

1º Trabalho Prático

Métodos de Programação I
LESI/LMCC
Universidade do Minho
Ano Lectivo de 2002/2003

1 Preâmbulo

Este trabalho deve ser realizado por grupos com um máximo de três alunos. O trabalho deve ser entregue até ao dia 24 de Outubro 2002 na Recepção do *Departamento de Informática* (ext. 4430) e nele deve constar a listagem do código desenvolvido assim como um pequeno relatório.

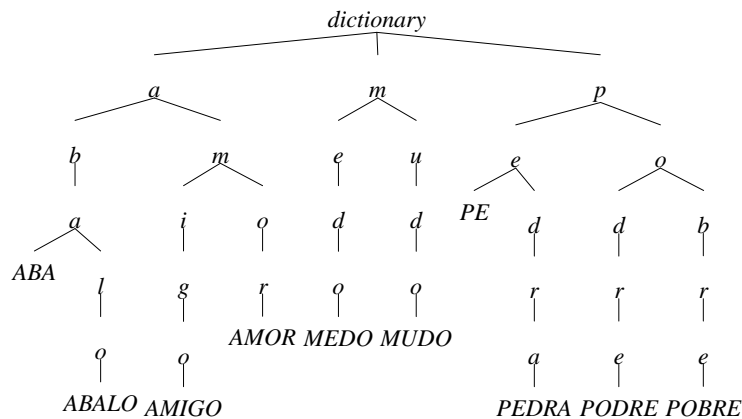
A entrega poderá também ser feita de forma electrónica, de acordo com as instruções dos monitores e docentes das aulas práticas.

2 O problema

Recorde os seguintes requisitos do 4º Trabalho Prático da disciplina de Paradigmas da Programação II, ano lectivo de 2001/02 ¹:

2.5 Dicionário

Pretende-se implementar um sistema de manutenção de um dicionário. Este terá uma estrutura muito peculiar em memória. Será construída uma árvore em que cada nodo terá apenas uma letra da palavra e cada folha da respectiva árvore terá uma pequena string com a definição de cada palavra inserida.



Deve ser possível inserir palavras (palavra e definição) e procurar a definição de determinada palavra. O programa deverá permitir a gravação do dicionário em disco para carregamento posterior.

¹Agradece-se aos docentes desta disciplina a disponibilização de material e colaboração neste trabalho.

3 O que se pretende

Considere o programa em C dado em anexo (pág.3) que é fornecido como resposta aos requisitos acima. O que se pretende neste trabalho é a conversão (“inversão”) desse programa em C para HASKELL, por forma a evidenciar uma comparação entre as capacidades expressivas das duas linguagens. Há que ter em conta os pontos seguintes:

- O programa em HASKELL a desenvolver deverá ser o mais simples possível, tanto em tamanho como em complexidade. Contudo, uma vez compilado pelo GHC, deve produzir o mesmo efeito que o programa C de que se partiu.
- Deverá ainda ser suficientemente paramétrico de modo a que, para diferentes instanciações de tipos (polimorfismo), possa representar outros “tipos” de problemas. Uma situação que deverá obriatoriamente ser considerada é a sua instanciação por forma a representar sistemas de ficheiros hierárquicos do tipo Unix ou Windows.
- Poderão ser definidas funções auxiliares ou outras desde que o resultado seja código HASKELL permaneça elegante e legível.

Tal como é habitual nos trabalhos desta disciplina, fornece-se de seguida um “kit” para arranque do projecto. O enunciado concluir-se-á com sugestões para valorização do trabalho a realizar.

4 Material fornecido como “kit” para o projecto

Entre na página da disciplina e descarregue, descomprimindo-o (eg. via unzip), o ficheiro `mpi0203mp.zip` que contém o respectivo material pedagógico. Para além de outros ficheiros relevantes para a disciplina, deverá obter:

1. `mpi0203t1.lhs` - trata-se do ficheiro que está a ler neste momento, escrito em “*literate HASKELL*”. Isto significa que:

- se o carregar no HUGS, este interpretador carregará o código HASKELL nele contido e interpretá-lo-á. Neste momento o único código HASKELL presente é, a título de exemplo,

```
main = print("trabalho ainda por realizar")
```

- se o processar via \LaTeX (ou $\PDF\LaTeX$) obterá este mesmo documento em *PDF*. Sugestão: experimente

```
latex mpi0203t1.lhs
dvips -o mpi0203t1.ps mpi0203t1
ps2pdf mpi0203t1.ps
```

ou, mais simplesmente, `pdflatex mpi0203t1.lhs`.

Se não está habituado a \LaTeX : em LINUX, faz parte da distribuição standard e é só experimentar; em Windows, sugere-se a instalação de *MiKTeX* (<http://www.miktex.org/>).

2. `mpi0203t1.sty` - trata-se de um ficheiro importado por `mpi0203t1.lhs`, contendo funções de processamento de texto expressas em sintaxe \LaTeX (como poderá ver, é normal programar *funcionalmente* em \LaTeX).
3. `mpi0203t1.pdf` - trata-se do resultado do processamento de `mpi0203t1.lhs`, fornecido para sua conveniência, caso não tenha \LaTeX acessível.

O trabalho consiste em editar ficheiro `mpi0203t1.lhs`, acrescentando-lhe não só texto seu (por exemplo, preâmbulo, conclusões, detalhes de execução, pistas para trabalho futuro, etc.) mas todo o código HASKELL que vai desenvolver, introduzindo-o pela ordem que lhe parecer mais natural².

²Veja como o fazer inspeccionando o material pedagógico da disciplina em 2001/2, eg. `mpi0102t1.lhs`.

5 Sugestões para Valorização

- Construa uma solução híbrida em que parte das funções estão em C (eg. main) e as outras estão em HASKELL.
- Investigue sobre “literate programming” em C e adapte mpi0203t1.lhs por forma a poder ser submetido directamente ao seu compilador de C.
- Inclua no polimorfismo da sua solução dois tipos de dados conhecidos: árvores de decisão e árvores genealógicas (ver trabalhos anteriores desta disciplina).

Anexo

```
#define _GNU_SOURCE

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct _cell_ {
    char letter;
    char *description;
    struct _cell_ *child, *next;
} Cell;

Cell* new_cell(void) {
    Cell *tree;
    tree = (Cell*)malloc(sizeof(Cell));
    tree->letter = ' ';
    tree->description = NULL;
    tree->child = NULL;
    tree->next = NULL;
    return tree;
}

void list_(Cell *tree, char* prefix) {
    if (tree) {
        char *p;
        if (tree->description) {
            printf("%s%c: %s\n", prefix, tree->letter, tree->description);
        }
        /* NOTE: THIS LINE ONLY WORKS ON LINUX/GCC */
        asprintf(&p, "%s%c", prefix, tree->letter);
        list_(tree->child, p);
        free(p);
        list_(tree->next, prefix);
    }
}

void list(Cell *tree) {
    list_(tree, "");
}

char* search(Cell *tree, char *word) {
    if (!word || !tree || *word == '\0') return NULL;
    if (tree->letter == *word) {
```

```

    if (*(word+1) == '\0') {
        return tree->description;
    } else {
        return search(tree->child, word+1);
    }
} else {
    return search(tree->next, word);
}
}

Cell* add(Cell *tree, char *word, char *description) {
    if (!word || *word == '\0') return NULL;
    if (tree) {
        if (tree->letter == *word) {
            tree->child = add(tree->child, word+1, description);
            if (!tree->child) tree->description = strdup(description);
        } else {
            if (tree->letter < *word) {
                Cell *cell = new_cell();
                cell->letter = *word;
                cell->child = add(cell->child, word+1, description);
                cell->next = tree;
                tree = cell;
            } else {
                tree->next = add(tree->next, word, description);
            }
        }
    } else {
        tree = new_cell();
        if (!tree) return NULL;
        tree->letter = *word;
        tree->child = add(tree->child, word+1, description);
        if (!tree->child) {
            tree->description = strdup(description);
        }
    }
    return tree;
}

int main(void) {
    Cell *tree;

    tree = NULL;

    tree = add(tree, "dormir", "acto de ressonar :)");
    tree = add(tree, "dormirei", "forma do verbo dormir");
    tree = add(tree, "dorminhoco", "aquele que dorme muito");

    list(tree);
    return 0;
}

```