Segundo Trabalho de Oficina de Computação (CI067)
Primeiro semestre de 2015

Histórico das revisões deste enunciado

07/05 - Eliminação da exigência de entrega de um Makefile.

04/05 - Correção do nome do diretório onde ficam os arquivos da entrega da solução.

27/05 - Inclusão de critérios de avaliação e especificação das diferentes formas de entrega da solução para as Turmas A e K.

1 Especificação do trabalho

Fazer um programa em linguagem C para ler uma moldura de palavras cruzadas simplificadas a partir da entrada padrão e preencher as lacunas da moldura com letras de palavras que obedecem às restrições da moldura. Mesmo que exista mais de uma solução para uma instância de problema, apenas uma delas deve ser dimpressa. Um arquivo de memória secundária pode ser usado como facilitador da entrada de dados. Para ilustrar a execução do programa com o nome de usuário `grs12`, veja o seguinte comando:

```
time ./grs12 < moldura1.txt
```

Veja ainda o texto de saída correspondente abaixo.

```
d o m * v
a * a t a
o * r i o
* * * a *

real    0m0.180s
user    0m0.001s
sys     0m0.016s
```

Note que o utilitário `time` do UNIX foi aplicado para que seja estimado o tempo de execução do programa. Note também o direcionamento de dados do arquivo `moldura1.txt` para compor a entrada padrão do programa executável `grs12`.
O formato dos dados de entrada é padronizado. Para o exemplo anterior, os dados de entrada são como segue.

```
[[ x1 x2  x3  *   x4 ]
 [ x5 *   x6  x7  x8 ]
 [ x9 *  x10 x11 x12 ]
 [ *  *   *  x13  *  ]]
pal[x1 x2 x3] pal[x6 x7 x8] pal[x10 x11 x12] pal[x1 x5 x9]
pal[x3 x6 x10] pal[x7 x11 x13] pal[x4 x8 x12]
dif[[x1 x2 x3] [x6 x7 x8] [x10 x11 x12] [x1 x5 x9]
     [x3 x6 x10] [x7 x11 x13] [x4 x8 x12]]
```

A primeira estrutura é matricial e representa uma moldura retangular de N linhas por M colunas (4×5 para o exemplo acima) da palavra cruzada. As lacunas $x_1, x_2, \ldots, x_k$ ($k = 13$ no exemplo acima) deverão ser preenchidas pelo algoritmo para formar palavras válidas do dicionário (ver abaixo). Onde há definição de lacuna (variável) no cruzamento entre uma linha e uma coluna, a letra a ser preenchida precisa coincidir para ambas as palavras (vertical e horizontal). Adicionalmente, as ocorrências dos asteriscos (`*`) indicam que não haverá preenchimento de letra na posição.
As estruturas que se seguem à da moldura representam restrições sobre os elementos da moldura. Elas podem ser de dois tipos:

- $pal[x_a x_b \ldots x_p]$ em que necessariamente a sequência de letras $x_a x_b \ldots x_p$ deve formar uma palavra existente no dicionário;
- $dif[[x_a x_b \ldots x_p] \ldots [x_i x_j \ldots x_q]]$ em que necessariamente cada sequência de letras é diferente de qualquer outra sequência da lista.

Tais restrições completam as representações básicas que derivam dos cruzamentos de palavras da moldura. Vale ressaltar que a quantidade de restrições e o tamanho da lista de itens de cada restrição só é conhecida em tempo de execução. Isso sugere que estruturas de dados alocadas dinamicamente na meméria devem ser utilizadas.
Note ainda que o domínio (valores possíveis) de cada uma das lacunas $x_1, x_2, \ldots, x_k$ compreende apenas as letras minúsculas do alfabeto (`a` até `z`, incluindo `k`, `w` e `y`).
O dicionário de palavras é bem pequeno e só contém itens de 3 (três) letras para facilitar o trabalho de desenvolvimento. Ele pode ser encontrado em:
`http://www.inf.ufpr.br/alexd/ofcomp/dados-t2-2015-1/dicionario_T2_2015_1.txt`
O desempenho de seu programa será avaliado por meio do consumo de tempo de execução. Para medir o consumo de tempo, veja o exemplo acima. Alguns arquivos de nome `moldura*.txt` contendo descrições de molduras e restrições para teste podem ser encontradas no link abaixo. Apenas os sete primeiros casos devem terminar sua execução com garantia em um tempo pequeno (poucos segundos). Os demais são apenas para ilustrar como a complexidade de tempo exponencial rapidamente cobra seu preço em relação ao tamanho da moldura de entrada.
`http://www.inf.ufpr.br/alexd/ofcomp/dados-t2-2015-1/`
Obsevações importantes:

- assuma que a quantidade de restrições sobre uma moldura varia bastante, o que sugere que arrays (vetores estáticos) não sejam usados como estruturas de dados mas, sim, tipos estruturados (com `typedef` e `struct`) com alocação dinâmica de memória (`malloc`) em linguagem C;
- não gaste seu tempo implementando qualquer mecanismo (*e.g.*, tabela hash ou qualquer estrutura de árvore) para acelerar a busca na moldura lida do arquivo TXT.

2 Produto a ser entregue

O trabalho deve ser desenvolvido INDIVIDUALMENTE.
O aluno deve entregar um pacote de software completo contendo os fontes em linguagem C dos programas solicitados e documentação. O pacote deve ser arquivado e compactado com `tar` e `bzip2` em um arquivo chamado `login.tar.bz2`, onde `login` é *login* do aluno nos sistemas computacionais do DINF/C3SL.
O pacote deve ter a seguinte estrutura de diretórios e arquivos:

- `./login/`: diretório principal (ao ser executado, o programa deve assumir que este é o diretório corrente);
- `./login/LEIAME`;
- `./login/src/`: diretório contendo todos os arquivos fonte em linguagem C (`*.c`).

Note que a extração dos arquivos em `login.tar.bz2` deve criar o diretório `login` contendo todos os arquivos e diretórios acima.

2.1 Documentação

O pacote deve conter um arquivo de documentação em texto plano (ASCII). Este arquivo, chamado `LEIAME`, deve conter as seguintes informações:

- autoria do software, isto é, nome e RA do aluno;
- lista dos arquivos e diretórios contidos no pacote e sua descrição (breve);
- um capítulo especial descrevendo os algoritmos e as estruturas de dados utilizadas, as alternativas de implementação consideradas e/ou experimentadas e os motivos que o levaram a optar pela versão entregue, as dificuldades encontradas e as maneiras pelas quais foram contornadas.
- *bugs* conhecidos;
- outras informações que forem julgadas importantes.

2.2 Arquivos Fonte

O programa deve ser implementado exclusivamente em linguagem C , e deve ser composto de no mínimo: a função `main`, e no mínimmo mais 3 (três) funções que devem ser criadas e usadas pelo programa como parte do algoritmo completo que resolve o problema.

3 Prazos de entrega

A entrega do trabalho será feita em 2 (duas) fases:

Fase 1 - 28/04/2015, 19:00h : O aluno deve entregar o programa com seu esqueleto definido em pelo menos 90%. Isto inclui definição de tipos de dados, constantes, funções e o código da função `main` indicando a chamadas das funções e a estrutura geral do programa. As funções deverão OBRIGATORIAMENTE ter um comentário indicando o objetivo da função, descrição dos parâmetros e dos valores de retorno, se houver, seu cabeçalho e protótipo.

Fase 2 - 14 de maio de 2015, 22:30H: O aluno deve ter seu programa compilando e executando e usar o tempo de aula para corrigir falhas e responder aos questionamentos que o professor fará durante a aula. ALUNOS QUE NÃO COMPARECEREM NESTA AULA TERÃO NOTA 0 (ZERO). Ao final da aula, o programa deverá ser entregue no estado em que estiver e será considerado o estado final do programa para correção pelo professor.

ATENÇÃO:	O não cumprimento de qualquer de uma das fases acima implicará em nota 0 (zero) para o aluno.

4 Meios de entrega

Os meios de entrega são diferentes para cada uma das duas turmas.

4.1 Turma A - professor Aldri

Para o professor Aldri, o trabalho deve ser enviado como anexo por e-mail, assim:

- A mensagem com o anexo deve ser enviada para `Aldri Santos <ci067ufpr@gmail.com>`, com o Assunto (*Subject*): **CI067 - Trabalho 02 2015**.
- No corpo da mensagem DEVE CONSTAR OBRIGATORIAMENTE o Nome e Número de Registro Acadêmico (RA) do aluno autor do trabalho;
- O aluno deverá considerar o trabalho como entregue SOMENTE APÓS RECEBER DO PROFESSOR UMA MENSAGEM DE CONFIRMAÇÃO DE RECEBIMENTO dentro de 24 horas desde o envio da mensagem;

4.2 Turma K - professor Alexandre

O trabalho deverá ser arquivado no diretório "`~alexd/CI067/`" (veja detalhes que seguem). O nome do arquivo terá como prefixo o nome-de-usuário do aluno no sistema do laboratório e, como extensão, ".tar.bz2" para indicar que seu conteúdo possui arquivos empacotados (pelo comando `tar`) e comprimidos (pelo comando `bzip2`).
Assim, por exemplo, se o seu nome de usuário no sistema é grs12 então o nome do arquivo será grs12.tar.bz2 (dentro do diretório "~alexd/CI067/"). Siga as instruções abaixo para entender como os arquivos serão empacotados e comprimidos:

1. faça tantos arquivos com o programa-fonte quantos você preferir para separar funções e as definições de headers para o código completo;
2. garanta que o código-fonte dos arquivos será compatível com o compilador gcc (GNU C) das servidoras de processamento do Departamento de Informática;
3. o nome do arquivo executável deverá ser o mesmo nome de usuário do aluno (exemplo: `grs12`)
4. coloque os arquivos em um diretório chamado `grs12`;
5. empacote e comprima esse diretório com os comandos abaixo para criar o arquivo `grs12.tar.bz2` assim:

   ```
   tar cf grs12.tar grs12
   bzip2 grs12.tar
   ```

6. finalmente, copie o arquivo `grs12.tar.bz2` para "`~alexd/CI067/`" tal como consta acima.

Não se esqueça de proteger completamente o arquivo criado, de maneira a permitir a leitura do mesmo apenas por você! Isso deve ser feito aplicando `chmod og-rwx grs12.tar.bz2` antes de efetuar a cópia com a preservação das permissões, assim: `cp -fp grs12.tar.bz2 ~alexd/CI067/` (se este comando terminar sem erros, isso já significa uma confirmação de entrega).
Não se preocupe com as permissões do professor que irá corrigir o trabalho. A correção dos trabalhos será parcialmente automatizada, sendo assim, é importante que todos os arquivos com as soluções individuais estejam no diretório citado acima, dentro do prazo estipulado.

5 Critério de Avaliação

APENAS OS TRABALHOS QUE COMPILAREM SEM ERROS SERÃO CORRIGIDOS. Se o trabalho não compilar ou acusar falha de segmentação (*Segmentation fault*) prematura durante os testes realizados pelo professor (sem que qualquer operação se efetue a contento), isso resultará em NOTA 0 (ZERO). Também receberão NOTA 0 (ZERO) trabalhos plagiados de qualquer fonte, e/ou com códigos idênticos ou similares. Além disso, apenas trabalhos entregues no prazo marcado receberão nota.
Legibilidade, elegância, eficiência, modularidade, coesão e acoplamento entre módulos, e uso adequado de estruturas de dados serão levados em conta na avaliação.
Os algoritmos de manipulação das estruturas de dados (inserção, ordenação, etc.) devem ser tão eficientes quanto possível, usando todos os conceitos vistos nas disciplinas de Algoritmos do Curso. Estruturas dinâmicas abordadas em Alg II, devidamente utilizadas, colaboram fortemente para uma avaliação positiva.
Caso seu programa possua *bugs* conhecidos, descreva-os no arquivo `LEIAME`. A documentação de um *bug* pode evitar que o aluno receba nota Zero.
Os itens de avaliação do trabalho e respectivas pontuações são:

Qualidade da documentação: 15 pontos

Qualidade do código: 25 pontos

Implementação: 60 pontos

Defesa: A defesa do trabalho será oral, e definirá a nota individual de cada aluno, de acordo com seu conhecimento a respeito do trabalho.

O item Qualidade da documentação se refere ao arquivo `LEIAME`. O item Qualidade do código inclui os comentários no código fonte e grau de portabilidade do código. O item Eficiência da implementação inclui análise de estruturas de dados, de algoritmos utilizados e seu tempo de execução.

6 Casos Omissos

Quaisquer dúvidas a respeito da especificação, entrega ou avaliação do trabalho deverão ser encaminhadas aos professores da disciplina, pessoalmente ou através de mensagens eletrônicas.
Bom trabalho!