Cours INGI2132 : Contenu du rapport final

Baudouin Le Charlier

9 mai 2009

1 Consignes à respecter

- Dans la rédaction de votre rapport, essayez d'être aussi concis que possible. Vous ne serez pas évalués à la quantité de pages produites, au contraire. Essayez d'être clairs tout en allant à l'essentiel.
- Soignez particulièrement les explications permettant de comprendre la structure des différents composants de votre programme. Évitez tout ce qui s'apparente à du remplissage : paraphrase en français des algorithmes, spécifications vagues. À la place donnez des explications qui montrent clairement le lien entre la théorie du cours et l'application pratique que vous en avez faite.
  Il n'est pas nécessaire que l'entièreté de votre code soit documenté (a posteriori, par obligation...) mais, bien sûr, vous pouvez conserver dans le code les spécifications et autres commentaires qui vous ont servi concrètement pour réaliser votre implémentation.
- Soyez clairs concernant l'état d'achèvement des différentes parties de votre programme. Il est tout à fait acceptable que vous ayez omis de réaliser certaines parties prévues au départ par manque de temps, mais je vous demande de le dire clairement pour éviter de nous faire perdre du temps à deviner ce qui fonctionne et ce qui reste à l'état de "travaux futurs".

2 Plan du rapport

Votre rapport doit être structuré en respectant le plan donné ci-dessous. Chaque point ci-dessous correspond à un chapitre de votre rapport et chaque sous-point correspond à un aspect du problème que vous devez traiter obligatoirement. Mais les sous-points ne sont pas nécessairement des sous-chapitres ou des sections. Vous pouvez structurer le contenu des chapitres de la manière qui vous semble la plus logique.

1. **Première page (couverture) du rapport**
   (a) **Le numéro de votre groupe**
   (b) La liste des membres du groupe et leur rôle dans le groupe (A, B ou C, voir à la fin de ce document)
2. **Introduction**
   (a) Méthode d'analyse syntaxique imposée (LL(1) ou WP)

(b) État d'achèvement du projet

(c) Résumé et plan du rapport

3. **Présentation générale de votre langage**

   (a) Explications concernant les particularités éventuelles du langage, sources d'inspirations de la syntaxe, aspects sémantiques...

   (b) Première version de la syntaxe concrète, sous forme BNF. Cette version est "approximative" et "introductive" dans le sens qu'elle ne précise pas nécessairement le statut exact des terminaux de la grammaire (caractères ou symboles de plusieurs caractères) ni les contraintes sur l'utilisation des espaces, caractères séparateurs de lignes, tabulations, etc. Il sera utile de donner des exemples élaborés de constructions telles que les expressions, les conditions, les instructions, détachés du cadre d'un programme complet.

   (c) Exemples de programmes avec explication du but du programme et, éventuellement, d'aspects particuliers du langage illustrés par le programme.

4. **Analyse lexicale et version rigoureuse de la syntaxe concrète**

   Avant de rédiger ce chapitre, relisez le chapitre sur l'analyse lexicale dans mes notes de cours et, surtout, la section 5.4, aux pages 39 à 45.

   (a) Définir avec précision les textes d'entrée corrects du compilateur.

      i. Préciser quels sont les caractères permis dans un fichier source et, notamment, les hypothèses sur l'"encodage" du fichier source.

      ii. Définir les différentes catégories de "symboles de base" du langage. Pour rappel, pour l'analyseur syntaxique, un programme de votre langage est une suite de tels symboles de base (et non une suite de caractères).

      iii. Ajouter les conventions précises sur l'utilisation des espaces et autres caractères de séparation (et, éventuellement, les commentaires).

   (b) Définir de manière précise et exhaustive un ensemble de symboles terminaux servant de base à l'écriture d'une grammaire BNF constituant une version rigoureuse de la grammaire du point 3(b).

   (c) Définir cette grammaire. À ce stade, elle ne doit pas encore être LL(1) ou WP. Elle doit rester aussi lisible que possible.

   (d) Représentation des symboles lexicaux en Java, c'est-à-dire comment chaque "symbole de base" est représenté et comment la représentation fait le lien avec les terminaux de la grammaire.

   (e) Spécification rigoureuse des méthodes publiques de l'analyseur lexical (celles qui sont utilisées par l'analyseur syntaxique).

   (f) Principe d'implémentation de l'analyseur lexical.

   (g) Description des tests effectués sur l'analyseur lexical. Il faut essayer de rester concis tout en justifiant que les tests couvrent raisonnablement les cas corrects et les erreurs possibles.

5. **Vérificateur de grammaires LL(1) ou WP**

   Dans cette partie, vous faites abstraction de votre langage particulier pour décrire la façon dont vous avez choisi de représenter une grammaire context free en général et dont vous avez implémenté les algorithmes de vérification des grammaires LL(1) ou WP.

   (a) Représentation en Java des grammaires et de leurs éléments.

   (b) Implémentation des algorithmes de vérification : comment vous avez structuré vos classes et quelles sont les méthodes importantes que vous avez réalisées pour la vérification.

   (c) Faire le lien entre la théorie et votre implémentation. Justifier la correction de l'un ou l'autre algorithme "difficile".

   (d) Tests des vérificateurs. Donner des exemples de grammaires correctes et incorrectes, y-compris celles qui vous ont été proposées lors de la "démo" de la semaine 14.

6. **Analyse syntaxique pure et grammaire spécialisée LL(1) ou WP**

   Dans cette partie, on s'intéresse à une version préliminaire de l'analyseur syntaxique qui se contente d'accepter ou de refuser le texte d'entrée, sans construire un arbre syntaxique.

   (a) Donner la grammaire LL(1) ou WP du langage en notation BNF.

   (b) Expliquer les changements qui ont été apportés à la grammaire BNF initiale (du point 4(c)).

   (c) Parler des difficultés rencontrées pour transformer la grammaire.

   (d) Indiquer éventuellement les changements apportés au langage lui-même pour faciliter l'obtention d'une grammaire LL(1) ou WP.

   (e) Code Java de l'analyseur syntaxique "pur", documenté avec une spécification claire et complète, un invariant et des spécifications pour les méthodes utilisées comme sous-problèmes par le code de l'analyseur syntaxique

7. **Traduction du programme en code interne**

   (a) *Création de l'arbre syntaxique du programme*

      i. Structures de données utilisées pour l'arbre syntaxique (classes Java)

      ii. Code Java de l'analyseur syntaxique complété avec les structures de données et les instructions de création de l'arbre syntaxique

   (b) *Traduction du programme en syntaxe abstraite structurée*

      i. Donner un aperçu des règles de traduction de l'arbre syntaxique en un arbre simplifié correspondant à la syntaxe abstraite structurée (classes `CProg` et consorts).

   (c) *Génération du code interne*

   Ici, normalement, il n'y a quasi rien à mettre puisqu'il suffit d'utiliser la fonction `Cprog.translate()`.

i. Difficultés éventuelles rencontrées.

ii. Si vous avez préféré réimplémenter votre propre version de la fonction `Cprog.translate()`, expliquez les principes que vous avez utilisés (en spécifiant avec précision les méthodes de traduction).

iii. Donner quelques exemples de code généré. Choisir des programmes de la démo.

8. **Sémantique opérationnelle**

   (a) Rappeler la syntaxe abstraite non structurée de SLIP (équivalant au code interne).

   (b) Tenir compte des remarques sur le premier rapport préliminaire.

   (c) Voir aussi les instructions du rapport préliminaire.

   (d) Ne pas oublier que cette sémantique est la base de la sémantique de votre langage "concret". Donc, en cas d'ajout de nouvelles caractéristiques à votre langage "concret", qui ne se trouvent pas au niveau de la sémantique opérationnelle de départ, expliquer comment relier ces nouvelles caractéristiques à la sémantique de base ou étendre celle-ci.

9. **Interpréteur du code interne**

   (a) Reprendre et améliorer la description du second rapport préliminaire.

   (b) Faire clairement le lien avec la sémantique opérationnelle.

   (c) Indiquer les déviations éventuelles par rapport à la sémantique opérationnelle. (Idéalement, il ne doit pas y en avoir.)

10. **Mode d'emploi de votre compilateur**

    (a) Tout ce qu'il faut savoir pour utiliser votre compilateur sans se tromper.

    (b) Description complète de la commande à exécuter pour lancer votre compilateur

    (c) Messages d'erreurs ou autres délivrés par le compilateur

11. **Exemples de programmes dans votre langage**

    (a) Un exemple complet comportant

        i. un programme représentatif de toutes les constructions possibles dans votre langage,

        ii. sa traduction en syntaxe abstraite structurée (via `toString()`),

        iii. sa traduction en code interne,

        iv. un exemple d'exécution

    (b) D'autres exemples d'exécution. Utiliser les programmes de la demo plus éventuellement un ou deux autres s'ils présentent des aspects vraiment intéressants et spécifiques de votre langage. Donner :

        i. le code source du programme dans votre syntaxe concrète,

ii. les diagnostics (messages) du compilateur (vous pouvez prévoir quelques "programmes" incorrects syntaxiquement),

iii. des résultats d'exécution.

12. **Conclusion**
    (a) Ce que vous retenez du cours et du projet.
    (b) Joies, remords et regrets

3 Ce qu'il faut remettre et Quand.

1. Une version papier du rapport doit m'être remise pour le mercredi 20 mai 2009 à 14h au plus tard. Une caisse sera placée devant mon bureau à partir du vendredi 15 mai pour l'y déposer.
2. Le code Java de votre projet doit être livré sur iCampus selon la procédure habituelle pour le lundi 18 mai à 12h.

4 Évaluation

1. Comme décidé au départ, le projet sera comptabilisé pour cinquante pour cent des points de la cote finale du cours. Les points obtenus pour le projet seront publié sur le site iCampus du cours au plus tard le 31 mai.
2. Pour rappel également, les cotes seront personnalisés entre les membres des groupes selon la qualité de réalisation des tâches dont ils ont la responsabilité (voir grille ci-dessous). L'évaluation des différentes parties du projet sera basée sur le contenu du rapport, sur les observations faites lors de la "démo" et, accessoirement, sur l'élégance et la correction du code final (de chaque partie). Le rapport doit donc associer un nom d'étudiant à chaque lettre A, B ou C dans le tableau ci-dessous.

Groupe de 3 étudiants			Groupe de 2 étudiants	
A	B	C	A	B
1	2		1	2
3	3	3	3	3
7	4	6	5, 6	4
5	8	9	9	7, 8
10a	10b	10c	10c	10ab

Liste des tâches			
1	Sémantique opérationnelle	6	Vérificateur de grammaire
2	Interpréteur	7	Analyseur syntaxique pur
3	Tout le reste	8	Création de l'arbre syntaxique
4	Analyse lexicale	9	Traduction en code interne
5	Syntaxe LL(1) ou WP	10a	Tests du vérificateur
10b	Tests des analyseurs	10c	Tests de l'interpréteur