Théorie des langages de programmation – Le projet

Alain Chillès – 祁冲

ParisTech Shanghai Jiao Tong
上海交大–巴黎高科卓越工程师学院

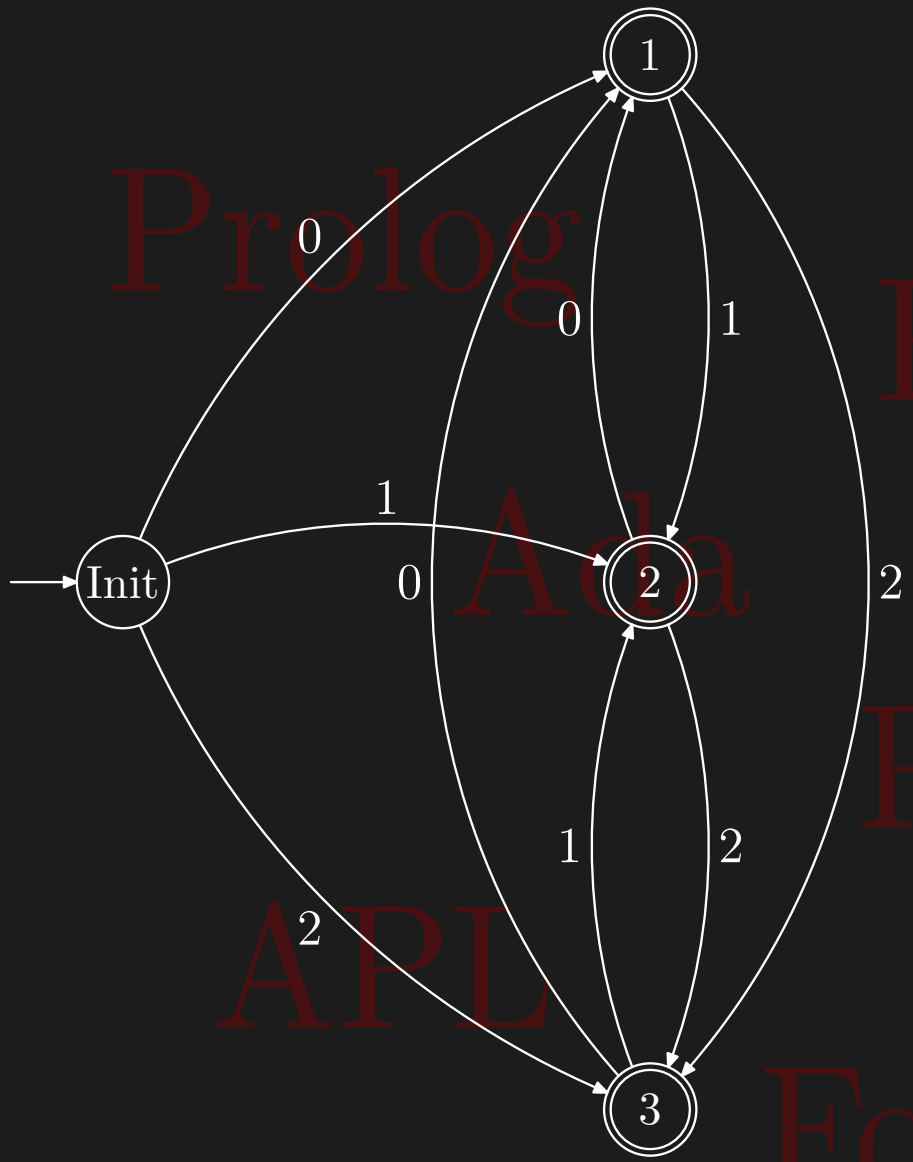
1er novembre 2019 – 2019年11月1日 – 己亥年十月初五

Situation générale

On veut écrire un compilateur d'automate qui

- Lit un fichier contenant la description de l'automate
- Produit en machine virtuelle un code capable de reconnaître ou pas tout mot qui lui sera fourni
- Permet d'exécuter les machines virtuelles écrites

Exemple de situation – AFD



Exemple de situation – AFD

Fichier Zpile.txt

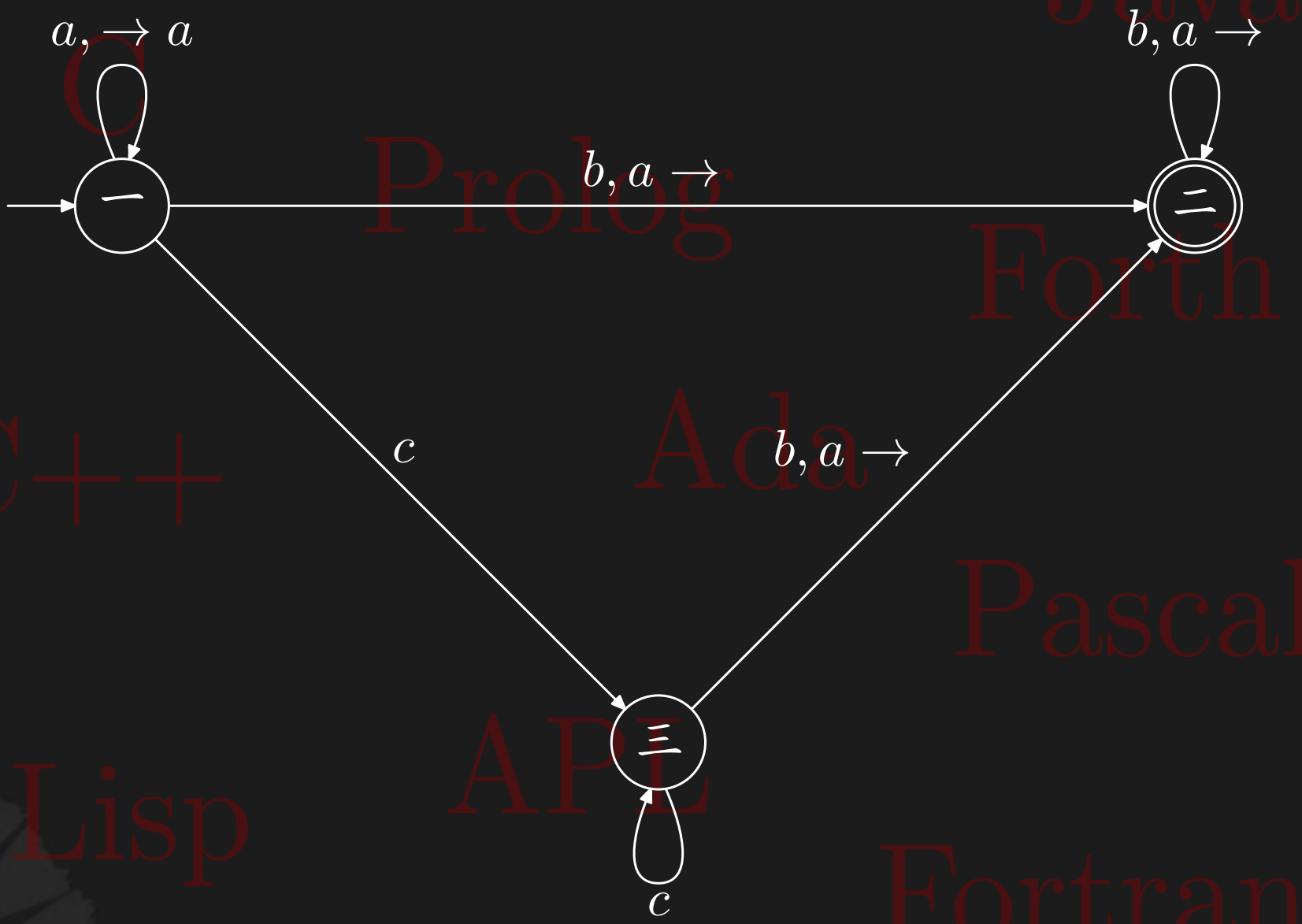
```
1  Automate(0)={
2  // Cet automate n'a pas de pile
3  // Il correspond à l'automate du cours 5, page
   ↪  5
4      etats=["1","2","3","Init"] // Le 0 est en
       ↪  dernier, c'est plus clair
5  // Chaque état est repéré par son numéro dans
   ↪  la liste etats
6  // Cette liste commence à l'indice 0
7      initial=3 // L'état Init
```

Exemple de situation – AFD

Fichier Zpile.txt

```
9   // final est une liste, même s'il n'y a qu'un état final
10      final=[0,1,2]
11      transitions=[(3 → 0,`0`),(3 → 1,`1`),(3 → 2,`2`),
12          (0 → 1,`1`),(0 → 2,`2`),(1 → 0,`0`),(1 → 2,`2`),
13          (2 → 0,`0`),(2 → 1,`1`)]
14  }
15
```

Exemple de situation – AFD avec une pile



Exemple de situation – AFD avec une pile

Fichier Upile.txt

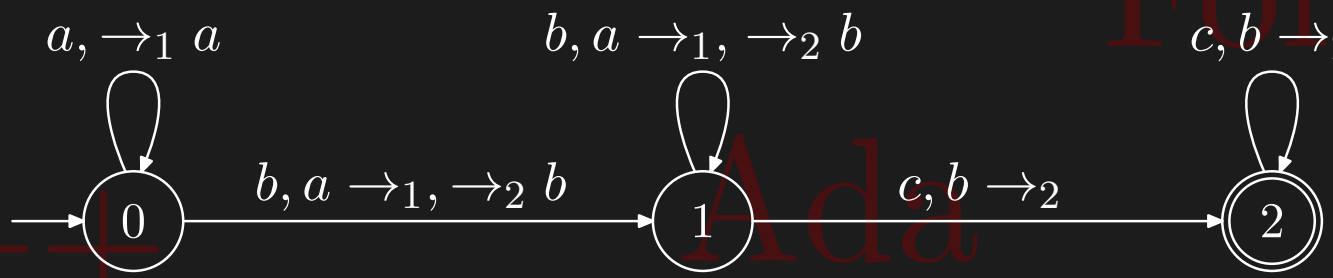
```
1  /* Cet automate a une pile, il permet de
   ↪  programmer toutes les grammaires BNF et
2  de simuler la reconnaissance des langages
   ↪  algébriques*/
3  Automate(1) ={
4  /* Cet automate a une pile
5  Il correspond à l'automate reconnaissant le
   ↪  langage
6  a^n.c^p.b^n, où n>0 */
7      etats =["一","二","三"] // Les noms
       ↪  peuvent être différents de numéros
8  // Chaque état est repéré par son numéro dans
   ↪  la liste etats
9      initial= 0
```

Exemple de situation – AFD avec une pile

Fichier Upile.txt

```
10  // final est une liste, même s'il n'y a qu'un
    ↪  état final
11      final =[1]
12      transitions=[(0 → 0, `a`,(→, `a`)),
13          (0 → 1,`b`, (`a`,→)),
14          (2 → 1,`b`, (`a`,→)),
15          (0 → 2,`c`, ()), // Pas d'action de
            ↪  pile
16          (2 → 2, `c`), // Pas d'action de pile
17          (1 → 1,`b`,(`a`,→))]
18  }
```

Exemple de situation – AFD avec deux piles



Exemple de situation – AFD avec deux piles

Fichier Dpile.txt

```
1  /* Cet automate a deux piles */
2
3  Automate(2) = {
4  /* Cet automate a deux piles
5  // Il correspond à l'automate reconnaissant le
   ↪  langage
6  // a^n.b^n.c^n où n>0 (cours 10, page 18) */
7       etats = [`0`, `1`, `2`]
8  // On pourrait mettre aussi [`a`, `b`,`c`]
9  // Chaque état est repéré par son numéro dans
   ↪  la liste etats
10      initial = 0
```

Exemple de situation – AFD avec deux piles

Fichier Dpile.txt

```
12  // final est une liste, même s'il n'y a qu'un
    ↪  état final
13      final= [2]
14      transitions=[(0 → 0, `a`, (→,`a`)), //
        ↪  Pas d'action sur la pile 2
15      // On pourrait aussi écrire (0 → 0, `a`,
        ↪  (→,`a`),())
16          (0 → 1, `b`, (`a`, →), (→, `b`)),
17          (1 → 1, `b`, (`a`, →), (→,` b`)),
18          (1 → 2, `c`, (), (`b`, →)), // Pas
            ↪  d'action sur la pile 1
19          (2 → 2, `c`, (), (`b`, →))]
20  }
```

Parties du projet

- Un rapport (écrit en `.pdf`) décrivant les résultats demandés, les choix faits, les problèmes rencontrés et les solutions apportées
- Des fichiers de code (`.c` et `.h`)

Le tout sera livré sur `Moodle` dans une unique archive `.zip`

Le projet – Analyse lexicale

On veut pouvoir lire les fichiers donnés (`Zpile.txt`, `Upile.txt` et `Dpile.txt`).

- Dans le rapport : description des lexèmes, des expressions régulières associées à chaque lexème
- Les fichiers de code : un fichier `analyseur_lexical.c` et un fichier pour tester

Le projet – Analyse syntaxique

À partir de la liste ou du tableau de lexèmes :

- Dans le rapport : Définir la notion d'arbre syntaxique, produire une grammaire BNF des automates
- Les fichiers de code : Produire un parser transformant une liste ou un tableau de lexèmes en arbre syntaxique et une fonction `print_arbre` permettant d'imprimer un arbre syntaxique avec les parenthèses bien placées (fichier `analyseur_syntaxique.c` et un fichier pour tester)

Le projet – analyse sémantique

- Dans le rapport : décrire les choix sémantiques faits et les vérifications effectuées lors de cette phase
- Les fichiers de code : fichier `analyseur_semantique.c` et un fichier pour tester

Le projet – compilation et exécution

▶ La table des symboles contiendra le nom des états et son adresse dans la machine virtuelle

▶ Une machine virtuelle sera un tableau d'entiers. On pourra remplir la machine de la manière suivante :

- Automate sans pile : chaque état sera représenté par :

$$n0, \underbrace{c_1, n_1, \ldots\ldots, c_{n0}, n_{n0}}_{n0 \text{ couples de la forme } (c,k)}$$

où $n0$ est le nombre de transitions partant de l'état, c le caractère déclenchant la transition et k l'adresse de l'état activé par la transition.

Le projet – compilation et exécution

Supposons que l'état 12 s'appelle "Ici" et qu'il soit codé dans la VM à partir de l'indice 30. (L'état 15 est à l'adresse x, l'état 18 à l'adresse y et l'état 5 à l'adresse z)

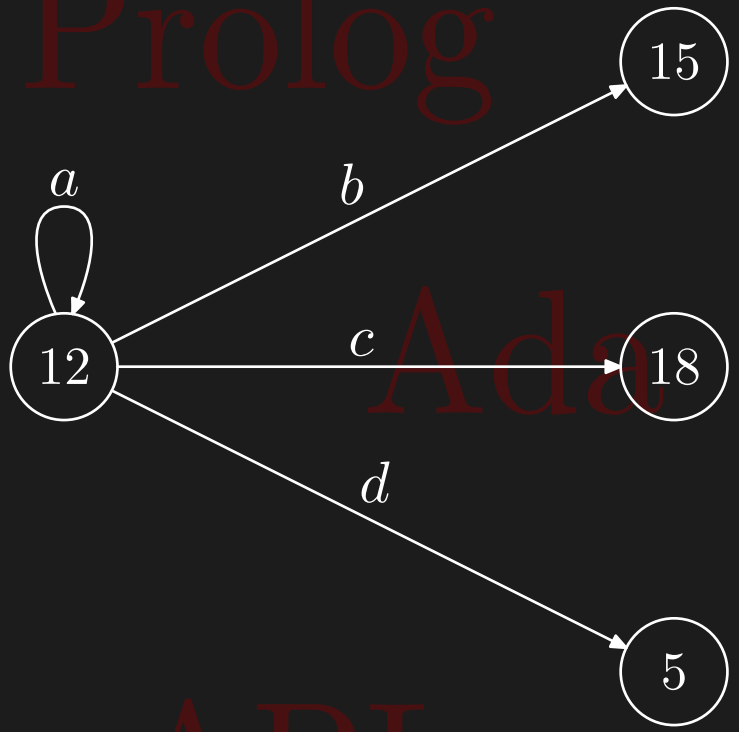


Table des symboles : Nom = "Ici", adresse = 30

VM :

29	30	31	32	33	34	35	36	37	38	39
···	4	97	30	98	x	99	y	100	z	···

Le projet – compilation et exécution

▶ La table des symboles contiendra le nom des états et son adresse dans la machine virtuelle

▶ Une machine virtuelle sera un tableau d'entiers. On pourra remplir la machine de la manière suivante :

- Automate à une pile : chaque état sera représenté par :

$$n0, \underbrace{c_1, n_1, p_1, a_1 \ldots \ldots, c_{n0}, n_{n0}, p_{n0}, a_{n0}}_{n0 \text{ quadruplet de la forme } (c,k,p,a)}$$

où $n0$ est le nombre de transitions partant de l'état, c le caractère déclenchant la transition, k l'adresse de l'état activé par la transition, p le caractère à mettre ou à enlever de la pile (0 si on ne fait rien), $a \in \{-1, 0, 1\}$ l'action à faire sur la pile (-1 : on enlève, 0 : on ne fait rien, 1 : on ajoute le caractère sur la pile).

Le projet – compilation et exécution

▶ La table des symboles contiendra le nom des états et son adresse dans la machine virtuelle

▶ Une machine virtuelle sera un tableau d'entiers. On pourra remplir la machine de la manière suivante :

- Automate à deux piles : chaque état sera représenté par :

$$n0, \underbrace{c_1, n_1, p_1, a_1, q_1, b_1 \ldots\ldots, c_{n0}, n_{n0}, p_{n0}, a_{n0}, q_{n0}, b_{n0}}_{n0 \text{ 6-uplet de la forme } (c,k,p,a,q,b)}$$

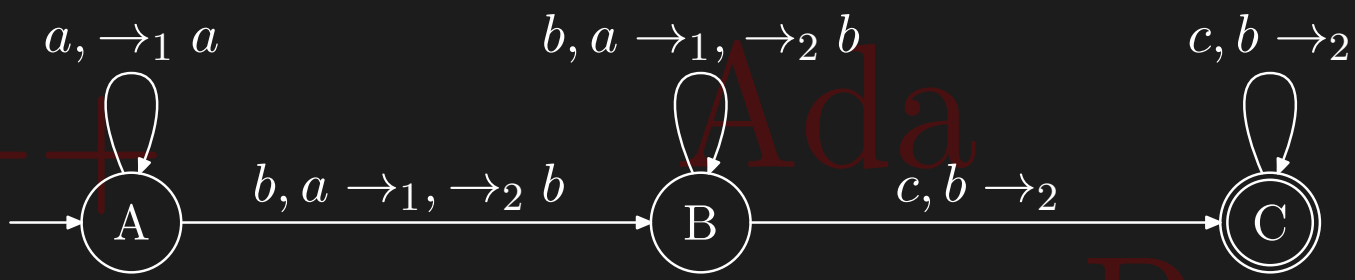
où $n0$ est le nombre de transitions partant de l'état, c le caractère déclenchant la transition, k l'adresse de l'état activé par la transition, p le caractère à mettre ou à enlever de la première pile (0 si on ne fait rien), $a \in \{-1, 0, 1\}$ l'action à faire sur la première pile (-1 : on enlève, 0 : on ne fait rien, 1 : on ajoute le caractère sur la pile), q le caractère à mettre ou à enlever de la deuxième pile et b l'action à faire sur la deuxième pile.

Le projet – compilation et exécution

- Le compilateur prendra un fichier `.txt` et écrira dans un fichier `VM` la machine virtuelle et dans un autre fichier, la table des symboles. (Fichier `compile_automate.c`).
- L'exécuteur prendra le fichier `VM` et sera capable de reconnaître ou pas un mot saisi au clavier. (Fichier `Executeur.c`).
- Le mode debug permettra de suivre l'évolution de l'automate (avec les noms qui sont dans la table des symboles). Ce sera une option de l'exécuteur...

Le projet – compilation et exécution

Qu'est-ce que le mode debug ? Soit l'automate suivant (à deux piles) et soit le mot d'entrée abc lecture de gauche à droite).



Le projet – compilation et exécution

Terminal

```
$ Executeur -debug VM
Donner le mot d'entrée : abc
  -> État : A    Pile 1 : Vide    Pile 2 : Vide
a -> État : A    Pile 1 : a       Pile 2 : Vide
b -> État : B    Pile 1 : Vide    Pile 2 : b
c -> État : C    Pile 1 : Vide    Pile 2 : Vide
Le mot abc est accepté !
```

Le projet – compilation et exécution

Terminal

```
$ Executeur -debug VM
Donner le mot d'entrée : abbc
  -> État : A    Pile 1 : Vide    Pile 2 : Vide
a -> État : A    Pile 1 : a       Pile 2 : Vide
b -> État : B    Pile 1 : Vide    Pile 2 : b
b -> Erreur : Pile 1 vide !
Le mot abbc est refusé !
```

Le projet – compilation et exécution

Terminal

```
$ Executeur -debug VM
Donner le mot d'entrée : aaabbc
  -> État : A    Pile 1 : Vide    Pile 2 : Vide
a -> État : A    Pile 1 : a       Pile 2 : Vide
a -> État : A    Pile 1 : aa      Pile 2 : Vide
a -> État : A    Pile 1 : aaa     Pile 2 : Vide
b -> État : B    Pile 1 : aa      Pile 2 : b
b -> État : B    Pile 1 : a       Pile 2 : bb
c -> État : C    Pile 1 : a       Pile 2 : b
Le mot aaabbc est refusé ! Pile 1 et Pile 2 non
 ↪  vides
```

Le projet – les contraintes

- Des fichiers `.c` bien commentés
  - Les fichiers doivent pouvoir être compilés sans erreur ni warning par le compilateur `linux` (`gcc`). Attention à bien vérifier que c'est le cas, certains étudiants ont, l'année dernière, livré des codes qui ne se compilaient pas sous `Linux` et ont perdu beaucoup de points !
- Un rapport qui explique les choix et présente les résultats d'exécution sur certains mots. Format du fichier = pdf, tout autre format générera des points en moins.

Le tout sera livré sur `Moodle` sous la forme d'une archive `.zip`

Le projet – avertissements

- Toute tricherie sera sévèrement punie
- La qualité du travail sera appréciée fortement, même si, à la fin, cela ne marche pas tout le temps (mettre en ce cas les problèmes rencontrés et les solutions essayées dans le rapport)
- Un projet raisonnablement travaillé et respectant les consignes aura une bonne note

Je rappelle que

$$\text{projet} = 80\% \text{ et attitude} = 20\%$$

Le projet – rendez-vous

Des séances de questions/réponses et d'aide à la résolution des problèmes auront lieu à 18h les jeudis

- 21 novembre (analyse lexicale)
- 28 novembre (analyse syntaxique)
- 5 décembre (analyse sémantique)
- 12 décembre (compilation et exécution)

Le projet est à livrer avant le 3 janvier 2020

Je peux aussi être contacté par mail à l'adresse `alain.chilles@gmail.com` ou par `Wechat` (Alain-QiChong)