Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

HE IG VD

Recueil d'exercices d'Algorithmes et Structures de Données (ASD)

Partie 2 : Structures linéaires

Version 1.0
10 mai 2023

Table des matières

Chapitre X : Allocation dynamique 1
Exercice X.1 new et delete 1
Exercice X.2 ::operator new, new(p), … 2
Exercice X.3 vector::emplace et … 3
Solutions 4
Chapitre 4 : Structures linéaires 5
Exercice 4.1 std::vector à capacité inchangée 5
Exercice 4.2 std::vector à capacité qui varie 7
Exercice 4.3 Complexités de std::vector 8
Exercice 4.4 Tableau d'entiers redimensionnable 10
Exercice 4.5 Tableau générique redimensionnable (struct) 11
Exercice 4.6 Tableau générique redimensionnable (class) 12
Exercice 4.7 Pile avec une liste simplement chainée 12
Exercice 4.8 Pile avec une liste simplement chainée (2) 13
Exercice 4.9 Tri par sélection d'une `std::list` 13
Exercice 4.10 Tri par sélection d'une `std::forward_list` 14
Exercice 4.11 Inversion d'une `std::forward_list` 14
Exercice 4.12 Inversion d'une `std::list` 14
Exercice 4.13 Insertion dans un tas 15
Exercice 4.14 Suppression du sommet d'un tas 15
Exercice 4.15 Création d'un tas 15
Exercice 4.16 Tri par tas 16
Exercice 4.17 Tas dans la librairie `<algorithm>` 16
Exercice 4.18 Coût en mémoire des structures de donnée 18
Exercice 4.19 Itérateurs et splice 19
Exercice 4.20 Algorithme de Dijkstra 22
Solutions 23

Chapitre X : Allocation dynamique

Exercice X.1 new et delete

Soit le code suivant

```
#include <iostream>

using namespace std;

class C {
   int i;
public:
   C() : i(0) { cout << "CD " << flush; }
   C(int i) : i(i) { cout << "C" << i << " " << flush; }
   ~C() { cout << "D" << i << " " << flush; }
};

int main() {
   auto p1 = f();  CD
   auto p2 = f(2); C2
   g(p2); D2
   p2 = f(3); C3
   g(p1); D0
   g(p2); D3
}
```

Ecrivez les fonctions f, g et éventuelles surcharges de sorte que le programme affiche ce qui suit à l'exécution

```
CD C2 D2 C3 D0 D3
```

```
C* f() { return new C(); }
C* f(int i) { return new C(i); }
void g(C* p) { delete p; }
```

Exercice X.2 ::operator new, new(p), …

Soit le code suivant

```
#include <iostream>

using namespace std;

class C {
   int i;
public:
   C() : i(0) { cout << "CD " << flush; }
   C(int i) : i(i) { cout << "C" << i << " " << flush; }
   ~C() { cout << "D" << i << " " << flush; }
};

int main() {
  void *p1 = f(), *p2 = f();
  f(p1); CD
  f(p2,4); C4
  g((C*)p2); D4
  f(p2,1); C1
  g((C*)p1); D0
  ::operator delete(p1);
  g((C*)p2); D1
  ::operator delete(p2);
}
```

Ecrivez les fonctions f, g et éventuelles surcharges de sorte que le programme affiche ce qui suit à l'exécution

```
CD C4 D4 C1 D0 D1
```

```
void f(void* p) { new(p) C; }
void f(void* p, int i) { new(p) C(i); }
void g(C* i) { i->~C(); }
void* f() { return ::operator new(sizeof(C)); }
```

Exercice X.3 vector::emplace et …

Soit la classe C suivante

```
class C {
   int i;
public:
   C() : i(0) { cout << "CD " << flush; }
   C(int i) : i(i) { cout << "C" << i << " " << flush; }
   C(C const& c) : i(c.i) { cout << "Cp" << i << " " << flush; }
   C& operator=(C const& c) { i = c.i; cout << "=" << i << " " << flush;
return *this; }
   ~C() { cout << "D" << i << " " << flush; }
};
```

Qu'affiche le code suivant à chacune de ses lignes

Code	Affichage
`int main() {`	
`  vector<C> v(2);`	C0 C0
`  v.pop_back();`	D0
`  v.push_back(C(1));`	C1 Cp1 D1
`  v.clear();`	D1 D0
`  v.emplace_back(2);`	C2
`  v.emplace_back(3);`	C3
`  v.front() = C(4);`	C4 =4 D4
`  v.pop_back();`	D3
`}`	D4

Solutions

Exercice X.1 new et delete

```
C* f() { return new C(); }

C* f(int i) { return new C(i); }

void g(C* p) { delete p; }
```

Exercice X.2 ::operator new, new(p), …

```
void* f() { return ::operator new(sizeof(C)); }

void f(void* p) { new(p) C; }

void f(void* p, int i) { new(p) C(i); }

void g(C* p) { p->~C(); }

// ou à partir de C++17, void g(C* p) { std::destroy_at(p); }
```

Exercice X.3 vector::emplace et …

`int main() {`	
`  vector<C> v(2);`	`CD CD`
`  v.pop_back();`	`D0`
`  v.push_back(C(1));`	`C1 Cp1 D1`
`  v.clear();`	`D1 D0`
`  v.emplace_back(2);`	`C2`
`  v.emplace_back(3);`	`C3`
`  v.front() = C(4);`	`C4 =4 D4`
`  v.pop_back();`	`D3`
`}`	`D4`

Chapitre 4 : Structures linéaires

Exercice 4.1 std::vector à capacité inchangée

Soit la classe C ci-dessous, dont vous pouvez trouver une copie ici :
https://gist.github.com/ocuisenaire/2eaef33afb297bae02334b799299d020

```
class C {
   int i;
public:
   C();                                 // CD
   C(int i);                            // C#
   C(C const& c);                       // Cc#
   C(C && c) noexcept;                  // Cm#, c.i vaut -1 après
   C& operator=(C const& c);            // =c# ou =c#x si auto-affectation
   C& operator=(C && c) noexcept;       // =m# ou =m#x si auto-affectation,
                                        // c.i vaut -2 après
   ~C();                                // D#
   friend std::ostream & operator<< (std::ostream & out, C const& c) {
     return out << c.i << std::flush;
   }
};                                      // où # affiche la valeur de i

C::C()           : i(0)   { cout << "CD " << flush; }

C::C(int i)      : i(i)   { cout << "C" << i << " " << flush; }

C::C(C const& c) : i(c.i) { cout << "Cc" << i << " " << flush; }

C::C(C && c) noexcept : i(c.i) { c.i = -1;
                                 cout << "Cm" << i << " " << flush; }

C& C::operator=(C const& c) {
   if (&c == this) { cout << "=c" << i << "x " << flush; }
   else { i = c.i; cout << "=c" << i << " " << flush; }
   return *this;
}

C& C::operator=(C && c) noexcept {
   if (&c == this) { cout << "=m" << i << "x " << flush; }
   else { i = c.i; c.i = -2; cout << "=m" << i << " " << flush; }
   return *this;
}

C::~C() { cout << "D" << i << " " << flush; }
```

Indiquez ce qu'affiche chaque ligne du code suivant.

`int main() {`	
`  vector<C> v;`	
`  v.reserve(10);`	
`  v.emplace_back();`	
`  v.emplace_back(1);`	
`  v.emplace_back(C(2));`	
`  v.push_back(3);`	
`  v.push_back(C(4));`	
`  v[0] = v[4];`	
`  v[1] = v[1];`	
`  v[2] = C(2);`	
`  v[3] = 3;`	
`  v[4] = std::move(v[4]);`	
`  v.insert(v.begin()+2,C(5));`	
`  v.erase(v.begin()+1);`	
`  v.clear();`	
`}`	

Exercice 4.2 std::vector à capacité qui varie

Avec la même classe C qu'à l'exercice 4.1, qu'affiche le code suivant si le compilateur utilisé multiplie par un facteur 2 la capacité quand c'est nécessaire ? Notez l'absence d'appel à la méthode `vector<C>::reserve(…)`

`int main() {`	
`  vector<C> v;`	
`  v.emplace_back();`	
`  v.emplace_back(1);`	
`  v.emplace_back(C(2));`	
`  v.emplace_back(3);`	
`  v.insert(v.begin()+2,C(4));`	
`  v.insert(v.begin()+3,C(5));`	
`  v.erase(v.begin()+1);`	
`}`	

Exercice 4.3 Complexités de std::vector

Quelle est la complexité des extraits de fonctions suivantes en fonction de N (M ou L)

`void f1(vector<int>& v, size_t N) {` `  size_t M = v.size();` `  for(size_t i = 0; i < N; ++i) {` `  v.insert(v.begin(), i);` `  v.pop_back();` `  }` `}`	
`vector<int> f2(size_t N) {` `  vector<int> v;` `  for(size_t i = 0; i < N; ++i) {` `  v.insert(v.begin(), i);` `  v.pop_back();` `  }` `  return v;` `}`	
`vector<int> f3(size_t N) {` `  vector<int> v;` `  for(size_t i = 0; i < N; ++i) {` `  v.push_back(i*i);` `  }` `  return v;` `}`	
`vector<int> f4(size_t N) {` `  vector<int> v;` `  for(size_t i = 0; i < N; ++i) {` `  v.insert(v.begin() + v.size() / 2, i);` `  }` `  return v;` `}`	
`void f5(vector<int>& v, vector<int>& w, int L) {` `  size_t N = v.size(), M = w.size();` `  for(size_t i = 0; i < L; ++i) {` `  swap(v, w);` `  }` `}`	

```
vector<vector<int>> f6(size_t N) {
   vector<vector<int>> v;

   for(int i = 1; i <= N; ++i)
      v.emplace_back(i);

   for(auto& w : v)
      generate(w.begin(),w.end(),
           [N]() {
              return rand() % N;
           });

   sort(v.begin(),v.end(),
        [](vector<int> const& a,
           vector<int> const& b) {
         return accumulate(a.begin(),a.end(),0) <
                accumulate(b.begin(),b.end(),0);
   });

   return v;
}
```

```
vector<vector<int>> f7(size_t N) {
   vector<vector<int>> v;

   for(int i = 1; i <= N; ++i)
      v.emplace_back(i);

   for(auto& w : v)
      generate(w.begin(),w.end(),
           [N]() {
              return rand() % N;
           });

   sort(v.begin(),v.end(),
        [](vector<int> const& a,
           vector<int> const& b) {
           return a.front() < b.front();
        });

   return v;
}
```

Exercice 4.4 Tableau d'entiers redimensionnable

Définissez la fonction `inserer_en_fin` qui ajoute l'entier `val` en fin de tableau dont les données, taille et capacité sont stockée dans la structure `Tableau`, en réallouant si nécessaire la mémoire avec une capacité double.

```
#include <iostream>
using std::cout, std::endl, std::ostream;

struct Tableau {
   int* data = nullptr;
   size_t capacite = 0;
   size_t taille = 0;
};

void inserer_en_fin(Tableau& t, int val);
ostream& operator<<(ostream& out, Tableau const& t);

int main() {
   Tableau t1 { .data = nullptr, .capacite = 0, .taille = 0};
   for(int i = 0; i < 5; ++i)
   {
      cout << t1.taille << "/" << t1.capacite << " : " << t1 << endl;
      inserer_en_fin(t1,i*i);
   }
   delete t1.data;
}

ostream& operator<<(ostream& out, Tableau const& t) {
   out << "{";
   for(size_t i = 0; i < t.taille; ++i) {
      if(i) out << ", ";
      out << t.data[i];
   }
   return out << "}";
}
```

Le programme ci-dessus doit alors afficher

```
0/0 : {}
1/1 : {0}
2/2 : {0, 1}
3/4 : {0, 1, 4}
4/4 : {0, 1, 4, 9}
```

Veillez à ce que la fonction `inserer_en_fin` offre une garantie forte sur le contenu du tableau en cas d'exception levée par la réallocation de mémoire.

Exercice 4.5 Tableau générique redimensionnable (struct)

Même exercice que le 4.4, mais cette fois le tableau stocke des éléments de types quelconque. Allocation de la mémoire et construction doivent maintenant être séparés, de même que destruction et libération de la mémoire. Par ailleurs, il faut définir la fonction `detruire` en plus de la fonction `inserer_en_fin`. En utilisant la classe C de l'exercice 4.1, le code suivant

```
#include <iostream>
using std::cout, std::endl, std::ostream;
#include "ClasseEspion.h"

template<typename T>
struct Tableau {
   T* data = nullptr;
   size_t capacite = 0;
   size_t taille = 0;
};

template<typename T>
void inserer_en_fin(Tableau<T>& t, T val);

template<typename T>
void detruire(Tableau<T>& t) noexcept;

template<typename T>
ostream& operator<<(ostream& out, Tableau<T> const& t);

int main() {
   Tableau<C> tab { .data = nullptr, .capacite = 0, .taille = 0};
   for(int i = 0; i < 5; ++i)
   {
      cout << tab.taille << "/" << tab.capacite << " : " << tab << endl;
      inserer_en_fin(tab, C(i * i));
      cout << endl;
   }
   detruire(tab);
}

template<typename T>
ostream& operator<<(ostream& out, Tableau<T> const& t) {
   out << "{";
   for(size_t i = 0; i < t.taille; ++i) {
      if(i) out << ", ";
      out << t.data[i];
   }
   return out << "}";
}
```

Doit afficher le résultat suivant

```
0/0 : {}
C0 Cm0 D-1
1/1 : {0}
C1 Cm0 D-1 Cm1 D-1
2/2 : {0, 1}
C4 Cm0 Cm1 D-1 D-1 Cm4 D-1
3/4 : {0, 1, 4}
C9 Cm9 D-1
4/4 : {0, 1, 4, 9}
C16 Cm0 Cm1 Cm4 Cm9 D-1 D-1 D-1 D-1 Cm16 D-1
D16 D9 D4 D1 D0
```

La fonction `inserer_en_fin` doit offrir une garantie forte à condition que le constructeur de déplacement du type `T` soit `noexcept` ou absent. Il n'est nécessaire d'offrir aucune garantie s'il est présent mais pas `noexcept`.

Exercice 4.6 Tableau générique redimensionnable (class)

Réécrire le code de l'exercice 4.5 en faisant de Tableau une classe dont les données sont privées, et en transformant les différentes fonctions en méthodes de cette classe.

Exercice 4.7 Pile avec une liste simplement chainée

Soit la structure de maillon suivante

```
template<typename T>
struct Maillon {
   T val;
   Maillon* nxt;
};
```

Ecrire les fonctions génériques nécessaires pour que le code suivant compile et affiche `Hello, World!`

```
int main() {
   Maillon<string>* pile = nullptr;
   for (string const& s : { "!", "World", ", ", "Hello" } )
      push(pile,s);
   while(not empty(pile)) {
      cout << top(pile);
      pop(pile);
   }
   cout << endl;
}
```

Exercice 4.8 Pile avec une liste simplement chainée (2)

En utilisant la même structure de maillon qu'à l'exercice précédent, écrire la classe générique Pile pour que le code suivant compile et affiche `Hello, World!`

```
int main() {
   Pile<string> pile;
   for (string const& s : { "!", "World", ", ", "Hello" } )
      pile.push(s);
   while(not pile.empty()) {
      cout << pile.top();
      pile.pop();
   }
   cout << endl;
}
```

Exercice 4.9 Tri par sélection d'une `std::list`

En utilisant la méthode `splice` de `std::list` pour déplacer l'élément minimum plutôt que d'effectuer un échange comme lors du tri d'un tableau, écrivez la fonction générique `tri_selection` et l'`operator<<` qui permettent au programme suivant d'afficher `1 2 3 4 5 6 7.`

```
int main() {
   list<const int> v {3, 5, 4, 7, 6, 1, 2};
   cout << v << endl;
   tri_selection(v);
   cout << v << endl;
}
```

Notez qu'il s'agit d'une liste d'entiers constants, et que l'on ne peut donc modifier le contenu des maillons, mais seulement les déplacer.

Notez également que comme on déplace plutôt que d'échanger l'élément minimum, les états intermédiaires du conteneur sont

```
{ 3, 5, 4, 7, 6, 1, 2 }
{ 1, 3, 5, 4, 7, 6, 2 }
{ 1, 2, 3, 5, 4, 7, 6 }
{ 1, 2, 3, 5, 4, 7, 6 }
{ 1, 2, 3, 4, 5, 7, 6 }
{ 1, 2, 3, 4, 5, 7, 6 }
{ 1, 2, 3, 4, 5, 6, 7 }
```

Exercice 4.10 Tri par sélection d'une `std::forward_list`

En utilisant la méthode `splice_after` de `std::forward_list` pour déplacer l'élément minimum plutôt que d'effectuer un échange comme les tableaux, écrivez la fonction générique `tri_selection` et l'`operator<<` qui permettent au programme suivant d'afficher `1 2 3 4 5 6 7.`

```
int main() {
   forward_list<const int> v {3, 5, 4, 7, 6, 1, 2};
   cout << v << endl;
   tri_selection(v);
   cout << v << endl;
}
```

Exercice 4.11 Inversion d'une `std::forward_list`

En utilisant la méthode `splice_after` de `std::forward_list` et pas sa méthode `reverse`, écrivez la fonction générique `reverse` et l'`operator<<` qui permettent au programme suivant d'afficher `1 2 3 4 5 6 7.`

```
int main() {
   forward_list<const int> v {7, 6, 5, 4, 3, 2, 1};
   cout << v << endl;
   reverse(v);
   cout << v << endl;
}
```

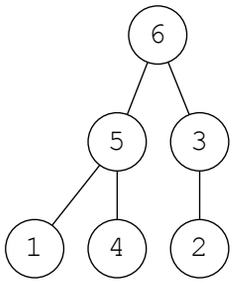
Exercice 4.12 Inversion d'une `std::list`

En utilisant la méthode `splice` de `std::list` et pas sa méthode `reverse`, écrivez la fonction générique `reverse` et l'`operator<<` qui permettent au programme suivant d'afficher `1 2 3 4 5 6 7.`

```
int main() {
   list<const int> v {7, 6, 5, 4, 3, 2, 1};
   cout << v << endl;
   reverse(v);
   cout << v << endl;
}
```

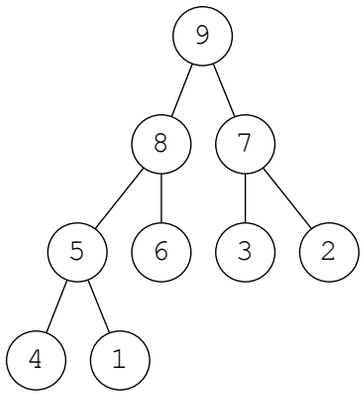
Exercice 4.13 Insertion dans un tas

Insérer dans cet ordre les valeurs 8, 10, 9, et 7 dans le tas suivant



Exercice 4.14 Suppression du sommet d'un tas

Supprimer 5 fois de suite le sommet du tas suivant



Exercice 4.15 Création d'un tas

Créez un tas avec l'algorithme en O(n) à partir des tableaux suivants

1. `[ 1, 2, 3, 4, 5, 6, 7 ]`
2. `[ 3, 5, 7, 1, 4, 6, 9, 8 ]`
3. `[ 1, 5, 7, 3, 4, 6, 8, 2, 9, 10 ]`

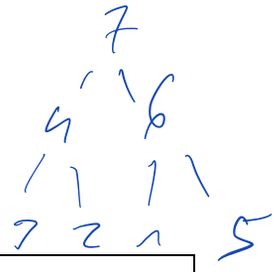
Exercice 4.16 Tri par tas

Triez par tas les tableaux suivants, en indiquant l'état du tableau après chaque descente / montée d'un élément dans le tableau

1. `[ 1, 3, 2, 5, 4, 6 ]`

2. `[ 1, 5, 3, 6, 4, 2, 7 ]`

3. `[ 3, 5, 2, 1, 6, 7, 4 ]`



Exercice 4.17 Tas dans la librairie `<algorithm>`

Qu'affichent les extraits de code suivants ?

`vector v {6, 4, 5, 3, 2, 1, 7};` `push_heap(v.begin(),v.end());` `for (auto e: v) cout << e << " ";`	7463215
`vector v {7, 6, 5, 3, 2, 1, 4};` `push_heap(v.begin(), v.end());` `for (auto e: v) cout << e << " ";`	
`vector v {7, 6, 4, 3, 2, 1, 5};` `for(int i = 0; i < v.size(); ++i)` `  push_heap(v.begin(),v.end());` `for (auto e: v) cout << e << " ";`	7653214
`vector v {6, 4, 5, 3, 2, 1};` `pop_heap(v.begin(),v.end());` `for (auto e: v) cout << e << " ";`	
`vector v {7, 6, 5, 3, 2, 1, 4};` `push_heap(v.begin(),v.end());` `pop_heap(v.begin(),v.end());` `for (auto e: v) cout << e << " ";`	

`vector v {1, 3, 5, 4, 2, 6};` `make_heap(v.begin(), v.end(),std::greater<int>());` `for (auto e: v) cout << e << " ";`	
`vector v {1, 2, 5, 3, 4, 6};` `pop_heap(v.begin(), v.end(),std::greater<int>());` `for (auto e: v) cout << e << " ";`	
`vector v {1, 3, 5, 4, 6, 2};` `push_heap(v.begin(), v.end(),std::greater<int>());` `for (auto e: v) cout << e << " ";`	
`vector v {6, 4, 5, 3, 1, 2};` `sort_heap(v.begin(), v.end());` `for (auto e: v) cout << e << " ";`	
`vector v {1, 2, 3, 6, 4, 5};` `sort_heap(v.begin(), v.end());` `for (auto e: v) cout << e << " ";`	

Note: `std::greater<T>` est un foncteur défini dans la librairie `<functional>` qui fournit une fonction équivalente à `operator>`.

Exercice 4.18 Coût en mémoire des structures de donnée

Quel est le coût en mémoire (en octets) des différentes structures de données présentes dans le code suivant, en fonction de la constante `N`.

```
assert(sizeof(void*) == 8);

array<int16_t,0> array1{};

vector<int16_t> vector1;

list<int16_t> list1;

forward_list<int16_t> forwardList1;

array<int32_t,N> array2{};

vector<int32_t> vector2(N);

list<int32_t> list2(array2.begin(),array2.end());

forward_list<int32_t> forwardList2(array2.begin(),array2.end());

array<int8_t,N> array3{};

array<array<int8_t,N>,N> array4{};

array<vector<int8_t>,N> array5;
array5.fill(vector<int8_t>(N));

vector<array<int8_t,N>> vector3(N);

vector<vector<int8_t>> vector4(N,vector<int8_t>(N));

list<int8_t> list3(array3.begin(), array3.end());

vector<list<int8_t>> vector5(N,list3);

list<vector<int8_t>> list4(array5.begin(),array5.end());

list<list<int8_t>> list5(vector5.begin(),vector5.end());

vector<int8_t> vector6;
for(int i = 0; i < N; ++i)
   vector6.push_back(0);
```

Exercice 4.19 Itérateurs et splice

Qu'affichent les extraits de code suivants ? Il est possible que certains d'entre eux ne compilent pas, aient un comportement indéterminé (en pratique le programme crashe), ou donnent une boucle infinie. Indiquez-le.

`vector v { 1, 2, 3, 4, 5};` `for(auto it = v.begin(); it != v.end(); ++it)` `  if(*it == 2)` `  v.push_back(0);` `for(int e : v) cout << e << " ";`	
`list v { 1, 2, 3, 4, 5};` `for(auto it = v.begin(); it != v.end(); ++it)` `  if(*it == 2)` `  v.push_back(0);` `for(int e : v) cout << e << " ";`	
`forward_list v { 1, 2, 3, 4, 5};` `for(auto it = v.begin(); it != v.end(); ++it)` `  if(*it == 2)` `  v.push_back(0);` `for(int e : v) cout << e << " ";`	
`list v { 1, 2, 3, 4, 5, 6};` `v.splice(next(v.begin(),1), v, next(v.begin(),3));` `for(int e : v) cout << e << " ";`	
`list v { 1, 2, 3, 4, 5, 6};` `v.splice(next(v.begin(),1), v,` `next(v.begin(),3),v.end());` `for(int e : v) cout << e << " ";`	
`forward_list v { 1, 2, 3, 4, 5, 6};` `v.splice_after(next(v.begin(),1), v,` `next(v.begin(),3));` `for(int e : v) cout << e << " ";`	

`` forward_list v { 1, 2, 3, 4, 5, 6}; v.splice_after(next(v.begin(),1), v, next(v.begin(),3),v.end()); for(int e : v) cout << e << " "; ```	
`` list v { 1, 2, 3, 4, 5}; for(auto it = v.begin(); it != v.end(); ++it) v.splice(v.begin(),v,it); for(int e : v) cout << e << " "; ```	
`` list v { 1, 2, 3, 4, 5}; auto it = v.begin(); while(it != v.end()) { auto nxt = next(it); v.splice(v.begin(), v, it); it = nxt; } for(int e : v) cout << e << " "; ```	
`` forward_list v { 1, 2, 3, 4, 5}; for(auto it = v.before_begin(); it != v.end(); ++it) v.splice_after(v.before_begin(),v,it); for(int e : v) cout << e << " "; ```	
`` forward_list v { 1, 2, 3, 4, 5}; for(auto it = v.before_begin(); next(it) != v.end(); ++it) v.splice_after(v.before_begin(),v,it); for(int e : v) cout << e << " "; ```	
`` vector v { 1, 2, 3, 4, 5}; for(auto it = v.begin(); it != v.end(); ++it) v.splice(v.begin(),v,it); for(int e : v) cout << e << " "; ```	

`` vector v { 1, 2, 3, 4, 5}; v.reserve(6); for(auto it = v.begin(); it != v.end(); ++it) if(*it == 2) v.push_back(0); for(int e : v) cout << e << " "; ```	
`` list v { 1, 2, 3, 4, 5}; for(auto it = v.begin(); it != v.end(); ++it) v.insert(it,*it); for(int e : v) cout << e << " "; ```	
`` vector v { 1, 2, 3, 4, 5}; v.reserve(10); for(auto it = v.begin(); it != v.end(); ++it) v.insert(it,*it); for(int e : v) cout << e << " "; cout << endl; ```	
`` vector v { 1, 2, 3, 4, 5}; v.reserve(10); for(auto it = v.begin(); it != v.end(); ++it) v.erase(it); for(int e : v) cout << e << " "; cout << endl; ```	
`` vector v { 1, 2, 3, 4, 5, 6}; v.reserve(10); for(auto it = v.begin(); it != v.end(); ++it) v.erase(it); for(int e : v) cout << e << " "; cout << endl; ```	

Exercice 4.20 Algorithme de Dijkstra

Appliquez l'algorithme d'évaluation d'expression de Dijkstra aux expressions suivantes. Indiquez le contenu des piles des valeurs et des opérateurs à chaque étape. Donnez l'expression équivalente en notation polonaise inverse.

1. `( 1 + ( 2 * ( 7 - ( ( 3 * 4 ) / 2 ))))`
2. `( ( 3 * 5 ) + ( 2 * ( 6 - ( 2 + 3 ) ) ) )`
3. `( ( 2 + ( 3 * 5 ) ) - ( ( 3 + 2 ) / 5 ) )`

Solutions

Exercice 4.1 - std::vector à capacité fixe

`int main() {`	
`  vector<C> v;`	
`  v.reserve(10);`	
`  v.emplace_back();`	`CD`
`  v.emplace_back(1);`	`C1`
`  v.emplace_back(C(2));`	`C2 Cm2 D-1`
`  v.push_back(3);`	`C3 Cm3 D-1`
`  v.push_back(C(4));`	`C4 Cm4 D-1`
`  v[0] = v[4];`	`=c4`
`  v[1] = v[1];`	`=c1x`
`  v[2] = C(2);`	`C2 =m2 D-2`
`  v[3] = 3;`	`C3 =m3 D-2`
`  v[4] = std::move(v[4]);`	`=m4x`
`  v.insert(v.begin()+2,C(5));`	`C5 Cm4 =m3 =m2 =m5 D-2`
`  v.erase(v.begin()+1);`	`=m5 =m2 =m3 =m4 D-2`
`  v.clear();`	`D4 D3 D2 D5 D4`
`}`	

Exercice 4.2 - std::vector à capacité qui change

`int main() {`	
`  vector<C> v;`	
`  v.emplace_back();`	`CD`
`  v.emplace_back(1);`	`C1 Cm0 D-1`
`  v.emplace_back(C(2));`	`C2 Cm2 Cm1 Cm0 D-1 D-1 D-1`
`  v.emplace_back(3);`	`C3`
`  v.insert(v.begin()+2,C(4));`	`C4 Cm4 Cm1 Cm0 Cm2 Cm3 D-1 D-1 D-1 D-1 D-1`
`  v.insert(v.begin()+3,C(5));`	`C5 Cm3 =m2 =m5 D-2`
`  v.erase(v.begin()+1);`	`=m4 =m5 =m2 =m3 D-2`
`}`	`D3 D2 D5 D4 D0`

Note : l'ordre exact des `Cm#` lors des insertions de `C(2)` et `C(4)` n'est pas garanti.

Exercice 4.3 – Complexités de std::vector

1. $O(N \cdot M)$
2. $O(N)$
3. $O(\mathrm{N})$
4. $O(N^2)$
5. $O(L)$
6. $O(N^2 \cdot \log N)$
7. $O(N^2)$

Exercice 4.4 Tableau d'entiers redimensionnable

```
#include <iostream>
using std::cout, std::endl, std::ostream;
#include <utility>
using std::swap;

struct Tableau {
   int* data = nullptr;
   size_t capacite = 0;
   size_t taille = 0;
};

void inserer_en_fin(Tableau& t, int val);
ostream& operator<<(ostream& out, Tableau const& t);

int main() {
   Tableau t1 { .data = nullptr, .capacite = 0, .taille = 0};
   for(int i = 0; i < 5; ++i)
   {
      cout << t1.taille << "/" << t1.capacite << " : " << t1 << endl;
      inserer_en_fin(t1,i*i);
   }
   delete t1.data;
}

ostream& operator<<(ostream& out, Tableau const& t) {
   out << "{";
   for(size_t i = 0; i < t.taille; ++i) {
      if(i) out << ", ";
      out << t.data[i];
   }
   return out << "}";
}

void inserer_en_fin(Tableau& t, int val) {
   if(t.taille == t.capacite) {
      size_t newcap = t.capacite ? t.capacite * 2 : 1;
      auto tmp = new int[newcap];
      // noexcept after this line
      memcpy(tmp,t.data,sizeof(int) * t.capacite);
      swap(t.data,tmp);
      t.capacite = newcap;
      delete tmp;
   }
   t.data[t.taille] = val;
   ++t.taille;
}
```

Exercice 4.5 Tableau générique redimensionnable (struct)

```
#include <iostream>
using std::cout, std::endl, std::ostream;
#include "ClasseEspion.h"

#include <utility>
using std::swap, std::move;
#include <memory>
using std::destroy_at;

template<typename T>
struct Tableau {
   T* data = nullptr;
   size_t capacite = 0;
   size_t taille = 0;
};

template<typename T>
void inserer_en_fin(Tableau<T>& t, T val);

template<typename T>
void detruire(Tableau<T>& t) noexcept;

template<typename T>
ostream& operator<<(ostream& out, Tableau<T> const& t);

int main() {
   Tableau<C> tab { .data = nullptr, .capacite = 0, .taille = 0};
   for(int i = 0; i < 5; ++i)
   {
      cout << tab.taille << "/" << tab.capacite << " : " << tab << endl;
      inserer_en_fin(tab, C(i * i));
      cout << endl;
   }
   detruire(tab);
}

template<typename T>
ostream& operator<<(ostream& out, Tableau<T> const& t) {
   out << "{";
   for(size_t i = 0; i < t.taille; ++i) {
      if(i) out << ", ";
      out << t.data[i];
   }
   return out << "}";
}


template<typename T>
```

```
void inserer_en_fin(Tableau<T>& t, T val) {
   if(t.taille == t.capacite) {
      size_t newcap = t.capacite ? t.capacite * 2 : 1;
      Tableau<T> tmp {
         .data = (T*) ::operator new(sizeof(C) * newcap),
         .taille = t.taille,
         .capacite = newcap
      };
      for(size_t i = 0; i < t.taille; ++i) {
         new(tmp.data+i) C(move(t.data[i]));
      }
      swap(t,tmp);
      detruire(tmp);
   }
   new(t.data + t.taille) C(move(val));
   ++t.taille;
}

template<typename T>
void detruire(Tableau<T>& t) noexcept {
   size_t i = t.taille;
   while (i > 0) {
      --i;
      destroy_at(t.data + i); // equivalent à t.data[i].~T();
   }
   ::operator delete(t.data);
   t.taille = t.capacite = 0;
   t.data = nullptr;
}
```

Exercice 4.6 Tableau générique redimensionnable (class)

```
#include <iostream>
using std::cout, std::endl, std::ostream;
#include <utility>
using std::swap, std::move;
#include <memory>
using std::destroy_at;
#include "ClasseEspion.h"

template<typename T>
class Tableau {
public:
   Tableau();
   void inserer_en_fin(T val);
   size_t taille() const { return _taille; }
   size_t capacite() const { return _capacite; }
   T& operator[](size_t i) { return _data[i]; }
   T const& operator[](size_t i) const { return _data[i]; }
   void swap(Tableau& tab);
   ~Tableau();
private:
   T* _data;
   size_t _capacite;
   size_t _taille;
};

template<typename T>
ostream& operator<<(ostream& out, Tableau<T> const& t);

int main() {
   Tableau<C> tab;
   for(int i = 0; i < 5; ++i)
   {
      cout << tab.taille() << "/" << tab.capacite() << " : " << tab << endl;
      tab.inserer_en_fin(C(i * i));
      cout << endl;
   }
}

template<typename T>
ostream& operator<<(ostream& out, Tableau<T> const& t) {
   out << "{";
   for(size_t i = 0; i < t.taille(); ++i) {
      if(i) out << ", ";
      out << t[i];
   }
   return out << "}";
}
```

```
template<typename T>
Tableau<T>::Tableau() : _data(nullptr), _taille(0), _capacite(0) {
}

template<typename T>
void Tableau<T>::inserer_en_fin(T val) {
   if (_taille == _capacite) {
      size_t newcap = _capacite ? _capacite * 2 : 1;
      Tableau<T> tmp;
      tmp._data = (T*) ::operator new(sizeof(C) * newcap);
      tmp._taille = _taille;
      tmp._capacite = newcap;
      for (size_t i = 0; i < _taille; ++i) {
         new(tmp._data + i) C(move(_data[i]));
      }
      swap(tmp);
   }
   new(_data + _taille) C(move(val));
   ++_taille;
}

template<typename T>
void Tableau<T>::swap(Tableau<T>& tab) {
   std::swap(_data, tab._data);
   std::swap(_taille, tab._taille);
   std::swap(_capacite, tab._capacite);
}

template<typename T>
Tableau<T>::~Tableau() {
   size_t i = _taille;
   while (i > 0) {
      --i;
      destroy_at(_data + i); // equivalent à t.data[i].~T();
   }
   ::operator delete(_data);
}
```

Note hors matière d'ASD : il est possible d'également offrir une garantie forte si le constructeur de déplacement de `T` n'est pas `noexcept`. Pour cela, il faut utiliser la librairie `<type_traits>` et remplacer la ligne.

```
   new(tmp._data + i) C(move(_data[i]));
```

par

```
   if constexpr (std::is_nothrow_move_constructible<T>::value) {
      new(tmp._data + i) C(move(_data[i]));
   } else {
      new(tmp._data + i) C(_data[i]);
   }
```

Exercice 4.7 Pile avec une liste simplement chainée

```
#include <iostream>
using std::cout;
#include <ostream>
using std::endl;
#include <string>
using std::string;

template<typename T>
struct Maillon {
   T val;
   Maillon* nxt;
};

template<typename T>
void push(Maillon<T>*& first, T const& t) {
   first = new Maillon<T>{ .val = t, .nxt = first };
}

template<typename T>
void pop(Maillon<T>*& first) {
   if(first == nullptr) return;
   auto tmp = first;
   first = first->nxt;
   delete tmp;
}

template<typename T>
T const& top(const Maillon<T>* first) {
   return first->val;
}


template<typename T>
bool empty(const Maillon<T>* first) {
   return first == nullptr;
}

int main() {
   Maillon<string>* pile = nullptr;
   for (string const& s : { "!", "World", ", ", "Hello" } )
      push(pile,s);
   while(not empty(pile)) {
      cout << top(pile);
      pop(pile);
   }
   cout << endl;
}
```

Exercice 4.8 Pile avec une liste simplement chainée (2)

```
#include <iostream>
using std::cout;
#include <ostream>
using std::endl;
#include <string>
using std::string;

template<typename T>
class Pile {
   struct Maillon {
      T val;
      Maillon* nxt;
   };
   Maillon* first;
public:
   Pile() : first(nullptr) {}

   void push(T const& t) {
      first = new Maillon{.val = t, .nxt = first};
   }

   void pop() {
      if (first == nullptr) return;
      auto tmp = first;
      first = first->nxt;
      delete tmp;
   }

   T const& top() const {
      return first->val;
   }

   bool empty() const {
      return first == nullptr;
   }
};

int main() {
   Pile<string> pile;
   for (string const& s : { "!", "World", ", ", "Hello" } )
      pile.push(s);
   while(not pile.empty()) {
      cout << pile.top();
      pile.pop();
   }
   cout << endl;
}
```

Exercice 4.9 Tri par sélection d'une *std::list*

```
#include <iostream>
using std::cout;
#include <ostream>
using std::ostream, std::endl;
#include <list>
using std::list;
#include <algorithm>
using std::min_element;

template<typename Iterator>
ostream& displayContainer(ostream& out,
                          Iterator first, Iterator last ) {
   out << "{ ";
   for(auto i = first; i != last; ++i) {
      if(i != first) out << ", ";
      out << *i;
   }
   return out << " }";
}

template<typename T>
ostream& operator<<(ostream& out, std::list<T> const& v) {
   return displayContainer(out,v.begin(),v.end());
}

template<typename T>
void tri_selection(list<T>& v) {
   auto i = v.begin();
   while (next(i) != v.end()) {
      auto j = min_element(i,v.end());
      v.splice(i, v, j);
      i = next(j);
   }
}

int main() {
   list<const int> v {3, 5, 4, 7, 6, 1, 2};
   cout << v << endl;
   tri_selection(v);
   cout << v << endl;
}
```

Exercice 4.10 Tri par sélection d'une *std::forward_list*

```
#include <iostream>
using std::cout;
#include <ostream>
using std::ostream, std::endl;
#include <forward_list>
using std::forward_list;
#include <iterator>
using std::next;

// même function display_container qu'à l'exercice 4.9

template<typename T>
ostream& operator<<(ostream& out, std::forward_list<T> const& v) {
   return displayContainer(out,v.begin(),v.end());
}

template<typename T>
void tri_selection(forward_list<T>& v) {
   for (auto pi = v.before_begin(); next(pi,2) != v.end(); ++pi) {
      auto pmin = pi;
      for(auto pj = pi; next(pj) != v.end(); ++pj) {
         if(*next(pj) < *next(pmin))
            pmin = pj;
      }
      v.splice_after(pi, v, pmin);
   }
}

int main() {
   forward_list<const int> v {3, 5, 4, 7, 6, 1, 2};
   cout << v << endl;
   tri_selection(v);
   cout << v << endl;
}
```

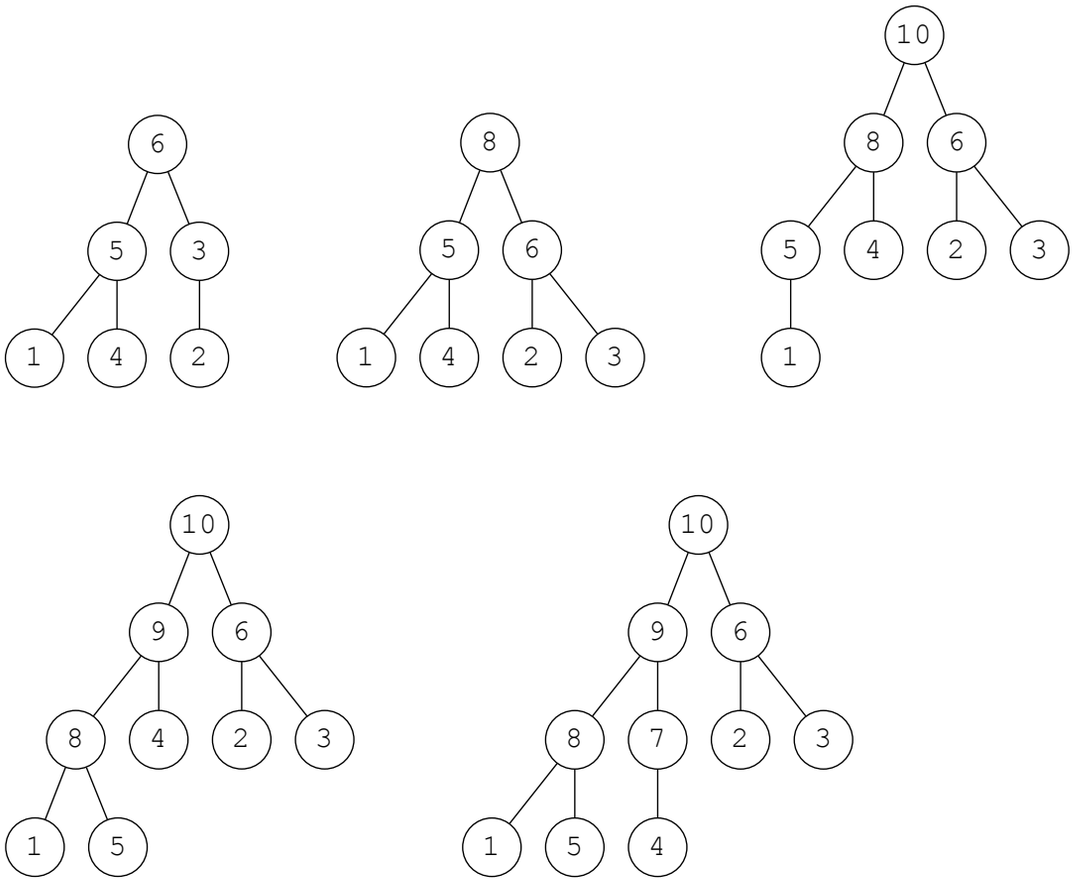
Exercice 4.11 Inversion d'une *std::forward_list*

```
// même #include et operator<< qu'à l'exercice 4.10
#include <iterator>
using std::next;

template<typename T>
void reverse(forward_list<T>& v) {
   auto first = v.begin();
   while(next(first) != v.end())
      v.splice_after(v.before_begin(),v,first);
}

int main() {
   forward_list<const int> v {7, 6, 5, 4, 3, 2, 1};
   cout << v << endl;
   reverse(v);
   cout << v << endl;
}
```

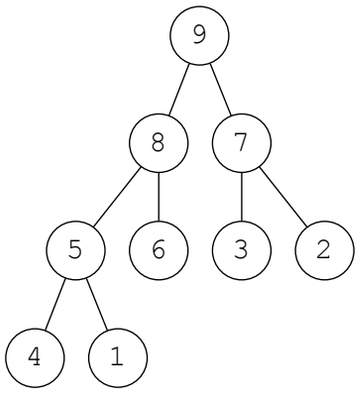
Exercice 4.12 Inversion d'une *std::list*

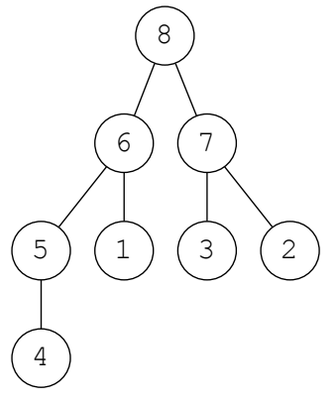
```
// même #include et operator<< qu'à l'exercice 4.9
#include <iterator>
using std::next;

template<typename T>
void reverse(list<T>& v) {
   auto first = v.begin();
   while(next(first) != v.end())
      v.splice(v.begin(),v,next(first));
}

int main() {
   list<const int> v {7, 6, 5, 4, 3, 2};
   cout << v << endl;
   reverse(v);
   cout << v << endl;
}
```

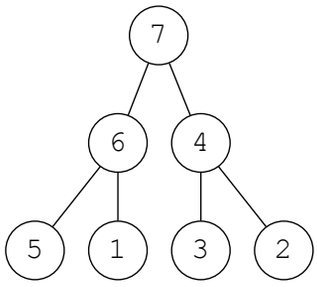
Exercice 4.13 Insertion dans un tas

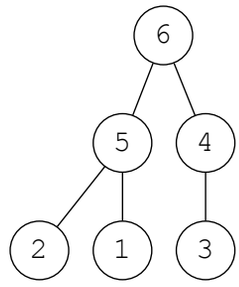


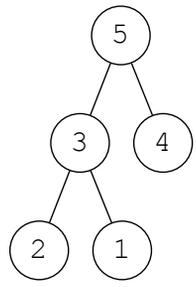
Exercice 4.14 Suppression du sommet d'un tas

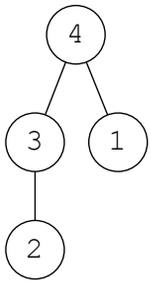






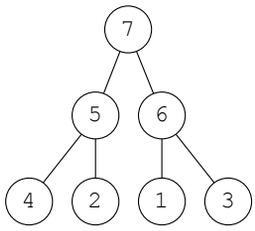




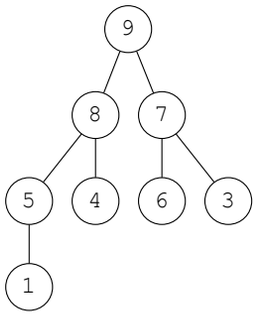


Exercice 4.15 Création d'un tas

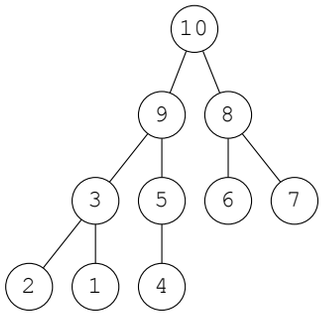
1. `[ 1, 2, 3, 4, 5, 6, 7 ] -> [7, 5, 6, 4, 2, 1, 3]`



2. `[ 3, 5, 7, 1, 4, 6, 9, 8 ] -> [9, 8, 7, 5, 4, 6, 3, 1]`



3. `[ 1, 5, 7, 3, 4, 6, 8, 2, 9, 10 ] -> [10, 9, 8, 3, 5, 6, 7, 2, 1, 4]`



Exercice 4.16 Tri par tas

1. [1, 3, 2, 5, 4, 6]
   [1, 3, 6, 5, 4, 2]
   [1, 5, 6, 3, 4, 2]
   [6, 5, 2, 3, 4, 1]
   [6, 5, 2, 3, 4, 1]
   [5, 4, 2, 3, 1] [6]
   [4, 3, 2, 1] [5, 6]
   [3, 1, 2] [4, 5, 6]
   [2, 1] [3, 4, 5, 6]
   [1] [2, 3, 4, 5, 6]

2. [1, 5, 3, 6, 4, 2, 7]
   [1, 5, 7, 6, 4, 2, 3]
   [1, 6, 7, 5, 4, 2, 3]
   [7, 6, 3, 5, 4, 2, 1]
   [7, 6, 3, 5, 4, 2, 1]
   [6, 5, 3, 1, 4, 2] [7]
   [5, 4, 3, 1, 2] [6, 7]
   [4, 2, 3, 1] [5, 6, 7]
   [3, 2, 1] [4, 5, 6, 7]
   [2, 1] [3, 4, 5, 6, 7]
   [1] [2, 3, 4, 5, 6, 7]

3. [3, 5, 2, 1, 6, 7, 4]
   [3, 5, 7, 1, 6, 2, 4]
   [3, 6, 7, 1, 5, 2, 4]
   [7, 6, 4, 1, 5, 2, 3]
   [7, 6, 4, 1, 5, 2, 3]
   [6, 5, 4, 1, 3, 2] [7]
   [5, 3, 4, 1, 2] [6, 7]
   [4, 3, 2, 1] [5, 6, 7]
   [3, 1, 2] [4, 5, 6, 7]
   [2, 1] [3, 4, 5, 6, 7]
   [1] [2, 3, 4, 5, 6, 7]

Exercice 4.17 Tas dans la librairie <algorithm>

1. 7 4 6 3 2 1 5
2. 7 6 5 3 2 1 4
3. 7 6 5 3 2 1 4
4. 5 4 1 3 2 6
5. 6 4 5 3 2 1 7
6. 1 2 5 4 3 6
7. 2 3 5 6 4 1
8. 1 3 2 4 6 5
9. 1 2 3 4 5 6
10. 2 3 6 4 5 1

Exercice 4.18 Coût en mémoire des structures de donnée

```
array<int16_t,0> array1{};
// 0 octets

vector<int16_t> vector1;
// 24 octets (3 pointeurs)

list<int16_t> list1;
// 24 octets (2 pointeurs + 1 size_t)

forward_list<int16_t> forwardList1;
// 8 octets (1 pointeur)

array<int32_t,N> array2{};
// 4*N octets
// avec 4 = sizeof(int32_t)

vector<int32_t> vector2(N);
// 24 + 4*N octets

list<int32_t> list2(array2.begin(),array2.end());
// 24 + 20*N octets (1 int32_t et 2 pointeurs par élément)

forward_list<int32_t> forwardList2(array2.begin(),array2.end());
// 8 + 12*N octets (1 int32_t et 1 pointeur par élément)

array<int8_t,N> array3{};
// 1*N
// avec 1 = sizeof(int8_t)

array<array<int8_t,N>,N> array4{};
// 1*N^2
```

```
array<vector<int8_t>,N> array5;
array5.fill(vector<int8_t>(N));
// N * ( 24 + 1*N ) = 24*N + 1*N^2

vector<array<int8_t,N>> vector3(N);
// 24 + 1*N^2

vector<vector<int8_t>> vector4(N,vector<int8_t>(N));
// 24 + N * ( 24 + 1*N ) = 24 + 24*N + 1*N^2

list<int8_t> list3(array3.begin(), array3.end());
// 24 + 17*N
// avec 17 = 1 * sizeof(int8_t) + 2 * sizeof(void*)

vector<list<int8_t>> vector5(N,list3);
// 24 + N * (24 + 17*N) = 24 + 24*N + 17*N^2

list<vector<int8_t>> list4(array5.begin(),array5.end());
// 24 + N * ( 40 + 1 * N ) = 24 + 40*N + 1*N^2
// avec 40 = 2 pointeurs par élément + sizeof(vector<int8_t>)

list<list<int8_t>> list5(vector5.begin(),vector5.end());
// 24 + N * ( 40 + 17*N ) = 24 + 40*N + 17*N^2
// avec 40 = 2 pointeurs par élément + sizeof(list<int8_t>)

vector<int8_t> vector6;
for(int i = 0; i < N; ++i)
   vector6.push_back(0);
// 24 + 1*M
// avec M le plus petite puissance de F >= N, avec F le facteur
// multiplicatif d'augmentation de la capacité, typiquement F=2
```

Exercice 4.19 Itérateurs et splice

`vector v { 1, 2, 3, 4, 5};` `for(auto it = v.begin(); it != v.end(); ++it)` `  if(*it == 2)` `  v.push_back(0);` `for(int e : v) cout << e << " ";`	`Indéterminé.` L'itérateur n'est plus valide après que push_back augmente la capacité
`list v { 1, 2, 3, 4, 5};` `for(auto it = v.begin(); it != v.end(); ++it)` `  if(*it == 2)` `  v.push_back(0);` `for(int e : v) cout << e << " ";`	`1 2 3 4 5 0`
`forward_list v { 1, 2, 3, 4, 5};` `for(auto it = v.begin(); it != v.end(); ++it)` `  if(*it == 2)` `  v.push_back(0);` `for(int e : v) cout << e << " ";`	`Ne compile pas` Le conteneur `forward_list` n'a pas de méthode `push_back`
`list v { 1, 2, 3, 4, 5, 6};` `v.splice(next(v.begin(),1), v, next(v.begin(),3));` `for(int e : v) cout << e << " ";`	`1 4 2 3 5 6`
`list v { 1, 2, 3, 4, 5, 6};` `v.splice(next(v.begin(),1), v,` `  next(v.begin(),3),v.end());` `for(int e : v) cout << e << " ";`	`1 4 5 6 2 3`
`forward_list v { 1, 2, 3, 4, 5, 6};` `v.splice_after(next(v.begin(),1), v,` `  next(v.begin(),3));` `for(int e : v) cout << e << " ";`	`1 2 5 3 4 6`
`forward_list v { 1, 2, 3, 4, 5, 6};` `v.splice_after(next(v.begin(),1), v,` `next(v.begin(),3),v.end());` `for(int e : v) cout << e << " ";`	`1 2 5 6 3 4`

`` list v { 1, 2, 3, 4, 5}; for(auto it = v.begin(); it != v.end(); ++it) v.splice(v.begin(),v,it); for(int e : v) cout << e << " "; ```	Boucle infinie La liste oscille entre 1 2 3 4 5 et 2 1 3 4 5 car le splice ramène it en première position
`` list v { 1, 2, 3, 4, 5}; auto it = v.begin(); while(it != v.end()) { auto nxt = next(it); v.splice(v.begin(), v, it); it = nxt; } for(int e : v) cout << e << " "; ```	5 4 3 2 1
`` forward_list v { 1, 2, 3, 4, 5}; for(auto it = v.before_begin(); it != v.end(); ++it) v.splice_after(v.before_begin(),v,it); for(int e : v) cout << e << " "; ```	Indéterminé L'itérateur va une position trop loin, le dernier splice_after crashe.
`` forward_list v { 1, 2, 3, 4, 5}; for(auto it = v.before_begin(); next(it) != v.end(); ++it) v.splice_after(v.before_begin(),v,it); for(int e : v) cout << e << " "; ```	4 2 1 3 5
`` vector v { 1, 2, 3, 4, 5}; for(auto it = v.begin(); it != v.end(); ++it) v.splice(v.begin(),v,it); for(int e : v) cout << e << " "; ```	5 4 3 2 1
`` vector v { 1, 2, 3, 4, 5}; v.reserve(6); for(auto it = v.begin(); it != v.end(); ++it) if(*it == 2) v.push_back(0); for(int e : v) cout << e << " "; ```	1 2 3 4 5 0

`` list v { 1, 2, 3, 4, 5}; for(auto it = v.begin(); it != v.end(); ++it) v.insert(it,*it); for(int e : v) cout << e ; ```	`1122334455`
`` vector v { 1, 2, 3, 4, 5}; v.reserve(10); for(auto it = v.begin(); it != v.end(); ++it) v.insert(it,*it); for(int e : v) cout << e << " "; cout << endl; ```	`Indéterminé` Les 5 premières itérations donnent `1 1 1 1 1 1 2 3 4 5`, l'insert suivant augmente la capacité et invalide l'itérateur Sans ce problème d'itérateur, on aurait une boucle infinie
`` vector v { 1, 2, 3, 4, 5}; v.reserve(10); for(auto it = v.begin(); it != v.end(); ++it) v.erase(it); for(int e : v) cout << e << " "; cout << endl; ```	`Indéterminé` La boucle efface les éléments 1, 3 et 5, mais ne s'arrête pas car alors it vaut `v.end()+1`. Crash très probable sur l'erase suivant
`` vector v { 1, 2, 3, 4, 5, 6}; v.reserve(10); for(auto it = v.begin(); it != v.end(); ++it) v.erase(it); for(int e : v) cout << e << " "; cout << endl; ```	`2 4 6`

Exercice 4.20 Algorithme de Dijkstra

1. `( 1 + ( 2 * ( 7 - ( ( 3 * 4 ) / 2 ))))`

Notation polonaise inverse : `1 2 7 3 4 * 2 / - * +`

Algorithme :

```
Valeurs : 1              Operateurs :
Valeurs : 1              Operateurs : +
Valeurs : 1 2            Operateurs : +
Valeurs : 1 2            Operateurs : + *
Valeurs : 1 2 7          Operateurs : + *
Valeurs : 1 2 7          Operateurs : + * -
Valeurs : 1 2 7 3        Operateurs : + * -
Valeurs : 1 2 7 3        Operateurs : + * - *
Valeurs : 1 2 7 3 4      Operateurs : + * - *
Valeurs : 1 2 7 12       Operateurs : + * -
Valeurs : 1 2 7 12       Operateurs : + * - /
Valeurs : 1 2 7 12 2     Operateurs : + * - /
Valeurs : 1 2 7 6        Operateurs : + * -
Valeurs : 1 2 1          Operateurs : + *
Valeurs : 1 2            Operateurs : +
Valeurs : 3              Operateurs :
```

2. `( ( 3 * 5 ) + ( 2 * ( 6 - ( 2 + 3 ) ) ) )`

Notation polonaise inverse : `3 5 * 2 6 2 3 + - * +`

Algorithme :

```
Valeurs : 3              Operateurs :
Valeurs : 3              Operateurs : *
Valeurs : 3 5            Operateurs : *
Valeurs : 15             Operateurs :
Valeurs : 15             Operateurs : +
Valeurs : 15 2           Operateurs : +
Valeurs : 15 2           Operateurs : + *
Valeurs : 15 2 6         Operateurs : + *
Valeurs : 15 2 6         Operateurs : + * -
Valeurs : 15 2 6 2       Operateurs : + * -
Valeurs : 15 2 6 2       Operateurs : + * - +
Valeurs : 15 2 6 2 3     Operateurs : + * - +
Valeurs : 15 2 6 5       Operateurs : + * -
Valeurs : 15 2 1         Operateurs : + *
Valeurs : 15 2           Operateurs : +
Valeurs : 17             Operateurs :
```

3. ((2 + (3 * 5)) - ((3 + 2) / 5))

Notation polonaise inverse : `2 3 5 * + 3 2 + 5 / -`

Algorithme :

```
Valeurs : 2             Operateurs :
Valeurs : 2             Operateurs : +
Valeurs : 2 3           Operateurs : +
Valeurs : 2 3           Operateurs : + *
Valeurs : 2 3 5         Operateurs : + *
Valeurs : 2 15          Operateurs : +
Valeurs : 17            Operateurs :
Valeurs : 17            Operateurs : -
Valeurs : 17 3          Operateurs : -
Valeurs : 17 3          Operateurs : - +
Valeurs : 17 3 2        Operateurs : - +
Valeurs : 17 5          Operateurs : -
Valeurs : 17 5          Operateurs : - /
Valeurs : 17 5 5        Operateurs : - /
Valeurs : 17 1          Operateurs : -
Valeurs : 16            Operateurs :
```