Exercices
machine caractère

Ces exercices font suite à la première présentation du langage de programmation C++ est écrite dans le cadre du cours C++ organisé au club informatique de Loriol-sur-Drôme (http://ciloriol.fr).

Ils sont écrits et présentés au sein du club par Jean-Claude CATY (jc<année>.caty@laposte.net).

Table des matières

1. Compter les « e » sur le ruban d’une machine caractère....3
Ligne « if (lvMachine.caractereCourant () == 'e') { »....3
Résultats sur des exemples....4
2. Compter les mots....5
Ensemble des caractères formant un mot....5
Un automate à états finis....5
Automate : un squelette de boucle....6
Automate : L’initialisation....6
Automate : La condition....6
Automate : Le traitement....6
Traitement de l’état ETAT_SEP....7
Traitement de l’état ETAT_MOT....8
Automate : Avancer....8
Compter les mots....8
3. Compter les « le »....10
Initialisation....10
Condition....10
Traitement....10
Avancement....10
Compter les ‘le’....10
4. Notions abordées....12
5. Licence....13

1. Compter les « e » sur le ruban d'une machine caractère

Reprenons à partir du programme « compterCar ». Ce programme, présenté dans un premier cours, compte tous les caractères d'un ruban d'une machine caractère.

```
#include <iostream>
#include <cstdint>
#include "machineCaractere.h"

using namespace std;

int main (int pvArgc, char* paArgv[]) {

     uint16_t lvCompteur;

    cout << "Lancement du programme " << paArgv[0] << endl;

     for (int lvNoArg = 1; lvNoArg < pvArgc; lvNoArg++) {

        string lvChaine (paArgv[lvNoArg]);
        CMachineCaractere lvMachine (lvChaine);
        cout << "La machine caractères est initialisée avec \"" << lvChaine << "\"" << endl;
        lvCompteur = 0;

          while (!lvMachine.marque ()) {

            lvCompteur++;
            lvMachine.avancer ();
            };

        cout << "La machine caractère contient " << lvCompteur << " caractères" << endl;
        };
    return 0;
    }
```

Rappelons la boucle principale comptant les caractères :

```
lvCompteur = 0;

while (!lvMachine.marque ()) {

    lvCompteur++;
    lvMachine.avancer ();
    };
```

À cette boucle, comptant les caractères sans distinction aucune, nous allons ajouter une condition pour incrémenter le compteur. Comme nous cherchons à compter les 'e' du texte, nous encadrons l'incrémentation du compteur (instruction lvCompteur++) de la condition « *Si le caractère courant est un 'e'* ». En langage C++, cela donne :

```
lvCompteur = 0;

while (!lvMachine.marque ()) {

    if (lvMachine.caractereCourant () == 'e') {
        lvCompteur++;
        };
    lvMachine.avancer ();
    };
```

Décortiquons la ligne supplémentaire :

Ligne « if (lvMachine.caractereCourant () == 'e') { »

- if (*condition*) { *traitement* }

  L'instruction « `if` » vérifie si la condition « *condition* » est réalisée et dans l'affirmative, exécute les instructions symbolisée ici par le mot « *traitement* ». L'ensemble des instructions concernées est encadré par les accolades « {} ». Dans notre exemple, le traitement est le même que celui déjà vu dans le programme où nous comptions le nombre de caractères : `lvCompteur++;`

- lvMachine.caractereCourant () == 'e'

La condition est simple : nous incrémentons le compteur uniquement si le caractère courant de la machine caractère est un ‘e’. L’expression conditionnelle doit renvoyer un « booléen » (type `bool` en C++), c’est à dire une information dont la valeur est soit vraie (`true` en C++), soit fausse (`false` en C++).

Le caractère courant de la machine caractère est donné par la fonction « caractereCourant ». Le caractère courant de notre machine caractère « lvMachine » s’écrit donc « lvMachine.caractereCourant () ». Ce caractère courant doit être comparé à la lettre ‘e’. La condition d’égalité s’écrit « == ». Elle ne doit pas être confondue avec l’affectation « = » comme dans l’instruction « `lvCompteur = 0;` ». L’affectation attribue une valeur à une variable. La condition d’égalité vérifie que deux expressions sont égales. Afin de différencier ces deux sémantiques, les écritures diffèrent également.

Résultats sur des exemples

Commande : ./compterE "Elle n'est pas veuve" "Quelle belle epee !" ""

La machine caractères est initialisée avec "Elle n'est pas veuve"
La machine caractère contient 4 'e'

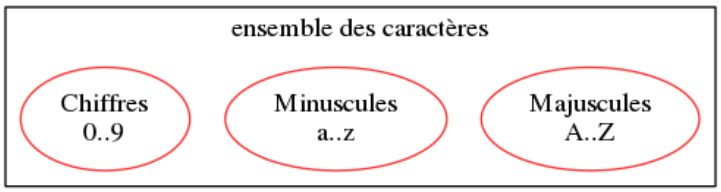
La machine caractères est initialisée avec "Quelle belle epee !"
La machine caractère contient 7 'e'

La machine caractères est initialisée avec ""
La machine caractère contient 0 'e'

2. Compter les mots

Comme pour le comptage des ‘e’, nous allons initialiser le programme « compterMots » à partir du programme compterCar. Mais avant de plonger dans le code, nous schématiser le traitement sous forme de schéma. Ce schéma est un automate à nombre d’états finis. La conversion en code sera ensuite facile à mettre en œuvre. Inversement, étant donné un code implémentant un automate, il sera facile de reconstituer le schéma si nous n’avons pas celui-ci.

Ensemble des caractères formant un mot



Comme précisé dans le libellé de l’exercice, un mot est constitué d’une suite de lettres majuscules ou minuscules non accentuées ou de chiffres.

Notons dès à présent que l’ensemble des chiffres, l’ensemble des majuscules et l’ensemble des minuscules sont des sous-ensembles de l’ensemble des caractères. Par ailleurs ces trois sous-ensembles sont disjoints. Enfin, l’union de ces trois sous-ensembles ne forment pas la totalité de l’ensemble des caractères. Autrement dit, il existe des caractères qui n’appartiennent à aucun de ces trois sous-ensembles. Nous les considérerons comme **séparateurs**.

Un automate à états finis

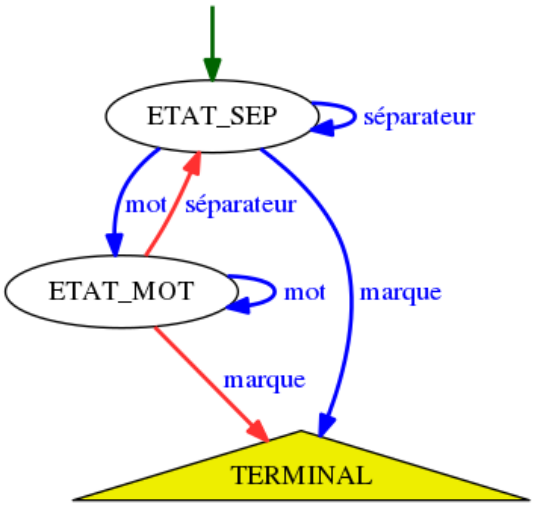


Illustration 1: automate dessiné automatiquement via l'application graphviz (www.graphviz.org)

Le traitement de comptage des mots peut être modélisé par le schéma ci contre. Nous entrons dans le schéma par la flèche verte. Le schéma impliquera une boucle. L’initialisation de la boucle devra prendre en compte cette entrée par la flèche verte.

L’automate est constitué de trois états :

- L’état ETAT_SEP précise que nous sommes dans le sous-ensemble des séparateurs.
- L’état ETAT_MOT précise que nous sommes dans un mot correspondant donc à un mot composé de l’un des trois sous-ensembles « Majuscule », « Minuscule » ou « Chiffres ».
- L’état TERMINAL indiquant que l’automate se termine. Nous retrouverons cette terminaison dans la condition de sortie de la boucle.

Outre les états, l’automate est constitué de transitions entre états.

- Dans tous les cas, lorsque la machine caractères arrive sur la marque, il y a transition vers l’état TERMINAL.
- Lorsque le caractère courant est un séparateur, il y a transition vers l’état ETAT_SEP.
- Lorsque le caractère courant appartient à l’un des trois sous-ensemble Majuscule, Minuscule ou Chiffre, il y a transition vers l’état ETAT_MOT.
- Enfin, deux de ces transitions (en rouge sur le schéma) impliquent la fin d’un mot. C’est lors de ces transitions qu’il faudra comptabiliser le mot.

Automate : un squelette de boucle

Appliquons le schéma algorithmique d'une boucle « Tant que » au cas de l'automate à états finis :

Initialiser
Tant que l'état n'est pas terminal
    Selon état :
        - État ETAT_SEP : Traitement associé à un séparateur
        - État ETAT_MOT : Traitement associé à un mot
    Avancer

Automate : L'initialisation

L'initialisation est décomposée en deux éléments : l'initialisation du compteur et l'initialisation de l'automate.

L'initialisation du compteur comporte la déclaration et la mise à zéro du compteur :

```
uint16_t lvCompteur;
lvCompteur = 0;
```

Nous laissons au lecteur le soin de placer les instructions aux emplacements les plus judicieux dans le code source du programme.

Concernant l'initialisation de l'automate, nous introduisons la notion d'énumération permettant une meilleure compréhension du code source ainsi que des contrôles de la part du compilateur.

Notre automate étant composé de trois états, nous définissons l'énumération « `enEtat` » de ces trois états. Il s'agit d'un type de donnée défini par le programmeur spécifique à cet automate. Par ailleurs, nous déclarons la variable « `lvEtat` » du type de l'énumération « `enEtat` » qui est ensuite initialisée à l'état premier de l'automate :

```
enum enEtat { TERMINAL, ETAT_SEP, ETAT_MOT };
enEtat lvEtat
lvEtat = ETAT_SEP;
```

Automate : La condition

Le parcours de l'automate sera donc une boucle. Comme toute boucle, une condition de sortie est nécessaire. Si vous regardez attentivement le schéma de l'automate, vous constaterez que les transitions vers l'état TERMINAL sont liées au fait que nous avons atteint la marque sur la machine caractère. Nous avons donc deux possibilités pour écrire la condition de sortie :

- soit nous avons atteint l'état TERMINAL.
- soit nous avons atteint la marque.

Dans un souci de cohérence, puisque nous travaillons sur un automate, nous allons choisir l'état TERMINAL.

Notre code seront donc de la forme :

```
while (lvEtat != TERMINAL) { … };
```

Les points de suspension représente le traitement et l'avancement de la boucle.

Automate : Le traitement

Le traitement de l'automate consiste à parcourir les états de l'automate en fonction des caractères rencontrés sur le ruban de la machine caractère. Il reproduit fidèlement le dessin de l'automate et selon la valeur de l'état, passe les transitions vers d'autres états.

Ce traitement prend donc la forme suivante :

```
switch (lvEtat) { // Selon la valeur de lvEtat
    case ETAT_SEP : { // État "séparateur"
        // Traitement de l'état ETAT_SEP
        break;
        }
    case ETAT_MOT : { // État "Mot"
```

```
            // Traitement de l'état ETAT_MOT
            break;
            };
        };
```

Ce traitement est constitué d'une instruction « `switch` (lvEtat) » pouvant être traduite par « selon la valeur de l'état ». Suivent ensuite des blocs de la forme « `case` ETAT_SEP `:` `{` … `break;` `}` » représentants le traitement concernant un état bien précis (ETAT_SEP dans l'exemple). Ce bloc est terminé par une instruction « `break` » indiquant au compilateur de terminer l'instruction « `case` » et ne pas prolonger sur l'instruction « `case` » suivante. Le squelette de l'instruction « switch » associée à ses instructions « case » étant posé, voyons maintenant le traitement associé à chaque état : ETAT_SEP et ETAT_MOT respectivement les états correspondants à un séparateur et à un mot.

Traitement de l'état ETAT_SEP

Dans cet état, nous avons trois transitions à définir.

1. Si nous avons atteint la marque, nous basculons vers l'état TERMINAL. Voici le code correspondant :

```
    if (lvMachine.marque ()) {
        lvEtat = TERMINAL;
        }
```

2. Si nous avons un début de mot, nous basculons vers l'état ETAT_MOT. Un début de mot correspondant à l'un des trois sous-ensembles « minucules », « majuscules » ou « chiffres », nous testons successivement ces trois possibilités après avoir lu le caractère courant de la machine caractères. Pour lire le caractère courant, la documentation de la machine caractères précise que la marque ne doit pas être atteinte. Ces trois tests ne doivent donc être réalisés que si la marque n'est pas atteinte. Comme nous venons de tester la marque, l'instruction « `if` » est prolongée par un « `else` » (sinon).

   Le test concernant les minuscules s'écrit « `((lvCar >='a') && (lvCar <= 'z'))`. En français, nous exprimons cette condition « le caractère courant est supérieur ou égal à 'a' et inférieur ou égal à 'z' ». l'opérateur et s'écrivant « `&&` ». Si ce test est vérifié, nous basculons l'état vers ETAT_MOT.

   En complétant avec des tests similaires pour majuscules et chiffres, nous obtenons le code suivant.

```
    else {
        char lvCar = lvMachine.caractereCourant ();
        if ((lvCar >='a') && (lvCar <= 'z')) lvEtat = ETAT_MOT;
        if ((lvCar >='A') && (lvCar <= 'Z')) lvEtat = ETAT_MOT;
        if ((lvCar >='0') && (lvCar <= '9')) lvEtat = ETAT_MOT;
        };
```

   Notons que, pour chaque instruction « if », nous n'avons pas entouré d'accolades (« {} ») l'instruction « lvEtat = ETAT_MOT; ». Sans accolades, l'instruction if ne concerne que la seule instruction suivante. Il s'agit d'un raccourci d'écriture quelquefois bien commode. Rien n'interdit au programmeur d'entourer l'instruction d'accolades s'il juge l'écriture mieux compréhensible.

3. Enfin, si le caractère courant de la machine caractères est un séparateur, nous restons dans le même état. Dans un tel cas, l'état reste inchangé et nous n'avons donc rien à écrire. De plus, dans cet état, nous ne traitons que le cas des séparateurs. Nous n'avons donc pas de mot à comptabiliser et donc pas de compteur à actualiser.

Le traitement de l'état ETAT_SEP est donc :

```
case ETAT_SEP : { // État "séparateur"
    if (lvMachine.marque ()) {
        lvEtat = TERMINAL;
        }
    else {
        char lvCar = lvMachine.caractereCourant ();
        if ((lvCar >='a') && (lvCar <= 'z')) lvEtat = ETAT_MOT;
        if ((lvCar >='A') && (lvCar <= 'Z')) lvEtat = ETAT_MOT;
        if ((lvCar >='0') && (lvCar <= '9')) lvEtat = ETAT_MOT;
        };
    break;
    }
```

Traitement de l'état ETAT_MOT

Dans cet état, nous avons trois transitions à définir. Globalement, au vu du schéma, l'état ETAT_MOT ressemble beaucoup à l'état ETAT_SEP. La différence principale concerne l'actualisation du compteur de mots. Ce compteur est mis à jour lorsque nous quittons l'état ETAT_MOT. D'après le schéma, ous avons donc trois transitions à prendre en compte.

1. Si nous avons atteint la marque, nous basculons vers l'état TERMINAL. Voici le code correspondant :

```
    if (lvMachine.marque ()) {
        lvEtat = TERMINAL;
        }
```

2. Si le caractère courant de la machine caractères n'est ni une minuscule, ni une majuscule, ni un chiffre, nous revenons à l'état ETAT_SEP. Ici, nous allons raisonner à l'inverse. Nous allons d'abord supposer que nous avons un séparateur et nous basculons donc inconditionnellement vers l'état ETAT_SEP. Ensuite, comme précédemment, nous revenons à l'état ETAT_MOT si avons rencontré une minuscule, une majuscule ou un chiffre.

Enfin, nous allons actualiser le compteur. Cette actualisation n'a lieu que si nous avons une transition soit vers l'état TERMINAL, soit vers l'état ETAT_SEP. Autrement dit, si nous quittons l'état ETAT-MOT. Le code correspondant est donc :

```
    if (lvEtat != ETAT_MOT) lvCompteur++;
```

Le code final du traitement de l'état ETAT_MOT est donc :

```
case ETAT_MOT : { // État "Mot"
    if (lvMachine.marque ()) {
        lvEtat = TERMINAL;
        }
    else {
        char lvCar = lvMachine.caractereCourant ();

        lvEtat = ETAT_SEP;
        if ((lvCar >='a') && (lvCar <= 'z')) lvEtat = ETAT_MOT;
        if ((lvCar >='A') && (lvCar <= 'Z')) lvEtat = ETAT_MOT;
        if ((lvCar >='0') && (lvCar <= '9')) lvEtat = ETAT_MOT;
        };
    if (lvEtat != ETAT_MOT) lvCompteur++;
    break;
    }
```

Automate : Avancer

Suite au traitement de l'automate, nous revenons à notre boucle et devons nous occuper d'avancer. Ceci consiste à passer au caractère suivant de la machine caractères si nous n'avons pas atteint la marque :

```
    if (!lvMachine.marque ()) lvMachine.avancer ();
```

Compter les mots

Nous sommes maintenant en mesure d'écrire le programme complet. Vous noterez comment ajouter un « s » à « mot » si le compteur est supérieur à 1 via l'instruction « (condition) ? Valeur_si_vraie : valeur_si_fausse ».

```
#include <iostream>
#include <cstdint>
#include "machineCaractere.h"

using namespace std;

int main (int pvArgc, char* paArgv[]) {

    enum enEtat { TERMINAL, ETAT_SEP, ETAT_MOT };

    uint16_t lvCompteur;
    enEtat lvEtat;

    cout << "Lancement du programme " << paArgv[0] << endl;
```

```
for (int lvNoArg = 1; lvNoArg < pvArgc; lvNoArg++) {

    string lvChaine (paArgv[lvNoArg]);
    CMachineCaractere lvMachine (lvChaine);

    cout << "La machine caractères est initialisée avec \"" << lvChaine << "\"" << endl;

    lvCompteur = 0;
    lvEtat = ETAT_SEP;

    while (lvEtat != TERMINAL) {
        switch (lvEtat) { // Début de l'automate
            case ETAT_SEP : { // État "séparateur"
                if (lvMachine.marque ()) {
                    lvEtat = TERMINAL;
                    }
                else {
                    char lvCar = lvMachine.caractereCourant ();
                    if ((lvCar >='a') && (lvCar <= 'z')) lvEtat = ETAT_MOT;
                    if ((lvCar >='A') && (lvCar <= 'Z')) lvEtat = ETAT_MOT;
                    if ((lvCar >='0') && (lvCar <= '9')) lvEtat = ETAT_MOT;
                    };
                break;
                }
            case ETAT_MOT : { // État "Mot"
                if (lvMachine.marque ()) {
                    lvEtat = TERMINAL;
                    }
                else {
                    char lvCar = lvMachine.caractereCourant ();

                    lvEtat = ETAT_SEP;
                    if ((lvCar >='a') && (lvCar <= 'z')) lvEtat = ETAT_MOT;
                    if ((lvCar >='A') && (lvCar <= 'Z')) lvEtat = ETAT_MOT;
                    if ((lvCar >='0') && (lvCar <= '9')) lvEtat = ETAT_MOT;
                    };
                if (lvEtat != ETAT_MOT) lvCompteur++;
                break;
                }
            }
        if (!lvMachine.marque ()) lvMachine.avancer ();
        };

    cout << "La machine caractère contient " << lvCompteur << ((lvCompteur > 1) ? "mots" : "mot") << endl;
    };
return 0;
}
```

3. Compter les « le »

Pour compter le nombre de « le » (en caractères minuscules), nous allons revenir vers une boucle classique. Nous pourrions également dessiner un automate et écrire le programme correspondant. Dans notre boucle, nous allons reprendre les fondamentaux et travailler successivement sur les différentes étapes d'une boucle : Initialisation, Condition, Traitement et Avancement.

Pour chacune de ces étapes, nous allons focaliser notre attention sur trois variables :

- Une variable lvMachine représentant la machine caractère sur le ruban de laquelle nous allons compter les « le ».
- Une variable lvCompteur devenue habituelle dans nos boucles.
- Une variable lvCarPdt représentant le caractère précédant le caractère courant.

Initialisation

La variable lvMachine est initialisée dans sa propre boucle avec les paramètres issus de la ligne de commande. Nous l'avons déjà fait et ne revenons donc pas sur cette initialisation.

La variable lvCompteur est initialisée à 0.

Lors du traitement, nous incrémenterons le compteur lorsque le caractère courant sera 'e' **et** le caractère précédent sera 'l'. Le premier caractère du ruban pourra être 'e'. Dans un tel cas, il ne faudra pas que le caractère précédent – qui n'existe pas encore – soit un 'l' afin d'éviter d'incrémenter indûment le compteur. Nous initialisons donc la variable lvCarPdt à tout caractère sauf un 'l'.

Condition

La condition d'arrêt de notre boucle est similaire aux programmes précédent : nous testons si la fin du ruban de lvMachine est atteinte.

Traitement

Le traitement consiste à lire le caractère courant sur le ruban et à incrémenter le compteur lorsque les caractères précédent et courant sont respectivement un 'l' et un 'e'.

Avancement

Pour avancer, nous devons tout d'abord écrire que le caractère courant devient le caractère précédent et passer sur le caractère suivant du ruban de lvMachine.

Compter les 'le'

```
#include <iostream>
#include <cstdint>
#include "machineCaractere.h"

using namespace std;

int main (int pvArgc, char* paArgv[]) {

    cout << "Lancement du programme " << paArgv[0] << endl;

     for (int lvNoArg = 1; lvNoArg < pvArgc; lvNoArg++) {

        uint16_t lvCompteur;
        char lvCar;
        char lvCarPdt;
        string lvChaine (paArgv[lvNoArg]);
        CMachineCaractere lvMachine (lvChaine);

        cout << "La machine caractères est initialisée avec \"" << lvChaine << "\"" << endl;
```

```
    lvCompteur = 0;
    lvCarPdt = 'a'; // doit être différent de 'l'

      while (!lvMachine.marque ()) {

        lvCar = lvMachine.caractereCourant ();
        if ((lvCarPdt == 'l') && (lvCar == 'e')) {
            lvCompteur++;
            };

        lvCarPdt = lvCar;
        lvMachine.avancer ();
        };

    cout << "La machine caractère contient " << lvCompteur << " 'le'" << endl;
    };

return 0;
}
```

4. Notions abordées

Au travers des exercices sur la machine caractères, nous avons abordé les notions suivantes :

- Boucles for, while
- Les instructions if else, switch case break
- La technique de l’automate à nombre d’états fini.
- L’énumération enum

Dans un prochain cours, nous allons voir comment paramétrer et construire les programmes des exercices de la machine caractères avec l’utilitaire cmake.

5. Licence