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

Especificação do Trabalho
Bingo

1 Introdução

Esse 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 desse trabalho é implementar um simulador de jogo de bingo seguindo as especificações descritas nesse documento e utilizando os conceitos ensinados em sala para manter boas práticas de programação.

Bingo é um jogo popular, no qual existe uma banca responsável por correr as pedras e existem jogadores com cartelas. Cada cartela possui vários números que podem ser sorteados pela banca. Antes de começar o jogo, os jogadores compram (ou pegam) suas cartelas com a banca. Quando o jogo começa, a banca passa a sortear uma pedra por vez. A cada pedra sorteada, os jogadores marcam os respectivos números em suas cartelas que contêm a pedra sorteada. O primeiro jogador a marcar todas as pedras de uma de suas cartela, ganha o jogo.

2 Descrição do Trabalho

Nesse trabalho, um programa para simular um jogo de bingo deverá ser implementado. 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 cartela, diferentes números de participantes, 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.

De maneira geral, o programa irá iniciar pela linha de comando (terminal) e irá ler um arquivo contendo as configurações do jogo, como por exemplo o tamanho das cartelas, o nome dos jogadores e a quantidade de cartelas de cada um deles, entre outros dados. A localização desse arquivo será passada como parâmetro (pela linha de comando) para o programa. O primeiro passo do jogo será criar as cartelas (com números “aleatórios”) e distribui-las para os jogadores. As cartelas criadas deverão ser salvas em um arquivo que será considerado uma das saídas do programa. Com as cartelas em mãos, o programa deverá iniciar o jogo efetivamente, ou seja, começar a sortear as pedras para que os jogadores possam marcar em suas cartelas. Como estamos fazendo um simulador, o próprio programa fará o papel dos jogadores e marcará as cartelas. Durante o jogo, o estado das cartelas será exibido na tela. O jogo terminará quando algum jogador ganhar. No final do jogo, o programa deverá exibir o resultado (jogador vencedor ou jogadores empatados). Além disso, o programa deverá salvar um arquivo contendo as estatísticas do jogo (também considerado uma saída do programa), ou seja, contendo uma lista das cartelas do jogo seguindo uma ordenação específica a ser descrita adiante.

Perceba que em um jogo de bingo real, a banca sorteia os números e os jogadores (i.e., as pessoas jogando) marcam em suas cartelas, porém no ambiente simulado, tudo isso ocorrerá dentro do programa sem interação com o usuário. Ou seja, programa sorteará os números, marcará aquele número nas cartelas dos jogadores, e verificará quem ganhou sem que o usuário interfira.

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. Ao iniciar, o programa deverá executar uma série de operações na sequência, e informar ao usuário o estado atual do jogo. As operações a serem realizadas pelo programa são descritas a seguir (obedecendo a ordem): criar jogo, realizar jogo, gerar arquivo de estatísticas para análise, além de uma função bônus opcional. A descrição dos parâmetros de inicialização e das operações a serem realizadas pelo programa são apresentadas em detalhes a seguir.

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 contendo os arquivos a serem processados (ex. de execução: "*./trabalhoprog2 /maquinadorprofessor/test_1*"). Considere um caminho com tamanho máximo de 1000 caracteres. O programa deverá verificar a presença do parâmetro. Caso o usuário tenha esquecido de informar o nome do diretório, o programa deverá exibir (ex. "ERRO: O diretorio de arquivos de configuração nao foi informado!"), e finalizar sua execução.

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 programa deverá procurar o arquivo de configurações do jogo (descrito em mais detalhes abaixo), denominado *jogo_config.txt*. Ele está localizado dentro do diretório *input* que é uma subpasta do caminho informado pelo usuário durante a chamada do programa, e ler sua informação para memória. Caso o programa não consiga ler o arquivo, ele deverá ser finalizando e imprimir a mensagem informando que não conseguiu ler o arquivo (OBS: a mensagem deve conter o caminho e nome do arquivo que ele tentou ler). Caso a leitura seja bem sucedida, em seguida, o programa deverá criar as cartelas de cada jogador e salvar um arquivo, denominado *cartelas_jogador.txt*, no diretório *output* que é uma subpasta do caminho informado pelo usuário durante a chamada do programa. Se houver falha na operação, o programa deverá imprimir "ERRO: Nao foi possível salvar arquivo com as cartelas dos jogadores."

- Organização do arquivo de configuração do jogo (*jogo_config.txt*): A primeira linha do arquivo *jogo_config.txt* conterá cinco números inteiros separados por ";". O primeiro número representará a semente aleatória da banca (explicado mais adiante). O segundo representará a quantidade de pedras da banca a serem sorteadas (no máximo 900). O terceiro e o quarto representam a quantidade de linhas e de colunas das cartelas respectivamente (no máximo 20 cada). O quinto representará a quantidade de jogadores (no máximo 20). As linhas subsequentes apresentarão os nomes dos jogadores e suas respectivas quantidades de cartelas (no máximo 10 cada).

```
3;90;6;5;3
Tripa Seca;3
Geronimo da silva;1
Fulano de Tal;2
```

O programa deverá criar tantas cartelas quanto necessário para suprir as demandas dos jogadores. As cartelas terão um *id* representado por um inteiro começando de zero e incrementando sequencialmente de acordo com a quantidade de cartelas de cada jogador descritas no arquivo de configurações. No exemplo acima, o primeiro jogador receberia as cartelas com *id* 0, 1 e 2, o segundo receberia a com *id* 3 e o terceiro as com *id* 4 e 5. Para garantir a reprodutibilidade dos casos de teste passados como exemplo, as cartelas deverão ser criadas na ordem de seus *id*, ou seja, a cartela 0 será criada primeiro, depois a 1 e assim por diante. Os números que comporão cada cartela deverão ser gerados "aleatoriamente" com o tipo

tGeradorAle que será fornecido juntamente com esse documento e não deve ser alterado no trabalho. Ele deverá ser incluído e compilado juntamente com o programa de vocês. O tipo possui duas funções: *ReiniciaGerador* que dois recebe parâmetros, uma semente que determina a aleatoriedade e uma quantidade de números a serem sorteados; e *GeraProxNumero* que retorna o próximo número sorteado. A função *ReiniciaGerador* deverá necessariamente ser chamada antes da criação de cada cartela passando o *id* da cartela como semente e a quantidade de pedras lida do arquivo de configuração como parâmetros. Uma vez iniciado o gerador, a função *GeraProxNumero* deverá ser chamada repetidamente para gerar todos os números necessários para cada cartela. Os números de cada cartela deverão ser posicionados obedecendo a seguinte ordem: preencher a cartela do menor número para o maior começando na coluna da esquerda de cima para baixo, e passando sequencialmente pelas outras colunas também de cima para baixo (ver exemplo de uma cartela no arquivo *cartelas_jogador.txt*).

- Organização do arquivo de cartelas do jogador (*cartelas_jogador.txt*): Esse arquivo deverá conter uma lista dos nomes dos jogadores seguido de suas respectivas cartelas. As cartelas devem ser impressas com o *id* e os valores que a compõem. Os números das cartelas deverão ser impressos com 3 dígitos completando com zeros a esquerda e posicionados entre "|". Cada linha da cartela deverá ser impressa com um TAB (ou seja, um "\t" antes da impressão da linha). Veja exemplo da formatação nos arquivos fornecidos com a especificação. Veja abaixo um exemplo de parte de um arquivo de *cartelas_jogador.txt*.

```
Jogador:Thiago Oliveira dos Santos
Cartela ID:0
        |001|020|035|057|072|
        |002|021|036|059|077|
        |003|022|044|068|080|
        |005|027|045|069|082|
        |013|029|050|070|089|
        |014|031|054|071|090|

Cartela ID:1
        |002|010|025|042|063|
        |004|011|028|043|069|
        |006|014|029|049|079|
        |007|016|030|056|081|
        |008|017|033|059|082|
        |009|018|037|061|084|
...
...
...
```

Realizar jogo: Uma vez preparado, a banca poderá começar a sortear pedras até o fim do jogo. Após cada sorteio, os jogadores deverão marcar a pedra sorteada em suas cartelas e verificar se alguma delas foi totalmente preenchida. O jogo acabará quando algum jogador preencher uma cartela completamente. Os números sorteados pela banca deverão ser gerados pelo tipo *tGeradorAle* descrito acima. Antes de começar a sortear os números, a função *ReiniciaGerador* deverá ser chamada passando a semente da banca lida do arquivo de configuração do jogo e a quantidade de pedras também lida do mesmo arquivo. O sorteio deverá ser feito com a função *GeraProxNumero*. A cada sorteio o estado do jogo deverá ser impresso na tela. O estado do jogo impresso na tela poderá ser redirecionado para arquivo para permitir comparações com os arquivos fornecidos como exemplo, *saida.txt* (igual fazemos no BOCA).

- Organização da informação do estado a ser impresso na tela (ex. *saida.txt*): Após a marcação das cartelas, o estado do jogo deverá ser impresso na tela informando a pedra sorteada, o nome de cada jogador com suas respectivas cartelas. Posições marcadas na cartela deverão ser impressas com "---". No final, o programa deverá imprimir "Jogador Venceu!" seguido na próxima linha do nome do vencedor, ou "Jogadores

Empataram!" seguido nas próximas linhas dos nomes dos vencedores empatados (na ordem informada no arquivo de configuração do jogo). Veja abaixo um exemplo de parte de um arquivo de *saida.txt*.

```
Pedra:70
Jogador:Tripa Seca
Cartela ID:0
        |001|020|035|057|072|
        |002|021|036|059|---|
        |003|022|044|068|080|
        |005|027|045|069|082|
        |013|029|050|070|089|
        |014|031|054|071|090|

Cartela ID:1
        |002|010|025|042|063|
        |004|011|028|043|069|
        |006|014|029|049|079|
        |007|016|030|056|081|
        |008|017|033|059|082|
        |009|---|037|061|084|

...
...
...

Jogador:Geronimo da silva
Cartela ID:3
        |---|---|---|---|---|
        |---|---|---|---|---|
        |---|---|---|---|---|
        |---|---|---|---|---|
        |---|---|---|---|---|
        |---|---|---|---|---|


Jogador:Fulano de Tal
Cartela ID:4
        |004|---|---|050|---|
        |---|024|041|051|---|
        |016|026|042|---|073|
        |---|028|043|063|075|
        |---|029|045|---|---|
        |---|030|---|068|086|

Cartela ID:5
        |003|---|036|054|---|
        |006|026|---|---|078|
        |---|028|040|---|080|
        |---|029|---|066|082|
        |---|031|045|068|085|
        |019|035|046|---|087|


Jogador Venceu!
Geronimo da silva
```

Gerar arquivo de estatísticas para análise: Nessa operação, o programa deverá utilizar as informações das jogadas coletadas durante a partida e exportar para um arquivo denominado *estatisticas_jogo.txt*. Esse arquivo deverá conter uma lista de todas as cartelas daquele jogo, ordenadas (decrescentemente) pela quantidade de pedras marcadas na cartela e tendo o nome do jogador dono da cartela como critério de desempate (crescentemente). A informação de cada cartela deverá estar em uma linha com os dados: quantidade de pedras marcada, *id* da cartela, e o nome do jogador ao qual a cartela pertence. Os campos deverão ser separados por " - ". Ver abaixo um exemplo do arquivo *estatisticas_jogo.txt*.

```
30 - 3 - Geronimo da silva
12 - 4 - Fulano de Tal
```

```
11 - 0 - Tripa Seca
10 - 5 - Fulano de Tal
9 - 2 - Tripa Seca
8 - 1 - Tripa Seca
```

Função bônus (opcional): Essa função deverá gerar um arquivo adicional denominado *bonus.txt* (a ser armazenado no mesmo local dos outros arquivos de saída). Ele deverá conter uma lista dos jogadores ordenada (decrescentemente) pela quantidade de pedras da cartela que teve menos pedras marcadas daquele jogador, e tendo o nome do jogador como critério de desempate (crescentemente). Ver abaixo um exemplo do arquivo *bonus.txt*. Nesse exemplo, a quantidade da cartela do jogador Geronimo com menor números de pedras marcadas foi 30, a do Fulano foi 10 e a do Tripa foi 8. Eles estão ordenados como 30, 10 e 8, como não teve empate, não precisou utilizar o nome.

```
Geronimo da silva (30)
Fulano de Tal (10)
Tripa Seca (8)
```

Alguns arquivos de entrada e seus respectivos arquivos de saída serão fornecido para o aluno como exemplo. O aluno deverá utilizar tais arquivos para testes durante a implementação do trabalho. É responsabilidade do aluno criar novos arquivos para testar outras possibilidades do programa e garantir seu correto funcionamento. O trabalho será corrigido usando 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 uso de arquivos com formatação diferente poderá acarretar em incompatibilidade durante a correção do trabalho e consequentemente na impossibilidade de correção do mesmo (nota zero). Portanto, siga estritamente o formato estabelecido.

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.

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 o dia **15/07/2017**. O assunto da mensagem deverá seguir o padrão: [TRABALHO_PROG2_2017_1] <NomeDoAluno>.

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

- [TRABALHO_PROG2_2017_1] FulanoDaSilva

Todos os arquivos do programa (*.c e *.h), 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 ".exe" sob o risco de bloqueio de email contendo arquivos executáveis. O código será compilado posteriormente em uma outra máquina. A correção poderá ser feita de forma automática, 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

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 análise depende de jogar. 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 do 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
Criar jogo	Inicializar o jogo Gerar corretamente arquivo de cartelas dos jogadores	3
Jogar	Realizar o jogo Gerar resultados corretamente	5

Gerar análise	Gerar corretamente arquivo estatisticas para análise	2
Legibilidade e Modularização	Falta de: Uso de comentários Identação do código Uso de funções Uso de tipos de dados definidos pelo usuário e respectivos arquivos (.c e .h)	-2
Função bônus	Gerar corretamente a lista pedida	1

Com intuito de identificar alunos que tiveram o trabalho feito por terceiros, a nota obtida com a avaliação acima (NI) será ponderada com uma nota de entrevista (NE) sobre o trabalho. A nota final do trabalho será calculada com NI x NE, onde NE vai de zero a um. A entrevista avaliará o conhecimento do aluno a respeito do programa desenvolvido. 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. A necessidade da entrevista será avaliada no decorrer do semestre, portanto sejam honestos e façam o trabalho com seu próprio esforço.

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 as especificações do trabalho. Caso seja notada qualquer tipo de inconsistência nos arquivos de testes disponibilizados, comunique imediatamente ao professor para que ela seja corrigida.