Processamento de Linguagens (3º ano de Curso)

Trabalho Prático N 2

Relatório de Desenvolvimento

Filipe Costa Oliveira
a57816

30 de Maio de 2016

Conteúdo

Capítulo 1

Introdução

O presente trabalho prático foca-se no desenvolvimento de um compilador, que tem como fonte uma linguagem de alto nível (também esta desenvolvida especificamente para este trabalho prático) , gerando código para uma máquina de stack virtual.
Um compilador comum divide o processo de tradução em várias fases. Para o propósito específico desta unidade curricular iremos focar-nos nas seguintes:

- 1ª Fase de tradução – Análise Léxica, que agrupa sequências de caracteres em tokens. Recorreremos nesta fase à definição das expressões regulares que permitem definir os tokens.
- 2ª Fase de tradução – Reconhecimento(Parsing) da estrutura gramatical do programa, através do agrupamento dos tokens em produções. Recorreremos à definição de uma gramática independente de contexto por forma a definir as estruturas de programa válidas a reconhecer pelo parser. Denote que juntamente com o parsing é realizada a análise semântica, assim como a geração de código associando regras às produções anteriormente descritas.

Começaremos portanto por definir uma linguagem de programação imperativa simples, que chamaremos Algebra. A Algebra permitirá:

- declarar e manusear variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.
- declarar e manusear variáveis estruturadas do tipo array (a 1 ou 2 dimensões) de inteiros, em relação aos quais é apenas permitida a operação de indexação. Aos arrays de duas dimensões, por se tratar de uma linguagem algébrica, chamaremos matrizes, dada a fácil associação a este tipo de variável à sua definição análoga da álgebra linear.
- efetuar instruções algorítmicas básicas como a atribuição de expressões a variáveis.
- ler do standard input e escrever no standard output.
- efetuar instruções para controlo do fluxo de execução – condicional e cíclica – que possam ser aninhadas.
- definir e invocar subprogramas sem parâmetros mas que possam retornar um resultado atómico.

Na nossa linguagem de programação por questões de estruturação e percepção, teremos como premissa que as varíaveis deverão ser declaradas no início do programa, não podendo haver re-declarações, nem utilizações sem declaração prévia. Não será permitida a declaração e associação de um valor inteiro na mesma instrução. Achamos essa solução pouco elegante. Assim, todas as variáveis terão o valor zero após a declaração.
Será desenvolvido portanto o compilador para a Algebra, com base na GIC criada acima e recurso ao Gerador Yacc/ Flex. O compilador de Algebra irá gerar pseudo-código, Assembly da Máquina Virtual VM cuja documentação completa está disponibilizada em anexo.

Por forma a facilitar e validar o trabalho, à medida que as funcionalidades forem descritas serão apensados exemplos ilustrativos.
Por fim, serão apresentados um conjunto de testes mais complexos(programas-fonte diversos e respectivo código produzido), que tentam testar de uma forma mais alargadas as funcionalidades da Algebra, sendo estes:

- lidos 3 números, escrever o maior deles.
- ler N (valor dado) números e calcular e imprimir o seu somatório.
- contar e imprimir os números pares de uma sequência de N números dados.
- ler e armazenar os elementos de um vetor de comprimento N, imprimido os valores por ordem crescente após fazer a ordenação do array por trocas diretas.
- ler e armazenar os elementos de uma matriz NxM, calculando e imprimindo de seguida a média e máximo dessa matriz.
- invocar e usar num programa uma função.

Capítulo 2

Concepção da Linguagem Algebra

2.1 Concepção/desenho da Resolução

Comecemos por descrever as funcionalidades da linguagem Algebra. Tal como descrito anteriormente a Algebra permitirá:

- declarar e manusear variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.
- declarar e manusear variáveis estruturadas do tipo array (a 1 ou 2 dimensões) de inteiros, em relação aos quais é apenas permitida a operação de indexação. Aos arrays de duas dimensões, por se tratar de uma linguagem algébrica, chamaremos matrizes, dada a fácil associação a este tipo de variável à sua definição análoga da álgebra linear.
- efetuar instruções algorítmicas básicas como a atribuição de expressões a variáveis.
- ler do standard input e escrever no standard output.
- efetuar instruções para controlo do fluxo de execução – condicional e cíclica – que possam ser aninhadas.
- definir e invocar subprogramas sem parâmetros mas que possam retornar um resultado atómico.

2.1.1 Uma introdução às variáveis

Temos então que as variáveis poderão ser de 3 tipos: inteiros simples, arrays de inteiros, e matrizes de inteiros. Dessa premissa sabemos à partida que o código gerado para a nossa máquina virtual terá que suportar o tipo de variável inteiro. Sabemos ainda que os tipos de dados mais complexos (arrays e matrizes) apenas é permita a realização de operações de indexação.

Na nossa linguagem de programação por questões de estruturação e percepção, teremos como premissa que as variáveis deverão ser declaradas no início do programa, não podendo haver re-declarações, nem utilizações sem declaração prévia. Não será permitida a declaração e associação de um valor inteiro na mesma instrução(à lá C). Achamos essa solução pouco elegante. Assim, todas as variáveis terão o valor zero após a declaração.

Podemos então aceitar como exemplo as declarações do tipo:

```
1 int a;
2 int auxiliar_1;
3 int array_1d[10];
4 int exemplo_2d[40,2];
```

Dado que toda a porção de código de alto nível julgamos essencial a possibilidade de existência de comentários. Atente no exemplo anterior agora com comentários que facilitam a percepção:

```
// variaveis do tipo inteiro
int a;
int auxiliar_1;
// arrays de inteiros
int array_1d[10];
// matrizes
int exemplo_2d[40,2];
```

Tal como poderá confirmar pela última declaração do exemplo anterior a declaração do tamanho das matrizes é feita da seguinte forma: **nome_variavel[nºlinhas,nºcolunas]**.

A forma de armazenamento e acesso às variáveis será posteriormente discutida nas secções seguintes deste relatório. Neste momento temos especial interesse na especificação da estrutura correcta de programas da nossa linguagem.

Expressões Regulares e acções resultantes

Podemos desde já enumerar as expressões regulares necessárias à produção dos tokens que permitam à GIC o agrupamento dos tokens em produções:

```
%{

%}

letter      [a-zA-Z]
digit       [0-9]
ignore      [\ \t\r\n]

%option yylineno

%%

[\%\,\{\}\+\-\(\)\=\>\<\!\;\/\*\[\]\|\&\_]      { return(yytext[0]); }
int                                             { return (TYPE_INT); }

{letter}({letter}|{digit}|\_)*                  { yylval.var = strdup(yytext); return(id); }
{digit}+                                        { yylval.qt = atoi(yytext); return(num); }
\"[^"]+\"                                       { yylval.var = strdup(yytext); return(string); }
\/\/[^\n]*                                      { printf("%s\n",yytext);  }
{ignore}                                        { ; }

%%

int yywrap(){
  return(1);
}
```

Como é perceptível pela expressão regular correspondente, vulgo {letter}({letter}|{digit}|_)*, as variáveis do tipo inteiro terão sempre de ser iniciadas por uma letra (maiúscula ou minúscula), sendo que como segundo caracter poderão ter um número, letra, ou _.

Um olhar mais atento às expressão regulares definidas até ao momento permitem-nos identificar um outro tipo de variável – **string**, até ao momento não apresentada. Denote que na nossa linguagem não é permitido realizar operações sobre strings (como concatenação ou comparação), apenas será permitir ler e escrever

Produções da GIC

Capítulo 3

Conclusão

Relativamente ao estado final do projecto acredito que foram cumpridos todos os requisitos, sendo que o segundo exercício foi sem dúvida o mais desafiante dada a enorme quantidade de dados e o tipo de dados em si a serem analisados. Reconhecer por si só quais as sequências de caracteres válidas foi um desafio.

Naturalmente que a partir da alínea 2.2.b a alínea 2.2.c foi de extrema facilidade, uma vez que todo o trabalho de análise já estava realizado.

Foi ainda tido em conta a possiblidade de recuperar de erros de leitura na alínea 2.2.b o que facilitou o input correct de dados e posterior tratamento. O recurso à biblioteca Glib, recomendada pelo professor José João num aula laboratorial permitiu-me ambientar ainda mais com código desenvolvido por terceiros e sua correcta análise e integração nos meus projectos.

Faço um balanço positivo do trabalho prático, pois, apesar de ser extremamente "time consuming"retirei muito conhecimento no que da análise de dados e processamento de linguagens diz respeito.

Apêndice A

Código do Programa da alínea 1a

Apêndice B

Código do Programa da alínea 2a

Apêndice C

Código do Programa da alínea 2b

Apêndice D

Código do Programa da alínea 3a