	Centro Tecnológico Departamento de Informática
Disciplina: Programação II	Código: INF09330
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos

Especificação do Trabalho
Caça-palavras para dois jogadores

1 Introdução

Este trabalho tem como objetivo avaliar, de forma prática, os conceitos ensinados no curso de Programação II. O trabalho pode ser feito de forma incremental, isto é, à medida que o conteúdo for ensinado. A meta deste trabalho é implementar um caça-palavras para dois jogadores seguindo as especificações descritas neste documento e utilizando os conceitos ensinados em sala para manter boas práticas de programação.

Neste caça-palavras, cada jogador possui uma lista de palavras escondidas em um tabuleiro (representado por uma matriz) e deve tentar encontrar as próprias palavras, considerando uma quantidade limitada de tentativas, e antes do adversário. Caso encontre uma palavra de seu adversário, ao invés de uma sua, ele gera ponto para seu oponente.

2 Descrição do Trabalho

Neste trabalho, você deverá implementar o caça-palavras para dois jogadores. O programa deverá rodar no terminal, assim como todos os outros exercícios feitos em sala. Considere que os jogos podem ser realizados com configurações variadas (i.e., diferentes tamanhos de tabuleiro, diferentes listas de palavras, diferentes limites de jogadas, etc.), e que esses dados (chamados de configuração do jogo) serão passados para o programa antes do jogo iniciar através de um arquivo de configuração chamado *config.txt*.

Você coletará alguns dados durante o jogo que, a princípio, não são necessários para o jogo em si, mas são necessários para ganhar os pontos do trabalho. Perceba também que o jogo é completamente determinístico, portanto não há elementos aleatórios a serem considerados. Ao rodar o jogo várias vezes com as mesmas configurações e as mesmas jogadas, os resultados sempre serão os mesmos.

O primeiro passo do jogo é ler o arquivo de configuração (*config.txt*) e preencher a matriz de letras. Em seguida, o jogo perguntará o nome dos jogadores. Então, cada jogador terá, alternadamente, a sua vez de jogar informando as coordenadas da posição da matriz que deseja revelar. Cada posição selecionada deverá ser apresentada com letra maiúscula. Assim que o jogador informa uma posição, o programa deverá verificar se ela contém a primeira letra de uma palavra. Em caso positivo, a palavra deverá ser revelada por completo seguindo a regra: as letras ainda ocultas na matriz deverão ser apresentadas em minúsculo, ao passo que as letras já reveladas na matriz em jogadas anteriores (incluindo a jogada atual) deverão ser apresentadas em maiúsculo (ver exemplos adiante). Esse processo se repetirá até que todas as palavras de um jogador sejam encontradas, ou até que se atinja o número máximo de turnos do jogo.

Você deve considerar que cada palavra pode estar escrita de cima para baixo, de baixo para cima, e assim sucessivamente na horizontal e nas diagonais, mas nunca haverá duas palavras iniciando na mesma posição da matriz. Ao final do jogo, o programa deverá exibir o vencedor e criar alguns arquivos contendo estatísticas do jogo. Cada vez que um jogador encontra uma palavra pertencente à sua lista, recebe 2 pontos; caso a palavra pertença à lista do oponente, esse é quem

recebe 1 ponto. O vencedor será aquele que, ao término do jogo, tiver mais pontos. Em caso de empate, imprimir empate.

2.1 Funcionamento Detalhado do Programa

O programa será chamado de *trabalhoprog2* (após a compilação), portanto o projeto no Netbeans também deverá ser chamado de *trabalhoprog2*. Sua execução será feita através da linha de comando (i.e. pelo cmd, console, ou terminal) e permitirá a passagem de parâmetros, descritos mais adiante.

Parâmetros de inicialização (passados na chamada do programa pela linha de comando): Para garantir o correto funcionamento do programa, o usuário deverá informar, ao executar o programa pela linha de comando, o caminho do diretório de teste que contém o arquivo de configuração (ex. de execução: "*./trabalhoprog2 /maquinadorprofessor/test_1*"). Nesse mesmo diretório, serão gerados os arquivos de saída, porém, estes deverão ir para uma pasta específica, chamada "saida", dentro desse diretório informado (ex. da pasta dos arquivos de saída do exemplo anterior seria "*/maquinadorprofessor/test_1/saida*"). Considere um caminho com tamanho máximo de 1000 caracteres. O programa deverá verificar a presença desse parâmetro informado na linha de comando. Caso o usuário tenha esquecido de informar o nome do diretório, o programa deverá exibir (ex. "ERRO: O diretório de arquivos de configuração não foi informado"), e finalizar sua execução. As mensagens de erro facilitarão a detecção de possíveis erros.

Criar jogo: Nessa operação, o programa deverá ler informações do arquivo de configuração do jogo para a memória e preparar o ambiente de jogo. O arquivo de configuração terá sempre o nome *config.txt*, e estará sempre no diretório de teste informado pela linha de comando. Caso o programa não consiga ler o arquivo, ele deverá ser finalizado e imprimir uma mensagem informando que não conseguiu ler o arquivo (OBS: a mensagem deve conter o caminho e nome do arquivo que ele tentou ler).

- Organização do arquivo de configuração do jogo (*config.txt*):
    - A primeira linha do arquivo conterá um inteiro que indica o número máximo de turnos do jogo, ou seja, número de jogadas para cada jogador. Por exemplo, se o inteiro for 10, cada jogador poderá jogar 10 vezes;
    - A segunda linha do arquivo conterá um inteiro que indica o tamanho da matriz (no máximo 100x100). Isto é, se o inteiro for 5, a matriz será 5x5;
    - Em seguida, teremos a matriz propriamente dita, com todas as letras sempre em minúsculo, sendo cada linha da matriz dada em uma linha do arquivo e seus elementos dispostos sem separação por espaço;
    - A próxima linha após o fim da matriz conterá um inteiro que indica o número de palavras de cada jogador (no máximo 25). Isto é, se o inteiro for 5, haverão 10 palavras, sendo as primeiras 5 do primeiro jogador e as restantes do segundo;
    - Em seguida, teremos a lista de palavras (uma em cada linha e cada uma com no máximo 16 letras e todas em minúsculo);
    - Exemplo de arquivo *config.txt*:

```
10
5
apamp
zebra
arean
ruavd
scasa
5
azar
casa
panda
ave
area
zebra
mapa
peru
reus
rua
```

- Caso a leitura do arquivo de configuração seja bem-sucedida, em seguida, o programa deverá perguntar o nome dos jogadores na entrada/saida padrão. Os nomes devem ser simples (ou seja, apenas uma palavra e conter no máximo 16 caracteres). O formato da entrada e saída esperado é descrito abaixo, sendo as linhas 1 e 3 exemplos das saídas e as linhas 2 e 4 exemplo das entradas:

L	Terminal
1	Nome do Jogador 1:
2	Renan
3	Nome do Jogador 2:
4	Thiago

Realizar o jogo: A cada jogada, o jogo deve, primeiramente, mostrar a lista de palavras restantes (i.e., aquelas que não foram encontradas ainda) na ordem em que foram lidas do arquivo de configuração e, a pontuação de cada jogador (inicialmente, 0). Exemplo da saída do terminal:

```
Terminal
```

```
|            Palavras Restantes           |
|Renan           (00)|Thiago          (00)|
|--------------------|--------------------|
|azar                |zebra               |
|casa                |mapa                |
|panda               |peru                |
|ave                 |reus                |
|area                |rua                 |
```

E, então, mostrar o tabuleiro. Inicialmente, nenhuma letra terá sido revelada. Então todas conterão " - ", pois estão ocultas. Exemplo da saída do terminal:

```
Terminal
```

```
    00 01 02 03 04
00| -  -  -  -  - |
```

```
01| -  -  -  -  - |
02| -  -  -  -  - |
03| -  -  -  -  - |
04| -  -  -  -  - |
```

Em seguida, o jogo pede ao primeiro jogador que informe as coordenadas para jogar. Exemplo da saída (linha 1) e entrada (linha 2) no terminal:

L	Terminal
1	Renan por favor entre com as coordenadas para a sua jogada:
2	0 0

No exemplo acima, o jogador inseriu as coordenadas (0,0). Como a posição (0,0) da matriz contém a primeira letra da palavra azar, o jogo exibe a lista de palavras com a palavra azar removida, e dois pontos a mais para o Renan por ter acertado uma palavra de sua lista. Em seguida, exibe novamente o tabuleiro com a palavra revelada com a letra A em maiúsculo. Exemplo da saída do terminal:

```
Terminal
```

```
|             Palavras Restantes          |
|Renan           (02)|Thiago          (00)|
|--------------------|--------------------|
|casa                |zebra               |
|panda               |mapa                |
|ave                 |peru                |
|area                |reus                |
|                    |rua                 |

    00 01 02 03 04
00| A  -  -  -  - |
01| z  -  -  -  - |
02| a  -  -  -  - |
03| r  -  -  -  - |
04| -  -  -  -  - |
```

Depois, o jogo pede ao segundo jogador que informe as coordenadas para jogar. Exemplo da saída (linha 1) e entrada (linha 2) no terminal:

L	Terminal
1	Thiago por favor entre com as coordenadas para a sua jogada:
2	4 1

Neste exemplo, o segundo jogador inseriu as coordenadas (4,1), que contém a primeira letra da palavra casa. Como esta palavra pertence a Renan, Renan ganhou um ponto, e a palavra casa foi removida da lista de palavras de Renan e revelada com a letra C em maiúsculo. Exemplo da saída do terminal:

```
Terminal
```

```
|             Palavras Restantes          |
|Renan           (03)|Thiago          (00)|
|--------------------|--------------------|
|panda               |zebra               |
|ave                 |mapa                |
```

```
|area                |peru                |
|                    |reus                |
|                    |rua                 |

    00 01 02 03 04
00| A  -  -  -  - |
01| z  -  -  -  - |
02| a  -  -  -  - |
03| r  -  -  -  - |
04| -  C  a  s  a |
```

Assim, termina o primeiro turno e volta a ser a vez do primeiro jogador. Caso ele escolha uma coordenada que não acerta nenhuma palavra, apenas a letra é revelada em maiúsculo. Por exemplo, se Renan jogar na coordenada (1,2), o jogo mostrará:

Terminal

```
|             Palavras Restantes          |
|Renan           (03)|Thiago          (00)|
|--------------------|--------------------|
|panda               |zebra               |
|ave                 |mapa                |
|area                |peru                |
|                    |reus                |
|                    |rua                 |

    00 01 02 03 04
00| A  -  -  -  - |
01| z  -  -  -  - |
02| a  -  -  -  - |
03| r  -  -  -  - |
04| -  C  a  s  a |
```

Novos turnos acontecem até que seja atingido o número máximo (neste exemplo, 10), ou até que a lista de palavras de um dos dois jogadores acabe. Uma vez que o jogo acabe, um jogador ganha se ele tiver mais pontos ou o jogo empata se eles tiverem quantidades iguais de pontos. O jogo deve imprimir o resultado ("<nome_do_jogador> Ganhou!!!" ou "Empate") logo após mostrar o seu estado atual de palavras, pontos e tabuleiro (como feito para cada jogada). Exemplo da saída do terminal:

Terminal

```
|            Palavras Restantes          |
|Renan           (07)|Thiago          (06)|
|--------------------|--------------------|
|ave                 |                    |

    00 01 02 03 04
00| A  P  A  M  P |
01| Z  E  B  R  a |
02| A  r  E  a  n |
03| R  U  a  V  d |
04| s  C  a  s  a |

Renan Ganhou!!!
```

Note que o jogo acabou quando a lista de palavras de Thiago terminou, mas quem ganhou foi Renan, já que possui mais pontos. Exemplos completos de entrada e respectiva saída esperada (no terminal e nos arquivos gerados) são fornecidos junto com essa especificação.

Tratamento de Jogadas não ordinárias: Nos casos de teste, haverá casos em que os jogadores farão jogadas repetidas (i.e., tentar jogar em uma posição anteriormente jogada por ele ou pelo seu adversário) e, jogadas inválidas (i.e., jogadas em posições fora do tabuleiro). As jogadas serão sempre dois inteiros separados por um espaço e terminados com uma quebra de linha.

Nos casos de jogadas repetidas, seu programa deverá imprimir a seguinte mensagem para jogador:

```
Terminal
Coordenadas já jogadas.<nome_do_jogador> por favor entre com novas
coordenadas para a sua jogada:
```

Solicitando uma nova jogada. Um tratamento similar deve ser feito no caso de jogadas inválidas, porém a mensagem deverá ser a seguinte:

```
Terminal
Coordenadas fora do tabuleiro.<nome_do_jogador> por favor entre com
novas coordenadas para a sua jogada:
```

Gerar arquivo de inicialização: Assim que o programa ler o arquivo de configuração e os nomes dos jogadores, ele deverá escrever, na pasta de saída do caso de teste em questão, um arquivo (*Inicializacao.txt*). O arquivo conterá, para cada jogador, o nome do jogador, a lista de palavras (na ordem em que aparece no arquivo de configuração), a maior palavra (em caso de empate deve ser considerada a primeira que aparece na ordem em que foi dada a lista), o número de palavras iniciadas com cada letra (considerando todas as primeiras letras) em ordem alfabética (mostrando apenas letras que iniciam alguma palavra). Exemplo do arquivo de *Inicializacao.txt:*

```
--Jogador 1--
Nome: Renan
Palavras:
azar
casa
panda
ave
area
Maior Palavra:
panda
Letras Iniciais:
a -> 3
c -> 1
p -> 1

--Jogador 2--
Nome: Thiago
Palavras:
zebra
mapa
peru
reus
rua
Maior Palavra:
zebra
Letras Iniciais:
m -> 1
p -> 1
r -> 2
z -> 1
```

Gerar arquivo de estatísticas para análise: Ao final do jogo, o programa deverá escrever, na pasta de saída do caso de teste em questão, um arquivo (*Estatisticas.txt*) contendo algumas estatísticas básicas sobre a partida. Para isso, será necessário que ao longo do jogo o programa armazene algumas informações durante o jogo, para que seja possível gerar essas estatísticas. Esse arquivo deverá conter, para cada jogador, o número de pontos feitos, o número de pontos recebidos, e o total de pontos. Em seguida, deverá exibir a lista de palavras com o número de pontos obtidos com cada palavra (2 quando a palavra foi encontrada pelo dono, 1 quando a palavra foi encontrada pelo outro jogador e 0 quando ela não foi encontrada), e o número da jogada em que a palavra foi encontrada, assumindo-se a primeira jogada a partir de 1 e considerando-se somente as jogadas válidas. As palavras devem estar ordenadas pelos pontos que foram obtidos (decrescente) e usando a palavra, ordem alfabética (crescente), como desempate. Exemplo do arquivo de *Estatisticas.txt:*

```
--------------
--- PONTOS ---
--------------

Renan
Feitos: 6
Recebidos: 1
Total: 7

Thiago
Feitos: 2
```

```
Recebidos: 4
Total: 6

--------------------------
--- PALAVRAS POR PONTOS ---
--------------------------

|P|    Palavra     | T |
|-|----------------|---|
|2|area            |013|
|2|azar            |001|
|2|panda           |011|
|2|zebra           |010|
|1|casa            |002|
|1|mapa            |015|
|1|peru            |009|
|1|reus            |005|
|1|rua             |007|
|0|ave             |000|
```

Gerar arquivo com o mapa ordenado (Bônus): Ao final do jogo, o programa deverá escrever, na pasta de saída do caso de teste em questão, um arquivo (*MapaOrdenado.txt*) contendo algumas estatísticas básicas sobre a partida. Cada letra será substituída pelo número representando a ordem em que a palavra a qual ela pertence foi revelada (i.e., primeira palavra a ser revelada recebe 1, a segunda recebe 2, e assim por diante). As palavras que forem aparecendo primeiro terão prioridade, ou seja, se uma palavra tiver sobreposição com outra já escrita, ela não deverá sobrescrever o que já estiver preenchido. As células que não forem reveladas, deverão exibir "--". Exemplo do arquivo de *MapaOrdenado.txt:*

```
01 05 09 09 07
01 05 06 03 06
01 05 03 08 07
01 03 04 -- 07
03 02 02 02 02
```

Alguns casos de teste serão fornecidos para o aluno como exemplo, incluindo arquivos de entrada e seus respectivos arquivos de saída. Os arquivos de entrada serão *config.txt*, contendo as configurações do jogo, e *entrada.txt*, contendo os nomes dos jogadores e as jogadas a serem redirecionadas para a entrada padrão (esse arquivo pode conter mais jogadas do que o necessário para finalizar o jogo e cabe ao programa finalizar na hora certa). Os arquivos de saída serão de dois tipos: os gerados pelo programa na pasta "saida" do diretório de teste (*Estatisticas.txt*, *Inicializacao.txt* e *MapaOrdenado.txt*), contendo as informações descritas acima, e o redirecionado da saída padrão, denominado *saida.txt* nos casos de teste fornecidos, contendo as informações impressas no terminal. O aluno deverá utilizar tais arquivos para testes durante a implementação do trabalho. É responsabilidade do aluno criar novos arquivos para testar outras possibilidades de erro do programa e garantir seu correto funcionamento. O trabalho será corrigido usando, além dos arquivos fornecidos, outros arquivos (específicos para a correção e não disponibilizados para os alunos) seguindo a formatação descrita nesse documento e utilizada nos arquivos exemplos. Em caso de dúvida, pergunte ao professor. O trabalho será corrigido via script, portanto o uso de arquivos com nome ou formatação diferente do especificado poderá acarretar em incompatibilidade durante a correção do trabalho e consequentemente na

impossibilidade de sua correção (recebendo nota zero). Portanto, siga estritamente o formato estabelecido para as entradas e saídas.

2.2 Implementação

A implementação deverá seguir os conceitos de modularização e abstração apresentados em sala. O trabalho terá uma componente subjetiva que será avaliada pelo professor para verificar o grau de uso dos conceitos ensinados. Portanto, além de funcionar, o código deverá estar bem escrito para que o aluno obtenha nota máxima.

É extremamente recomendado utilizar algum programa para fazer as comparações do resultado final do programa, isto é, os arquivos de saída gerados, poderão ser comparados com o arquivo de saída esperada (fornecido pelo professor) utilizando o comando *diff*, como visto em sala. O *meld* é uma alternativa gráfica para o *diff*, se você preferir. Esse programa faz uma comparação linha a linha do conteúdo de 2 arquivos. Diferenças na formatação poderão impossibilitar a comparação e consequentemente a correção do trabalho. O programa será considerado correto se gerar a saída esperada idêntica à fornecida com os casos de teste.

3 Regras Gerais

O trabalho deverá ser feito individualmente e pelo próprio aluno, isto é, o aluno deverá necessariamente conhecer e dominar **todos** os trechos de código criados. O aluno deverá necessariamente utilizar o Netbeans para desenvolvimento do trabalho!

Cada aluno deverá trabalhar independente dos outros, não sendo permitido a cópia ou compartilhamento de código. O professor irá fazer verificação automática de plágio. Trabalhos identificados como iguais, em termos de programação, serão penalizados com a nota zero. Isso também inclui a pessoa que forneceu o trabalho, sendo, portanto, de sua obrigação a proteção de seu trabalho contra cópias ilícitas. **Proteja seu trabalho e não esqueça cópias do seu código nas máquinas de uso comum (ex. laboratório).**

3.1 Entrega do Trabalho

O trabalho deverá ser entregue por email (para: todsantos@inf.ufes.br) até as 23:59 do dia **07/07/2019**. O assunto da mensagem deverá seguir o padrão: [TRABALHO_PROG2_2019_1] <NomeDoAluno>.

Exemplo do assunto da mensagem para o aluno Fulano da Silva:

- [TRABALHO_PROG2_2019_1] FulanoDaSilva

O arquivo do programa (*.c), incluindo o diretório “nbproject” e o arquivo “Makefile” gerados pelo netbeans deverão ser compactados em um único arquivo <nome_do_aluno>.zip e enviado como anexo da mensagem. Lembrando que o anexo não deverá conter arquivos compilados (".o") ou executáveis (e.g., ".exe") sob o risco de bloqueio de email contendo arquivos executáveis. O código será compilado posteriormente em uma outra máquina. A correção será feita de forma automática (via *script*), portanto a entrega incorreta do trabalho, ou seja, fora do padrão, poderá acarretar na impossibilidade de correção do trabalho e consequentemente na atribuição de nota zero. A pessoa corrigindo não terá a obrigação de adivinhar nome de arquivos, diretórios ou outros. **Siga estritamente o padrão estabelecido!**

Dica para teste de envio do trabalho: siga os passos anteriores; envie um email para você mesmo; descompacte o conteúdo para uma pasta; abra um terminal e mude o diretório corrente para a pasta de descompactação; execute o comando "make all" que deverá compilar todo o projeto e gerar um arquivo executável em algum lugar da pasta "./dist/Release/.../*trabalhoprog2.exe*"; coloque os arquivos de teste em um diretório qualquer (ex. "/minhapasta/test_1"); execute o programa gerado passando o caminho do diretório com os arquivos de teste, e veja se funciona corretamente. Por fim, abra o projeto com o Netbeans para ver se está tudo como esperado. **Se não funcionou para você, não vai funcionar para mim!**

Os dados necessários para executar o comando "make" são gerados automaticamente pelo Netbeans dentro da pasta do projeto.

3.2 Pontuação e Processo de Correção

Trabalhos entregues após o prazo não serão corrigidos (nota zero). O trabalho será pontuado de acordo com sua implementação e a tabela abaixo. Os pontos descritos na tabela não são independentes entre si, isto é, alguns itens são pré-requisitos para obtenção da pontuação dos outros. Por exemplo, gerar o arquivo de estatísticas depende de realizar a simulação corretamente. Código com falta de legibilidade e modularização pode perder ponto conforme informado na tabela. Erros gerais de funcionamento, lógica ou outros serão descontados como um todo.

Percebam que no melhor dos casos os pontos da tabela abaixo somam 11 ao invés de 10. Isso foi feito propositalmente para ajudar os alunos esforçados com um ponto extra. Esse ponto, caso obtido, irá complementar uma das notas, do trabalho ou das provas parciais do semestre. Prioridade será dada para a nota com maior peso, porém não ultrapassando a nota máxima.

Item	Quesitos	Ponto
Inicialização	Ler o arquivo de configuração Gerar corretamente o arquivo de Inicialização	1
Lógica do jogo	Receber corretamente as entradas dos jogadores Mostrar as informações do jogo como especificado Computar os pontos e mostrar o ganhador da partida	5
Criar estatísticas	Gerar corretamente o arquivo com as estatísticas.	3
Tratamento de jogadas não ordinárias	Tratamento de jogadas repetidas Tratamento de jogadas fora do tabuleiro	1
Legibilidade e Modularização	Falta de: • Uso de comentários; • Identação do código; • Uso de funções;	-2

	• Uso de tipos de dados definidos pelo usuário	
Função bônus	Gerar corretamente o Mapa Ordenado	1

Processo de correção do trabalho: O trabalho será corrigido, considerando-se 3 turnos: primeira entrega, aula de correção e entrevista. A Primeira Entrega ocorrerá na data definida na seção de Entrega do Trabalho. Ela gerará uma nota da Primeira Entrega (NPE) que servirá como base para a nota final do trabalho, ou seja, o trabalho será pontuado de acordo com o que for entregue nesta etapa. A Aula de Correção ocorrerá no dia **08/07/2019** (horário de aula). Nesta Aula, o aluno terá a possibilidade, dentro do tempo de aula, de corrigir erros encontrados no trabalho. Funcionalidades novas implementadas nessa etapa não serão contabilizadas. As partes corrigidas ganharão um percentual (0% a 100%) do ponto cheio, de acordo com o tipo de correção feita pelo aluno, ou seja, ela gerará a nota da Aula de Correção (NAC), em que NPE ≤ NAC ≤ 10. Caso o aluno perca essa aula, a correção será feita baseada somente na primeira entrega, ou seja, NAC = NPE. Para fazer a correção, o aluno receberá todos os casos de teste escondidos do trabalho. Ele receberá também o *script* que rodará o programa de forma automática (seguindo as regras definidas nesta especificação) para que ele possa consertar possíveis erros de implementação (por exemplo, formatação das saídas, nomes de arquivos, lógica, etc.). No final dessa aula, o aluno deverá enviar o programa corrigido com comentários explicando cada alteração feita. Cada comentário deverá ser iniciado com a tag "//CORRECAO:" para indicar o que foi alterado. Modificações sem as devidas explicações e tags poderão ser desconsideradas e não pontuadas. Vale a pena ressaltar que não será possível aceitar entregas fora do prazo, dado que os casos de teste escondidos serão liberados na Aula de Correção. A Entrevista ocorrerá em uma data posterior a aula de correção e fora do horário de aula (a ser agendado). Ela tem o intuito de identificar alunos que tiveram o trabalho feito por terceiros ou, por exemplo, que tiveram ajuda demasiada. A nota, NAC, obtida ao final das duas primeiras etapas acima será ponderada com uma nota de entrevista (NE) sobre o trabalho. A nota final do trabalho será calculada com NAC x NE, em que $0 \leq NE \leq 1$. A entrevista avaliará o conhecimento do aluno a respeito do programa desenvolvido. Como pode ser visto, a nota da entrevista não servirá para aumentar a nota do trabalho, mas sim para diminuir quando for identificada uma falta de conhecimento sobre o trabalho entregue.

4 Considerações Finais

Não utilizar acentos no trabalho.

5 Erratas

Esse documento descreve de maneira geral as regras de implementação do trabalho. É de responsabilidade do aluno garantir que o programa funcione de maneira correta e amigável com o usuário. Qualquer alteração nas regras do trabalho será comunicada em sala e no portal do aluno. É responsabilidade do aluno frequentar as aulas e se manter atualizado em relação às especificações do trabalho. Caso seja notada qualquer tipo de inconsistência nos arquivos de testes disponibilizados, comunique imediatamente ao professor para que ela seja corrigida e reenviada para os alunos.