UNIVERSIDADE DO MINHO

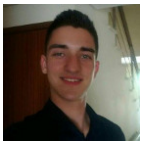
DEPARTAMENTO DE INFORMÁTICA

PROCESSAMENTO DE LINGUAGENS

Relatório TP3 - Dicionário de Informática

Autores:

Alexandre Pinho (A82441)



Joel Gama (A82202)

10 de Junho de 2019

Conteúdo

1 Introdução

Este relatório surge no contexto da UC de Processamento de Linguagens do terceiro ano do Mestrado Integrado de Engenharia Informática. O trabalho proposto consiste na criação de um programa que processa um conjunto de textos procurando termos de um dicionário de termos na área da Informática. Como resultado final é pretendido obter textos anotados segundo um dicionário pré-definido.

Nas próximas páginas, é mostrada a linguagem utilizada para especificar um dicionário, a implementação de um *parser* para ela através do *flex* e do *yacc*, o algoritmo utilizado no processamento dos textos e quais os resultados produzidos pelo programa. Por fim, é incluída uma descrição da interface por linha de comando do programa, uma discussão do trabalho futuro, e uma breve conclusão.

2 Descrição do problema

Neste trabalho prático o problema apresentado consiste em desenvolver um sistema de anotação (automática) de textos de informática. O sistema deve aceitar um dicionário e a lista de textos a serem anotados.

Desta forma, pretende-se ter como resultado um ficheiro LaTeX que reproduz o texto dos ficheiros dados com todas os termos que estejam presentes no dicionário sublinhados e associados a uma nota de rodapé com o termo em inglês. Por fim, no final do documento será gerado um apêndice com a lista de termos encontrados e respectivos tradução, significado e sinónimos.

Especificação do conteúdo de uma entrado de um dicionário:

- Termo: sequência de caracteres a serem anotados no texto
- Designação em inglês: tradução do termo
- Significado: frase que explica o significado do termo
- Sinónimos: lista de sinónimos do termo

3 Resolução do problema

3.1 Gramática

A linguagem utilizada para descrever um dicionário segue uma gramática independente de contexto (GIC), com três símbolos não-terminais e oito símbolos terminais, incluindo sinais de pontuação.

```
DicI : Def DicI
     | &
     ;

Def : TERMO '(' TERMO ')' ':' DESC '[' Sinonimos ']'


Sinonimos -> TERMO ',' Sinonimos
           | TERMO
           | &
           ;
```

Várias decisões foram feitas tendo em vista a clareza e usabilidade da linguagem, tanto na escolha dos símbolos terminais como na sintaxe da gramática em si.

O terminal `TERMO` representa um termo, uma tradução do termo, ou um sinónimo, e é escrito entre apóstrofos. O terminal `DESC` é a descrição do significado de um termo, e deve ser escrito entre aspas. Esta notação permite que um termo contenha espaços e outros caracteres sem introduzir ambiguidade na gramática.

A verbosidade relativa da gramática (os parênteses curvos à volta da designação em inglês, os dois pontos, os parênteses retos à volta dos sinónimos, e as vírgulas a separar), apesar de desnecessários para a desambiguação da gramática (se for adicionado um símbolo no fim da produção `Def`), servem para clarificar o significado de cada parte. Assim, consegue-se intuitivamente compreender o que significam as entradas do dicionário.

Ao listar os sinónimos é possível introduzir um vírgula depois do último sinónimo. Esta possibilidade pode induzir em erro se for esquecida a introdução do último sinónimo na lista. No entanto, como se prevê que o ficheiro que descreve o dicionário será automaticamente gerado por um programa, ou construído por uma pessoa com recurso ao ”copy-paste”, tal escolha previne mais erros humanos do que causa, e simplifica a construção do programa que gera o dicionário.

A seguir consta um exemplo de uma definição de um termo:

```
'termo' ('designação comum em inglês'): "significado"
['sinónimo 1', 'sinónimo 2',]
```

3.2 Construção do dicionário

O primeiro passo para o processamento dos textos é a construção do dicionário do termos a serem anotados.

3.2.1 Estruturas utilizadas

Para guardar as entradas do dicionário em memória central é utilizada uma tabela de hash cujas chaves são os termos a anotar e os valores contêm a informação relativa ao respetivo termo.

A estrutura que contém essa informação, denominada de `Termo`, contém os seguintes campos (definição da estrutura em `C`):

```
typedef struct termo {
    char *termo;
    char *designacao_ingles;
    char *significado;
    GSequence *sinonimos; // sequência de sinónimos (char *)
} *Termo;
```

3.2.2 FLEX

A análise léxica é feita através do FLEX, que dado um conjunto de padrões e correspondentes ações produz os *tokens* necessários para a análise sintática.

Assim são processados três padrões: um para *tokens* representados por um único caractere; um para os termos (`TERMO`); e um para o significado (`DESC`).

```
[\(\):\[\],]        {return yytext[0];}
```

Para os termos e os significados, é possível incluir dentro da sequência de caracteres o próprio delimitador utilizando um *backslash* atrás.

```
'([^']|(\\'))*' {
    yylval.s=escape(yytext);
    return TERMO;
}

\"(\n|[^"]|\\\")*\" {
    yylval.s=escape(yytext);
    return DESC;
}
```

O função `escape()`, para além de excluir o primeiro e o último caractere da sequência, remove também os *backslashes* utilizados para fazer sequências de escape.

```
char *escape(char *s) {
    char *r = (char *)malloc(sizeof(char)*(strlen(s)+1));
    int j = 0;
    for (uint32_t i = 0; i < strlen(s)+1; i++) {
        int backslash = yytext[i] == '\\';
        int last_backslash = (i != 0 && yytext[i-1] == '\\');
        if (!backslash) {
            r[j++] = yytext[i];
        } else if (last_backslash) {
            r[j++] = yytext[i];
            s[i] = 'X';
        }
    }
    r[j - 2] = '\0';
    return r + 1;
}
```

De resto, todos os caracteres considerados espaço vazio são ignorados, e se algum caractere não tiver sido reconhecido por uma das regras anteriores, então é impressa uma mensagem de erro é o programa é terminado.

```
[ \t\n] {}
.       {fprintf(stderr, "sati: syntax error on dictionary file\n"); exit(1);}
```

3.2.3 YACC

Adicionando ações às produções da gramática especificada anteriormente, obtem-se um procedimento que cria o dicionário a partir do ficheiro indicado pelo utilizador.

Os *tokens* TERMO e DESC têm um valor de *string* (`char *`). O símbolo não-terminal Sinonimos é uma sequência de *strings* (`GSequence *`), e o não-terminal Def é uma entrada do dicionário – `Termo`.

```
%union {char *s; GSequence *seq; Termo t;};

%token <s> TERMO
%token <s> DESC

%type <seq> Sinonimos
%type <t> Def
```

Para construir a lista de sinónimos são utilizadas as funções `g_sequence_new()` e `g_sequence_prepend()` para adicionar um sinónimo à lista já existente. Os sinónimos são adicionados à cabeça da lista para manter a ordem pela qual foram listados no ficheiro, já que a produção tem recursividade à direita.

```
Sinonimos : TERMO ',' Sinonimos {$$ = $3; g_sequence_prepend($$, $1);}
          | TERMO {$$ = g_sequence_new(NULL); g_sequence_prepend($$, $1);}
          | & {$$ = g_sequence_new(NULL); }
```

Depois de ter todos os componentes de uma entrada no dicionário, eles são postos numa estrutura do tipo `Termo` utilizando a função `termo_new()`.

```
Def : TERMO '(' TERMO ')' ':' DESC '[' Sinonimos ']'
        {$$ = termo_new($1, $3, $6, $8);}
```

Por fim, cada termo é adicionado ao dicionário representado pela tabela de hash `dicionario`, sendo a sua chave o termo a ser procurado no texto.

```
DicI : Def DicI {Termo t=$1; g_hash_table_insert(dicionario, t->termo, t); }
     | &
     ;
```

3.3 Estratégias de processamento do texto

Depois de construído o dicionário, ele é utilizado para identificar que texto deve ser anotado e respetiva definição incluída no apêndice.

3.3.1 Estruturas utilizadas

O algoritmo implementado emprega cinco estruturas de dados principais:

- `GHashTable *dicionario;` – o dicionário de termos
- `GSequence *procura_atual;` – a lista de termos a serem procurados
- `GTree *matches;` – conjunto de termos presentes no texto, ordenados e indexados pela sua posição inicial
- `GHashTable *apendice_por_posicao;` – conjunto das ocorrências dos termos presentes no texto indexadas pela sua posição
- `GHashTable *apendice_por_termo;` – conjunto de termos presentes no texto

Para representar uma procura de um termo, existe a estrutura `Procura`:

```
typedef struct procura {
    int posicao_inicial;
    char *termo;
    int posicao;
    int ignorar;
} *Procura;
```

O campo `posicao_inicial` indica a posição, no texto a ser processado, do primeiro caractere reconhecido. O campo `termo` é o termo a ser procurado, o campo `posicao` indica o índice do próximo caractere a ser reconhecido. E o campo `ignorar` serve apenas na indicar que a estrutura deve ser retirada da lista, pois não é possível o fazer durante a iteração.

3.3.2 Algoritmo de deteção de termos

O algoritmo que processa o texto e determina as ocorrências dos termos é baseado numa máquina de estados finitos não determinística (AFD). Ele mantém todos os termos (estados) a serem procurados, até chegar (ou não) a um estado aceitador. O algoritmo segui os próximos passos:

Para cada termo a ser procurado, o algoritmo verifica se o próximo caractere a serem reconhecido é igual ao próximo caractere do texto a ser processado. Se não for, retira da lista `procura_atual`. Se for, incrementa a sua posição e testa se o próximo caractere do termo é `'\0'`, o que significa que foi feito o reconhecimento completo do termo.

Nesse caso, o termo deve ser adicionado ao conjunto de *matches*. Neste conjunto apenas pode existir um reconhecimento por cada posição. Se um termo é um prefixo de um termo com maior comprimento e ambos forem reconhecidos na mesma posição, o anterior será adicionado primeiro a este conjunto, mas o último, o de maior comprimento, irá substituir-lo quando for reconhecido depois dele. Assim, é garantido o reconhecimento dos termos mais longos em cada ocasião de conflito.

Em cada iteração, são adicionados à lista de termos a serem reconhecidos todos os termos do dicionário cujo primeiro caractere coincide com o atual caractere do texto. Isto é feito iterando pelo dicionário e fazendo a comparação para cada termo.

3.3.3 Produção dos resultados

No fim de serem detetados todos os termos a serem anotados, o programa itera mais uma vez pelo texto de cada ficheiro dado e imprime o código LaTeX necessário para produzir os efeitos desejados, nomeadamente sublinhar os termos assinalados, adicionar uma nota de rodapé para cada um, e adicionar um apêndice no fim do documento.

Para sublinhar e adicionar a nota de rodapé, são impressos os comandos LaTeX `\underline` e `\footnote` (com o termo e a designação em inglês) quando uma ocorrência de um termo é encontrada na posição `i`. Para não imprimir os caracteres do termo duas vezes, a posição é incrementada de acordo com o comprimento do termo.

```
char *termo = g_hash_table_lookup(apendice_por_posicao, (gpointer)i);
if (termo) {
    Termo t = g_hash_table_lookup(dicionario, termo);
    printf("\\underline{%s}\\footnote{%s}", termo, t->designacao_ingles);
    i += strlen(termo) - 1;
}
```

Para o apêndice, é utilizado o comando `\appendix`, e de seguida impressas as entradas do dicionário que foram utilizadas em um ou mais textos de input, utilizando o conjunto das chaves do tabela de *hash* `apendice_por_termo`.

Para além dos caracteres que não fazem parte de nenhum termo reconhecido, é necessário também formatar caracteres que têm significado especial no LaTeX.

```
void tex_escape(char c) {
    switch (c) {
        case '&':
        case '%':
        case '$':
        case '#':
        case '_':
        case '{':
        case '}':
            printf("\\%c", c);
            break;
        case '~':
            printf("\\textasciitilde ");
            break;
        case '^':
            printf("\\textasciicircum ");
            break;
        case '\\':
            printf("\\textbackslash ");
            break;
        case '\n':
            printf("\n\n");
            break;
        default:
            printf("%c", c);
    }
}
```

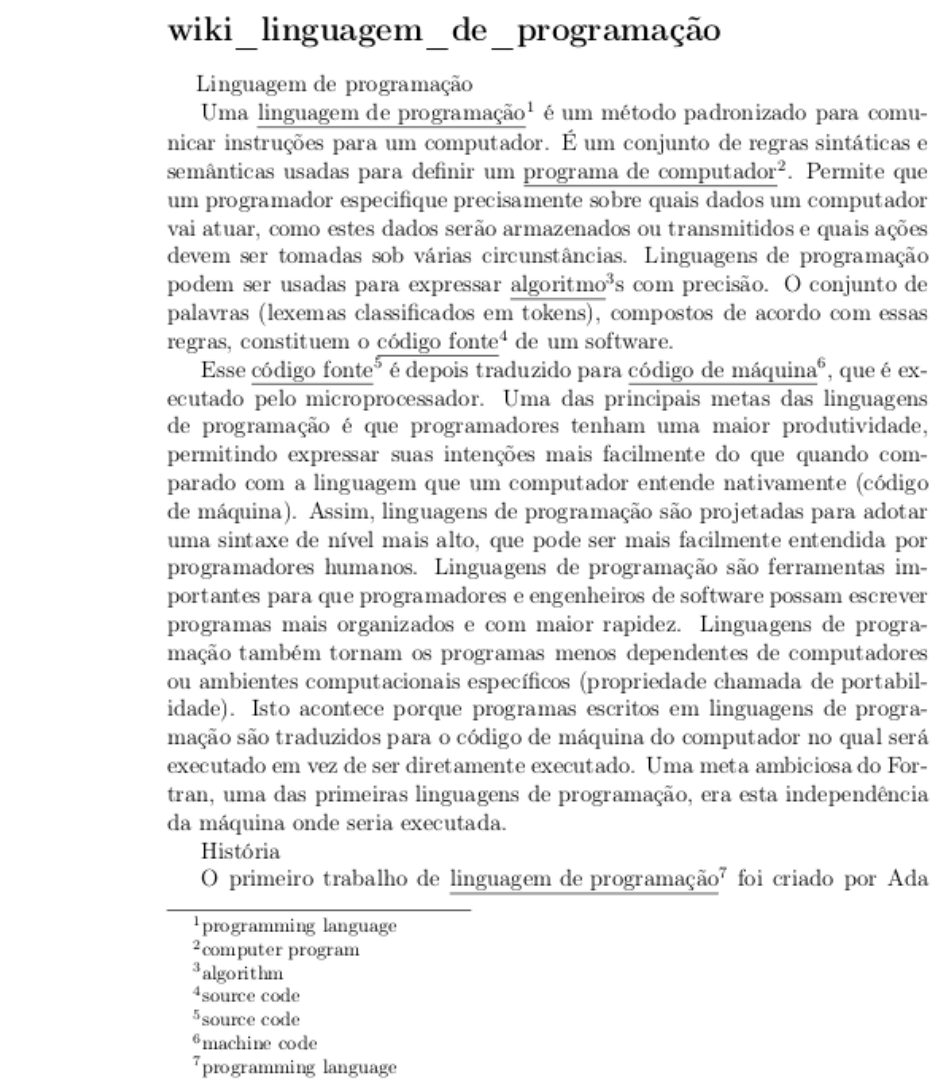
4 Resultados - Ficheiro LaTeX

Como resultado do processamento dos textos é esperado um ficheiro LaTeX. Neste ficheiro os termos encontradas que possuem uma definição no dicionário são sublinhados. É também adicionada numa nota de rodapé a designação comum do termo em inglês. Por fim, no final do ficheiro encontra-se um apêndice com todos os termos encontrados, com os significados, traduções e sinónimos associados.

```
'linguagem de programação' ('programming language') :
    "Método padronizado para comunicar instruções para um computador."
    [
    ]

'programa de computador' ('computer program') :
    "Conjunto de instruções que descrevem uma tarefa a ser realizada por um computador."
    [
    'programa',
    'programa informático',
    'executável',
    ]

'algoritmo' ('algorithm') :
    "Uma sequência finita de ações executáveis que visam obter uma solução para um determinado tipo de problema."
    [
    'operação',
    ]

'código fonte' ('source code') :
    "Conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das linguagens de programação existentes, de maneira lógica."
    [
    'código',
    'fonte',
    'programa',
    ]

'código de máquina' ('machine code') :
    "Uma sequência de bytes que se tratam de instruções a serem executadas pelo processador."
    [
    ]
```

Figura 1: Dicionário utilizado.

4.1 Resultado - PDF



wiki_linguagem_de_programação

Linguagem de programação

Uma linguagem de programação[1] é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador[2]. Permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias. Linguagens de programação podem ser usadas para expressar algoritmo[3]s com precisão. O conjunto de palavras (lexemas classificados em tokens), compostos de acordo com essas regras, constituem o código fonte[4] de um software.

Esse código fonte[5] é depois traduzido para código de máquina[6], que é executado pelo microprocessador. Uma das principais metas das linguagens de programação é que programadores tenham uma maior produtividade, permitindo expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente (código de máquina). Assim, linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos. Linguagens de programação são ferramentas importantes para que programadores e engenheiros de software possam escrever programas mais organizados e com maior rapidez. Linguagens de programação também tornam os programas menos dependentes de computadores ou ambientes computacionais específicos (propriedade chamada de portabilidade). Isto acontece porque programas escritos em linguagens de programação são traduzidos para o código de máquina do computador no qual será executado em vez de ser diretamente executado. Uma meta ambiciosa do Fortran, uma das primeiras linguagens de programação, era esta independência da máquina onde seria executada.

História

O primeiro trabalho de linguagem de programação[7] foi criado por Ada

[1] programming language
[2] computer program
[3] algorithm
[4] source code
[5] source code
[6] machine code
[7] programming language

Figura 2: Excerto do texto depois do processamento.

algoritmo (algorithm)

Uma sequência finita de ações executáveis que visam obter uma solução para um determinado tipo de problema.

- operação

código de máquina (machine code)

Uma sequência de bytes que se tratam de instruções a serem executadas pelo processador.

código fonte (source code)

Conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das linguagens de programação existentes, de maneira lógica.

- código
- fonte
- programa

linguagem de programação (programming language)

Método padronizado para comunicar instruções para um computador.

programa de computador (computer program)

Conjunto de instruções que descrevem uma tarefa a ser realizada por um computador.

- programa
- programa informático
- executável

Figura 3: Apêndice gerado.

5 Utilização do programa

O programa construído lê os ficheiros do dicionário definido e dos textos a serem processados como argumentos, e produz o ficheiro LaTeX pretendido, colocando-o no standard output.

O formato da interface por linha de comando do programa está apresentado a seguir (os primeiros dois argumentos são obrigatórios):

```
./sati <dicionário> <texto1> [<texto2>...]
```

Por exemplo (com redirecionamento do output):

```
./sati lp_dict wiki_linguagem_de_programação >lp.tex
```

Assim, é possível compilar o ficheiro LaTeX produzido e obter um PDF com o aspeto dad Figuras 2 e 3.

6 Trabalho futuro

Em termos de trabalho futuro existem alguns aspetos que podem ser melhor trabalhados.

O programa não lida com caracteres *utf8* com mais de um *byte*, uma vez que a implementação é complexa. A complexidade deriva do facto de se estar a trabalhar com uma linguagem de baixo nível como o C.

O *match* feito pelo programa é *case sensitive*, isto é, os termos apenas são encontrados se estiverem com o mesmo tipo de letra (maiúscula ou minúscula). Este aspeto poderia ser alterado para reconhecer, por exemplo, termos no início de uma frase.

As notas de rodapé são adicionadas em todas as ocorrências do termo, mesmo na mesma página, quando poderiam ser adicionadas apenas na primeira ocorrência da palavra, ou apenas uma vez por página.

O algoritmo de deteção de termos atual tem complexidade assimptótica $O(n * m)$, onde n é o comprimento do texto a processar e m é o número de entradas do dicionário. Um dicionário de termos informáticos pode ter centenas de entradas, tornando o algoritmo pouco eficiente.

Na prática, seria possível melhorar este desempenho para $O(n)$ ao manter uma tabela de hash onde as chaves representam o *lookahead* (de pelo menos dois ou três caracteres) de um certo conjunto de termos, de forma a evitar fazer m comparações com os termos, e também para reduzir a quantidade de termos que são procurados, pois o *lookahead* seria mais que um único caractere.

7 Conclusões

Este trabalho revelou-se importante para os membros do grupo, no sentido em que permitiu um aprofundamento dos conhecimentos adquiridos nas aulas teóricas e práticas da disciplina.

Utilizando as ferramentas *flex* e *yacc*, foi desenvolvida uma linguagem de domínio específico especificada por uma GIC (Gramática Independente de Contexto). Para além disso, foi desenhado um algoritmo de deteção de termos baseado numa máquina de estados finitos não determinística (AFD).

Em suma, em relação ao trabalho realizado, os objetivos estabelecidos foram alcançados e, por isso, faz-se uma apreciação global positiva do projeto.