

Cours n°9

Textes, langages formels et templates



Sommaire

1. Représentation des textes

1.1 Classe Node

1.2 Classe Edge

2. Analyse des textes

2.1 Symboles et motifs

2.2 Règles de production

Bibliothèques de templates

Regex

Expression régulière

www.cplusplus.com/reference/regex/

Common Text Transformation Library (cttl)

Classe de chaînes de caractères avec marques et segments

Comparaison, insertion, modification et analyse à l'aide de grammaires EBNF

sourceforge.net/projects/cttl/

Yet another Object-Oriented Lex (YooLex)

Construction d'analyseurs lexicaux (cf. outil unix lex)

sourceforge.net/projects/yoolex/

Automata Standard Template Library (astl)

Construction et utilisation d'automates d'états finis

sourceforge.net/projects/astl/

DicoLib

Création et gestion de dictionnaires

sourceforge.net/projects/dicolib/

Constructeurs et Accesseurs

Gestion de marques dans la chaîne

Nœud Identifié (numéro) correspondant à une position (offset) dans la chaîne

Constructeurs

Construction d'un noeud dans la string str à la position pos

node(string str, int pos)

Construction d'une copie du nœud n

node(node<T> n)

Construction d'une copie du nœud n avec une position différente

node(node<T> n, int pos)

Accesseurs

Accès au Conteneur **input<T> parent()**

Accès au numéro **int identity()**

Position **int offset()**

Numéro de ligne correspondant à la position **int line()**

Caractère d'indice i relativement à la position du noeud **char operator[] (int i)**

Gestion des positions des marques

Méthodes modifiant la position

<code>int offset(int p)</code>	Position du nœud à la valeur p
<code>int go_bof()</code>	Position du nœud au début du conteneur
<code>int go_eof()</code>	Position du nœud à la fin du conteneur
<code>int go_line_next()</code>	Position du nœud au début de la ligne suivante
<code>int go_line_previous()</code>	Position du nœud au début de la ligne précédente
<code>int go_line_home()</code>	Position du nœud au début de la ligne courante
<code>int go_line_home(int l)</code>	Position du nœud au début de la ligne l
<code>int go_line_end()</code>	Position du nœud à la fin de la ligne courante
<code>int go_line_end(int l)</code>	Position du nœud à la fin de la ligne l

Méthodes d'insertion de texte

Insertion d'un texte au début du nœud, décalage des positions des nœuds suivants

`void insert_stay(string s)`

Insertion d'un texte avant le nœud, décalage des positions des nœuds suivants

`int insert_go(string s)`

1.1 CLASSE NODE

Lecture, création du conteneur et des marques

```
string texte;  
cttl::file2string("Data/livres/adelaide.txt", texte);  
node<> p1( texte ); p1.go_bof();  
node<> p2( texte ); p2.go_eof();  
cout << "ce texte comporte " << p2.offset() << " caractères et " ;  
cout << p2.line()-1 << " lignes" << endl;  
  
// lecture du caractère j de la ligne i  
int i,j; cin >> i >> j; p1.go_line_home(i+1);  
cout << p1.offset() << " " << p1[j] << endl;  
  
// insertion et décalage  
p1.insert_stay("insertion");  
cout << p1.offset() << " " << p2.offset() << endl;  
p1.insert_go("destruction");  
cout << p1.offset() << " " << p2.offset() << endl;
```

```
ce texte comporte 46355 caractères et 192 lignes  
10 4  
4329 c  
4329 46364  
4340 46375
```

1.1 CLASSE NODE

Création du conteneur et conteneur de marques

```
ifstream fEntree("Data/livres/adelaide.txt");
string phrase, texte;
vector<node<> > vp;
int n = 0;
while (!fEntree.eof()) {
    getline(fEntree, phrase); phrase += "\n";
    vp.push_back(*(new node<>(texte)));
    vp[n].go_eof(); vp[n].insert_stay(phrase);
    n++;
}

for (int i = 0; i < n; i++)
    cout << i << " " << vp[i].offset() << endl;
```

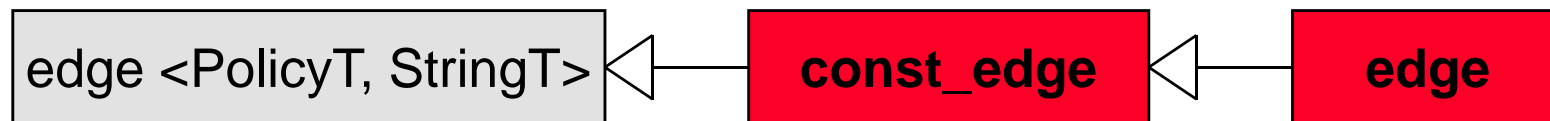
```
0 0
1 141
2 142
3 1160
4 1161
5 2800
6 2801
```

Composants

Deux spécialisations du patron de classes edge

const_edge : pas d'accesseurs en écriture (sauf casse)

edge : accesseurs en lecture et en écriture



PolicyT (white space policy) : modélisation des commentaires

Constructeurs et Accesseurs (const_edge)

Gestion des segments de la chaîne (sous-chaînes)

Couple de marques (first et second) avec modélisation des commentaires

Constructeurs

Construction d'un segment entre deux marques d'un même conteneur

const_edge (node<T> md, node<T> mf)

Construction d'un segment dans la string str entre deux positions

const_edge (string str, int m1, int m2)

Construction d'un segment à partir d'un autre segment

const_edge (const_edge<T1, T2> e)

Accesseurs

Texte du segment

string text()

Longueur du segment

int length()

Passage du texte en majuscules

void text_tolower()

Passage du texte en minuscules

void text_toupper()

Gestionnaire des séparateurs

PolicyT space_policy()

1.2 CLASSES CONST_EDGE ET EDGE

Lecture, segmentation et création des segments

```
string texte;  
cttl::file2string("Data/livres/adelaide.txt",texte);  
int i1 = -1, i2, n = 0;  
vector<const_edge<> > vs;  
while ((i2 = texte.find('\n', i1+1)) != string::npos) {  
    node<> p1(texte, i1+1); p1.offset(i1+1);  
    node<> p2(texte, i2); p2.offset(i2);  
    vs.push_back(*(new const_edge<>(p1, p2))); n++; i1 = i2;  
}  
// lecture de la ligne i  
int i; cin >> i;  
cout << vs[i].first.offset() << " " << vs[i].second.offset() << endl;  
cout << vs[i].text() << endl;
```

6

2801 3981

Elle se dit: je ferai un heureux. J'aurai un esclave qui me devra tout, et le premier succès et le premier bonheur et la première gloire et la première expérience. Il m'adorera; et, si je l'adore, je ne le lui dirai pas comme je le sens, et je régnerai sur lui. Je l'entraînerai où il me plaira qu'il aille et je le connaîtrai à fond: tête et coeur, bien et mal, vices et vertus. Des premiers je

Constructeurs et Accesseurs (edge)

Modification directe des textes d'un segment des

Constructeurs

Construction d'un segment entre deux marques d'un même conteneur

edge (`node<T>` md, `node<T>` mf)

Construction d'un segment dans la string str entre deux marques (numéro)

edge (`string` str, `int` m1, `int` m2)

Construction d'un segment à partir d'un autre segment

edge (`edge<T1, T2>` s) **edge** (`const_edge<T1, T2>` s)

Accesseurs `text`.

Modification du texte du segment `void text(string)`

Echange des textes entre segments `void text_swap(edge<T1, T2> s)`

Modélisation des commentaires (white space policy)

Commentaire

Chaînes de caractères n'appartenant pas au langage
Pas de prise en compte dans une analyse
Annotation dans un texte

Modélisation

Paramètre d'un segment de texte

Définition par défaut (pas de commentaire)

Espace ou caractères de formatage

Commentaires C/C++/Java

`policy_default`

`policy_space< flag_follow_space >`

`policy_space< flag_cpp_comments`

Description par l'utilisateur de segments
de texte de commentaires (2 passes)

`policy_space< flag_follow_region >`

Insertion d'un segment de commentaires

`void region_insert()`

Destruction d'un segment de commentaires

`void region_erase()`

Texte du segment sans commentaires

`string region_difference ()`

Notation Extended Backus-Naur Form (EBNF)

Ensemble de règles de productions (rule)

variables,

symboles auxiliaires [] optionnel, {} zéro ou plus

\$start = \$mcmd | \$dcmd | \$icmd | \$ecmd;

\$mcmd = bouger \$dir;

\$dcmd = detruire [\$item];

\$icmd = inserer;

\$ecmd = fin [inserer];

\$dir = haut | bas | gauche | droite;

\$item = caractere | mot | ligne | page;

Phrases du langage

{bouger haut, detruire mot, inserer, fin }

Fonction `symbol()`

Symbole terminal du langage

Implémentation avec une fonction lexème

Caractère quelconque `symbol()`

Symbole vide `symbol(true)`

Symbole stop `symbol(false)`

Caractère `c` `symbol(char c)`

Tableau de caractères `tc` `symbol(char* tc)`

Chaîne de caractères `s` `symbol(string s)`

symbole optionnel `*symbol(...)`

symbole multiple `+symbol(...)`

Opérateur de concaténation de symboles `(symbol(...) + symbol(...))`

Opérateur logique entre symboles `(symbol(...) | symbol(...))`

Position des symboles terminaux

Début du motif `first(char* tc)` `first(string s)`

Début du segment `begin()`

Fin du segment `end()`

2.1 SYMBOLES ET MOTIFS

Recherche de motifs dans un segment

Méthodes associées à un motif

`int match(edge<T1, T2> s)` Appariement du segment avec un motif
`int find(edge<T1, T2> s)` Recherche d'un motif dans le segment
`int bang_find(edge<T1, T2> s)` Recherche successive de tous les motifs
 Retour de la position du début du motif, position de la fin dans l'attribut first de s

```

string texte; file2string("Data/livres/adelaide.txt",texte);
// recherche d'une chaine de caracteres
edge<> e = *(new edge<>(texte, 0, texte.size()));

int p = symbol("Hautcastel").find(e);
cout << p << " " << texte.substr(p, e.first.offset()-p) << endl;

// recherche en boucle du motif a{n}
while ((p = (symbol('a') + +symbol('n')).bang_find(e)) !=string::npos)
    cout << p << " " << texte.substr(p, e.first.offset()-p) << " ";
  
```

```
10 Hautcastel
```

```

24 an 144 ann 172 ann 254 an 377 an 455 an 493 ann 649 an 672 an
682 an 821 an 1005 an 1028 an 1086 an 1167 an 1245 an 1259 an 1324
  
```

Fonction entity()

Classe de symboles terminaux

Caractère quelconque	entity()
Caractère c	entity(char c) éq. symbol(char c)
Caractère du tableau tc	entity(char* tc)
Caractère de la chaîne de caractères s	entity(string s)
Caractère alphanumérique	entity(isalnum)
Caractère alphabétique	entity(isalpha)
Caractère décimal	entity(isdigit)
Caractère majuscule	entity(islower)
Caractère minuscule	entity(isupper)
Caractère de ponctuation	entity(ispunct)
Classe de symboles optionnelle	*entity(...)
Classe de symboles multiple	+entity(...)
Opérateur de concaténation de classes	(entity(...) + entity(...))
Opérateur logique entre classes	(entity(...) entity(...))

2.1 SYMBOLES ET FORMES

Recherche de noms propres dans un segment

```

int NomPropre(const_edge<>& edge_ ) {
    string maj = "ABCDEFGHJKLMNOPQRSTUVWXYZ";
    string min = "abcdefghijklmnopqrstuvwxyzéèà";
    return ( entity(maj) + +entity(min) ).bang_find(edge_);
}

int main() {
    string texte;
    file2string("Data/livres/adelaide.txt", texte);

    // recherche en boucle du motif maj{min}
    const_edge<> e = *(new const_edge<>(texte, 0, texte.size()));
    int p;
    while ((p = NomPropre(e)) != std::string::npos)
        cout << p << " " << texte.substr(p, e.first.offset()-p) << " ";}

```

```

0 Madame 10 Hautcastel 158 Frédéric 167 Rothbanner 250 Hermansburg
276 Ce 313 Rien 430 Le 438 Ma 512 Bernstein 677 Le 863 Son 867
Altesse 875 Royale 968 Bref 1035 Au 1161 Cependant 1388 De 1400
Rothbanner 1571 Ce 1616 Pour 1937 Hermansburg 2062 Elle 2147 Pour

```

Représentation des règles de production par des fonctions d'appariement

```
static size_t sp(const_edge<>& e) { return symbol(' ') match(e); }
```

Appel des fonctions d'appariement avec l'adaptateur de fonction rule()

```
static size_t start(const_edge<>& e) { return rule(sp) match(e); }
```

rule(&r) avec r est une fonction globale ou une fonction membre statique

rule(O,&C::r) avec O est une classe-grammaire, C une instance de O et r une fonction membre non statique

Appel avec trace pour le déverminage de grammaires

CTTL_RULE(&C::r), CTTL_MEMBER_RULE(O,&r), CTTL_STATIC_RULE(r)

2.2 REGLES DE PRODUCTION

Définition de la grammaire

```
class Grammaire {
static size_t item( const_edge<policy_space<> >& e) {
    return (symbol("caractere") | symbol("mot") | symbol("ligne") |
            symbol("page") ).match(e); }
static size_t dir( const_edge<policy_space<> >& e) {
    return (symbol("haut") | symbol("bas") | symbol("gauche") |
            symbol("droite") ).match(e); }
static size_t ecmd( const_edge<policy_space<> >& e) {
    return ( symbol("fin") + *symbol("insérer") ).match(e); }
static size_t icmd( const_edge<policy_space<> >& e) {
    return symbol("insérer").match(e); }
static size_t dcmd( const_edge<policy_space<> >& e) {
    return ( symbol("détruire") + *rule(item) ).match(e); }
static size_t mcmd( const_edge<policy_space<> >& e) {
    return ( symbol("bouger") + rule(dir) ).match(e); }

public:
static size_t start( const_edge<policy_space<> >& e) {
    return ( rule(mcmd) | rule(dcmd) | rule(icmd) + rule(ecmd)
            ).match(e); }
};
```

2.2 REGLES DE PRODUCTION

Appartenance à la grammaire

```
void appartient(string s) {
    const_edge<policy_space> > e = *(new const_edge<policy_space> >(s,
    0, s.size()));

    if (rule(Grammaire::start).match(e) == 0)
        cout << s << " appartient au langage" << endl;
    else
        cout << s << " n'appartient pas au langage" << endl;
}

int main() {
    appartient ("bouger bas");
    appartient ("detruire caractere");
    appartient ("bouger caractère");

    return 0;
}
```

bouger bas appartient au langage

detruire caractere appartient au langage

bouger caractère n'appartient pas au langage