ED – Estrutura de Dados – 2015/2

T3 – 3o. Trabalho Prático de Implementação

17 de novembro de 2015

1 Objetivo

O objetivo deste trabalho é desenvolver um corretor ortográfico simples, utilizando estruturas de dados diferentes para armazenar a lista de palavras do corretor e comparando o desempenho do programa ao se utilizar cada estrutura.

O programa a ser desenvolvido recebe como argumento o nome de um arquivo contendo a lista de palavras válidas do idioma desejado (dicionário) e um arquivo contendo o texto a ser verificado. Como saída, o seu programa deve imprimir na tela a lista de palavras do texto que não estavam no dicionário.

2 Requisitos da implementação

Nesta seção estão descritos os formatos dos arquivos de entrada e o formato da saída que seu programa deve exibir. *ATENÇÃO*: siga com muito cuidado os padrões estabelecidos nesta seção. Desta forma você terá a garantia que conseguirei usar o seu trabalho durante a correção. Trabalhos fora do padrão aqui estabelecido, no mínimo, perderão pontos, podendo até mesmo não serem corrigidos.

2.1 Formato de execução do programa

Seu programa deve ser executado de forma semelhante a dos primeiros trabalhos. Agora, porém, o seu trabalho recebe como entrada (por linha de comando) os nomes de dois arquivos: primeiro, o nome do dicionário a ser utilizado; depois, o nome do arquivo a ser verificado. Por exemplo, a linha de comando abaixo verifica o arquivo `t3_en` usando o dicionário `en`:

```
./trab3 data/dict/en data/text/t3_en
```

ATENÇÃO: Para evitar complicações com a codificação de caracteres extendidos (á, ç, etc), serão usados como entrada somente dicionários e textos em inglês. Isso facilitará (e muito) o desenvolvimento do trabalho.

2.2 Formato do arquivo de entrada

O arquivo de dicionário contém exatamente uma palavra por linha. Não assuma nenhuma constante para o número máximo de palavras possíveis no dicionário. De forma semelhante, leia cada linha do arquivo usando uma variável suficientemente grande para armazenar uma palavra, mas tente ser o mais econômico possível ao armazenar as palavras na estrutura de dados que indexará o seu dicionário. Use, para isto, alocação dinâmica (`malloc`) e a biblioteca de manipulação de *strings* da linguagem de C.

O arquivo contendo o texto a ser verificado não possui nenhum padrão previamente estabelecido. Não crie restrições arbitrárias e desnecessárias como, por exemplo, um limite máximo de tamanho para cada linha do arquivo.

Observe nos arquivos de exemplo disponibilizados que o texto a ser verificado contém sinais de pontuação, palavras em caixa-alta, palavras com primeira letra maiúscula, etc. Seu programa deve tratar todos estes casos. Para facilitar a sua implementação, seguem algumas dicas sobre como proceder:

- O processo de divisão do arquivo em palavras deve ser feito somente usando a função `fscanf` da biblioteca de *IO* da linguagem C.

- Após ler uma palavra do arquivo de entrada, os sinais de pontuação (`: ; , . ! ?` etc) estarão sempre mais à direita da palavra. Retire a pontuação usando a função `strtok` da biblioteca de *strings*.
- Não é obrigatório tratar casos mais complexos, como frases entre parênteses. Note que nenhum arquivo de exemplo possui estes casos. Você deve, no entanto, tratar palavras com apóstrofo.
- Tente primeiro buscar a palavra no dicionário da forma como ela apareceu no texto.
- No caso de palavras com partes em caixa-alta, busque primeiro a palavra na forma em que ela apareceu no texto. Caso não a encontre, converta a palavra para caixa-baixa e refaça a pesquisa. Caso também não a encontre, imprima na saída a palavra na **forma original**. Provavelmente você necessitará da função `strcpy` da biblioteca de *strings* para isto.

Para converter uma palavra para caixa-baixa, faça como no exemplo a seguir.

```
#include <ctype.h>
#include <stdio.h>

char* lowercase(char *str) {
  int  i = 0;
  while (str[i]) {
    str[i] = tolower(str[i]);
    i++;
  }
  return str;
}

int main (int argc, char *argv[]) {
  char word[] = "TO BE CONVERTED";
  printf("%s\n", lowercase(word));
  return 0;
}
```

Caso necessite, `http://www.cplusplus.com/reference/` contém mais documentação sobre as bibliotecas padrão da linguagem C, inclusive `string.h` que vocês usarão bastante neste trabalho.

2.3 Formato da saída

Seu programa deve produzir como saída na tela apenas a lista de palavras do arquivo texto que não existam no arquivo de dicionário. As palavras devem ser impressas uma por linha, na ordem em que apareceram no texto original e exatamente no formato original (letras maiúsculas, minúsculas, etc).

Valide o seu programa usando os gabaritos disponibilizados na página da disciplina. Garanta que seu programa gerará exatamente o mesmo resultado do gabarito usando o utilitário `diff` (no Linux). Exemplo:

```
./trab3 data/dict/en data/text/t3_en | diff data/output/t3_en -
```

ATENÇÃO: seu programa deve produzir como saída (NA TELA) SOMENTE o resultado no formato acima. Nada além disto. Ou seja, nenhuma mensagem e nenhuma formatação adicional deverá ser exibida. Isto é absolutamente necessário porque será usada uma bateria de testes para validação de seu trabalho, que verifica se sua resposta está correta baseado na saída do seu programa.

2.4 Estruturas de dados do índice de palavras

Você deve implementar duas versões do seu programa, que terão como diferença apenas a forma de implementação da estrutura de dados que guarda as palavras do dicionário. As duas variações que você deve implementar:

- Usando uma lista linear. Você pode escolher a forma de implementação da lista que achar melhor.
- Usando uma árvore de busca binária (*binary search tree – BST*).

As duas estruturas foram descritas em detalhes nas aulas. Compare o tempo de execução do seu programa para cada variação.

Para compilar o trabalho para cada versão, você deve criar no `Makefile` do seu projeto dois *targets* especiais, cada um preparado para gerar o sistema usando uma versão do módulo (arquivo .c) que implementa o dicionário. Os *targets* devem ser chamados de `list` e `bst`. O seu trabalho será compilado da seguinte forma:

```
$ make clean list
$ make clean bst
```

Note que você também deverá implementar um *target* chamado `clean` em seu `Makefile`.

2.5 Considerações gerais sobre a implementação

- Modularize o seu código adequadamente. Crie arquivos **.c** e **.h** para cada módulo do seu sistema. Em especial, crie arquivos exclusivos para manipular as estruturas de dados dos tipos abstratos de dados que você estiver representando.
- Utilize guarda de inclusão em todos os seus arquivos **.h**.
- Implemente os seus TADs Lista e BST como tipos opacos.
- Durante a correção, o seu trabalho será testado com uma bateria de testes diferente da disponibilizada como exemplo.
- Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Isto ajudará no momento da correção.
- Seu programa deve ser, obrigatoriamente, compilado com o utilitário `make`. Trabalhos enviados sem um `Makefile` não serão corrigidos. Crie um arquivo `Makefile` que gere como executável para o seu programa um arquivo de nome `trab3`.

3 Regras para desenvolvimento e entrega do trabalho

- **Data da Entrega**: O trabalho deve ser entregue até às 23:55 h do dia 15/12/2015 (terça-feira). Não serão aceitos trabalhos após essa data.
- **Grupo**: O trabalho é *individual*.
- **Linguagem de Programação**: Você deverá implementar o seu trabalho na linguagem C.
- **Ferramentas**:
  - O seu trabalho será corrigido no Linux.
  - Use a ferramenta Valgrind (`http://valgrind.org`) para garantir a ausência de *memory leaks*.
  - Crie um `Makefile` conforme explicado acima.
    Manual do Make: `http://www.gnu.org/software/make/manual/make.html`.
- **Como entregar**: Pela atividade criada no Moodle. Envie um arquivo compactado com todo o seu trabalho. Envie também um arquivo de texto simples contendo um relatório dos tempos de execução do seu programa com as duas variações das estruturas.

4 Avaliação

- O trabalho vale 20 pontos.
- Trabalhos com erros de compilação receberão nota zero.
- Caso seja detectado plágio (entre alunos ou da internet), todos os envolvidos receberão nota zero.

- Se o seu trabalho tiver *memory leaks* haverá desconto de pontos na nota do trabalho. Tal desconto será proporcional à gravidade das falhas detectadas no código.
- Serão levadas em conta, além da correção da saída do seu programa, a clareza e simplicidade de seu código.
- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre plágio, acima.)