



Travail Pratique 2

par

Guillaume Lahaie
LAHG04077707

et

Sylvain Labranche
LABS02059007

remis à

Emmanuel Chieze

dans le cadre du cours :

Construction et maintenance de logiciels
INF3135-20

Département d'informatique
18 décembre 2012

```
1  /* Chaine.c
2  * Par Guillaume Lahaie et Sylvain Labranche
3  *     LAHG04077707     LABS02059007
4  *
5  * Date de remise: 18 décembre 2012
6  *
7  * Dernières modifications: 12 décembre 2012
8  *
9  * Implémentation du module chaine.h, pour des chaines de caractères
10 * dynamiques.
11 */
12
13 #define TEXTE_PAS 10      //Valeur pour agrandir l'allocation-mémoire de la chaine.
14 #include "chaine.h"
15 #include <stdlib.h>
16 #include <string.h>
17 #include <assert.h>
18
19 //Longueur indique le nombre de caractères dans la chaine, en comprenant le \0.
20 struct chaine {
21     char *texte;
22     unsigned longueur;
23     unsigned longueurMax;
24 };
25
26
27 Chaine chaineCreeVide() {
28
29     Chaine chaine = (Chaine)malloc(sizeof(struct chaine));
30     if(!chaine) {
31         return NULL;
32     }
33
34     chaine->texte = (char *)malloc(TEXTE_PAS*sizeof(char));
35     if(!chaine->texte) {
36         free(chaine);
37         return NULL;
38     }
39
40     //Insère \0 pour la fin de la chaine.
41     chaine->texte[0] = '\0';
42     chaine->longueur = 1;
43     chaine->longueurMax = TEXTE_PAS;
44
45     return chaine;
46 }
47
48 Chaine chaineCreeCopie(char * ch, unsigned n) {
49
50     assert(n <= (strlen(ch)) && "n plus grand que la longueur de ch");
51
52     Chaine chaine = (Chaine)malloc(sizeof(struct chaine));
53     if(!chaine)
54         return NULL;
55
56     chaine->texte = (char *)malloc((n+1)*sizeof(char)); //+1 pour \0
57     if(!chaine->texte) {
58         free(chaine);
59         return NULL;
60     }
61
62     strncpy(chaine->texte, ch, n);
63
64     //On s'assure que la chaine se termine par \0
65     chaine->texte[n] = '\0';
66
67     chaine->longueur = chaine->longueurMax = n+1;
68
69     return chaine;
```

```
70 }
71
72 //J'utilise une chaine temporaire pour éviter de perdre la chaine passée en
73 //argument si le realloc ne fonctionne pas. Le seul cas où la fonction
74 //retourne faux est si la Chaine est pleine et que realloc échoue.
75 int chaineAjoute(Chaine chaine, unsigned char c) {
76
77     assert(chaine != NULL && "Erreur: chaine NULL");
78
79     char * temp; //Pour réalloc: on ne perd pas de données en cas d'échec
80
81     if(chaine->longueur >= chaine->longueurMax) {
82         temp = (char *)realloc(chaine->texte,
83                                 (chaine->longueurMax + TEXTE_PAS)*sizeof(char));
84
85         if(!temp)
86             return 0; //Problème avec realloc
87
88         chaine->texte = temp;
89         chaine->longueurMax += TEXTE_PAS;
90     }
91     chaine->texte[chaine->longueur-1] = c;
92     chaine->texte[chaine->longueur++] = '\0';
93     return 1;
94 }
95
96 //La fonction retourne NULL si l'allocation de mémoire a échoué.
97 char * chaineValeur(Chaine chaine) {
98
99     assert(chaine != NULL && "Erreur: chaine NULL");
100
101     char * retour = (char*)malloc((chaine->longueur)*sizeof(char));
102
103     if(!retour) {
104         return NULL;
105     }
106
107     strcpy(retour, chaine->texte);
108
109     return retour;
110 }
111
112 //Retourne le nombre de caractères de la chaine, sans compter le
113 //caractère de fin de chaine.
114 unsigned chaineLongueur(Chaine chaine) {
115     assert(chaine != NULL && "Erreur: chaine NULL");
116     return chaine->longueur-1;
117 }
118
119 void chaineSupprime(Chaine chaine) {
120     assert(chaine != NULL && "Erreur: chaine NULL");
121
122     free(chaine->texte);
123     free(chaine);
124     return;
125 }
126
```

```

1  /* balise.c
2  * Par Guillaume Lahaie et Sylvain Labranche
3  *     LAHG04077707     LABS02059007
4  *
5  * Date de remise: 18 décembre 2012.
6  *
7  * Dernières modifications: 15 décembre 2012.
8  *
9  * Implémentation du fichier en-tête balise.h. La fonction baliseCree
10 * contient la majorité du code du module. D'après les données du TP,
11 * on suppose que la chaîne utilisée pour la fonction baliseCree contient
12 * une balise valide, donc avec les délimiteurs tel qu'attendu (<>), et
13 * le nom et les attributs valides. Nous effectuons une vérification pour
14 * s'assurer que la chaîne passée contient bien un '<' en premier caractère
15 * et un '>' en dernier caractère. Si ce n'est pas le cas, on retourne NULL.
16 *
17 * Nous vérifions quand même que le nom de la balise ne contient pas de
18 * caractères illégaux grâce à la fonction verifierChamp. On pourrait pousser
19 * beaucoup plus loin la validation, par exemple en vérifiant que le nom ne
20 * débute pas par xml, mais c'est hors de l'étendue du TP. De plus, comme le
21 * comportement n'est pas spécifié pour les erreurs de balise, nous affichons
22 * tout simplement un message d'erreur sur la sortie stderr.
23 *
24 * Le comportement des fonctions si l'allocation dynamique de mémoire
25 * échoue n'est pas spécifié dans le TP. Si cela se produit, les fonctions
26 * retournent immédiatement NULL. Cela amène une confusion pour la fonction
27 * baliseLitAttributs, car la fonction pourrait retourner NULL car la balise
28 * ne contient pas d'attributs, ou pour une erreur d'allocation de mémoire.
29 *
30 */
31
32
33 //Chaîne des caractères interdits dans le nom d'une balise xml
34 //selon Wikipedia
35 #define VERIF_NOM "!\"#$%&'()*+,-/;<=>?@[\\]^_`{|}~ , "
36 #define MESS_ERR_VERIF "Erreur lors de la vérification d'un champ\n."
37 #define MESS_ERR_CHAMP "Erreur: nom de balise mal formé.\n"
38
39 #include "balise.h"
40 #include <assert.h>
41 #include <stdlib.h>
42 #include <stdio.h>
43 #include <string.h>
44
45
46 struct balise {
47     Chaîne nom;
48     Chaîne attribut;
49     TypeBalise type;
50 };
51
52 //obtenirType:
53 //Retourne le type de balise de la chaîne, selon l'énumération TypeBalise.
54 TypeBalise obtenirType(char *nom);
55
56 //verifierChamp:
57 //Vérifie qu'un champ d'une balise contient pas de caractères illégaux.
58 //Retourne faux s'il y a une erreur, vrai sinon. Si l'allocation de mémoire
59 //ne fonctionne pas, retourne -1.
60 int verifierChamp(Chaîne chaîne, char *carInterdits);
61
62 //baliseCree:
63 //La fonction obtient le champ balise->type, et les champs balise->nom et
64 //balise->attribut si nécessaire, selon le type de balise. Elle utilise
65 //la fonction strtok pour obtenir le nom et les attributs. Si l'allocation
66 //de mémoire ne fonctionne pas à un certain moment de la fonction, elle
67 //retourne alors NULL.
68 Balise baliseCree(Chaîne nom) {
69

```

```
70 Balise balise;
71 char * jeton;
72 char * attributs;
73 char * valeurNom;
74 int champ;
75
76 if(!(balise = (Balise)malloc(sizeof(struct balise)))) {
77     return NULL;
78 }
79 if(!(valeurNom = chaineValeur(nom))) {
80     free(balise);
81     return NULL;
82 }
83
84 //Vérification qu'il s'agit bien d'une balise et nom d'une chaine
85 //quelconque.
86 if(valeurNom[0] != '<' || valeurNom[chaineLongueur(nom)-1] != '>') {
87     free(valeurNom);
88     free(balise);
89     return NULL;
90 }
91
92 balise->nom = balise->attribut = NULL;
93 balise->type = obtenirType(valeurNom);
94
95 if (balise->type == FIN) {
96     //Trouver le nom à l'aide de jetons.
97     jeton = strtok(&valeurNom[2], ">");
98     if(!(balise->nom = chaineCreeCopie(jeton, strlen(jeton)))) {
99         free(balise);
100         free(valeurNom);
101         return NULL;
102     }
103 } else if (balise->type == DEBUT || balise->type == DEBUTFIN) {
104     jeton = strtok(&valeurNom[1], ">");
105     if(!(balise->nom = chaineCreeCopie(jeton, strlen(jeton)))) {
106         free(balise);
107         free(valeurNom);
108         return NULL;
109     }
110
111     //Traitement des attributs, si nécessaire.
112     if((jeton = strtok(NULL, ">")) != NULL) {
113
114         attributs = (char *)malloc(chaineLongueur(nom)*sizeof(char));
115         if(!attributs) {
116             baliseSupprime(balise);
117             return NULL;
118         }
119         attributs[0] = '\0';
120         while (jeton != NULL) {
121             strcat(attributs, jeton);
122             strcat(attributs, " \0");
123             jeton = strtok(NULL, ">");
124         }
125         if(balise->type == DEBUTFIN) {
126             //Pour enlever le '/' d'attributs
127             attributs[strlen(attributs)-2] = '\0';
128         }
129         free(attributs);
130         if(!(balise->attribut = chaineCreeCopie(attributs, strlen(attributs)))) {
131             baliseSupprime(balise);
132             return NULL;
133         }
134     }
135 }
136
137 //Vérifie que le nom est bien formé, si nécessaire, et qu'il
138 //n'est pas NULL
```

```
139     if(balise->type != COMMENTAIRES && balise->type != DIRECTIVE) {
140
141         assert(balise->nom != NULL && "Nom de balise NULL");
142         champ = verifierChamp(balise->nom, VERIF_NOM);
143         if(champ < 0) {
144             fprintf(stderr, "%s",MESS_ERR_VERIF);
145         } else if (champ == 0) {
146             fprintf(stderr, "%s",MESS_ERR_CHAMP);
147         }
148     }
149
150     free(valeurNom);
151     return balise;
152 } //fin baliseCree
153
154
155 Chaine baliseLitNom(Balise balise) {
156
157     assert(balise != NULL && "balise est NULL.");
158     assert(baliseLitType(balise) != DIRECTIVE && baliseLitType(balise)
159         != COMMENTAIRES && "Mauvais type de balise.");
160     char * temp = chaineValeur(balise->nom);
161     Chaine nomBalise = chaineCreeCopie(temp, chaineLongueur(balise->nom));
162     free(temp);
163     if(nomBalise == NULL) {
164         return NULL;
165     }
166     return nomBalise;
167 }
168
169 TypeBalise baliseLitType(Balise balise) {
170
171     assert(balise != NULL && "balise est NULL.");
172
173     return balise->type;
174 }
175
176 Chaine baliseLitAttributs(Balise balise) {
177
178     assert(balise != NULL && "La balise est nulle.");
179     assert(baliseLitType(balise) != DIRECTIVE && baliseLitType(balise) !=
180         COMMENTAIRES && "Mauvais type de balise.");
181
182     Chaine attribut;
183     char *temp; //Pour libérer l'allocation
184
185     //If sert ici à éviter de passer ue chaine NULL à chaineValeur.
186     if(balise->attribut == NULL) {
187         return NULL;
188     } else {
189         temp = chaineValeur(balise->attribut);
190         if(!temp) {
191             return NULL;
192         }
193         attribut = chaineCreeCopie(temp,
194             chaineLongueur(balise->attribut));
195         free(temp);
196
197         if(!attribut) {
198             return NULL;
199         }
200     }
201     return attribut;
202 }
203
204 void baliseSupprime(Balise balise) {
205
206     assert(balise != NULL && "La balise est nulle.");
207 }
```

```
208     if(balise->nom) {
209         chaineSupprime(balise->nom);
210     }
211     if(balise->attribut) {
212         chaineSupprime(balise->attribut);
213     }
214     free(balise);
215 }
216
217 TypeBalise obtenirType(char * nom) {
218
219     TypeBalise type;
220
221     if(nom[1] == '?') {
222         type = DIRECTIVE;
223     } else if (nom[1] == '!') {
224         type = COMMENTAIRES;
225     } else if (nom[1] == '/') {
226         type = FIN;
227     } else if (nom[strlen(nom)-2] == '/') {
228         type = DEBUTFIN;
229     } else {
230         type = DEBUT;
231     }
232     return type;
233 }
234
235 int verifierChamp(Chaine chaine, char *carInterdits) {
236
237     char * valeurChamp = chaineValeur(chaine);
238     int i, longueurChamp = chaineLongueur(chaine);
239     if(!valeurChamp) {
240         return -1;
241     }
242
243     for(i = 0; i < longueurChamp; i++) {
244         if(strchr(carInterdits, valeurChamp[i]) != NULL) {
245             free(valeurChamp);
246             return 0;
247         }
248     }
249     free(valeurChamp);
250     return 1;
251 }
252 }
```

```
1  /* fichierBalises.c
2  * Par Guillaume Lahaie et Sylvain Labranche
3  *      LAHG04077707      LABS02059007
4  *
5  * Date de remise: 18 décembre 2012
6  *
7  * Dernières modifications: 12 décembre 2012
8  *
9  * Implémentation de fichierBalises.h. Le module permet d'ouvrir un fichier
10 * et d'obtenir la prochaine balise ou le prochain extrait de texte. Comme
11 * pour le module.h, NULL peut être interpréter de différentes façons: soit
12 * il s'agit d'une situation concernant l'ouverture ou la lecture du fichier,
13 * soit il s'agit d'un problème d'allocation de mémoire.
14 *
15 */
16
17 #define BALISE_DEBUT '<'
18 #define BALISE_FIN '>'
19 #include <stdio.h>
20 #include <assert.h>
21 #include <ctype.h>
22 #include <stdlib.h>
23 #include "fichierBalises.h"
24
25 struct fichierBalises {
26     FILE *fic;
27     int position; //dernière position lue dans le fichier.
28 };
29
30
31 //fichierBalisesOuvre:
32 //retourne NULL dans deux cas: soit il est impossible d'ouvrir le
33 //fichier nom_fichier, soit l'allocation de mémoire a échoué.
34 fichierBalises fichierBalisesOuvre(char * nom_fichier) {
35
36     fichierBalises fib = (fichierBalises)malloc(sizeof(struct fichierBalises));
37     if(!fib) {
38         return NULL;
39     }
40
41     if(!(fib->fic = fopen(nom_fichier, "r"))) {
42         free(fib);
43         return NULL;
44     } else {
45         fib->position = 0;
46         return fib;
47     }
48 }
49
50 void fichierBalisesFerme(fichierBalises fichier) {
51
52     assert(fichier != NULL && "fichier est NULL");
53
54     fclose(fichier->fic);
55     free(fichier);
56     return;
57 }
58
59
60 //fichierBalisesLit:
61 //retourne une structure TypeInfo contenant soit la prochaine balise,
62 //soit le prochain texte. Si la balise est de type DIRECTIVE ou COMMENTAIRES,
63 //on lit le prochain extrait du fichier. Pour ce faire, j'utilise un appel
64 //récursif à fichierBalisesLit. La fonction retourne NULL si une
65 //allocation de mémoire échoue.
66 Info fichierBalisesLit(fichierBalises fichier) {
67
68     assert(fichier !=NULL && "fichier est NULL");
69     Info info;
```



```

70     Chaîne prochaine;
71     int prochainCar,
72         verif;           //Caractère de fin de texte ou balise.
73
74     if(!(info = (Info)malloc(sizeof(struct info)))) {
75         return NULL;
76     }
77
78     //On s'assure que le fichier est placé à la dernière position lue.
79     fseek(fichier->fic, fichier->position*sizeof(char), SEEK_SET);
80
81     //On obtient le prochain caractère qui n'est pas un blanc.
82     do {
83         prochainCar = fgetc(fichier->fic);
84         fichier->position++;
85     } while (isspace(prochainCar));
86
87     if(prochainCar == EOF) {
88         free(info);
89         return NULL;
90     } else if (prochainCar == BALISE_DEBUT) {
91         verif = BALISE_FIN;
92         info->type = BALISE;
93     } else {
94         verif = BALISE_DEBUT;
95         info->type = TEXTE;
96     }
97
98     if(!(prochaine = chaineCreeVide())) {
99         free(info);
100        return NULL;
101    }
102
103    chaineAjoute(prochaine, (unsigned char)prochainCar);
104
105    //On remplit la chaîne. Le seul moment où on n'ajoute pas à la chaîne est
106    //lors de la lecture d'un '<' avec un TypeInfo TEXTE.
107    do {
108        prochainCar = fgetc(fichier->fic);
109        if((prochainCar == verif && info->type == BALISE) || prochainCar != verif) {
110            if(!chaineAjoute(prochaine, (unsigned char)prochainCar)) {
111                //problème d'ajout à la chaîne.
112                free(prochaine);
113                free(info);
114                return NULL;
115            }
116            fichier->position++;
117        }
118    } while(prochainCar != verif && prochainCar != EOF);
119
120    if(info->type == TEXTE) {
121        info->contenu.texte = prochaine;
122    } else {
123        //analyse de la balise
124        info->contenu.balise = baliseCree(prochaine);
125        chaineSupprime(prochaine);
126    }
127
128    if(info->type == BALISE && (baliseLitType(info->contenu.balise) == COMMENTAIRES
129        || baliseLitType(info->contenu.balise) == DIRECTIVE)) {
130
131        baliseSupprime(info->contenu.balise);
132        free(info);
133        info = fichierBalisesLit(fichier);
134    }
135    return info;
136
137 }
138

```

```

1  /* extraitStructure.c
2  * Par Guillaume Lahaie et Sylvain Labranche
3  *     LAHG04077707     LABS02059007
4  *
5  * Dernière modification: 15 décembre 2012
6  *
7  * Remise: 18 décembre 2012
8  *
9  *
10 * Utilisation du programme: <nom_programme> <nom_fichier>.
11 *
12 * Le programme affiche les balises ouvrantes d'un fichier XML/HTML.
13 * Il vérifie aussi si le fichier est bien formé. Si le fichier est
14 * mal formé, le programme affiche un message d'erreur et arrête le
15 * traitement du fichier.
16 *
17 * Pour le moment, voici les vérifications faites par le programme:
18 * - Le fichier xml ne commence pas par une balise FIN.
19 * - Le fichier ne contient pas de balises fermées dans le mauvais ordre.
20 * - Toutes les balises DEBUT sont fermées à la fin du fichier.
21 * - Le programme s'assure que le fichier contient au moins une balise
22 *   ouvrante. S'il n'y a pas au moins une balise, ce n'est pas un fichier
23 *   XML ou HTML.
24 *
25 * Pour le moment, on suppose que ces situations ne se produisent pas dans
26 * les fichiers XML:
27 * - Il n'y a pas d'espace au début de la balise: par exemple toutes les
28 *   balises sont du format <texte..., et non < texte... Même chose pour les
29 *   balises de type DÉBUTFIN, il n'y a pas d'espace avant la fin de la
30 *   balise, donc du format .../>, et non .../ >.
31 * - Les noms et attributs sont biens formés, selon les spécifications XML.
32 *   Ils ne contiennent pas de caractères illégaux.
33 * - Les fichiers XML ne sont pas extrêmement longs. Donc, la pile fournie
34 *   ne sera jamais pleine, car nous n'avons pas de façon de vérifier si
35 *   la pile est pleine.
36 *
37 * Le programme a comme comportement, lors d'une erreur d'allocation dynamique
38 * de mémoire, d'afficher un message d'erreur sur stderr et d'arrêter le
39 * traitement.
40 */
41
42 #include <stdio.h>
43 #include <ctype.h>
44 #include <stdlib.h>
45 #include <string.h>
46 #include "pile.h"
47 #include "fichierBalises.h"
48
49 #define IMBRICATION "  "
50 #define ERR_ARG 1      //Erreur avec les arguments.
51 #define ERR_XML 2      //Erreur dans le fichier XML/HTML.
52 #define ERR_FICHIER 3  //Erreur lors de l'ouverture du fichier.
53 #define ERR_MEM 4      //Erreur d'allocation de mémoire
54 #define ERR_MESS_XML "Fichier XML ou HTML mal formé.\n"
55 #define ERR_MESS_FICHIER "Erreur lors de l'ouverture du fichier.\n"
56 #define ERR_MESS_MEM "Erreur lors de l'allocation de mémoire.\n"
57
58 //afficherErreur:
59 //Affiche un message d'erreur approprié sur stderr pour l'erreur noErreur.
60 void afficherErreur(int noErreur, const char *nomFichier);
61
62 //afficherNomBalise:
63 //Affiche le nom de la balise avec le niveau d'imbrication approprié.
64 //Si l'allocation de mémoire dynamique échoue, la fonction retourne 0.
65 int afficherBalise(int imbrication, const Balise balise);
66
67 //obtenirNomBalise:
68 //Retourne une chaîne char * du nom de la balise. Retourne NULL si
69 //l'allocation de mémoire dynamique échoue.

```

```

70 char *obtenirNomBalise(Balise balise);
71
72 //Libère l'allocation de mémoire pour une structure Info.
73 void libereInfo(Info info);
74
75 //Libère la mémoire de la pile lors d'une erreur dans le fichier, si
76 //nécessaire. La pile ne contient que des structures Info de type
77 //Balise, donc c'est le seul cas traité dans la fonction.
78 void viderPile(void);
79
80 int main(int argc, char *argv[]) {
81
82     int erreur = 0,                //Indique si une erreur a été détectée.
83         uneBalise = 0,            //Booléen: vrai si le fichier a au moins une
84                                   //balise ouvrante.
85         resultatCompare;          //Résultat de strcmp.
86     fichierBalises fichier;
87     Info texte,                   //Prochaine partie du texte
88         depile;                   //Structure dépilée.
89     int imbrication = 0;
90     TypeBalise typeBalise;
91     char *compare1, *compare2;    //Pour comparer les noms des balises
92
93     if(argc != 2){
94         afficherErreur(ERR_ARG, argv[0]);
95         return ERR_ARG;
96     }
97
98     fichier = fichierBalisesOuvre(argv[1]);
99
100    if (fichier == NULL) {
101        afficherErreur(ERR_FICHIER, argv[0]);
102        return ERR_FICHIER;
103    }
104
105    while ((texte = fichierBalisesLit(fichier)) != NULL) {
106        if (texte->type == BALISE) {
107            typeBalise = baliseLitType(texte->contenu.balise);
108            uneBalise = 1;
109            if (typeBalise == DEBUT || typeBalise == DEBUTFIN) {
110                if(!(afficherBalise(imbrication, texte->contenu.balise))) {
111                    erreur = ERR_MEM;
112                    break;
113                }
114                if(typeBalise == DEBUT) {
115                    pileEmpiler(texte);
116                    imbrication++;
117                } else {
118                    libereInfo(texte);
119                }
120            } else if (typeBalise == FIN) {
121                //Vérifier si la pile est vide avant de dépiler. Si elle est
122                //vide, c'est une erreur.
123                if(pileTaille() == 0) {
124                    libereInfo(texte);
125                    erreur = ERR_XML;
126                    break;
127                }
128                depile = (Info)pileDepiler();
129
130                compare1 = obtenirNomBalise(depile->contenu.balise);
131                compare2 = obtenirNomBalise(texte->contenu.balise);
132                libereInfo(texte);
133                libereInfo(depile);
134                if(!compare1 && !compare2) {
135                    erreur = ERR_MEM;
136                    break;
137                }
138                resultatCompare = strcmp(compare1, compare2);

```

```

139         free(compare1);
140         free(compare2);
141         if(resultatCompare) {
142             //Erreur - les noms des deux balises sont
143             //différents
144             erreur = ERR_XML;
145             break;
146         } else {
147             //pas d'erreur
148             imbrication--;
149         }
150     }
151 } else {
152     //Info de type texte, on ignore
153     libereInfo(texte);
154 }
155 }
156 erreur = (erreur?erreur:(uneBalise?0:ERR_XML));
157 fichierBalisesFerme(fichier);
158 if(pileTaille() > 0 || erreur) {
159     viderPile();
160     afficherErreur(erreur, argv[0]);
161 }
162 return erreur;
163 }
164
165 void afficherErreur(int noErreur, const char *nomFichier) {
166
167     printf("\n");
168     fprintf(stderr, "%s: Erreur %d\n", nomFichier, noErreur);
169     switch(noErreur) {
170         case ERR_ARG:         fprintf(stderr, "Mauvais arguments.\n");
171                             fprintf(stderr, "Usage: %s <fichier>\n",
172                                 nomFichier);
173                             break;
174         case ERR_FICHIER:     fprintf(stderr, "%s", ERR_MESS_FICHIER);
175                             break;
176         case ERR_XML:         fprintf(stderr, "%s", ERR_MESS_XML);
177                             break;
178         case ERR_MEM:         fprintf(stderr, "%s", ERR_MESS_MEM);
179                             break;
180         default:              fprintf(stderr, "Erreur inconnue.\n");
181                             break;
182     }
183     return;
184 }
185
186 int afficherBalise (int imbrication, const Balise balise) {
187     char * templ; //Pour eviter des fuites de mémoire
188     Chaine temp2;
189     int i;
190     for (i = 0; i < imbrication; i++) {
191         printf("%s", IMBRICATION);
192     }
193
194     if(!(temp2 = baliseLitNom(balise))) {
195         //Erreur - retourne 0
196         return 0;
197     } else {
198         if(!(templ = chaineValeur(temp2))) {
199             return 0;
200         } else {
201             printf("%s\n", templ);
202             chaineSupprime(temp2);
203             free(templ);
204         }
205     }
206 }
207 return 1;

```

```

208 }
209
210 char * obtenirNomBalise(Balise balise) {
211     char * retour;
212     Chaine nom = baliseLitNom(balise);
213     if(!nom) {
214         return NULL;
215     }
216     retour = chaineValeur(nom);
217     chaineSupprime(nom);
218     return retour;
219 }
220
221 void libereInfo(Info info) {
222     if(info->type == TEXTE) {
223         chaineSupprime(info->contenu.texte);
224         free(info);
225     } else {
226         baliseSupprime(info->contenu.balise);
227         free(info);
228     }
229     return;
230 }
231
232 void viderPile(void) {
233
234     Info info;
235     while(pileTaille() != 0) {
236         info = (Info)pileDepiler();
237         baliseSupprime(info->contenu.balise);
238         free(info);
239     }
240     return;
241 }

```

```
1 #Makefile pour TP2
2 #Par Guillaume Lahaie et Sylvain Labranche
3 #    LAHG04077707    LABS02059007
4 #
5 #    Dernières modifications: 18 décembre 2012
6 #    Remise: 18 décembre 2012
7
8 #Variables prédéfinies
9 CC = gcc
10 CFLAGS = -W -Wall
11
12 #Dépendances
13 extraitStructure: extraitStructure.o chaine.o fichierBalises.o balise.o pile.o
14
15 extraitStructure.o: extraitStructure.c fichierBalises.h pile.h
16
17 fichierBalises.o: fichierBalises.c fichierBalises.h balise.h chaine.h
18
19 balise.o: balise.c balise.h chaine.h
20
21 chaine.o: chaine.c chaine.h
22
23 pile.o: pile.h pile.c
24
25 #Je suppose ici que le répertoire contient seulement les fichiers .o
26 #de ce programme. Si le répertoire en contient d'autres, ils seront effacés.
27 clean:
28     rm *.o extraitStructure
29
```