Universidade do Minho

Licenciatura em Engenharia Informática

Processamento de Linguagens

Report 2007

Realizado por:
Daniel Carvalho a61008
Serafim Pinto a61056
Daniel Araújo a61058

3 de Junho de 2013

Resumo

Este relatório descreve o processo de desenvolvimento e o resultado obtido, como consequência da resolução do enunciado do trabalho prático número dois da Unidade Curricular Processamento de Linguagens. O enunciado aqui resolvido é sobre o "Report 2007: vamos escrever relatórios". Este documento trata de analisar e explicar os objectivos deste segundo trabalho. Assim, falaremos das decisões tomadas, os principais obstáculos encontrados e os resultados obtidos. Para além dos conhecimentos que já eram necessários para o primeiro projecto, foi necessário demonstrar conhecimentos sobre ferramentas *Flex* e *Yacc*, conhecimentos estes que se mostraram muito úteis e interessantes.

Conteúdo

1 Introdução

Pretende-se com este segundo trabalho usar o conhecimento adquirido nas aulas teóricas e práticas, sobretudo o uso das ferramentas Flex - *T*he Fast Lexical Analyzer e *Yet Another Compiler Compiler* que estão na base do desenvolvimento do nosso trabalho.

Assim sendo, temo como finalidade o desenvolvimento de um compilador que aceitará relatórios escritos numa determinada linguagem e gerará a respectiva versão HTML e gerar também uma versão em LaTeX.

À primeira vista, este trabalho pareceu-nos bastante interessante uma vez que, teremos que desenvolver um compilador que gera outra linguagem e estudar o seu funcionamento, e de todos os elementos do grupo esta é a primeira vez que qualquer um de nós fez algo parecido. Para além disto, as ferramentas que devemos utilizar (*Flex e Yacc*) parecem-nos bastantes poderosas e com muitas potencialidades e não temos dúvidas que o facto de as sabermos usar se irá revelar muito benéfico.

2 Descrição das ferramentas utilizadas

Para o desenvolvimento deste projecto necessitamos de utilizar ferramentas diferentes, sendo elas o **Flex**, o **Yacc**, a linguagem C e a biblioteca glib.

Assim, os dois primeiros componentes, como já foi referido, tratam-se de um analisador léxico e de um analisador sintáctico, respectivamente. Estes dois são utilizados, maioritariamente, em conjunto, isto é, o Yacc usa uma gramática formal de modo a analisar sintacticamente cada entrada enquanto que o Flex consegue fazer a distinção das expressões regulares. Estas expressões regulares distinguidas pelo Flex são reconhecidos como *tokens* pelo Yacc. O GCC, popular compilador da linguagem de programação C, é utilizado de modo a combinar o analisador léxico com o analisador sintáctico. A biblioteca glib do C, foi usada nas nossas estruturas de dados como se poderá ver mais adiante.

Falta fazer uma pequena referência ao uso de LaTeXe HTML que foram gerados pelo compilador, com o objectivo de poder obter um relatório escrito nestas duas linguagens.

3 Descrição da Linguagem desenvolvida

A nossa linguagem é baseada no esboço dado no enunciado, apenas com algumas coisas adicionais sugeridas pelo professor.
O nosso relatório estará dividido em três partes.

```
Report --> BREPORT FrontMatter Body BackMatter EREPORT
```

No que diz respeito ao **FrontMatter** acrescentamos todos os não-terminais propostos no enunciado: *Subtitle, Institution, Keywords, Aknowledgements, Table of contents, Table of figures e Table of tables.*

```
FrontMatter --> BFM Title SubTitle? Authors Date Institution? Keywords?
                Abstract Aknowledgements? Toc? Lof? Lot? EFM

Title --> BTITLE texto ETITLE
SubTitle --> BSUBTITLE texto ESUBTITLE
           | &
Authors --> Authors Author
          | Author
Author --> BAUTHOR Name Nident? Email? Url? Affilliation? EAUTHOR
Name --> BNAME texto ENAME
```

- Subtitle Apenas texto normal.
- Institution Apenas texto normal.
- Keywords Uma lista de palavras-chave do documento
- Aknowledgements Uma lista de parágrafos tal como o *Abstarct.*
- Table of contents É um inteiro que indica se o relatório conterá os índices respectivos.

Por exemplo, cada vez que quisermos adicionar um autor ao nosso relatório, basta escrever BAUTHOR.

```
typedef struct sAuthor
{
    char* name;
char* number;
char* mail;
} Author;
```

E de seguida adicionar os campos acima definidos na estrutura. Na nossa linguagem apenas é obrigatório sempre que se inicia o BAUHTOR, adicionar o nome(BNAME João ENAME). Isto porque nas regras do Yacc assim o definimos:

```
Author: BEGIN_AUTHOR Name Number Mail END_AUTHOR    {g_array_append_val(r.frontmatter.au

Authors: Author
    | Authors Author ;

Name: BEGIN_NAME text END_NAME  { a.name = strdup($2);};

Number: BEGIN_NUMBER text END_NUMBER    { a.number = strdup($2);}
    | ;

Mail: BEGIN_MAIL text END_MAIL  { a.mail = strdup($2);}
|;
```

Como podemos os campos NUMBER e MAIL, poderão não existir. O GArray, é usado através da glib para sempre for feito um BAUTOR seguido de um EAUTHOR, inserir no nosso array um author.

Então ficaria no ficheiro *input*:

```
BAUTHOR
BNAME
Mike Travis
ENAME
BNUMBER
77777
ENUMBER
BMAIL
mttttt@gmail.com
EMAIL
EAUTHOR
```

A seguir, a parte principal do relatório que conterá o nosso Body:

```
Body --> BBODY ChapterList EBODY
ChapterList --> ChapterList Chapter
              | Chapter Chapter --> BCHAP Title ElemList ECHAP
```

```
Section --> BSEC Title ElemListSec ESEC

ElemList --> ElemList Elem
           | Elem
```

```
Elem --> Paragraph
       | Float
       | List
       | CodeBlock
       | Section
       | Summary
```

O Body será constituído por uma lista de Chapters, e estes terão um Title e uma ElemList. Estes elementos aqui presentes como podemos ver na imagem, podem ser várias coisas: uma imagem, uma tabela, uma section, etc. O exemplo de do parágrafo:

```
typedef union uParagraphElem
{
    char* text;
Footnote footnote;
Ref ref;
Xref xref;
Citref citref;
Iterm iterm;
Bold bold;
Italic italic;
Underline underline;
InlineCode inlineCode;
Acronym acronym;
} Paragraph_Elem;
```

Um parágrafo é uma lista de vários elementos, desde o texto até palavras em itálico ou negrito por exemplo. Para começar um parágrafo basta escrever "BPARA"e lá dentro inserir qualquer um dos elementos apresentados acima, iniciados respectivamente pela etiqueta. Outro tipo de elementos dentro do body podem ser as figuras e tabelas que serão iniciadas como BFIGURE, e BTABLE.

Resumindo, o nossa linguagem é o esboço feito pelo professor, com mais algumas modificações que a tornam mais completa. De certo modo achamos que o que estava no enunciado era suficiente para tornar um relatório completo e usável. A nosso ver, não havia nada a faltar para acrescentar melhorias na linguagem.

4 Especificação FLEX

Quanto ao prodecimento a desenvolver para tratar da análise lexical, achámos necessário definir quatro estados como fizemos anteriormente no primeiro trabalho prático:

- keywords

```
<keywords>{
    [ \t\n]+        {;}
    [a-zA-Z]+       {yylval.str= strdup(yytext); return TEXT;}
    :{keywords}             {BEGIN(INITIAL); return KEYWORDS; }
    }
```

- table

```
<table>{
    [ \t\n]+            {;}
    {begin_row}         {return BEGIN_ROW;}
    {end_row}           {return END_ROW;}
    {begin_cell}        {return BEGIN_CELL;}
    {end_cell}          {return END_CELL;}
    {end_table}         {BEGIN(INITIAL); return END_TABLE;}
    [^\n \t][a-zA-Z0-9 ]*           {yylval.str = strdup(yytext); return(
    TEXT);}
    }
```

- list

```
<list>{
    [ \t\n]+        {;}
    ^.+             {yylval.str = strdup(yytext); return TEXT;}
    {end_list}      {BEGIN(INITIAL); return END_LIST;}
    }
```

- codeblock

```
<codeblock>{
    [ \t\n]+                    ;
    [\n.]{end_codeblock}        {yytext[strlen(yytext)-10]='\0'; yylval.
    str= yytext; BEGIN(INITIAL); return END_CODEBLOCK;}
    }
```

Ao introduzir estes quatro estados no *Flex*, conseguimos facilitar um pouco o nosso trabalho. No caso do **codeblock**, interessa-nos ter um estado porque isso nos permite guardar todo o que está dentro das etiquetas respectivas, inclusive outras etiquetas. Dentro do codeblock podemos fazer um BEGIN de outro tipo, sem compremeter o principal objectivo, que é colocar tudo o que está lá em formato de código. Para criar as **tabelas**, poupa-nos algum trabalho. Depois em relação aos estados **list** e **keywords**, que são muito parecidos, usamos estados porque assim basta inserir cada elemento da lista por linha sem precisar de usar algum tipo de separador, o que torna a nossa linguagem mai prática.

Além de estados, o restante *Flex* é bastante simples e intuitivo pois são apenas as etiquetas que nós definimos para a nossa linguagem. Como por exemplo, este exerto inicial:

```
begin_report "BREPORT"
end_report "EREPORT"
begin_frontmatter "BFM"
end_frontmatter "EFM"
begin_title "BTITLE"
end_title "ETITLE"
begin_subtitle "BSUBTITLE"
end_subtitle "ESUBTITLE"
begin_date "BDATE"
end_date "EDATE"
date_today "TODAY"

...


{begin_report}    {return BEGIN_REPORT;}
{end_report}    {return END_REPORT;}
{begin_frontmatter} {return BEGIN_FM;}
{end_frontmatter} {return END_FM;}
{begin_title} {return BEGIN_TITLE;}
{end_title} {return END_TITLE;}
{begin_subtitle} {return BEGIN_SUBTITLE;}
{end_subtitle} {return END_SUBTITLE;}
{begin_date} {return BEGIN_DATE;}
{end_date} {return END_DATE;}
{date_today} {return TODAY;}
```

Como podemos ver por esse pedaço de código, é facilmente visível e perceptível o significado de cada umas das etiquetas que iremos usar na nossa linguagem.

5 Utilização do YACC

Aqui encontrar-se-ão referenciados todos os *tokens* deinidos na especificação FLEX e estes serão palavras reservadas da nossa própria linguagem. Cada *token* terá uma tarefa específicaca quanto ao processamento da linguagem em uso uma vez que cada palavra indica um tipo de instrucão diferente.

Aqui é efetuada uma tradução direta entre os símbolos definidos na gramática apresentada e o código C. É desta forma que guardamos na estrutura de dados todo o conteúdo relevante e que nos permite gerar relatórios tanto em LaTeXcomo em HTML.

Ao utilizar o YACC podemos definir código C a ser exucutado no momento em que cada regra da gramática definida é verificada.

6 Estruturas na linguagem C

No que diz respeito às estruturas de dados de forma a armazenar o conteúdo do relatório para posteriormente ser possível imprimir o conteúdo após fazer o processamento da linguagem definida, são descritas de seguida as decisões e a forma de implementação destas no âmbito deste projecto. Assim sendo tornou-se necessária a criação de estruturas de dados para os dados relativamente a autores, nomeadamente o nome, o número e o mail.

```
typedef struct sAuthor
{
    char* name;
char* number;
char* mail;
} Author;
```

Relativamente ao resumo aagredecimentos e capitulos do relatório estes são implementados em GArray, uma estrutura de dados genérica pertencente à glib.

```
typedef struct sAbstract
{
    GArray* paragraphs;
} Abstract;

typedef struct sAknowledgements
{
    GArray* paragraphs;
} Aknowls;

typedef struct sBody
{
    GArray* chapters;
} Body;
```

A capa do relatório é definida pelo titulo, subtitulo, autores, data, resumo e agradecimentos da seguinte forma:

```
typedef struct sFrontMatter {
    char* title;
char* subtitle;
GArray* authors;
char* date;
```

```
Abstract abstract;
Aknowls aknow;
} FrontMatter;
```

Relativamente a figuras e tabelas usaram-se as seguintes estruturas de dados.

```
typedef struct sFigure
{
    char* caption;
char* path;
char* format;
float size;
} Figure;

typedef struct sTable
{
char* caption;
int linhas;
int colunas;
char*** text;
} Table;
```

A utilização de Unions facilitou o processo de criação da linguagem e de facilitação do uso . A razão principal para o uso deste tipo de estruturas é a possibilidade de representar uma informação de mais do que uma maneira. Como é o caso da nossa linguagem, em que, por exemplo, um capítulo pode conter vários topos de elementos.

```
typedef union uElems
{
    Paragraph paragraph;
Figure fig;
Table table;
List list;
Section section;
char* codeblock;
char* summary;
} ElemU;

typedef struct sElem
{
ElemU e;
int id;
}Elem;
```

Como se faz notar recorre-se a tipos primitivos do C ou a estruturas básicas de fácil implementação, para além das estruturas supracitadas existem outras estruturas todas bastante semelhantes às anteriores, como por exemplo estruturas para paragrafos, secções entre outras.

Resta falar do uso da biblioteca do C, que foi recomendada pelos docentes, a glib. No primeiro trabalho prático, usamos módulos de listas ligadas feitos por nós e agora escolhemos usar esta biblioteca, e também porque nunca nenhum de nós tinha experimentado. Fica de notar que facilitou o processo, comparativamente a trabalhos anteriores, e será concerteza usada por nós futuro.

7 Geração de Relatórios - LaTeXe HTML

Depois de escrita toda a gramática e definidas todas as nossas tags a usar. Fica a faltar gerar, com o recurso à linguagem C, um relatório em HTML. Todas a nossas funções que imprimem o HTML foram feitas no ficheiro **funcs.c**. Nós optamos por dividir e minimizar ao máximo a complexidade das funções para posteriormente ser mais fácil resolver algum problema. Deste modo, evitamos a implementação de funções demasiado grandes.
Fazendo a ligação das tags ao HTML, através da nossa linguagem decidimos o que gerar conforme o que definimos. A maioria das coisas já existiam no HTML, mas por exemplo para os índices não existe nada que os criem automaticamente.

8 Análise dos resultados obtidos

O resultado obtido com um exemplo apenas do frontmmater.

```
BREPORT
    BFM
BTITLE
Um relatório em HTML
ETITLE
BSUBTITLE
Vamos aprender HTML
ESUBTITLE
BAUTHOR
BNAME
Serafim Pinto
ENAME
BNUMBER
61056
ENUMBER
BMAIL
smcp92@gmail.com
EMAIL
EAUTHOR
BAUTHOR
BNAME
asda Pinto
ENAME
BNUMBER
54654
ENUMBER
BMAIL
asda@gmail.com
EMAIL
EAUTHOR
TODAY
BINST
Universidade do Minho
EINST
KEYWORDS
yacc
flex
PL
```

```
LEI
Engenharia
Informatica
:KEYWORDS
```

Title: Um relatório em HTML

Subtitle: Vamos aprender HTML

Authors:

Name	Number	Email
Serafim Pinto	61056	smcp92@gmail.com
asda Pinto	54654	asda@gmail.com

Date:

Segunda-Feira, 03 de Junho de 2013

Institution: Universidade do Minho

- yacc -- flex -- PL -- LEI -- Engenharia -- Informatica -

Um exemplo do corpo de um relatório com alguns elementos presentes aqui, como caso da figura, lista, tabela entre outros.

```
BBODY
     BCHAP
BTITLE
Um capítulo para testar
ETITLE
BPARA
lyfluyfjy
EPARA
BPARA
çukgfykhfçfk
EPARA
ECHAP
BCHAP
BTITLE
Mais capítulo para testar
ETITLE
BPARA
lyfluyfjy
EPARA
BSUMMARY
dsfdf
ESUMMARY
BCODEBLOCK
dsfdf
ECODEBLOCK
BLIST
items numa lista
mais items numa lista
mais items numa lista
mais items numa lista
ELIST
BPARA
çukgfykhfçfk
EPARA
ECHAP
BCHAP
BTITLE
Um último para testar
ETITLE
BPARA
```

lyfluyfjy
EPARA
BFIG
BGRAPH
1.jpg
EGRAPH
BCAPTION
rei-deus-patrao
ECAPTION
EFIG

Mais capítulo para testar

lyfluyfjy

dsfdf

dsfdf ECODEBLOCK

- items numa lista
- mais items numa lista
- mais items numa lista
- mais items numa lista

çukgfykhfçfk

Um último para testar

lyfluyfjy

rei-deus-patrao

çukgfykhfçfk

Dados	Mais dados	Teste
MAis dados	TESTE	TESTE2
AInda mais dados	para variar	ok
amostra	amostra2	amostra4

Epilogue

Exemplo de cenas

9 Makefile e Manpage

Para facilitar o processo de gestão quer de desenvolvimento quer de distribuição do software desenvolvido, seguimos os conselhos dos docentes e criamos um makefile com as funções mais comuns, tentado seguir os *standards* quer de nomenclatura quer das funcionalidades.
Assim sendo criamos 4 opções mais distintas no makefile: install, clean, test e delete.

Em relação à opção *install*, o que a sequência de comandos por nós definida faz é: compilar os ficheiros .c por ordem de dependência, fazer a linkagem, mover o escutável para a directoria /usr/bin e mover o ficheiro da man page para a directoria dos manuais. Além disso remove os ficeiros *.o* gerados durante o processo.

A opção *delete* executa a seguinte sequência: eliminar o executável e o ficheiro da man page.

A opção test corre corre o executável na diretoria atual utilizando como input um ficheiro que aparece na raiz da diretoria que contem os ficheiros com o código.

A opção *delete* limpa a raiz da directoria alguns ficheiros que o compilador gera que que não interessam, com por exemplo o lex.yy.c, y.tab.c e o y.tab.h.

Mais uma sugestão dada pelos docentes passava por criar uma manpage. Como o executável que criamos não tem opções ao executar pela linha de comandos, o manual que criámos é relativamente pobre e contem apenas uma descrição do trabalho. Para desenvolver esta parte utilizamos a aplicação para Mac OSX *Mandrake*.

10 Conclusão

Em relação a este trabalho prático e após o seu término podemos retirar algumas conclusões:

- Este trabalho permitiu, de uma forma agradável, consolidar os conhecimentos adquiridos acerca do par FLEX/YACC para a geração de compiladores.
- Olhando para o trabalho realizado, embora o resultado final seja uma linguagem funcional, verificamos que talvez com mais tempo pudéssemos gerar uma linguagem mais agradável para o utilizador.
- Apesar de não termos implementado a geração de relatórios em LaTeX, tal não seria muito díficil sendo o trabalho realizado análogo ao realizado para a geração de relatórios em HTML.