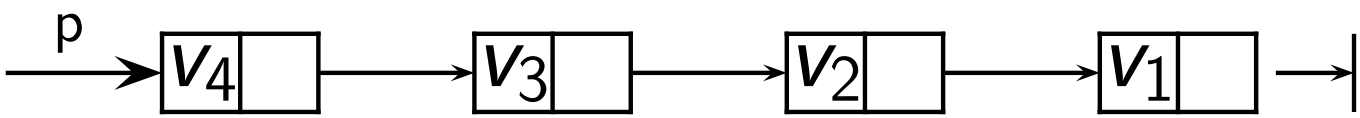
Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

pilegenerique2 : usage

```
with pilegenerique2;
with tableau2; use tableau2;
with Ada.Text_IO; use Ada.Text_IO;
with ada.integer_text_io;
use ada.integer_text_io;
procedure essai2 is
package piletableau is new pilegenerique2(tableau);
use piletableau;
t: tableau:=init;
p : piletableau.pile;
begin
for i in 1..10 loop
t:= init;
ajouter(p,t);
end loop;
afficher(sommet(p));
while(not vide(p)) loop
put(sommet(p)'length);
new_line;
supprimer(p);
end loop;
end essai2 ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

L'implémentation précédente a résolu le problème du contenant des piles (et des files).
Mais nos piles et files restent d'une taille fixée.
On va donc se servir de l'allocation dynamique pour construire piles (et files) de façon à n'utiliser que la mémoire nécessaire et suffisante : on empile sur une nouvelle zone de mémoire qui est libérée lors de son dépilement.
Pour cela les références de ces zones de mémoire seront stockées dans une zone de mémoire allouée dynamiquement : c'est le principe des structures chaînées, on stocke en même temps une valeur et une (des) référence(s) qui permettent l'accès à d'autres données du même type.
On aura une structure que l'on peut illustrer ainsi :



Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Déclaration

En Ada pour construire ce type on fait une déclaration ≪ circulaire ≫ :

```
type cellule ;
type pcell is access cellule ;
type cellule is  record
 v : item ;
 s : pcell ;
end record ;
```

Ici le type `item` peut être générique.
Le type `cellule` est tout d'abord déclaré de façon incomplète car il nécessite une référence au type cellule.
La déclaration de ce type `access` est donc faite de façon elle aussi incomplète puis le type `cellule` est entièrement déclaré.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Donc une variable de type `pcell` est une référence à une `cellule` qui est un article à deux champs, l'un contenant une valeur et l'autre une référence à une autre cellule.
On peut utiliser les structures chaînées pour un usage comparable aux tableaux **mais** on n'a pas d'**accès direct** aux valeurs stockées.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
with Ada.Numerics.Discrete_Random
procedure creerhasard  is
subtype TValeur is integer range -99 .. 99 ;
subtype Tlong is natural range 10..30 ;
package HasardValeur is
new Ada.Numerics.Discrete_Random (TValeur) ;
package Hasardlong is
new Ada.Numerics.Discrete_Random (Tlong) ;
use HasardValeur ; use Hasardlong ;
type cellule ;
type pcell is access cellule ;
type cellule is  record
 v : TValeur ;
 s : pcell ;
end record ;
G : HasardValeur.Generator ;
Gg : HasardHauteur.Generator ;
long :Tlong ;
list : pcell ;
begin
list :=null ;
Reset (G) ; Reset(Gg) ;
long :=Random(Gg) ;
for k in 1..long loop
    list := new cellule'(v => Random(G) , s=> list) ;
end loop;
end creerhasard;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

La structure de liste chaînée se prête particulièrement bien à la récursivité. En effet, on peut décrire une **liste** comme étant un item attachée à une **liste** : cette description fait appel à elle même.
On va donc créer un paquetage de liste en insérant des fonctionnalités le plus possible écrites récursivement.
Pour simplifier ce seront des listes d'entiers. Pour accentuer la manipulation récursive on ne fera figurer que des fonctions.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
package listedentier is
type cellule ;
type liste is access cellule ;
type cellule is  record
 val : integer ;
 suiv : liste ;
end record ;
function init return liste;
function ajouter(x:integer; l:liste) return liste;
function reste(l:liste) return liste;
function top(l:liste) return integer;
function vide(l:liste) return boolean;
function longueur(l:liste) return integer;
function concatener(l1, l2 : liste) return liste;
function miroir(l:liste) retrun liste;
function fusion(l1,l2:liste) return liste;
end listedentier ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
package body  listedentier is
function init return liste is
begin return null; end init;
function ajouter(x:integer; l:liste) return liste is
k : liste;
begin k:= new cellule'(x ,l) ;  return k; end ajouter;
function reste(l:liste) return liste is
return l.suiv; end reste;
function top(l:liste) return integer is
begin return l.val; end top;
function vide(l:liste) return boolean is
begin if l=null then return true;
else return false; end if; end vide;
function longueur(l:liste) return integer is
begin if vide(l) then return 0;
else return (1+longueur(reste(l));
end if; end longueur;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
function concatener(l1, l2 : liste) return liste is
k:liste;
begin
if vide(l1) then return l2;
else k := new cellule'(top(l1), concatener(reste(l1), l2));
       return k;
end if; end concatener;
function miroir(l:liste) return liste is
k, j : liste;
begin if longueur(l) <=1 then return l ;
         else j:= init; j:= ajouter(top(l),j);
                 k := concatener(miroir(reste(l)), j);
         end if;
end miroir;
function fusion(l1,l2:liste) return liste is
k:liste:=init;
begin if vide(1) then return l2;
elsif vide(l2) then return l1;
else if top(l1)<top(l2)
      then k:= ajouter(top(l1), k);
               k:= concatener(k,fusion(reste(l1), l2));
      else k:= ajouter(top(l2), k);
              k:= concatener(k,fusion(l1, reste(l2)));
end if; end if; end fusion;
end listedentier;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Dans le paquetage suivant on met en oeuvre des procédures qui modifient leurs paramètres

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

```
package listedentier2 is
type cellule ;
type liste is access cellule ;
type cellule is  record
 val : integer ;
 suiv : liste ;
end record ;
procedure init(l : in out liste);
procedure ajouter(x:in integer; l: in out liste) ;
function top(l:liste) return integer;
function vide(l:liste) return boolean;
function longueur(l:liste) return integer;
procedure concatener(l1, l2 : in out liste) ;
-- l1 devient la concatenation et l2 devient vide
procedure miroir(l: in out liste) ;
--  l est modifiée en son miroir
procedure fusion(l1,l2: in out liste ; l : out liste) ;
-- l1 et l2 deviennent nulles et l est la fusion
-- de l1 et l2 qui sont supposées triées
end listedentier2;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

des piles sans limite avec une liste chaînée

On peut remarquer que dans la procédure `creerhasard` les valeurs ajoutées dans la structure chaînée le sont en tête de structure à la façon d'une pile.
On va donc réécrire un paquetage générique `pile` en utilisant les structures chaînées et un paramètre générique non contraint.
Attention : aucune précaution ne sera prise pour un usage incorrect de la fonction `sommet` ou de la procédure `supprimer` (dépiler).

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

pile.ads

```
generic type item(<>) is private ;
package pile is
type itemref is access item ;
type cellulep ;
type pile is access cellulep ;
type cellulep is record
v : itemref ;
suiv : pile ;
end record ;
procedure init(p : in out pile ) ;
procedure ajouter(p : in out pile ; x : in item ) ;
procedure supprimer(l : in out pile ) ;
function sommet (p : pile) return item ;
function vide (p : pile) return boolean;
end pile  ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

pile.adb

```
with Ada.Unchecked_Deallocation;
package body pile  is
procedure init(p : in out pile ) is
begin p := null ; end init ;
procedure ajouter(p : in out pile ; x : in item ) is
iref: itemref;
begin
iref := new item'(x);
p := new cellulep'(iref,p) ;
end ajouter ;
procedure supprimer(p : in out pile ) is
procedure liberer is
new Ada.Unchecked_Deallocation(cellulep,pile) ;
procedure liberer is
new Ada.Unchecked_Deallocation(item,itemref) ;
temp : pile := p;
begin p := p.suiv ;
liberer(temp.v) ; liberer(temp) ;
end supprimer ;
function sommet (p : pile) return item is
a:itemref;
begin
a:=p.v;
return a.all ; end sommet ;
function vide (p : pile) return boolean is
begin  return (p=null ) ; end vide ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

et les files alors ? !

Pour les files, on a besoin de gérer les deux extrêmités : le début et la fin de la file. On peut concevoir un paquetage `file` avec un définition à trois champs : `v` pour la valeur, `debut` et `fin` pour les extrêmités.
La difficulté est que lorsque la file est vide les deux champs `debut` et `fin` sont égaux (à `null`). Et lorsque la file contient un unique élément alors de nouveau les deux champs `debut` et `fin` sont égaux.
Il faut donc soigneusement implémenter le fonction `vide`.
Une autre idée, simplifiant le type `file`, utilise une liste chaînée *en rond* de façon à ce que la dernière valeur ≪ pointe ≫ sur la première ; la file sera alors caractérisée par la référence à la dernière cellule permettant l'accès immédiat à la première qui suit la dernière.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

enfiler

```
procedure ajouter(f : in out file ; x : in item ) is
temp : file ; iref : itemref;
begin
iref := new item'(x);
if f=null then
f := new cellulef'(iref,f) ; f.suiv := f ;
else
temp := new cellulef'(iref,f.suiv) ;
f.suiv := temp ; f := temp ;
end if ;
end ajouter ;
```

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Type access
Déclaration
Création
Affectation
Récupération
Opérations
généricité : apport des pointeurs
Liste chaînée
Déclaration
un exemple
Manipulation de listes chaînées par des fonctions
Manipulation de listes chaînées par des procédures
Liste chaînée et pile
Liste chaînée et file
Une application des piles

Exercices

1. compléter file.ads et file.adb
2. problème de amstramgam : on a une ronde de n enfants, qui reste après amstramgam pic et pic et colegram ?
3. tester les procédures `inverser`, `copier` et `trier` avec le paquetage `pile` écrites par l'*utilisateur*.

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Trouver un chemin entre des points

On suppose que l'on a n points du plan que l'on numérotera $M_1, \cdots, M_n$. Certains points sont reliés par un segment d'autres non. Le schéma de liaison de ces points est donné par une matrice carrée $n \times n$ de booléens $\mathcal{A}$ telle que $\mathcal{A}_{i,j} =$ *true si et seulement si* les points M_i et M_j sont reliés. On cherche à déterminer un chemin possible entre M_1 et M_n (s'il existe).
Par exemple, si $n = 5$ et si on a exactement

$$\mathcal{A} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix},$$

alors il existe un chemin reliant M_1 à M_5 qui est M_1, M_4, M_5 ou bien M_1, M_3, M_4, M_5.

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

algorithme

Le principe de cette recherche de chemin est exhaustif : on part de M_1 puis on va à un des points qui lui sont reliés et on recommence à partir de ce nouveau point et ainsi de suite.
Attention :

à quoi ?

il ne faut pas tourner en rond !

Solution ?

Faire comme le petit poucet

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

algorithme

Le principe de cette recherche de chemin est exhaustif : on part de M_1 puis on va à un des points qui lui sont reliés et on recommence à partir de ce nouveau point et ainsi de suite.

Attention :

à quoi ?

il ne faut pas tourner en rond !

Solution ?

Faire comme le petit poucet

Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

algorithme

Le principe de cette recherche de chemin est exhaustif : on part de M_1 puis on va à un des points qui lui sont reliés et on recommence à partir de ce nouveau point et ainsi de suite.
Attention :
à quoi ?
il ne faut pas tourner en rond !
Solution ?
Faire comme le petit poucet

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

algorithme

Le principe de cette recherche de chemin est exhaustif : on part de M_1 puis on va à un des points qui lui sont reliés et on recommence à partir de ce nouveau point et ainsi de suite.
Attention :
à quoi ?
il ne faut pas tourner en rond !
Solution ?
Faire comme le petit poucet

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Lorsque l'on s'aperçoit que, d'un point, on a déjà vu tous les autres qui lui sont reliés, on doit revenir en arrière pour examiner un autre chemin possible depuis le point précédent.
Il faut donc garder en mémoire le trajet déjà parcouru depuis le point M_1 et pouvoir facilement remonter dans ce trajet : on utilisera pour cela

- une pile d'entiers (numéro des points)
- un tableau de taille de booléens indexé par les numéros des points

- la pile nous servira de mémoire (petits cailloux pour remonter le chemin en sens inverse)
- le tableau nous servira à marquer les points déjà visités

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Lorsque l'on s'aperçoit que, d'un point, on a déjà vu tous les autres qui lui sont reliés, on doit revenir en arrière pour examiner un autre chemin possible depuis le point précédent. Il faut donc garder en mémoire le trajet déjà parcouru depuis le point M_1 et pouvoir facilement remonter dans ce trajet : on utilisera pour cela

- une pile d'entiers (numéro des points)
- un tableau de taille de booléens indexé par les numéros des points

- la pile nous servira de mémoire (petits cailloux pour remonter le chemin en sens inverse)
- le tableau nous servira à marquer les points déjà visités

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

```
with text_io; with ada.integer_text_io; with pile;
with ada.numerics_discrete_random;
use text_io; use ada.integer_text_io;
procedure chemin is
nbre_point : constant integer := 100;
subtype intervalle is integer range 1..nbre_point;
type tableau is array(intervalle) of boolean;
type matrice is array(intervalle, intervalle) of boolean;
procedure init_carte(c : out matrice) is
subtype valeur is integer range -99..99;
package hasard is new  ada.numerics_discrete_random(valeur);
use hasard;
g : hasard.generator;
n : integer;
begin
reset(g);
for i in c'range loop
for j in c(i)'range loop
n:= random(g);
if n<0 then c(i,j):= false; else c(i,j):= true;
end if; end loop; end loop;
end init_carte;
```

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

```
package piledentier is new pile(integer);
use piledentier;
c: matrice; v : tableau; p : piledentier.pile;
M, j: integer;
begin
for i in v'range loop v(i) := false; end loop;
init_carte(c);
init(p);
ajouter(p,1); v(1) := true;
```

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

```
while (not vide(p)) and then (sommet(p)/= n) loop
M:= sommet(p);
j:= 1;
loop
if j>n then exit; end if;
if c(M,j) = true and v(j) = false then exit;
else j := j+1;
end loop;
if j<=n then ajouter(p,j); v(j) := true;
else supprimer(p);
end if;
end loop;
if vide(p) then put(" pas de chemin");
else inverser(p);
        while not vide(p) loop
        put(sommet(p),4);
        end loop;
        end if;
end chemin;
```

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Le problème des huit dames

Le problème des 8 reines est le suivant : il s'agit de placer sur un échiquier 8 reines de telle sorte que qu'aucune ne soit en prise avec les autres.
Donc on doit avoir une et une seule reine sur chaque ligne, chaque colonne, chaque diagonale.
On va utiliser la notion de backtracking (retour sur trace) : il s'agit de résoudre un problème par essai successif ; quand une suite d'essais aboutit à un cas impossible alors on revient sur l'avant dernier essai pour tenter autre chose.

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour un échiquier 4 x 4

On va remplir colonne par colonne de haut en bas.

Rappel sur les passages de paramètres

Paquetages

Généricité

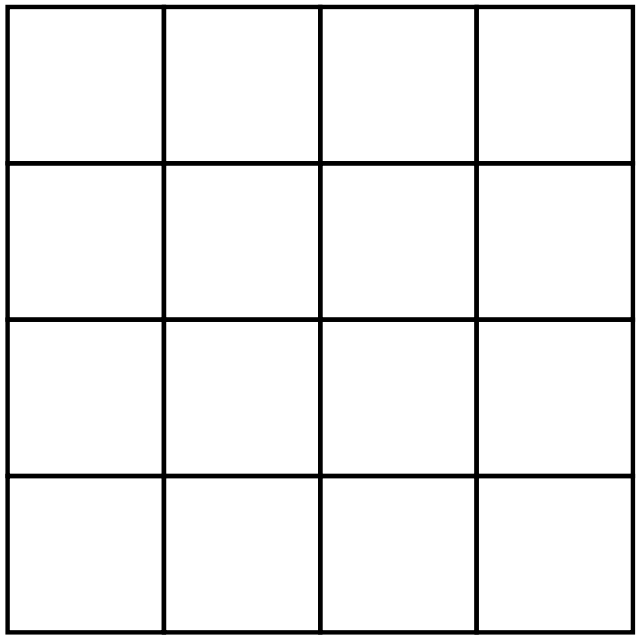
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

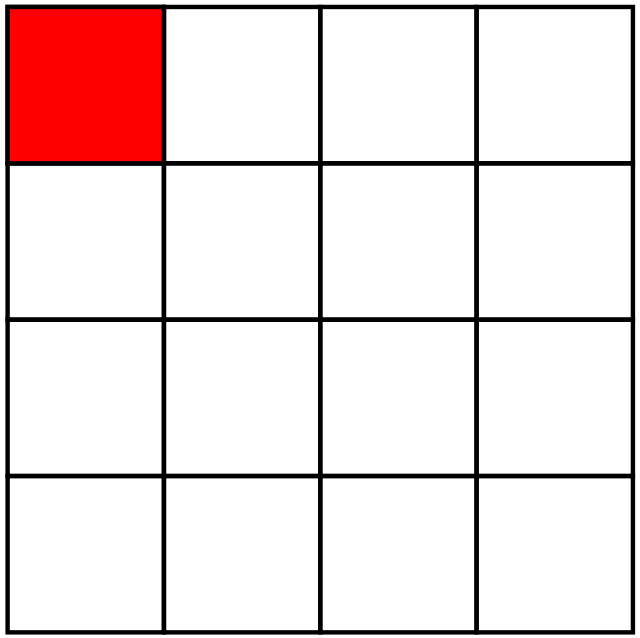
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

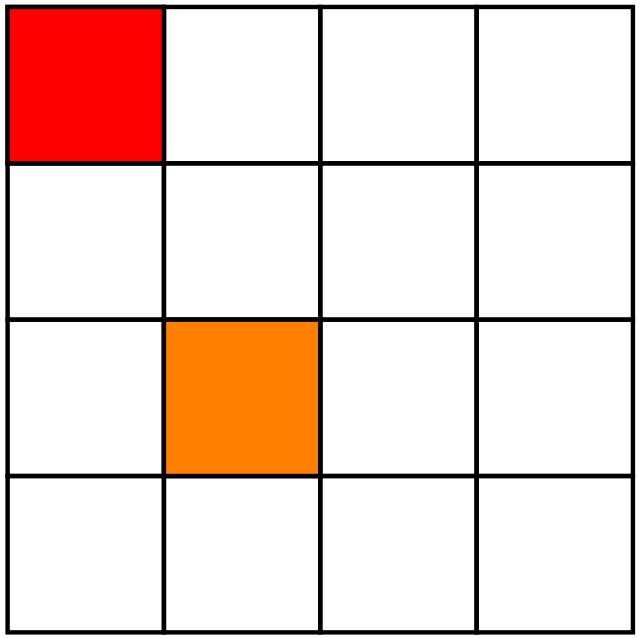
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

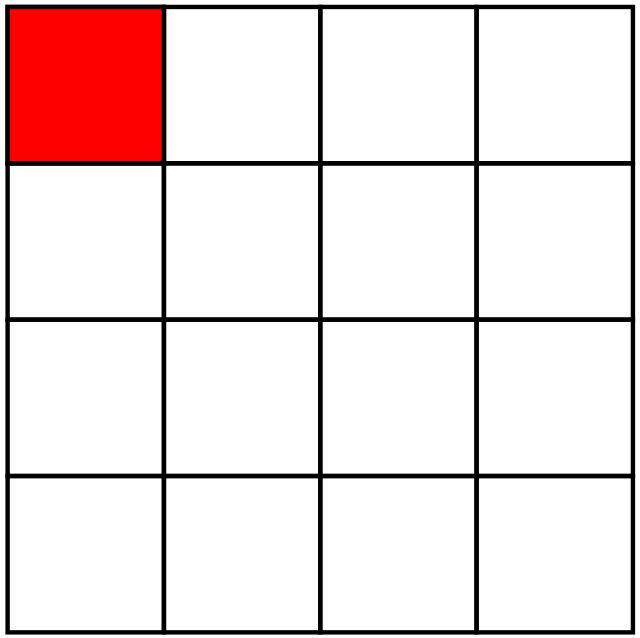
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

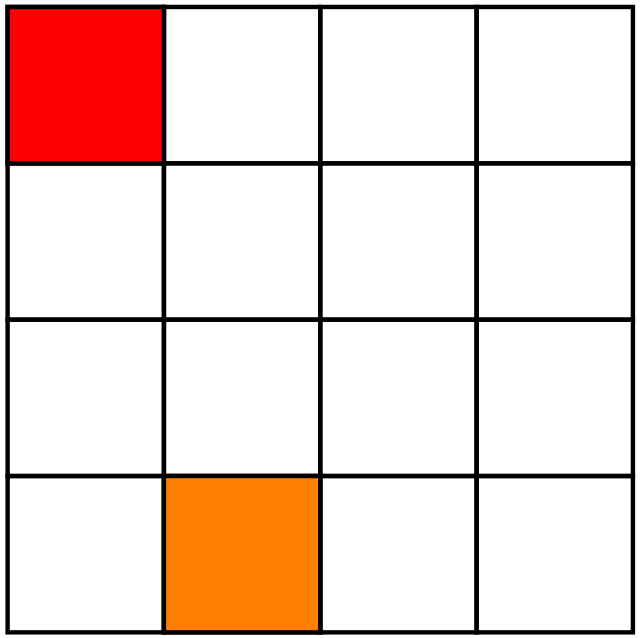
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

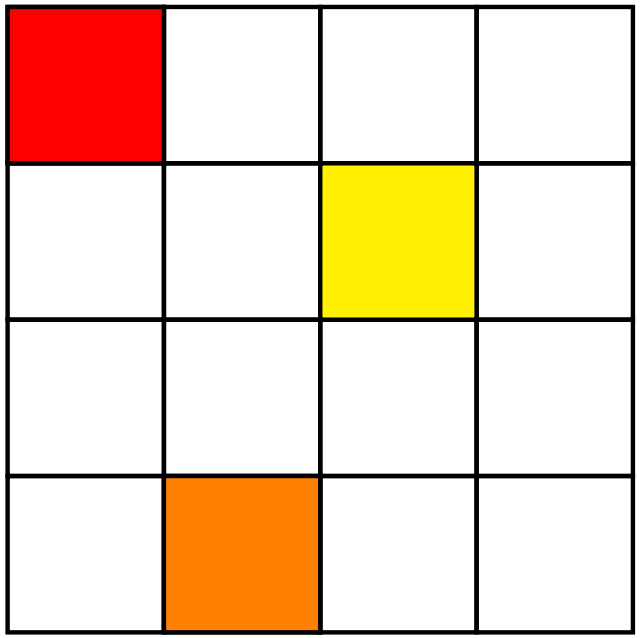
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

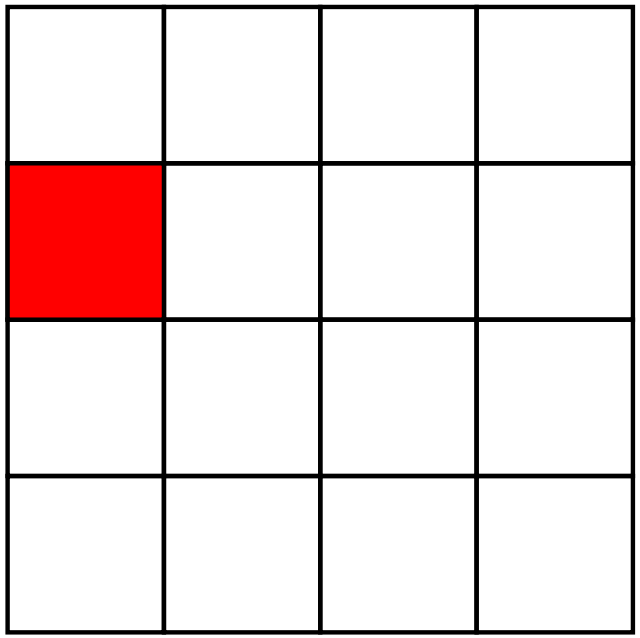
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

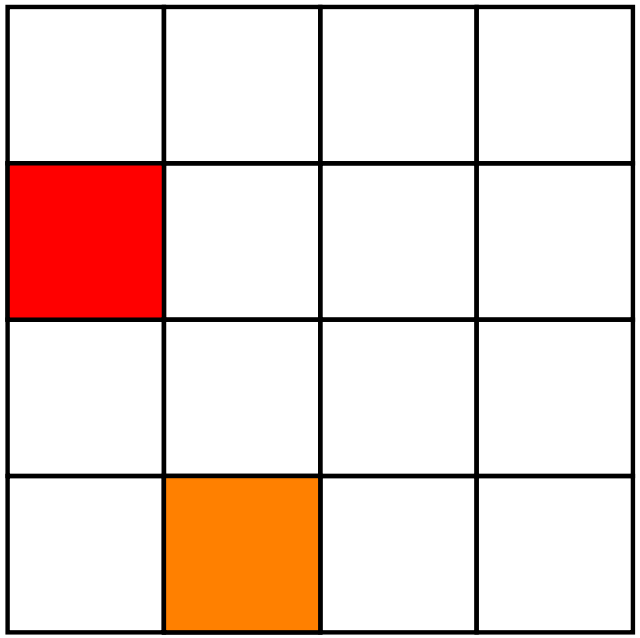
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

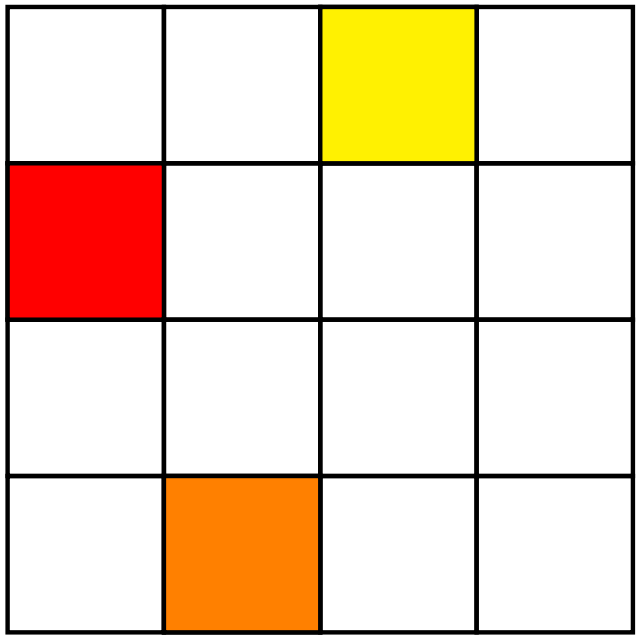
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres

Paquetages

Généricité

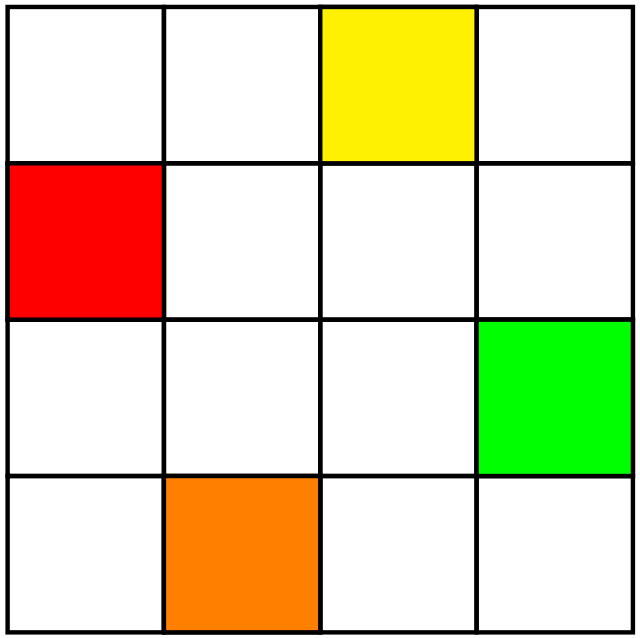
Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

exemple pour quatre reines



Rappel sur les passages de paramètres
Paquetages
Généricité
Exceptions
Variables dynamiques
Une application des piles
Une application de la récursivité : le backtracking

mise en oeuvre

Pour réaliser le backtracking on peut utiliser explicitement une pile. Mais on peut aussi profiter de la récursivité !
Algorithme
placer une reine en colonne c dans l'échiquier e :
si c est supérieur à la taille de e **alors** afficher e
sinon
pour chaque ligne l de la colonne c FAIRE
si la case (c,l) de e est valide **alors** mettre une reine dans e à la case (c,l) - **placer une reine en colonne c**+1 **dans e**
fin si
fin pour
fin si

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

La récursivité va bloquer la boucle **pour** dès qu'une case de la colonne en cours est remplie et l'appel récursif va passer à la colonne suivante.
Lorsqu'on atteint **fin pour** c'est que l'appel correspondant sur la colonne c n'a rien donné et se termine donc cet appel est dépilé de la pile des appels.

Rappel sur les passages de paramètres

Paquetages

Généricité

Exceptions

Variables dynamiques

Une application des piles

Une application de la récursivité : le backtracking

Par exemple pour l'échiquier 4x4 les appels successifs sont :

placer(e,1)

reine en 1, 1

↪ placer(e,2)

reine en 2, 3

↪ placer(e,3)

fin pour

reine en 2, 4

↪ placer(e,3)

reine en 3, 2

↪ placer(e,4)

fin pour

fin pour

fin pour

reine en 1,2

↪ placer(e,2)

reine en 2, 4

↪ placer(e,3)

reine en 3,1

↪ placer(e,4)