Projet d'Analyse Syntaxique

Université de Bordeaux, année 2014–2015

L3 Informatique et Math-Info.

1 Modalités

Ce projet est à réaliser par groupes de 3 ou 4 étudiant(e)s. Chaque membre doit contribuer de façon significative au travail de programmation, avoir connaissance des options choisies, des difficultés rencontrées, et des solutions retenues. Il/Elle exposera sa contribution individuellement lors d'une soutenance de 25mn (soutenances prévues la semaine du 11 mai).

Contrairement à d'autres projets (comme celui d'informatique théorique 2 ce semestre), on ne fournit pas de code de départ. Le projet sera commencé en TD. Il laisse d'autre part beaucoup d'options et de liberté à chaque groupe. Une partie est imposée, mais vous pourrez ajouter vos propres extensions si le temps vous le permet.

La notation tiendra compte du degré de réalisation du sujet, de la qualité du code (portabilité, lisibilité, documentation), du jeu de tests présenté, et de la qualité du rapport et de la soutenance. Les notes peuvent varier à l'intérieur d'un groupe en cas de travail trop inégal.

En plus du code source (avec jeux de tests), on demande un rapport court présentant le travail réalisé, illustré par des exemples, exposant les problèmes rencontrés et les choix effectués. Idéalement, le rapport sera présenté au format `html`, et généré partiellement par votre logiciel. Le projet doit être transmis à aspp3@labri.fr, avec comme sujet **[Projet ASPP3] Nom1 Nom2 Nom3 Nom4**. Le mail devra contenir une archive au format .tar.gz. Cette archive une fois décompressée contiendra un répertoire nommé Nom1-Nom2-Nom3-Nom4, qui lui-même contiendra les sources et le rapport. Enfin, vous devez utiliser un système de gestion de projets, comme subversion (savanne au CREMI) ou git.

Date **stricte** de remise du projet, code et rapport : 10 mai 2015, 12h00.

2 Objectif et description

Ce projet consiste à réaliser un logiciel permettant

- de **vérifier** la syntaxe et certains points sémantique de programmes,
- de produire une **documentation** de ces programmes et de **présenter** les programmes et la documentation comme un document `html` (à la manière de Doxygen).

On ne demande bien sûr pas un logiciel aussi abouti que Doxygen. Par ailleurs, le projet permettra d'avoir un document principal LaTeX, ce qui n'est pas le fonctionnement de Doxygen.

Ultimement, le programme développé devrait permettre de présenter le rapport du projet lui-même. Le format des documents produits sera du HTML5 et il sera fait appel à plusieurs technologies entourant ce type de document, comme CSS, Javascript (et des bibliothèques telles que jQuery), MathML etc... Le projet doit être écrit en langage **C**, en utilisant **flex** et **bison**.

Le logiciel que vous écrirez prend en entrée un programme à documenter. Ce programme à documenter est composé de plusieurs fichiers :

— un fichier LaTeX optionnel, permettant de décrire et d'organiser la documentation.

— des sources écrits dans le langage à analyser. Ces sources pourront contenir des commentaires pour guider le système de documentation (à nouveau, comme le permet le logiciel Doxygen).

Dans ce projet, les sources à analyser seront eux-mêmes écrits dans les langages de développement : **C**, et si le temps vous le permet, **flex**, **bison** et **JavaScript**. De cette façon, il sera possible d'écrire le rapport en utilisant le logiciel développé dans le projet.

Le projet se déroulera en plusieurs étapes.

— Il s'agira dans un premier temps d'utiliser une grammaire du C pour transformer le code C en document HTML. On ne demande pas d'écrire la grammaire. On peut utiliser pour bison la grammaire C11 de http://www.quut.com/c/ANSI-C-grammar-y.html par exemple. Un source flex est aussi disponible depuis la même page.

Le code devra être bien présenté, c'est-à-dire, proprement indenté et coloré. La présentation du code doit permettre de facilement le parcourir et l'appréhender. Cela passe par l'implémentation d'un certaines fonctionnalités, comme faire apparaître en surbrillance toutes les occurrences d'une variable lorsque le pointeur de la souris passe sur son nom et pouvoir rejoindre l'endroit du code où elle est déclarée lorsque l'on clique dessus.

Ces fonctionnalités sont décrites en Section 3.

— Dans un second temps, il faudra intégrer un langage de documentation, du type LaTeX, afin de pouvoir présenter une documentation structurée en HTML. Le langage inspiré de LaTeX est présenté en Section 4. Il s'agira également de pouvoir faire de liens internes et externes entre les documents. Les parties mathématiques du document seront exportées en MathML. On ne demande pas de traiter l'ensemble des expressions mathématiques, mais un sous-ensemble contenant au moins la section Basic Algebra de http://rypress.com/tutorials/mathml/.

— Enfin, dans un troisième temps, et cela dans le but de pouvoir obtenir l'intégralité du rapport du projet à partir du programme développé, il faudra étendre les fonctionnalités pour pouvoir documenter du code source flex, bison, JavaScript, HTML et CSS. Il n'est pas attendu que vous traitiez tous ces langages, mais que vous montriez votre capacité à réutiliser et à étendre les techniques que vous avez employées jusque-là. Vous pourrez également développer un langage permettant de configurer la présentation de la documentation de code. Vous avez également toute latitude pour développer les fonctionnalités qui vous semblent pertinentes dans le cadre du projet.

3 Présentation du code C

Il s'agit d'utiliser une grammaire du langage C pour transformer le code C en document HTML. On peut utiliser et adapter, au besoin, les sources suivants :

— http://www.quut.com/c/ANSI-C-grammar-l-2011.html.

— http://www.quut.com/c/ANSI-C-grammar-y.html.

On demande d'implémenter des fonctionnalités pour pouvoir facilement le parcourir et le lire. Pour réaliser ces fonctionnalités, on peut ajouter, dans le code HTML généré, de l'information, par

exemple dans des balises de type `<span class="mon_information">...</span>`. Les fonctionnalités demandées sont les suivantes :

1. Présenter le code correctement indenté.
2. Colorer syntaxiquement le code (mots-clés, identificateurs, types, commentaires, chaînes de caractères).
3. Faire apparaître en surbrillance toutes les occurrences d'une variable lorsque le pointeur de la souris passe sur son nom et pouvoir rejoindre l'endroit du code où elle est déclarée lorsque l'on clique dessus ; si la variable possède un commentaire pour la décrire (cf. ci-dessous, syntaxe des commentaires), la description sera également affichée.
4. Faire apparaître le prototype d'une fonction écrite par le programmeur sur un calque lorsque l'on passe le pointeur de la souris sur son nom, et pouvoir rejoindre l'endroit du code où elle est définie lorsque l'on clique dessus. Si la fonction possède un commentaire pour la décrire (cf. ci-dessous, syntaxe des commentaires), la description sera également affichée.
5. Faire apparaître la définition d'un type défini par le programmeur lorsque l'on passe le pointeur de la souris sur son nom, et pouvoir rejoindre l'endroit du code où il est défini lorsque l'on clique dessus. si le type est décrit par des commentaires (par exemple, si les champs d'une structure sont décrits, cf. ci-dessous, syntaxe des commentaires), la description sera également affichée.
6. Permettre de replier et de déplier un bloc de code `{...}`, où bien sûr les accolades ouvrante et fermante se correspondent.
7. Permettre à l'utilisateur de documenter son code dans les commentaires dédiés (du type `/**...*/`, voir Section 4), avec la même syntaxe que celle employée par Doxygen, et présenter cette documentation dans le fichier HTML. On demande d'implémenter les directives `\brief`, `\param`, `\return` de Doxygen.
8. Permettre la création d'étiquettes (toujours dans les commentaires) par une directive spéciale `\label{nom_d_etiquette}`, et la possibilité de mettre des hyperliens revenant sur une étiquette déjà définie, par la commande `\ref{nom_d_etiquette}`. Dans un même fichier, il ne pourra se trouver qu'une seule occurrence d'une étiquette avec un nom donné, par exemple une seule occurrence de l'étiquette `ma_fonction`. La commande `\ref{ma_fonction}` fait donc référence à l'étiquette `ma_fonction` du même fichier.
9. Permettre la possibilité de mettre des hyperliens revenant sur des étiquettes définies dans d'autres fichiers, par la commande `\ref[file=nom_fichier.c]{nom_d_etiquette}`.

4 Présentation de la documentation

Afin de présenter les documents, il nous faut deux types de commentaires. Un premier type de commentaires sera utilisé pour le code lui-même et ne participera pas à la mise en page du document, il pourra toutefois être mis en page avec le code. Un second type de commentaires servira de documentation pour le code. Ce type de commentaires contiendra des instructions de mises en forme inspirées de LaTeX ainsi que des formules mathématiques qui serons mises en forme en MathML. Les commentaires liés à la documentation seront de la forme `/** ...*/` (sur plusieurs lignes), ou de la forme `//*...` (jusqu'à la fin d'une ligne). Le commentaires C usuels délimités par `/* ...*/` et `//...` seront soit effacés soit mis en page avec dans la présentation même du code.

Pour articuler les différentes parties du projet, vous pourrez utiliser des scanners différents ou alors les fonctionnalités liées aux états de `flex`. Cela vous permettra de modifier le comportement du scanner suivant que il doive traiter du code C ou alors de commandes de mises en page.

La suite du sujet présente des fonctionnalités dont la difficulté d'implémentation va croissante. Nous attendons que vous implémentiez au moins les 5 premières parties. Une fois ces parties terminées, vous pourrez implémenter les deux dernières ou choisir de développer des fonctionnalités qui semblent plus intéressantes.

4.1 Paragraphes, environnement, commandes et caractères spéciaux

Pour la mise en pages organisée dans le fichier de type LaTeX, nous adoptons les conventions suivantes, issues de la syntaxe de LaTeX, auxquelles nous ajouterons des environnements ou commandes spécifiques :

1. pour séparer deux mots l'un de l'autre, on peut employer un nombre arbitraires d'espaces entre lesquels peut éventuellement s'insérer un unique saut de ligne,
2. pour séparer deux paragraphes, il sera nécessaire d'utiliser au moins deux sauts de lignes (éventuellement entrecoupés d'espaces).

Des environnements pourront être utilisés pour faire des mises en forme particulières. Ces environnements seront de la forme :

`\begin`{nom_env} [`opt1`]...[`optn`]
...
`\end`{nom_env}

avec entre accolades le nom de l'environnement, entre crochets, une série d'arguments optionnels, et entre les délimiteurs de l'environnement son contenu.

Les commandes seront de la forme suivante :

`\nom_commande`[opt1]...[optp]{arg1}...{argn}

avec entre accolades les arguments obligatoires de la commande et entre crochets, ces arguments optionnels. Il est a noté que commandes et environnements peuvent être enchâssés les uns dans les autres.

Un certains nombres de caractères ou de séquences de caractères sont ainsi réservés pour la syntaxe du langage de mise en forme. Il s'agit de :

\, \\, {, }, &, [,], $, et $$

On pourra imprimer $, &, [,], { et } en les faisant précéder d'un symbole \. Le symbole \ sera quant à lui représenté par `\backslash`.

4.2 Quelques commandes de présentation

On utilisera les mêmes commandes que LaTeX pour contrôler l'affichage de polices, comme la couleur, passer en italique, en gras, souligné. Par exemple, pour passer en gras, on pourra accepter les deux syntaxes suivantes : `\texttt{...}` ou `{\bf ...}`.

4.3 Environements de base

On reprendra également les environnements LaTeX permettant de mettre en page des listes numérotées ou non. Pour les listes numérotées :

```
\begin{enumerate}
\item
\item
\item
\end{enumerate}
```

Pour les listes sans numéro :

```
\begin{itemize}
\item
\item
\item
\end{itemize}
```

On permettra aussi de construire des tableaux :

```
\begin{tabular}{llc}
  Case 1,1&Case 1,2&Case 1,3\\
  Case 2,1&Case 2,2&Case 2,3\\
  ...
\end{tabular}
```

Enfin, on introduira les environnements pourécrire des équations, numérotées ou non :

```
\begin{equation}
  \label{eq:1}
  x = 2
\end{equation}
```

ou

```
\begin{equation*}
  \label{eq:1}
  x = 2
\end{equation*}
```

4.4 Un mécanisme de sections

En utilisant les délimiteurs, `\section{...}`, `\subsection{...}`, `\subsubsection{...}`, ...on souhaite pouvoir délimiter des parties du texte et faire apparaître son organisation hiérarchique. Il faudra également permettre de créer une table des matières à partir des sections définies dans le document.

4.5 Liens internes

Dans les commentaires, nous autoriserons la possibilité de créer des références avec la commande `\label{nom_ref}` qui pourra être citée n'importe où dans le document en utilisant la commande `\ref{nom_ref}{texte_lien}`. Il devra également être possible d'insérer une partie de code comprise entre deux références avec la commande `\citecode{nom_ref1}{nom_ref2}`. Pour des références vers des étiquettes posées dans d'autres fichiers, on utilisera `\citecode[file=fichier.c]{nom_ref1}{nom_ref2}` ou . `\citecode[file=fichier.h]{nom_ref1}{nom_ref2}`.

On pourra aussi faire référence à une ligne de code (celle juste sous un commentaire de la forme `/*...*/` qui contient l'ancre, ou celle sur la même ligne qu'un commentaire de la forme `//...`), cette ligne s'affichera en surbrillance à tous les endroits du document où elle est visible lorsque le pointeur de la souris passera sur le lien.

4.6 Fomules mathématiques

Les délimiteurs les portions de texte de la forme `$...$` permettront d'écrire des formules mathématiques qui seront traduites en MathML. Ces formules s'intégreront au texte, tandis que celles contenues entre double `$`, c'est-à-dire de la forme `$$...$$` ou dans un environement `equation` seront centrées par rapport au texte.

Pour la syntaxe des formules mathématiques et des symboles vous pourrez vous inpirer du chapitre 4 du manuel flshort.pdf. Pour MathML, nous vous conseillons ce tutoriel.

4.7 Extensions

Afin de rendre votre langage de documentation plus souple, vous pourrez ajouter la possibilité pour l'utilisateur de définir de nouvelles commandes ou de nouveaux environements en utilisant un mécanisme similaire à LaTeX, c'est-à-dire les commandes `\newcommand` et `\newenvironment`.