Algorithmique
Cours 7

IUT Informatique de Lens, 1ère Année
Université d'Artois

Frédéric Koriche
koriche@cril.fr
2011 - Semestre 1

Sommaire

L'objectif de ce cours est d'étudier les **chaînes** en algorithmique et programmation C++.

Sommaire

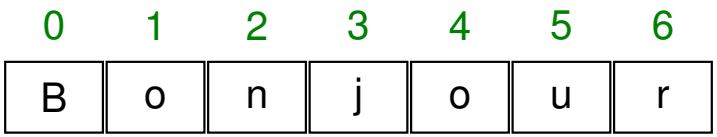
L'objectif de ce cours est d'étudier les **chaînes** en algorithmique et programmation C++.

Définition

Une chaîne est une séquence de caractères ascii.

Définition

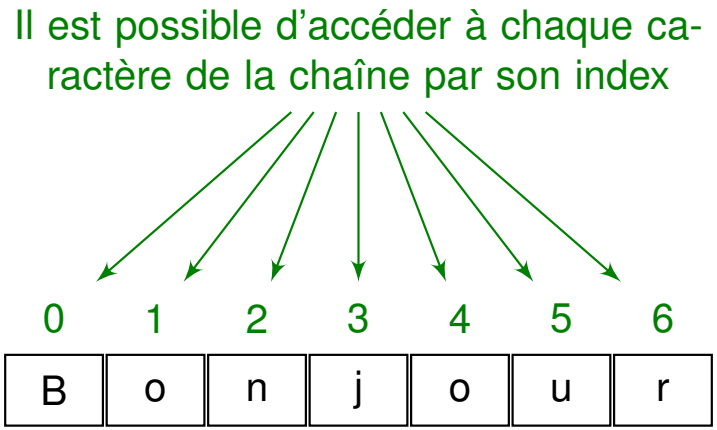
Une chaîne est une séquence de caractères ascii.



Une chaîne de sept caractères

Définition

Une chaîne est une séquence de caractères ascii.



Une chaîne de sept caractères

Déclaration en pseudo-code

Pour déclarer une chaîne en pseudo-code, nous utilisons le mot-clé **chaîne**.

Algorithme

```
variables
    chaîne nom

début
    afficher "Entrer votre nom"
    lire nom
fin
```

Déclaration en pseudo-code

Pour déclarer une chaîne en pseudo-code, nous utilisons le mot-clé **chaîne**.

Algorithme

```
variables
    chaîne nom

début
    afficher "Entrer votre nom"
    lire nom
fin
```

Déclaration d'une variable de type chaîne et appelée *nom*

Déclaration en pseudo-code

Pour déclarer une chaîne en pseudo-code, nous utilisons le mot-clé **chaîne**.

Algorithme

```
variables
    chaîne nom

début
    afficher "Entrer votre nom"
    lire nom
fin
```

Déclaration d'une variable de type chaîne et appelée *nom*

La chaîne de caractères saisie par l'utilisateur est affectée à la variable *nom*

Initialisation en pseudo-code

Pour initialiser une chaîne en pseudo-code, nous déclarons la variable chaîne et affectons sa valeur initiale.

Algorithme

```
variables
    chaîne nom ← "Burma"

début
    afficher "Entrer votre nom"
    lire nom
fin
```

Initialisation en pseudo-code

Pour initialiser une chaîne en pseudo-code, nous déclarons la variable chaîne et affectons sa valeur initiale.

Algorithme

```
variables
    chaîne nom ← "Burma"

début
    afficher "Entrer votre nom"
    lire nom
fin
```

Déclaration d'une variable de type chaîne et initialisée par la constante *Burma*

Initialisation en pseudo-code

Pour initialiser une chaîne en pseudo-code, nous déclarons la variable chaîne et affectons sa valeur initiale.

Algorithme

```
variables
    chaîne nom ← "Burma"

début
    afficher "Entrer votre nom"
    lire nom
fin
```

Déclaration d'une variable de type chaîne et initialisée par la constante *Burma*

La valeur de la chaîne *nom* est effacée, et remplacée par la valeur saisie par l'utilisateur

Initialisation en pseudo-code

Pour initialiser une chaîne en pseudo-code, nous déclarons la variable chaîne et affectons sa valeur initiale.

Algorithme

variables
  chaîne adresse ← "120, rue de la Gare"

début
  ...
fin

Une chaîne est une séquence arbitraire de caractères ascii, et peut donc être formée par plusieurs mots !

Accès aux éléments d'une chaîne en pseudo-code

Les caractères d'une chaîne peuvent être accédés par leur index en utilisant l'opérateur **[ ]**.

Algorithme

```
variables
    chaîne nom ← "Burma"
    caractère lettre

début
    lettre ← nom[1]

    lettre ← nom[10]
fin
```

Accès aux éléments d'une chaîne en pseudo-code

Les caractères d'une chaîne peuvent être accédés par leur index en utilisant l'opérateur **[ ]**.

Algorithme

```
variables
    chaîne nom ← "Burma"
    caractère lettre

début
    lettre ← nom[1]   ⟵ La variable lettre reçoit la valeur 'u'

    lettre ← nom[10]
fin
```

Accès aux éléments d'une chaîne en pseudo-code

Les caractères d'une chaîne peuvent être accédés par leur index en utilisant l'opérateur **[ ]**.

Algorithme

```
variables
    chaîne nom ← "Burma"
    caractère lettre

début
    lettre ← nom[1]
    lettre ← nom[10]
fin
```

La variable lettre reçoit la valeur 'u'

Erreur ! La chaîne nom ne possède que 6 caractères

Accès aux éléments d'une chaîne en pseudo-code

Les caractères d'une chaîne peuvent être accédés par leur index en utilisant l'opérateur **[ ]**.

Taille d'une chaîne

La taille d'une chaîne peut évoluer ! Pour accéder à la taille d'une chaîne, nous utilisons la fonction **longueur()**.

Algorithme

```
variables
    chaîne nom ← "Burma"
    caractère lettre
    entier i

début
    i ← 0
    lettre ← nom[i]

    i ← longueur(nom) - 1
    lettre ← nom[i]
fin
```

Accès aux éléments d'une chaîne en pseudo-code

Les caractères d'une chaîne peuvent être accédés par leur index en utilisant l'opérateur **[ ]**.

Taille d'une chaîne

La taille d'une chaîne peut évoluer ! Pour accéder à la taille d'une chaîne, nous utilisons la fonction **longueur()**.

Algorithme

```
variables
    chaîne nom ← "Burma"
    caractère lettre
    entier i

début
    i ← 0
    lettre ← nom[i]        ← La variable lettre reçoit la valeur 'B'
    i ← longueur(nom) - 1
    lettre ← nom[i]
fin
```

Accès aux éléments d'une chaîne en pseudo-code

Les caractères d'une chaîne peuvent être accédés par leur index en utilisant l'opérateur **[ ]**.

Taille d'une chaîne

La taille d'une chaîne peut évoluer ! Pour accéder à la taille d'une chaîne, nous utilisons la fonction **longueur()**.

Algorithme

```
variables
    chaîne nom ← "Burma"
    caractère lettre
    entier i

début
    i ← 0
    lettre ← nom[i]      ← La variable lettre reçoit la valeur 'B'
    i ← longueur(nom) - 1
    lettre ← nom[i]      ← La variable lettre reçoit la valeur 'a'
fin
```

Comparaison de chaînes en pseudo-code

Les chaînes peuvent être comparées en utilisant l'ordre *lexicographique*. Etant donné deux chaînes c_1 et c_2, nous avons $c_1 < c_2$ si et seulement si il existe un index i tel que :

- $c_1[j] = c_2[j]$ pour tout $j < i$, et
- soit longueur(c_1) < longueur(c_2), soit $c_1[i] < c_2[i]$

Algorithme

---

variables

chaîne $c_1 \leftarrow$ "café"
chaîne $c_2 \leftarrow$ "sucre"
chaîne $c_3 \leftarrow$ "cuiller"
chaîne $c_4 \leftarrow$ "Dans la tasse"
chaîne $c_5 \leftarrow$ "Dans la tasse de café"
booléen val

début

val $\leftarrow c_1 < c_2$

val $\leftarrow c_1 < c_3$

val $\leftarrow c_4 < c_5$

fin

Comparaison de chaînes en pseudo-code

Les chaînes peuvent être comparées en utilisant l'ordre *lexicographique*. Etant donné deux chaînes c_1 et c_2, nous avons $c_1 < c_2$ si et seulement si il existe un index i tel que :

- $c_1[j] = c_2[j]$ pour tout $j < i$, et
- soit longueur(c_1) < longueur(c_2), soit $c_1[i] < c_2[i]$

Algorithme

---

variables

chaîne $c_1 \leftarrow$ "café"
chaîne $c_2 \leftarrow$ "sucre"
chaîne $c_3 \leftarrow$ "cuiller"
chaîne $c_4 \leftarrow$ "Dans la tasse"
chaîne $c_5 \leftarrow$ "Dans la tasse de café"
booléen val

début

val $\leftarrow c_1 < c_2$ ⟵ val prend *vrai* car $c_1[0] < c_2[0]$

val $\leftarrow c_1 < c_3$

val $\leftarrow c_4 < c_5$

fin

Comparaison de chaînes en pseudo-code

Les chaînes peuvent être comparées en utilisant l'ordre *lexicographique*. Etant donné deux chaînes c_1 et c_2, nous avons $c_1 < c_2$ si et seulement si il existe un index i tel que :

- $c_1[j] = c_2[j]$ pour tout $j < i$, et
- soit longueur(c_1) < longueur(c_2), soit $c_1[i] < c_2[i]$

Algorithme

---

variables

chaîne $c_1 \leftarrow$ "café"
chaîne $c_2 \leftarrow$ "sucre"
chaîne $c_3 \leftarrow$ "cuiller"
chaîne $c_4 \leftarrow$ "Dans la tasse"
chaîne $c_5 \leftarrow$ "Dans la tasse de café"
booléen val

début

val $\leftarrow c_1 < c_2$ ⟵ val prend *vrai* car $c_1[0] < c_2[0]$

val $\leftarrow c_1 < c_3$ ⟵ val prend *vrai* car $c_1[1] < c_3[1]$

val $\leftarrow c_4 < c_5$

fin

Comparaison de chaînes en pseudo-code

Les chaînes peuvent être comparées en utilisant l'ordre *lexicographique*. Etant donné deux chaînes c_1 et c_2, nous avons $c_1 < c_2$ si et seulement si il existe un index i tel que :

- $c_1[j] = c_2[j]$ pour tout $j < i$, et
- soit longueur(c_1) < longueur(c_2), soit $c_1[i] < c_2[i]$

Algorithme

---

variables

chaîne $c_1 \leftarrow$ "café"
chaîne $c_2 \leftarrow$ "sucre"
chaîne $c_3 \leftarrow$ "cuiller"
chaîne $c_4 \leftarrow$ "Dans la tasse"
chaîne $c_5 \leftarrow$ "Dans la tasse de café"
booléen val

début

val $\leftarrow c_1 < c_2$ ⟵ val prend *vrai* car $c_1[0] < c_2[0]$

val $\leftarrow c_1 < c_3$ ⟵ val prend *vrai* car $c_1[1] < c_3[1]$

val $\leftarrow c_4 < c_5$ ⟵ val prend *vrai* car c_4 est le début de c_5

fin

Concaténation de chaînes en pseudo-code

Une chaîne peut être construite en lui ajoutant d'autres chaînes à la fin. L'opérateur de concaténation est noté **+**.

Algorithme

variables
- **chaîne** c_1 ← "Dans la "
- **chaîne** c_2 ← "tasse"
- **chaîne** c_3 ← "de café"
- **chaîne** phrase

début
- phrase ← c_1
- phrase ← phrase + c_2
- phrase ← phrase + ' ' + c_3

fin

Concaténation de chaînes en pseudo-code

Une chaîne peut être construite en lui ajoutant d'autres chaînes à la fin. L'opérateur de concaténation est noté **+**.

Algorithme

variables
- **chaîne** $c_1 \leftarrow$ "Dans la "
- **chaîne** $c_2 \leftarrow$ "tasse"
- **chaîne** $c_3 \leftarrow$ "de café"
- **chaîne** phrase

début
- phrase $\leftarrow c_1$ ⟵ la valeur de *phrase* est "Dans la "
- phrase $\leftarrow$ phrase + c_2
- phrase $\leftarrow$ phrase + ' ' + c_3

fin

Concaténation de chaînes en pseudo-code

Une chaîne peut être construite en lui ajoutant d'autres chaînes à la fin. L'opérateur de concaténation est noté **+**.

Algorithme

variables
- **chaîne** c_1 ← "Dans la "
- **chaîne** c_2 ← "tasse"
- **chaîne** c_3 ← "de café"
- **chaîne** phrase

début
- phrase ← c_1 ⟵ la valeur de *phrase* est "Dans la "
- phrase ← phrase + c_2 ⟵ la valeur de *phrase* est "Dans la tasse"
- phrase ← phrase + ' ' + c_3

fin

Concaténation de chaînes en pseudo-code

Une chaîne peut être construite en lui ajoutant d'autres chaînes à la fin. L'opérateur de concaténation est noté **+**.

Algorithme

variables
- **chaîne** $c_1 \leftarrow$ "Dans la "
- **chaîne** $c_2 \leftarrow$ "tasse"
- **chaîne** $c_3 \leftarrow$ "de café"
- **chaîne** phrase

début
- phrase $\leftarrow c_1$ ⟵ la valeur de *phrase* est "Dans la "
- phrase $\leftarrow$ phrase + c_2 ⟵ la valeur de *phrase* est "Dans la tasse"
- phrase $\leftarrow$ phrase + ' ' + c_3 ⟵ la valeur de *phrase* est "Dans la tasse de café"

fin

Résumé des opérations en pseudo-code

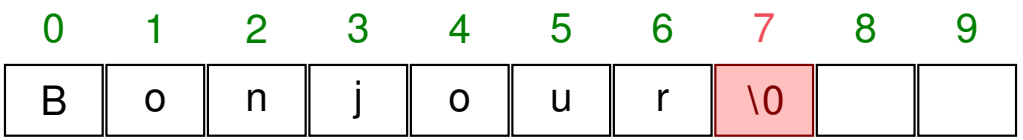
Opération	Type	Effet
$\leftarrow$	**chaîne**	Assigne une chaîne
$+$	**chaîne**	Concatène deux chaînes
$<, \leq, =, \neq, \geq, >$	**booléen**	Compare deux chaînes
vide()	**booléen**	Retourne vrai ssi la chaîne est vide
longueur()	**entier**	Retourne la longueur de la chaîne
efface()	aucun	Efface la chaîne

Les chaînes en C

En C, les chaînes sont impléméntées par des tableaux statiques de caractères se terminant par le caractère spécial \0 (backslash zero, appelé `null`). De tels tableaux sont appelés *C-chaînes*.

Les chaînes en C

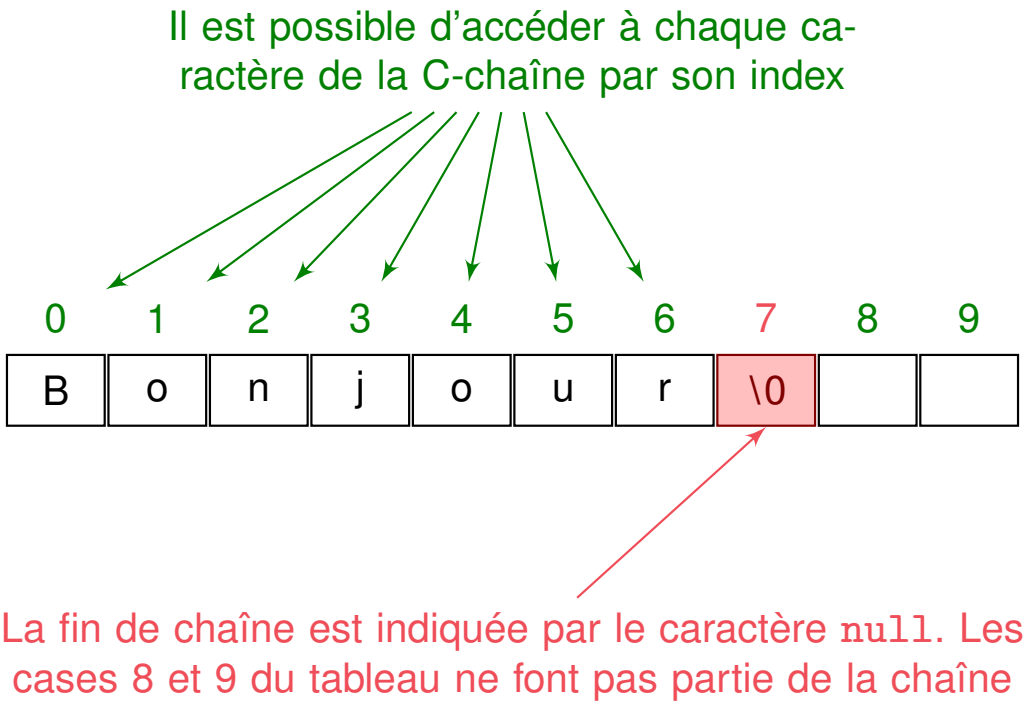
En C, les chaînes sont impléméntées par des tableaux statiques de caractères se terminant par le caractère spécial \0 (backslash zero, appelé `null`). De tels tableaux sont appelés *C-chaînes*.



Une C-chaîne de 10 caractères possibles contenant 7 caractères

Les chaînes en C

En C, les chaînes sont impléméntées par des tableaux statiques de caractères se terminant par le caractère spécial \0 (backslash zero, appelé `null`). De tels tableaux sont appelés *C-chaînes*.



Une C-chaîne de 10 caractères possibles contenant 7 caractères

Déclaration de chaînes en C

Les C-chaînes sont déclarées comme des tableaux statiques habituels.

```cpp
#include <iostream>
using namespace std;


int main()
{
char nom[20];
cin >> nom;
}
```

codeSource.cpp

Déclaration de chaînes en C

Les C-chaînes sont déclarées comme des tableaux statiques habituels.

```cpp
#include <iostream>
using namespace std;


int main()
{
char nom[20];
cin >> nom;
}
```

codeSource.cpp

Déclaration de chaînes en C

Les C-chaînes sont déclarées comme des tableaux statiques habituels.

```
#include <iostream>
using namespace std;


int main()
{
char nom[20];
cin >> nom;
}
```

codeSource.cpp

Déclaration d'une variable `nom` de type C-chaîne

Lecture d'une chaîne affectée à `nom`. Il faut que la chaîne lue fasse moins de 20 caractères !

Initialisation de chaînes en C

Les C-chaînes sont initialisées comme des tableaux statiques habituels en utilisant un ensemble de caractères, ou directement avec une chaîne constante.

```
#include <iostream>
using namespace std;


int main()
{
char nom[20] = {'B','u','r','m','a'};
char prenom[20] = "Nestor";
}
```

codeSource.cpp

Initialisation de chaînes en C

Les C-chaînes sont initialisées comme des tableaux statiques habituels en utilisant un ensemble de caractères, ou directement avec une chaîne constante.

```
#include <iostream>
using namespace std;


int main()
{
char nom[20] = {'B','u','r','m','a'};
char prenom[20] = "Nestor";
}
```

codeSource.cpp

Initialisation de chaînes en C

Les C-chaînes sont initialisées comme des tableaux statiques habituels en utilisant un ensemble de caractères, ou directement avec une chaîne constante.

```
#include <iostream>
using namespace std;


int main()
{
char nom[20] = {'B','u','r','m','a'};
char prenom[20] = "Nestor";
}
```

codeSource.cpp

Initialisation par un ensemble de caractères

Initialisation par une chaîne constante

Opérations sur les chaînes en C

Comme les C-chaînes sont des tableaux statiques, seule l'opération d'accès **[ ]** est possible.

```
#include <iostream>
using namespace std;


int main()
{
char nom[20] = "Burma";
char prenom[20] = "Nestor";
char metier[20];
metier = "Detective";
char signature[40];
signature = prenom + nom;
}
```

codeSource.cpp

Opérations sur les chaînes en C

Comme les C-chaînes sont des tableaux statiques, seule l'opération d'accès **[ ]** est possible.

```
#include <iostream>
using namespace std;


int main()
{
char nom[20] = "Burma";
char prenom[20] = "Nestor";
char metier[20];
metier = "Detective";
char signature[40];
signature = prenom + nom;
}
```

codeSource.cpp

Opérations sur les chaînes en C

Comme les C-chaînes sont des tableaux statiques, seule l'opération d'accès **[ ]** est possible.

```
#include <iostream>
using namespace std;


int main()
{
char nom[20] = "Burma";
char prenom[20] = "Nestor";
char metier[20];
metier = "Detective";
char signature[40];
signature = prenom + nom;
}
```

codeSource.cpp

Erreur ! L'assignation ne fonctionne pas sur les tableaux statiques en dehors de leur initialisation

Erreur ! La concaténation ne fonctionne pas sur les tableaux statiques

La chaîne C++

En C++, les chaînes sont définies à partir du type **string**. Il s'agit d'une classe de la **Standard Library** implémentant toutes les opérations du pseudo-code.

```
#include <iostream>
#include <string>
using namespace std;


int main()
{
string nom;
nom = "Burma";
}
```

codeSource.cpp

La chaîne C++

En C++, les chaînes sont définies à partir du type **string**. Il s'agit d'une classe de la **Standard Library** implémentant toutes les opérations du pseudo-code.

```
#include <iostream>
#include <string>
using namespace std;


int main()
{
string nom;
nom = "Burma";
}
```

codeSource.cpp

Utilisation de la classe `string`

Déclaration d'une variable `nom` de type `string`

Affectation de la valeur `"Burma"` à `nom`

Initialisation de chaînes en C++

En C++, les chaînes peuvent être initialisées en utilisant le constructeur ( ).

```
#include <iostream>
#include <string>
using namespace std;


int main()
{
string nom("Burma");
string prenom("Nestor");


cout ≪ prenom + " " + nom;
}
```

codeSource.cpp

Initialisation de chaînes en C++

En C++, les chaînes peuvent être initialisées en utilisant le constructeur ( ).

```cpp
#include <iostream>
#include <string>
using namespace std;


int main()
{
string nom("Burma");
string prenom("Nestor");


cout << prenom + " " + nom;
}
```

codeSource.cpp

Initialisation des variables `nom` et `prenom`

Affichage de la chaîne `Nestor Burma`

Taille des chaînes en C++

En C++, les chaînes sont implémentées comme des tableaux dynamiques. Leur taille est donnée par la fonction **size()**.

Taille des chaînes en C++

En C++, les chaînes sont implémentées comme des tableaux dynamiques. Leur taille est donnée par la fonction **size()**.

```cpp
#include <iostream>
#include <string>
using namespace std;


int main()
{
string nom("Burma");
int i = 0;


cout << nom[i];
i = nom.size() - 1;
cout << nom[i];
}
```

codeSource.cpp

Taille des chaînes en C++

En C++, les chaînes sont implémentées comme des tableaux dynamiques. Leur taille est donnée par la fonction **size()**.

```
#include <iostream>
#include <string>
using namespace std;


int main()
{
string nom("Burma");
int i = 0;


cout << nom[i];
i = nom.size() - 1;
cout << nom[i];
}
```

codeSource.cpp

Affichage du caractère 'B'

Dernier caractère de la chaîne

Affichage du caractère 'a'

Résumé des opérations en C++

Opération	Type	Effet
=	`string`	Assigne une chaîne
+	`string`	Concatène deux chaînes
<,<=,=,!=,>=,>	`bool`	Compare deux chaînes
empty()	`bool`	Retourne 1 ssi la chaîne est vide
size()	`unsigned int`	Retourne la longueur de la chaîne
clear()	`void`	Efface la chaîne

Note

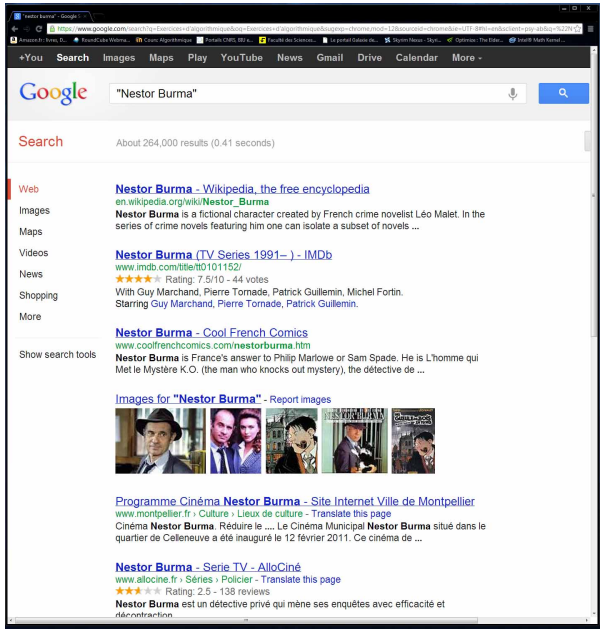
Les fonctions `empty()`, `size()` et `clear()` sont des méthodes de la classe `string`. Elles sont appelées en utilisant la variable suivie d'un point et suivie de la fonction. Par exemple l'instruction

```
nom.clear();
```

efface la valeur de la variable nom.

Le problème (String Matching)

- Données : une chaîne x (une phrase, un texte, une page web, ...) et une chaîne y (un mot, une expression, ...).
- Résultat : vrai si y est une sous-chaîne de x et faux sinon.



La recherche de sous-chaînes est employée dans un grand nombre d'applications, notamment les moteurs de recherche.

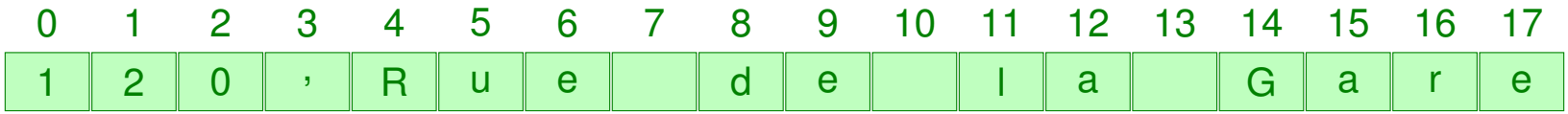
Le problème (String Matching)

- Données : une chaîne x (une phrase, un texte, une page web, ...) et une chaîne y (un mot, une expression, ...).
- Résultat : vrai si y est une sous-chaîne de x et faux sinon.

Le problème (String Matching)

- Données : une chaîne x (une phrase, un texte, une page web, ...) et une chaîne y (un mot, une expression, ...).
- Résultat : vrai si y est une sous-chaîne de x et faux sinon.

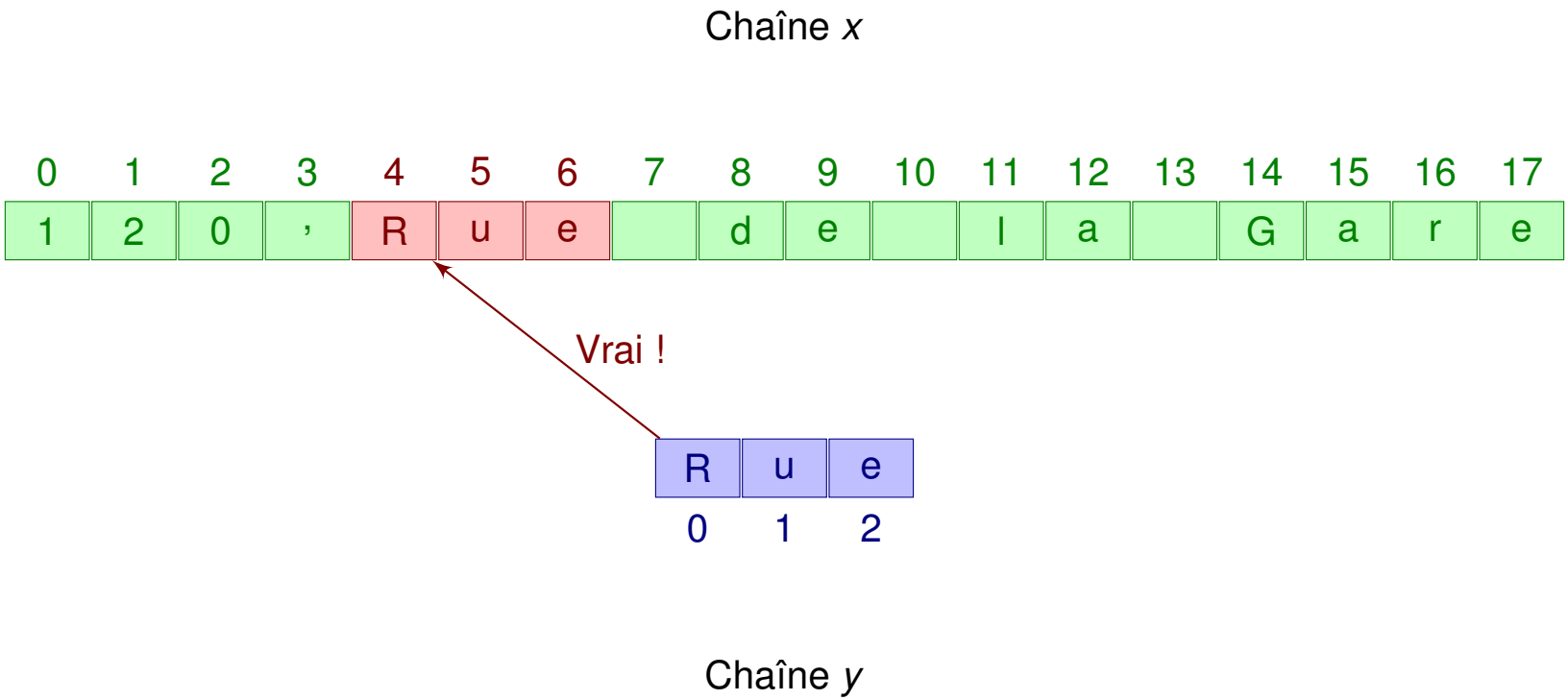
Chaîne x



Chaîne y

Le problème (String Matching)

- Données : une chaîne x (une phrase, un texte, une page web, ...) et une chaîne y (un mot, une expression, ...).
- Résultat : vrai si y est une sous-chaîne de x et faux sinon.



Le problème (String Matching)

- Données : une chaîne x (une phrase, un texte, une page web, ...) et une chaîne y (un mot, une expression, ...).
- Résultat : vrai si y est une sous-chaîne de x et faux sinon.

Chaîne x

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	2	0	,	R	u	e		d	e		l	a		G	a	r	e

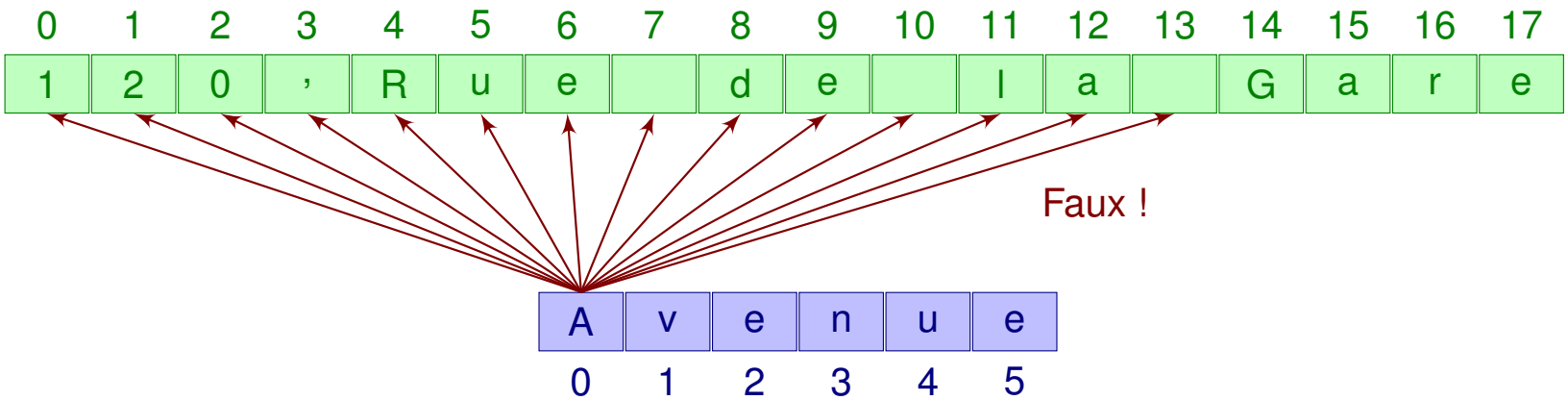
A	v	e	n	u	e
0	1	2	3	4	5

Chaîne y

Le problème (String Matching)

- Données : une chaîne x (une phrase, un texte, une page web, ...) et une chaîne y (un mot, une expression, ...).
- Résultat : vrai si y est une sous-chaîne de x et faux sinon.

Chaîne x



Chaîne y

Le problème (String Matching)

- Données : une chaîne x (une phrase, un texte, une page web, ...) et une chaîne y (un mot, une expression, ...).
- Résultat : vrai si y est une sous-chaîne de x et faux sinon.

Chaîne x

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	2	0	,	R	u	e		d	e		l	a		G	a	r	e

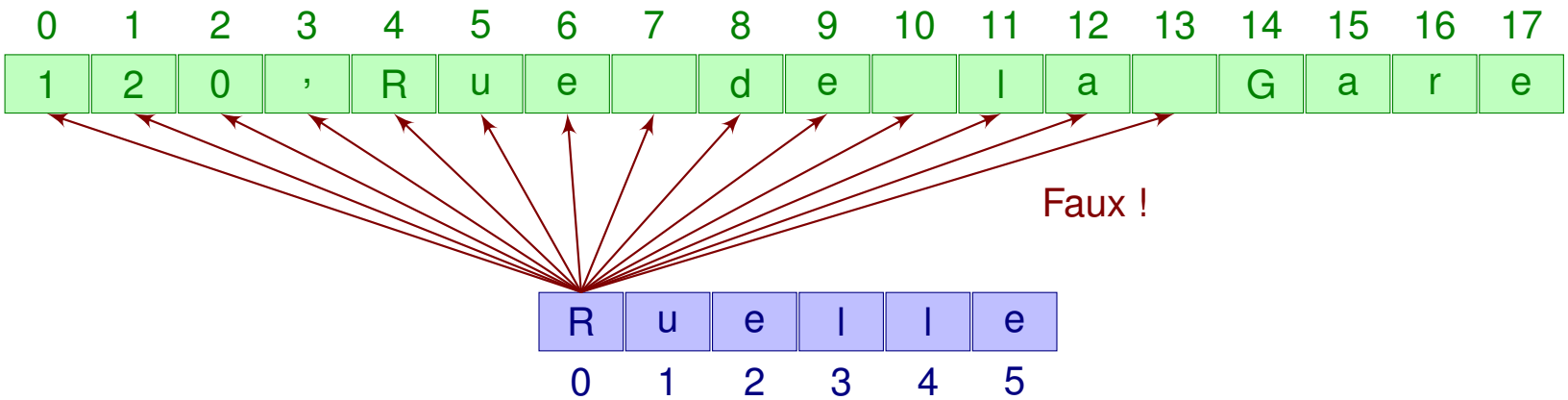
R	u	e	l	l	e
0	1	2	3	4	5

Chaîne y

Le problème (String Matching)

- Données : une chaîne x (une phrase, un texte, une page web, ...) et une chaîne y (un mot, une expression, ...).
- Résultat : vrai si y est une sous-chaîne de x et faux sinon.

Chaîne x



Chaîne y

Un algorithme pour la recherche de chaînes

Chaîne *x*

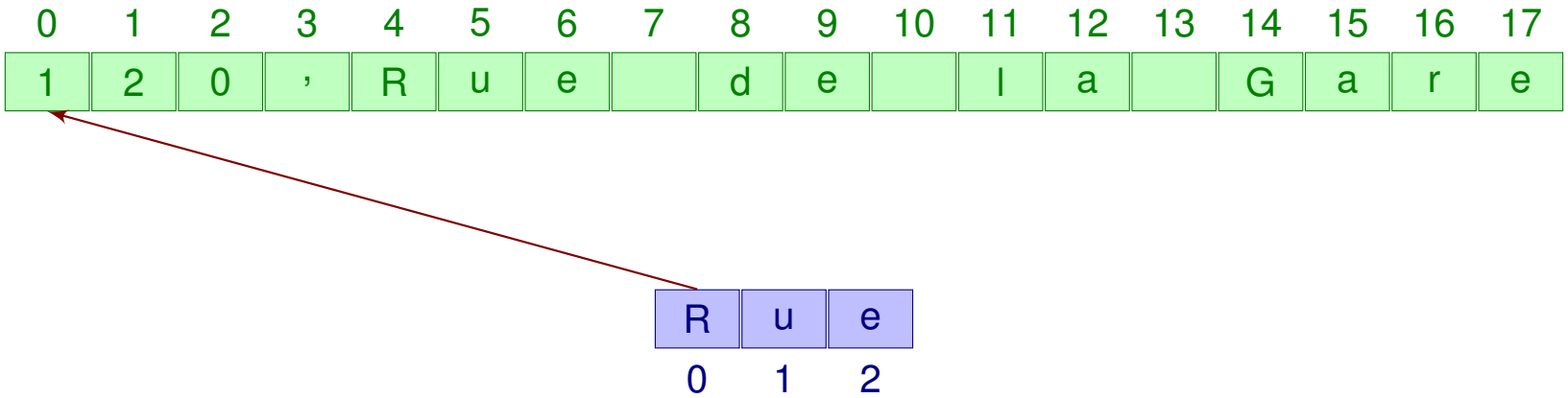
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	2	0	,	R	u	e		d	e		l	a		G	a	r	e

R	u	e
0	1	2

Chaîne *y*

Un algorithme pour la recherche de chaînes

Chaîne *x*

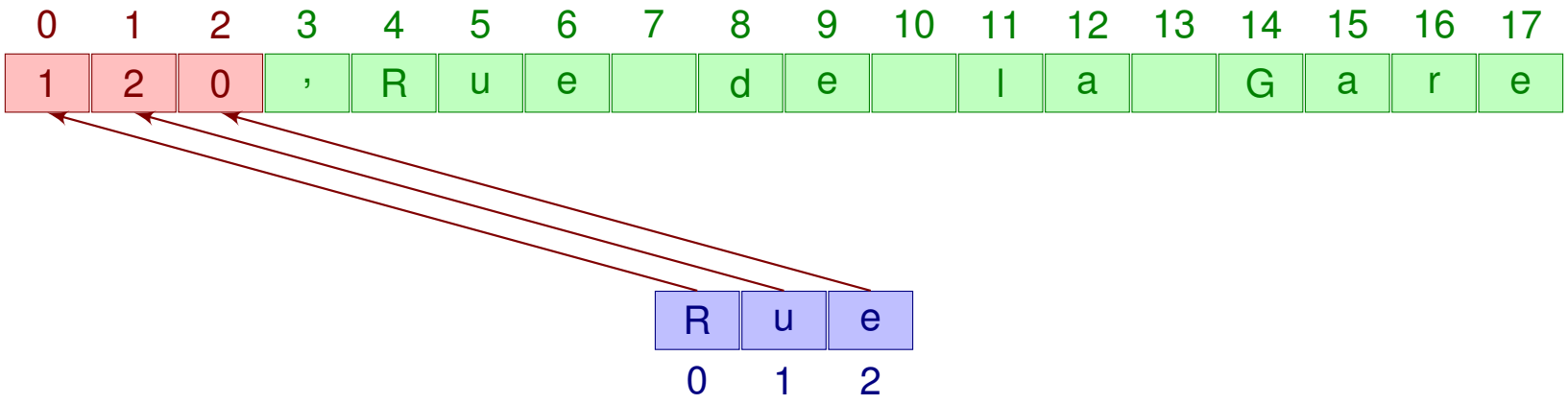


Chaîne *y*

On commence à l'index 0 de la chaîne *x* . . .

Un algorithme pour la recherche de chaînes

Chaîne *x*

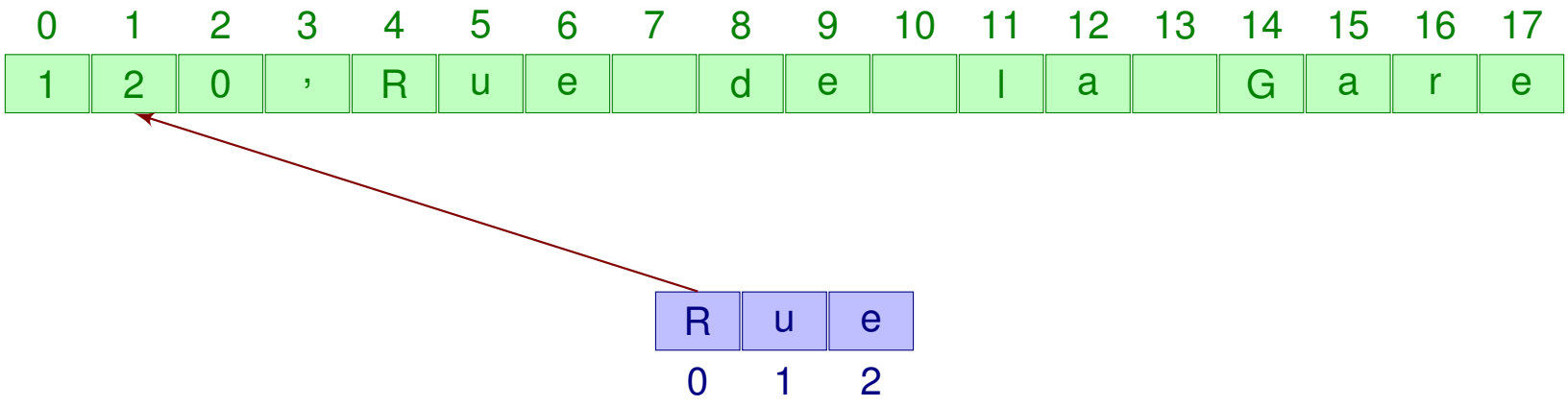


Chaîne *y*

. . . et on regarde si *x* et *y* coïncident à partir de 0

Un algorithme pour la recherche de chaînes

Chaîne *x*

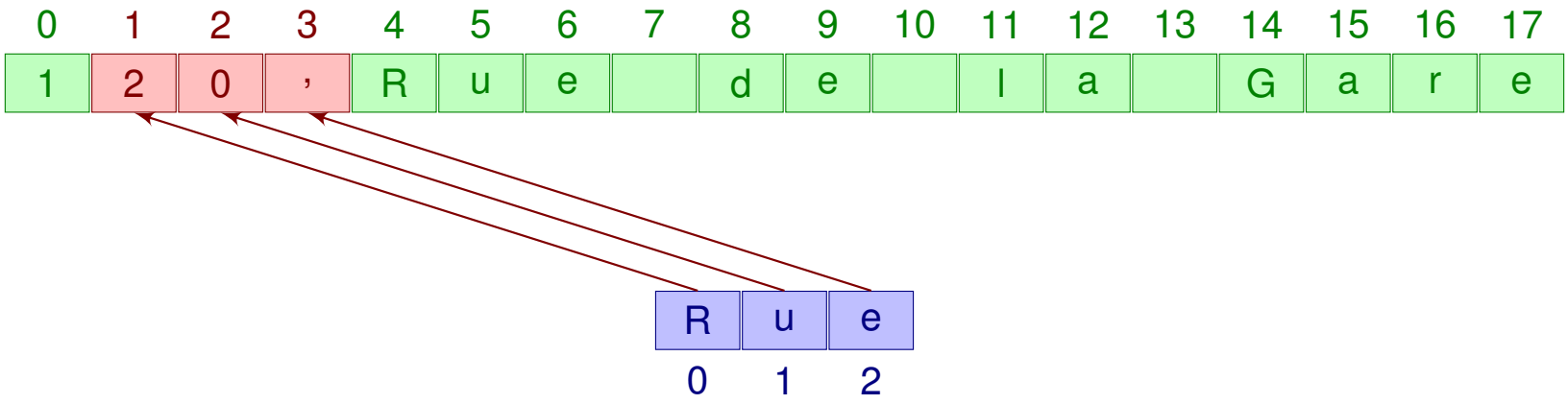


Chaîne *y*

Si ça ne marche pas, on recommence à l'index 1 de *x* . . .

Un algorithme pour la recherche de chaînes

Chaîne *x*

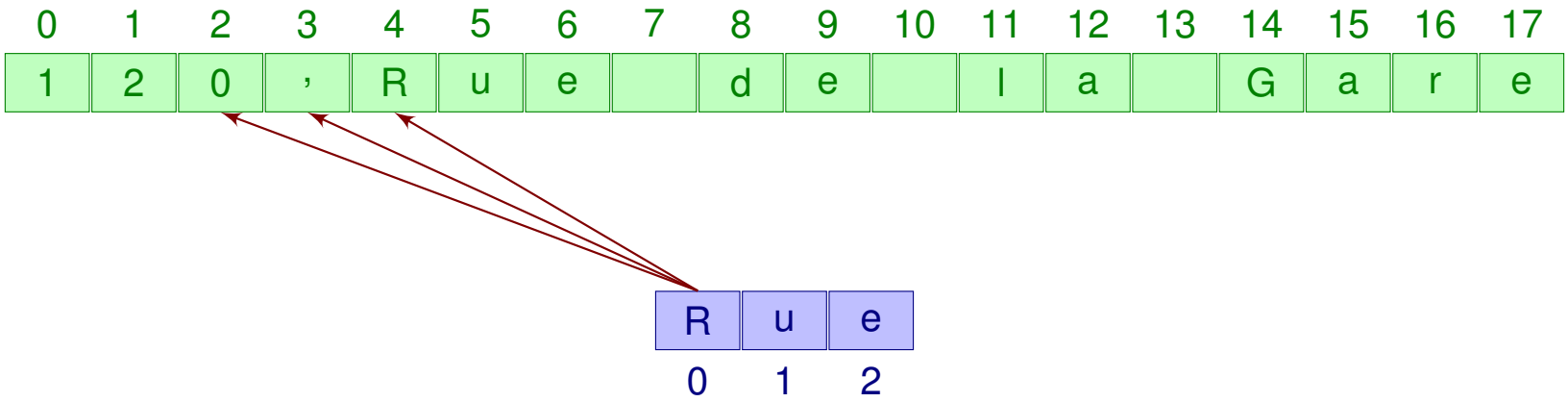


Chaîne *y*

. . . et on regarde si *x* et *y* coïncident à partir de 1

Un algorithme pour la recherche de chaînes

Chaîne *x*

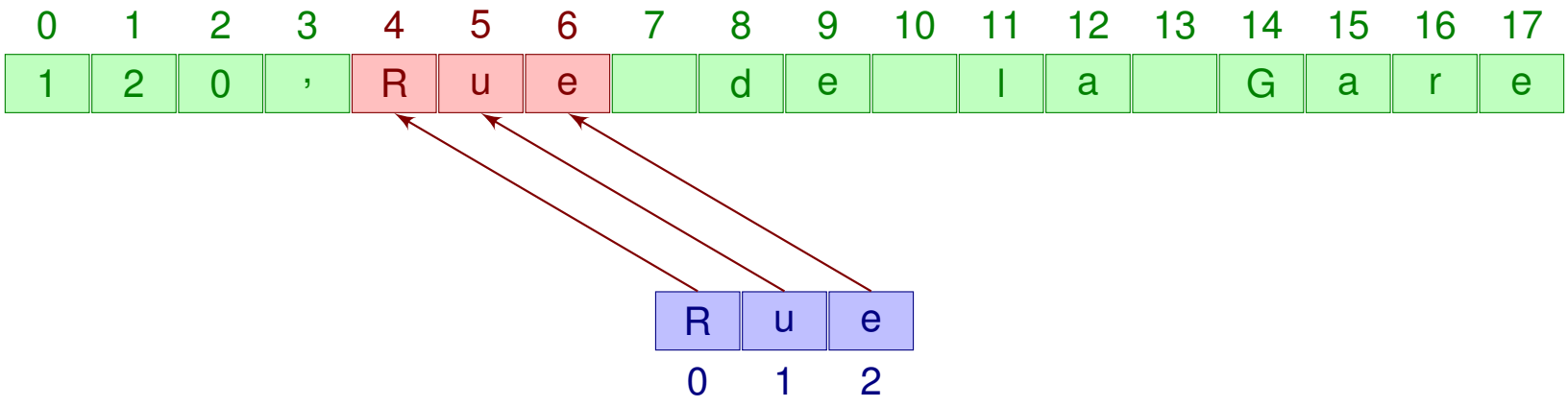


Chaîne *y*

. . . Et ainsi de suite . . .

Un algorithme pour la recherche de chaînes

Chaîne *x*



Chaîne *y*

. . . jusqu'à trouver *y* dans *x*, ou atteindre la fin de *x*.

Algorithme

```
variables
    chaîne x,y
    booléen trouvé
    entier i,j

début
    trouvé ← faux          ← Au début, on n'a rien trouvé
    i ← 0
    tant que (non trouvé) et (i ≤ longueur(x) - longueur(y))
    faire
        Tester si x et y coïncident à partir de i
    fin
fin
```

Algorithme

```
variables
    chaîne x,y
    booléen trouvé
    entier i,j

début
    trouvé ← faux
    i ← 0                                   ← Commencer à l'index 0 de x
    tant que (non trouvé) et (i ≤ longueur(x) - longueur(y))
    faire
        Tester si x et y coïncident à partir de i
    fin
fin
```

Algorithme

```
variables
    chaîne x,y
    booléen trouvé
    entier i,j

début
    trouvé ← faux
    i ← 0
    tant que (non trouvé) et (i ≤ longueur(x) - longueur(y))
    faire
        Tester si x et y coïncident à partir de i
    fin
fin
```

On incrémente i tant qu'on n'a rien trouvé et qu'on n'a pas atteint le dernier index possible

Algorithme

```
variables
    chaîne x,y
    booléen trouvé
    entier i,j, temp

début
    trouvé ← faux
    i ← 0
    tant que (non trouvé) et (i ≤ longueur(x) - longueur(y))
    faire
        trouvé ← vrai
        temp ← i
        j ← 0
        tant que (trouvé) et (j < longueur(y)) faire
            trouvé ← trouvé et x[i] = y[j]
            i ← i + 1
            j ← j + 1
        fin
        i ← temp + 1
    fin
    afficher trouvé
fin
```

Il faut que *trouvé* reste vrai pendant tout le test de *y*

Algorithme

```
variables
    chaîne x,y
    booléen trouvé
    entier i,j, temp

début
    trouvé ← faux
    i ← 0
    tant que (non trouvé) et (i ≤ longueur(x) - longueur(y))
    faire
        trouvé ← vrai
        temp ← i                    ← On mémorise l'index courant de i
        j ← 0
        tant que (trouvé) et (j < longueur(y)) faire
            trouvé ← trouvé et x[i] = y[j]
            i ← i + 1
            j ← j + 1
        fin
        i ← temp + 1
    fin
    afficher trouvé
fin
```

Algorithme

```
variables
    chaîne x,y
    booléen trouvé
    entier i,j, temp

début
    trouvé ← faux
    i ← 0
    tant que (non trouvé) et (i ≤ longueur(x) - longueur(y))
    faire
        trouvé ← vrai
        temp ← i
        j ← 0
        tant que (trouvé) et (j < longueur(y)) faire
            trouvé ← trouvé et x[i] = y[j]
            i ← i + 1
            j ← j + 1
        fin
        i ← temp + 1
    fin
    afficher trouvé
fin
```

On teste si x et y coïncident en incrémentant i et j

Algorithme

```
variables
    chaîne x,y
    booléen trouvé
    entier i,j, temp

début
    trouvé ← faux
    i ← 0
    tant que (non trouvé) et (i ≤ longueur(x) - longueur(y))
    faire
        trouvé ← vrai
        temp ← i
        j ← 0
        tant que (trouvé) et (j < longueur(y)) faire
            trouvé ← trouvé et x[i] = y[j]
            i ← i + 1
            j ← j + 1
        fin
        i ← temp + 1
    fin
    afficher trouvé
fin
```

On revient positionner *i* à *temp* et on l'incrémente de 1 pour le prochain test

Recherche de chaînes

Notre algorithme est loin d'être optimal (le plus rapide) pour la recherche de chaînes. Parmi les algorithmes les plus connus que vous étudierez :

- l'algorithme de Knuth-Morris-Pratt (KMP),
- l'algorithme de Boyer-Morre (BM).

Recherche de chaînes

Notre algorithme est loin d'être optimal (le plus rapide) pour la recherche de chaînes. Parmi les algorithmes les plus connus que vous étudierez :

- l'algorithme de Knuth-Morris-Pratt (KMP),
- l'algorithme de Boyer-Morre (BM).

Extensions

La recherche de chaîne est un problème de base qui ouvre la voie à de nombreux autres problèmes :

- Trouver le nombre d'occurrences d'un mot dans un texte,
- Déterminer si un texte contient un mot qui ressemble à l'expression recherchée (ex : "avenue" au lieu de "avennue", une expression mal orthographiée par l'utilisateur).