INF3105 – Piles (*Stacks*)

Jaël Champagne Gareau

Université du Québec à Montréal (UQAM)

Été 2024

http://cria2.uqam.ca/INF3105/

Sommaire

1 Introduction

2 Implémentation par un tableau (array)

3 Implémentation par chaîne de cellules

4 Exercices
- Pile avec liste de cellules

Les piles

- Type de données abstrait simple.
- Analogie : piles d'assiettes dans une cafétéria.
- Modèle *LIFO : last-in-first-out* (dernier arrivé, premier servi).

Exemples d'applications

- Programme récursif converti en programme non récursif.
- Évaluation d'opérations arithmétiques.
- Boutons précédent et suivant dans les navigateurs Internet.
- Commandes pushd et popd dans le Shell Bash (Linux et Unix).
- Interpréteurs de langage (pile d'appel de fonction (contexte)).
- Langage *Postscript* dans les imprimantes.
- Etc.

Interface **abstraite** d'une pile

empiler(e)	push(e)	Place e au sommet de la pile.
depiler()	pop()	Enlève l'élément au sommet de la pile.
sommet()	top()	Retourne le sommet de la pile.
taille()	size()	Retourne le nombre d'éléments dans la pile.
vide()	empty()	Retourne vrai si la pile est vide, sinon faux.

Interface **abstraite** en C++ d'une pile

```
1  template <class T> class Pile {
2    public:
3      Pile();
4      ~Pile();
5      int  taille() const; // fonction optionnelle
6      bool vide() const;
7      const T& sommet() const;
8      void empiler(const T& e);
9      // Au choix, l'une des fonctions suivantes :
10     T depiler(); // retourne l'objet qui était au sommet de la pile
11     void depiler(); // l'objet dépile n'est pas retourné
12     void depiler(T& e); // dépile l'élément dans l'objet e en référence
13 };
```

Représentation C++ d’une pile (version 1)

Fichier entête partiel pile.h

```
1  template <class T>
2  class PileTableau {
3    public:
4      PileTableau(int initCapacite = 100);
5      ~PileTableau();
6      int  taille() const;
7      bool vide() const;
8      const T& sommet() const;
9      void empiler(const T& e);
10     T depiler();
11
12   private:
13     T* data;
14     int capacite;
15     int s; // indice sur le sommet
16 };
```

Représentation C++ d'une pile (version 2)

Fichier entête partiel pile.h

```cpp
#include "tableau.h"
template <class T>
class PileTableau {
  public:
    PileTableau(); // optionnel
    ~PileTableau(); // optionnel
    int  taille() const;
    bool vide() const;
    const T& sommet() const;
    void empiler(const T& e);
    T depiler();

  private:
    Tableau<T> elements;
};
```

Fonctions (1)

```
1  template <class T>
2  int PileTableau<T>::taille() const {
3      return elements.taille();
4  }
5
6  template <class T>
7  bool PileTableau<T>::vide() const {
8      return elements.vide();
9  }
10
11 template <class T>
12 const T& PileTableau<T>::sommet() const {
13     return elements[elements.taille()-1];
14 }
```

Fonctions (2)

```
1  template <class T>
2  void PileTableau<T>::empiler(const T& element) {
3      elements.ajouter(element);
4  }
5
6  template <class T>
7  T PileTableau<T>::depiler() {
8      T result = sommet();
9      elements.enlever(elements.taille() − 1);
10     return result;
11 }
```

Analyse des opérations (sans réallocation)

Opération	**Complexité**
empiler(e)	O(1)
depiler()	O(1)
sommet()	O(1)
taille()	O(1)
vide()	O(1)
vider()	O(1) ou O(n)

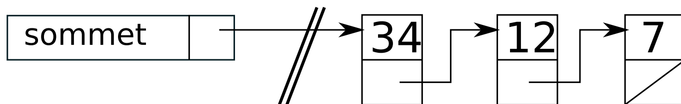
Analyse des opérations (avec réallocation)

Opération	Amortie	Pire cas
empiler(e)	O(1)	**O(n)**
depiler()	O(1)	O(1)
sommet()	O(1)	O(1)
taille()	O(1)	O(1)
vide()	O(1)	O(1)
vider()	O(1)	O(1) ou O(n)

Remarque

- Complexité héritée de `Tableau`.

Représentation d'une pile avec une chaîne de cellules



Représentation C++ d'une pile

```
1   template <class T>
2   class Pile {
3   public:
4       Pile();
5       ~Pile();
6
7       bool vide() const;
8       const T& sommet() const;
9       void empiler(const T&);
10      void depiler();
11  private:
12      struct Cellule {
13          Cellule(const T& c, Cellule* s) : contenu(c), suivante(s) {}
14          T contenu;
15          Cellule* suivante;
16      };
17      Cellule* sommet;
18  };
```

Constructeur et Destructeur

Constructeur - Version 1

```cpp
template <class T>
Pile<T>::Pile() {
    sommet = nullptr;
}
```

Constructeur - Version 2

```cpp
template <class T>
Pile<T>::Pile() : sommet(nullptr) {}
```

Destructeur

```cpp
template <class T>
Pile<T>::~Pile() {
    vider();
}
```

Fonctions

```
1  template <class T>
2  bool Pile<T>::vide() const {
3      return sommet == nullptr;
4  }
5
6  template <class T>
7  const T& Pile<T>::sommet() const {
8      assert(sommet != nullptr);
9      return sommet->contenu;
10 }
```

Empiler

```
1  template <class T>
2  void Pile<T>::empiler(const T& element) {
3      sommet = new Cellule(element, sommet);
4
5      // optionnel : test l'allocation de mémoire
6      assert(sommet != nullptr);
7  }
```

Depiler

```
1  template <class T>
2  void Pile<T>::depiler() {
3      assert(sommet != nullptr);
4      Cellule* suivante = sommet->suivante;
5      delete sommet;
6      sommet = suivante;
7  }
```

Depiler (version alternative)

```
1  template <class T>
2  T Pile<T>::depiler() {
3      assert(sommet != nullptr);
4      T element = sommet->contenu;
5      Cellule* ancien = sommet;
6      sommet = sommet->suivante;
7      delete ancien;
8      return element;
9  }
```

Analyse des opérations

Opération	Cas Moyen	Pire cas
empiler(e)	O(1)	O(1)
depiler()	O(1)	O(1)
sommet()	O(1)	O(1)
taille()	O(1)	O(1)
vide()	O(1)	O(1)
vider()	O(n)	O(n)

Remarques

- Hypothèse requise : allocation et désallocation de mémoire (opérateurs new et delete) en temps constant, i.e. O(1).
- Complexité spatiale : on a besoin d'un pointeur par cellule. Négligeable quand les objets sont gros.

Rappel

```
1   template <class T>
2   class Pile {
3   public:
4       Pile();
5       ~Pile();
6
7       bool vide() const;
8       const T& sommet() const;
9       void empiler(const T&);
10      void depiler();
11  private:
12      struct Cellule {
13          Cellule(const T& c, Cellule* s) : contenu(c), suivante(s) {}
14          T contenu;
15          Cellule* suivante;
16      };
17      Cellule* sommet;
18  };
```

Opérateur ==

```
1  template <class T>
2  bool Pile<T>::operator==(const Pile<T>& autre) const {
3
4
5
6
7
8
9
10
11
12
13 };
```

Opérateur =

```
1   template <class T>
2   Pile<T>& Pile<T>::operator=(const Pile<T>& autre) {
3
4
5
6
7
8
9
10
11
12      return *this;
13  };
```