Licenciatura em Engenharia Informática

Compiladores

Paulo Pereira
Aluno de 2º ano, 70560

Filipe Carvalho
Aluno de 2º ano, 70855

Vasco Ribeiro
Aluno de 2º ano, 70855

Pedro Mourão
Aluno de 2ºano, 70782

2021
Janeiro

Resumo

Neste trabalho prático iremos abordar o funcionamento de um reconhecedor assim como as fases de processamento de uma linguagem e de que forma foram implementadas essas fases de maneira que obtivessemos um reconhecedor funcional.
Conseguimos atingir os resultados pretendidos e reconhecer o código fonte que nos foi apresentado previamente como objetivo.

Contents

Chapter 1

Introdução

O problema que nos é apresentado é reconhecer um determinado código fonte através de um analisador léxico e sintático desenvolvido por nós.
O nosso principal objetivo é reconhecer o código-fonte em questão com sucesso.
Este documento irá ficar organizado por 7 tópicos:
• Introdução;
• Análise e especificação;
• Concepção/desenho da resolução;
• Codificação e testes;
• Conclusão;
• Anexos de ficheiros fonte;
• Equipa;

Chapter 2

Análise e especificação

2.1 Descrição informal do problema

Para a geração de um analisador léxico utilizamos o programa lex e para a geração de um analisador sintático utilizamos o programa yacc. O lex e o yacc são o complemento um do outro. O lex apenas através do mapeamento de expressões regulares em blocos de código e da leitura de tokens não consegue analisar sintáticamente uma entrada, necessitando do yacc. Para que o yacc consiga ler uma simples entrada de dados, requer uma série de tokens (enviados pelo Lex). De uma forma geral, o lex age como um pré-processador do yacc.

2.2 Especificação dos requisitos

2.2.1 Dados

Disponibilizamos do Lex e do Yacc para a criação deste analisador de código.

2.2.2 Pedidos

Queremos criar um analisador capaz de ler e verificar se um código em c está bem escrito permitindo o bom funcionamento do mesmo.

2.2.3 Relações

A partir do lex e do yacc somos capazes de criar expressões regulares e verificar se o código que estamos a analisar está correto, com o lex iremos fazer a análise léxica e descobrir os "tokens" subdividindo-os de forma a serem enviados para o yacc que irá fazer a sua análise sintática e verificar se os tokens tem a sintaxe totalmente correta.

Chapter 3

Concepção/desenho da resolução

3.1 Estruturas de dados

No lex, dividimos o documento em 3 secções, secção de definição, secção de regras e secção código c. Na secção de definição do lex apresentamos os includes e as definições de funções necessárias; A secção de regras do lex foi dividida em duas colunas, á esquerda apresentamos as expressões regulares, e á direita as respetivas ações; Por fim, na secção de código escrevemos as respetivas subrotinas definidas.
No yacc é feita uma divisão similar ao lex, alterando apenas a secção de regras onde definimos os tokens e a gramática.

3.2 Algoritmos

Usamos uma gramática deterministica, ou seja, top-down parser sem backtracking tornando o algoritmo mais eficiente.Em termos de recursividade, a gramática foi desenvolvida com recursividade á direita evitando "loops" de símbolos não-terminais.
A gramática desenvolvida é uma GIC (Gramática Independente de Contexto) pois possui apenas símbolos não terminais á esquerda da produção e símbolos terminais e não terminais á direita da produção.

Chapter 4

Codificação e testes

O nosso código efetua a leitura linha a linha daquilo que utilizador vai introduzindo, começando por verificar se a linha em questão é um include e de seguida alguma função (por exemplo printf). Caso isto não aconteça acaba pela confirmação que o código em questão é ou devia ser uma constante, acabando sempre a maioria das vezes com o token de um NUMBER ou CHAR. Depois da descoberta do token verificamos se este se encontra bem estruturado ou não.Caso haja algum erro o programa devolve o erro em questão e informa o que era necessário para a boa estruturação do mesmo. Desta forma ,a nosso ver , a gramática que nós construímos funciona do tipo include − >funções − >constantes − >id que é uma forma boa de verificar a construção de um código.

4.1 Alternativas, Decisões e Problemas de Implementação

Sendo que nunca tinhamos realizado qualquer trabalho utilizando lex e yacc tivemos inicialmente dificuldades em entender o funcionamento das produções criadas no ficheiro '.y'. Devido a estes problemas acabamos por optar pela criação de um reconhecedor que verifica o código linha a linha. Apesar de o reconhecedor apenas informar da existência de erros na última linha, tomamos a decisão de utilizar um algoritmo que nos indica as possíveis correções para o erro encontrado. A quantidade de warnings existentes após a compilação do yacc, foi um problema que não foi resolvido por completo.

4.2 Testes realizados e Resultados

Inicialmente testamos na condição da função 'for' o correto preenchimento de operadores entre números/caracteres.Tal como esperado o erro aconteceu quando introduzimos o símbolo & (sozinho) dentro da condição.Este erro não é só verificado no for, mas também nas outras funções e condições.
Na segunda figura observamos um erro depois da função printf devido á falta de um ';'. Por fim testamos o código disponibilizado no relatório e efetuamos o reconhecimento com sucesso.

```
Codigo:
int a;
int b;
for(i=0;i&7;i++){
Erro de parsing na ultima linha
Erro:syntax error, unexpected $undefined, expecting OPERADORES_COMPOSTOS or '<' or '>'
```

Figure 4.1: Teste 1

```
Codigo:
int a;
int b;

while(a<5){

a=a+1;

}

printf("Ola");

printf(a)
Erro de parsing na ultima linha
Erro:syntax error, unexpected EOL, expecting PTV
```

Figure 4.2: Teste 2

```
Codigo:
int main ( ) {
int num;
float real;
printf(”Vou fazer um ciclo”);
for(i=1;i<10;i=i+1) {
printf(i);
}

int  var=1 ;
float  x=14.5 ;

}
```

Figure 4.3: Teste 3

Chapter 5

Conclusão

Este documento foi dividio em vários tópicos de maneira que houvesse uma estrutura organizada e uma ideia clara sobre a forma que foi implementado o reconhecedor.
O reconhecedor desenvolvido, possui a capacidade de receber e ler caracteres vindos de um determinado código-fonte e transformá-los em tokens, onde irá reconhecer esses tokens e mapear em blocos de código.
Este reconhecedor analisa o código linha-a-linha e por fim mostra a localização do erro encontrado.
Num futuro próximo, poderíamos criar uma versão mais completa deste reconhecedor (Análise semântica,geração de código,otimização e geração de código final).
Conseguimos atingir os resultados pretendidos, analisando com sucesso o código-fonte disponibilizado pela docente.

Chapter 6

Anexos de ficheiros fonte

Ficheiros:

- AnalisadorLex.l
- AnalisadorYacc.y
- y.tab.c
- y.tab.h
- lex.yy.c

Chapter 7

Equipa

Paulo Pereira Aluno de 2º ano, 70560
Filipe Carvalho Aluno de 2º ano, 70855
Vasco Ribeiro Aluno de 2º ano, 70638
Pedro Mourão Aluno de 2ºano, 70782
Este projeto dividiu-se em diferentes tarefas para cada um dos elementos do grupo:
Lex: Filipe Carvalho;
Yacc: Paulo Pereira,Pedro Mourão;
Látex: Vasco Ribeiro;
Eventualmente existiu entreajuda entre os elementos do grupo pois para que pudessemos avançar mais rápido no projeto.