Cahier des charges

Projet Compilation

Version 0.1 draft

Préparé par MOREL Célestin, LO El Hadji Malick, ZHOU Alex et UNGARO Cosimo

Télécom Nancy

1er novembre 2023

Table des matières

Historique

Name	Date	Reason for changes	Version
Francois Charoy	29/06/2020	initial	0.1
Francois Charoy	29/10/2020	initial	0.1
Martine Gautier	30/03/2021	orthographe	0.1.1
Brigitte Wrobel-Dautcourt	22/05/2021	ponctuation, typo, orthographe, grammaire...	0.1.2
Martine Gautier	07/07/2021	ponctuation, typo, orthographe, grammaire, ...	0.1.3

1 Introduction

Ce cahier des charges est réalisé dans le cadre du projet de compilation 1 (PCL 1) de Telecom Nancy

1.1 Objectif

L'objectif du projet de compilation est de construire pas à pas un compilateur pour le langage canAda. Le projet PCL1 s'attachera à la "première moitié" de cette construction : les analyseurs lexicaux et syntaxiques. Le projet comprend également la définition complète de la grammaire de canAda, ainsi que la construction de l'arbre abstrait et sa visualisation

1.2 Audience et sections importantes

Les lecteurs potentiels de ce document sont les développeurs (élèves) d'une part et le(s) professeur(s) encadrant(s) d'autre part, qui doi(ven)t pouvoir contrôler que le projet répond bien aux attentes.

1.3 Portée du projet

Le compilateur doit donc prendre un texte en entrée, et en faire les analyses lexicales et syntaxiques. Il. en dressera également l'arbre abstrait, qui devra pouvoir être visualisé par l'utilisateur. L'analyse lexicale devra donc comprendre la découpe du texte en lexèmes et l'établissement de tokens. L'analyse syntaxique devra permettre de vérifier que le code est correct en "déroulant" la grammaire et en établissant l'arbre abstrait. En cas d'erreur, un message le plus précis possible sera établi et l'analyse continuera.

1.4 Conventions

<Décrivez les conventions de nommage et de structure utilisées au long du document ; indiquez de quelle façon ces conventions peuvent aider le lecteur.>

1.5 Références

Le Fascicule 0 contient les règles de la gestion des absences. https://intranext.telecomnancy.univ-lorraine.fr/xwiki/bin/download/FISE/WebHome/FASCICULE-0A_2019-2020%20_FISE.pdf

2 Description

2.1 Perspective du produit

Le projet PCL1 est organisé dans le cadre du module de "Traduction des langages" de deuxième année de Télécom Nancy. L'objectif est de construire la "première moitié" du compilateur, c'est-à-dire la traduction du texte original en lexème, son analyse grammaticale et en dresser l'arbre abstrait. Ainsi :
Le compilateur doit prendre un texte en entrée, et en faire les analyses lexicales et syntaxiques. Il en dressera également l'arbre abstrait, qui devra pouvoir être visualisé par l'utilisateur. L'analyse lexicale devra donc comprendre la découpe du texte en lexèmes et l'établissement de tokens. L'analyse syntaxique devra permettre de vérifier que le code est correct en "déroulant" la grammaire et en établissant l'arbre abstrait. En cas d'erreur, un message le plus précis possible sera établi et l'analyse continuera.

2.2 Principales fonctionnalités

Le compilateur devra donc :
— Prendre un texte en entrée
— Le découper en lexèmes
— Organiser ceux-ci en token
— Vérifier que le texte respecte la grammaire
⇒ Le cas échéant, renvoyer un message d'erreur le plus clair possible
— Construire l'arbre abstrait correspondant au programme

2.3 Présentation des utilisateurs

2.4 Environnement opérationnel

Le compilateur devra pouvoir être appelé depuis la ligne de commande sur des distributions Linux ou Mac.

2.5 Contraintes de conception et d'implantation

Le développement sera effectué en Python.
L'organisation du dossier de travail sera effectué selon un plan défini par l'équipe de développement et qui s'inspirera des recommandations disponibles sur :
— Aranacorp

— Guide to Python

2.6 Documentation

<Décrire le contenu, le mode de fourniture et les standards. >

2.7 Hypothèses/Dépendances

<Détailler toutes les hypothèses qui peuvent potentiellement impacter la spécification technique, incluant des facteurs externes >

3 Besoins fonctionnels

<Pour chaque besoin fonctionnel, donnez une description détaillée de son fonctionnement permettant de comprendre plus précisément son fonctionnement. La numérotation doit correspondre aux besoins utilisateurs.>

Requirement 1
Description : Recensement des identificateurs
Priorité : **high**
Détails, contraintes et conséquences

1. Un compilateur se doit de pouvoir transformer n'importe quel texte du langage en un programme interprétable. Il faudra pour cela dresser au préalable une liste exhaustives des éléments pouvant être rencontrés dans un programme Ada : mots-clés, ponctuation...

Requirement 2
Description : Découpage du texte en lexèmes
Priorité : **high**
Détails, contraintes et conséquences

1. Le texte doit être découpé en lexèmes
2. Pour cela on effectue une lecture caractère à caractère
3. À chaque nouveau caractère, on se demande s'il s'agit de la fin du lexème (par exemple : si le lexème correspond à un mot-clé, si l'on est au niveau d'un espace, d'un retour chariot...)
4. Si tel est le cas on transforme le lexèmes en token et on passe au suivant.
   Sinon on passe au caractère suivant.

Requirement 3
Description : Transformation des lexèmes en token
Priorité : **high**
Détails, contraintes et conséquences

1. On parcourt la liste des lexèmes
2. Pour chaque lexème on créé un token $<clé,valeur>$ où *clé* correspond au type de lexème et *valeur* comprend (éventuellement) la valeur du lexème

Requirement 4
Description : Découpage du texte en lignes

Priorité : **high**
Détails, contraintes et conséquences

1. Lorsque l'on parcourt le texte, on compte le numéro de ligne
2. En cas d'erreur, on doit être capable d'indiquer précisemment à quelle ligne se situe l'erreur

Requirement 5
Description : Retour erreur
Priorité : **high**
Détails, contraintes et conséquences

1. En cas d'erreur lexical, le programme doit renvoyer un message d'erreur le plus clair possible. Celui-ci devra notamment indiquer le numéro de ligne, et si possible indiquer de quel mot provient l'erreur
2. (*Optionnel*) L'analex pourra eventuellement tenter de proposer une correction avec le mot connu le plus proche en utilisant la distance de Hamming

Requirement 6
Description : Le compilateur doit pouvoir détecter les espaces et les commentaires
Priorité : **high**
Détails, contraintes et conséquences

1. Les commentaires et tabulations doivent être détectés et supprimés car ils ne sont pas pertinents pour la structure du programme
2. Les espaces doivent aussi être suppprimés, sauf s'ils sont nécessaires pour séparer les lexèmes

Requirement 7
Description : Le compilateur doit pouvoir construire l'arbre syntaxique à partir des tokens
Priorité : **high**
Détails, contraintes et conséquences

1.

4 Besoin des interfaces externes

4.1 Interfaces Utilisateurs

<Décrivez les interfaces utilisateur principales en incluant éventuellement des guides de conception, des contraintes et quelques exemples d'écrans si nécessaire.>

4.2 Interfaces Matérielles

<Décrivez les produits et les caractéristiques des appareils utilisés pour l'application ainsi que les interfaces de communication.>

4.3 Interfaces Logicielles

< Décrivez les logiciels ou applications, les composants ainsi que leur version, les bases de données et les outils qui seront utilisés avec l'application, quels sont les protocoles utilisés et les données échangées. >

4.4 Interfaces de communication

< Décrivez interfaces et les standards de communication utilisés ainsi que les standards de sécurité utilisés pour les échanges de données et le chiffrage si nécessaire. >

5 Besoins non fonctionnels

5.1 Performance

Requirement 1
Description : Le temps de réponse pour les actions de l'utilisateur doit être inférieur à 100ms dans 99% des cas.
Le système doit être réactif et ne pas ralentir le travail de l'utilisateur. Des tests de performance seront mis en place pour vérifier ce besoin ainsi qu'une solution de monitoring pour vérifier que la performance attendue est bien atteinte.

Requirement 2
Description : Performance
L'analyseur lexical doit fonctionner de manière efficiente pour analyser rapidement de grandes quantités de code.
La compilation d'un code ne doit pas dépasser

5.2 Sécurité

Requirement 3
Description : Le système doit fournir un moyen d'authentification à 2 facteurs

5.3 Qualité logicielle

5.4 Besoins liés au domaine

6 Appendices

6.1 Appendice A

6.2 Appendice B

Bibliographie