Relatório - Processamento de Linguagens
Report 2007: vamos escrever relatórios

Diogo Alves
A61030
a61030@alunos.uminho.pt

Helder Gonçalves
A61084
a61084@alunos.uminho.pt

Ricardo Branco
A61075
a61075@alunos.uminho.pt

5 de Junho de 2013

Resumo

Neste trabalho tem-se como objetivo criar um analisador léxico e um sintático, que "processa"/analisa o texto, que apanha as palavras reservadas, e de seguida verifica se a estrutura do relatório está bem construida. Enquanto analisa o texto este é guardado em estruturas de dados, e em listas ligadas para separar o código html do código latex, convertendo e criando por fim ficheiros HTML e/ou LaTeX com o nosso relatório convertido para cada uma das linguagens.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Para o segundo Trabalho Prático da Unidade Curricular de Processamento de Linguagens, a nossa escolha foi o enunciado 3 que tem como titulo: "Report 2007: vamos escrever relatórios".
Neste projeto, pretende-se que seja criado um compilador capaz de "converter"uma relatório escrito numa linguagem criada por nós, e já usada no trabalho prático 1 para a linguagem HTML ou/e LaTeX.
Portanto, neste documento irão estar presentes as nossas decisões, a estruturação do projecto, bem como as explicações e funcionamento do mesmo.

Capítulo 2

Sintaxe da Nossa Linguagem

Nós, como referido anteriormente, estamos a "continuar"o trabalho realizado no Trabalho Prático 1 (TP1) e por isso a sintaxe da linguagem manteve-se a do trabalho anterior, fazendo algumas alterações e acrescentando outras funcionalidades que são as seguintes:

Comandos	Descrição
Comandos a utilizar	

Capítulo 3

Estruturação do Trabalho

Este trabalho consiste em dois analisadores, um lexico, feito em flex que irá "apanhar"as palavras reservadas na nossa linguaguem e passar a informação do que reconheceu para o analisador sintático, feito em yacc, que irá verificar se a gramática obtida do documento que está a ser analisado está correta.
Depois disto, e no yacc, gravamos os dados em estruturas de dados. Por fim, vamos buscar os dados a essas estruturas e criamos um ficheiro HTML e/ou LaTeX com a nossa linguagem convertida para essas linguagens.

3.1 Flex

No ficheiro flex, temos uma condição de contexto exclusiva para os indices de figuras e tabelas, para criar caso exista os respectivos indices, caso contrário (se não existirem esses indices) cria o indice normal dos capitulos e secções.
Criamos também umas variáveis para as expressões regulares para facilitar e tornar o trabalho mais legivel, e apenas as escrevemos nos locais onde deveriam ter essas mesmas expressões.
Depois disto entramos na parte de flex mesmo, onde o compilador vai "filtrar"as tag's (palavras reservadas) da nossa linguagem, os textos, através das expressões regulares e mandar essa informação para o yacc para este verificar se a gramática se escontra correta.

3.2 Yacc

Em yacc temos a gramática para a estrutura de um relatório, como é formado e o que é permitido.

Começamos por incluir as bibliotecas necessárias para trabalhar com o C, incluindo uma criada por nós (preprocessador) que contém as estruturas e funções para guardar as informações retiradas do documento inicial.
Ainda nesta parte e depois dos includes declaramos as variáveis para as estruturas do preprocessador.
Após esta parte de C, criamos os tokens, types, uma estrutura union onde estão guardados os valores filtrados no flex que irão ser lidos/tratados pelo yacc e indicamos ao yacc onde ele vai começar a verificar a gramática.
Depois disto entramos na da gramática em si e das suas derivações, e o yacc vai começar em report que deriva na capa e no relatório em si (o corpo do relatório), e parte daí para o resto das derivações.
No fim, temos as funções para os erros e o main onde inicializamos a estrutura do relatório e iniciamos o yacc.

3.2.1 Estrutura do Relatório

Como referido anteriormente, o nosso relatório e composto por duas partes, a capa e o corpo do relatório.
A capa tem obrigatóriamente a seguinte estrutura:

1. Titulo
2. Autor
3. Data
4. Resumo

e opcionalmente:

- Mais do que um autor
- Sub Titulo
- Instituição
- Palavras Chave
- Agradecimentos
- Indice
- Indice de Figuras

- Indice de Tabelas

E o autor tem obrigatóriamente:

- Nome

e opcionalmente:

- Numero de identificação
- Email
- Url
- Afiliação

Neste caso, as opção fizemos em forma de escada, onde uma tinha-se a ela própria e à seguinte ou apenas a seguinte.
Depois no corpo do relatório temos obrigatoriamente:

- Capitulo ou lista de capitulos

Um capítulo tem obrigatoriamente:

- Titulo
- Lista de Elementos

A lista de elementos pode ser qualquer um entre:

- Parágrafo
- Corpo Flutuante
- Lista
- Secção
- Sumário
- Bloco de Código

Uma secção pode ter:

- Titulo
- Lista de Elementos

O paragrafo tem:

- Paragrafos
- Texto
- Elementos

Os elementos podem ser:

- FootNote
- Ref
- Xref
- CitRef
- Href
- Iterm
- Bold
- Italic
- Underline
- InlineCode
- Acronym

O sumário e os elementos anteriores aceitam:

- Texto

As listas podem ser:

- Enumerate
- Itemize

Se for Enumerate pode:

- Ter um item relativo ao mesmo
- Conter um itemize

Se for Itemize pode:

- Ter um item relativo ao mesmo
- Conter um Enumerate

O item pode ter texto ou todos os items referidos em cima no "Elementos".
Um corpo flutuante pode ser uma imagem ou uma tabela:
Se for uma imagem tem:

- O caminho da imagem
- Legenda

Se for tabela tem:

- Legenda
- Linhas

Uma linha contém uma lista de células.

3.3 PreProcessador

Neste modulo criado por nós, temos as estruturas necessárias para guardar os dados, bem como as funções para inicialização e para inserção dos dados nas mesmas. Temos támbem incluido um modulo de lista ligada genérica, também criado por nós, para guardar os dados nas estruturas, uma vez que podemos ter, por exemplo, vários autores, e também para o indice o que nos permite gerá-lo em apenas uma passagem pelo ficheiro de entrada, o que não seria possível sem uma estrutura de dados por causa de gerar o código para HTML.

3.3.1 Estruturas de Dados

As estruturas de dados usadas para efetuar o trabalho foram as seguintes:
Uma para as células da tabela, bem como para os indices e linhas da mesma e uma geral da tabela onde guarda a legenda e as linhas e estas guardam as células.
Temos outra estrutura para as imagens onde e guardada a legenda e caminho para a imagem, as keywords são também guardadas numa estrutura em que usa o modulo da lista ligada, uma vez que pode conter várias palavras chave.
Os autores têm uma estrutura própria com os dados de cada um e posteriormente será guardado numa lista ligada pois poderão ser vários, lista essa que está dentro da estrutura principal, que é a do relatório, onde são guardados também o titulo, o sub titulo, a intituição, os indices e a data, estes como dados únicos, pois haverá apenas um no relatório todo, depois temos todos os outros dados a serem gravados com listas ligadas, são eles, os autores, as palavras chave, os indices (um por lista ligada), outra para HTML e por fim para LaTeX.

3.3.2 Funções para as Estruturas

Para as estruturas temos funções de inicialização, que são as seguintes:

```
Autor init_Autor();
Cell init_Cell();
Row init_Row();
Table init_Table();
Image init_Image();
IndiceCell init_IndiceCell();
Image init_Image();
Keywords init_Keywords();
Report init_Report();
```

Funções para a capa:

```
void addTitulo(Report*,char*);
void addSTitulo(Report*, char*);
void addAutor(Report*,Autor*);
void addKey(Report*,char*);
```

E funções para o corpo do relatório:

```
void addRef(Report*,char*, char*,int);
void addHRef(Report*,char*,char*,int);
void addNegTag(Report*,int);
void addItTag(Report*,int);
void addUnderTag(Report*,int);
void fechoTag(Report*,char*,int);
void addTexto(Report*,char*,int);
void fechoParagrafo(Report*,int);
void addCapitulo(Report*,char*);
void addParagrafo(Report*,int);
void addEndTAG(Report*,char*,char*);
void addTextoNF(Report*,char*);
void addCodLinha(Report*,char*,int);
void addImagem(Report*,Image*);
void addItem(Report*,char*);
void addOrdList(Report*);
void addItemList(Report*);
void fechaOrdList(Report*);
void fechaItemList(Report*);
void addCelula(Row*,Cell*);
void addLinha(Table*,Row*);
void addTabela(Report*,Table*);
void addSeccao(Report*,char*,int);
void addResumo(Report*);
void addAgradecimentos(Report*);
void fechoResumo(Report*);
void fechoAgradecimentos(Report*);
```

3.4 Lista Ligada

Como referido na secção anterior, a lista ligada é usada nas estruturas para guardar os dados, pelos motivos referidos acima.

3.4.1 Estrura de dados

A lista ligada implementada por nós é genérica e esta é a sua estrutura:

```
typedef struct node {
    void* data;     /* Apontador para um tipo de dados*/
    struct node *next; /* Apontador para o próximo nodo*/
}Node;


typedef struct list {
    int size;     /* Número de elementos na lista ligada*/
    int (*compare) (void*,void*); /* Função de comparação*/
    size_t dataSize; /* Memória que o tipo de dados necessita*/
    Node *list; /* Apontador para o primeiro nodo da lista*/
}List;
```

3.4.2 Funções para controlo das Estruturas

```
List* init(size_t dataSize,int (*compare) (void*,void*));
List* insertHead (List *l, void *data);
int search (List *l, void *data);
int listDestroy(List *l);
List* sortedInsert (List *l, void *data);
List* insertTail (List *l, void *data);
List* removeNode (List*l, void *data);
void* pop(List* l);
```

3.5 Report

Neste modulo, temos as funções para gerar os ficheiros HTML e LaTeX, em que inclui o preprocessador e vai lá buscar os dados para criar os respectivos ficheiros.

3.6 Makefile

Com a makefile podemos criar o executável, fazendo apenas "make", podemos também eliminar os ficheiros .o, .c gerados pelo flex e yacc e o executável também fazendo "make clean", podemos ainda instalar o executável fazendo "make install", desinstala-o fazendo "make remove"e comprime os ficheiros principais fazendo "make tar".

Capítulo 4

Conclusão

Este trabalho prático permitiu-nos consolidar os conhecimentos obtidos nas aulas uma vez que precisamos de tudo o que temos vindo a dar, a forma como estruturar o compilador, como funcionava a gramática disponibilizada pelo docente, o que era suposto os analisadores lexico e semantico fazerem, como funcionavam e interagiam, e nesse aspeto as aulas ajudaram bastante pois tudo foi explicado lá.
O nível de dificuldade neste trabalho foi um pouco maior que no primeiro, o que era de esperar, e foi facilitado pelo facto de termos já grande parte da gramática, apenas tendo de fazer algumas alterações e acrescentar algumas derivações para ser compativel com o nosso primeiro trabalho.
O facto de escolher-mos este enunciado também facilitou, porque no primeiro trabalho já tinhamos definido a sintaxe da nossa linguagem e então apenas tivemos de alterar um pouco o flex e não pensar em toda a sintaxe/gramática novamente, já tinhamos usado estruturas de dados no primeiro trabalho e foi só altera-las minimamente e aplica-las neste trabalho, o que nos poupou bastante tempo e trabalho.
A nossa satisfação perante o que foi produzido neste trabalho é positiva, uma vez que ficamos a perceber melhor a estruturação do compilador dividindo o analisador em analisador lexico e sintatico, colocando depois um programa em C a fazer a gestão dos dados e escrever nos ficheiros HTML e LaTeX nas respetivas linguagens.