Especificação do Trabalho Prático

O trabalho prático da disciplina consiste em desenvolver um programa utilizando a linguagem de programação C. A seguir, encontram-se a descrição do problema, a forma de submissão do trabalho e os critérios de avaliação.

1. Descrição do problema

Uma empresa gostaria de oferecer a seus funcionários um jogo simples de adivinhação que permita que eles passem o tempo no horário de almoço. Nesta empresa, no entanto, os funcionários trabalham somente em modo texto (DOS, Console do Linux) e não há nenhum computador com um ambiente gráfico de janelas (Windows, Gnome). Desenvolva o programa de acordo com as especificações desta seção.

1.1. Menu de opções

Ao ser iniciado, seu programa deve cumprimentar o usuário e exibir um menu de opções. As opções possíveis são:

1. Novo Jogo;
2. Ver últimos 10 jogos;
3. Ver recordes (5 melhores);
4. Limpar histórico (jogos e recordes);
5. Sair.

Seu programa deve ler a opção escolhida e proceder como descrito nas demais seções, abaixo. Caso o usuário digite qualquer coisa diferente dos números de 1 a 5, você deve informar que a opção não existe e exibir o menu novamente.

1.2. Novo jogo

Escolhida a opção 1, seu programa deve começar um novo jogo de adivinhação. O computador deve escolher um número secreto de 1 a 100 aleatoriamente e o jogador deve adivinhá-lo, informando um número neste mesmo intervalo.

Se o jogador errar, o programa deve informar a ele se o número "chutado" foi maior ou menor do que o número secreto. Se o jogador acertar, o jogo deve parabenizá-lo e dizer com quantas tentativas ele conseguiu acertar o número.

Veja um exemplo a seguir. Neste jogo, o computador gerou o número 84 como secreto. Os números sublinhados foram os valores digitados pelo usuário:

```
Tente adivinhar um número de 1 a 100:
50
Muito baixo! Tente um valor mais alto:
75
Muito baixo! Tente um valor mais alto:
87
Muito alto! Tente um valor mais baixo:
81
```

```
Muito baixo! Tente um valor mais alto:
84
Parabéns! Você acertou depois de 5 tentativas!
```

Ao final de cada jogo, o programa deve perguntar ao jogador o seu nome, armazenar a informação deste jogo (nome, número secreto e tentativas) e apresentar o menu de opções novamente.

1.3. Ver últimos 10 jogos

Os últimos 10 jogos devem ser armazenados em memória pelo programa e quando o jogador escolher a opção 2 do menu, o computador deve exibir os últimos 10 jogos (ou menos, se menos de 10 jogos tiverem sido jogados), na ordem do mais antigo para o mais recente. Sobre cada jogo, o programa deve informar o nome do jogador, o número secreto adivinhado e o número de tentativas feitas até adivinhá-lo. Por exemplo:

```
Histórico:
Ten | Num | Jogador
006 | 084 | Fulano
001 | 050 | Beltrano
004 | 001 | Cicrano
011 | 015 | Vitor
```

O quadro acima exemplifica o histórico após quatro jogos diferentes: Fulano jogou primeiro, em seguida Beltrano, Cicrano e, por último, jogou Vitor. A primeira coluna mostra o número de tentativas, a segunda o número secreto e a terceira o nome do jogador. Para que as colunas fiquem alinhadas, é preciso imprimir os números (tentativas e número secreto) com 3 algarismos (usando zeros a esquerda).

Após exibir o histórico, o programa deve retornar ao menu de opções.

1.4. Ver recordes

Além de armazenar os últimos 10 jogos, o jogo deve armazenar os 5 melhores jogos de todos os jogados até o momento, sendo que quanto menos tentativas feitas para acertar, melhor o jogo. No caso de empate, ganha o jogo que foi jogado antes.

Ao escolher a opção 3 do menu, o jogador deve ver os 5 melhores jogos (ou menos, se menos de 5 jogos tiverem sido jogados até agora), na ordem do primeiro ao último do ranking. Sobre cada jogo, o programa deve informar a posição no ranking, o nome do jogador e o número de tentativas feitas até adivinhá-lo.

```
Recordes:
# | Ten | Jogador
1 | 001 | Beltrano
2 | 004 | Cicrano
3 | 006 | Fulano
4 | 011 | Vitor
```

O quadro acima mostra os recordes após os mesmos quatro jogos do exemplo anterior. Note, novamente, que o número de tentativas deve ser exibido com 3 algarismos.

Após exibir a lista de recordes, o programa deve exibir novamente o menu de opções.

1.5. Limpar histórico

Ao escolher a opção 4 do menu, o programa deve limpar todas as informações de histórico de jogos e de melhores jogos e imprimir a mensagem "Histórico zerado." Após limpar o histórico, caso o usuário selecione as opções 2 e 3 do menu antes de efetuar novos jogos as mesmas devem ser apresentadas vazias.

Após limpar o histórico o menu de opções deve ser exibido novamente.

1.6. Sair

Para sair do jogo, basta que o usuário escolha a opção 5 do menu. O programa deve ser encerrado sem imprimir nenhuma mensagem.

1.7. Mensagens

A empresa contratante gostaria de verificar que o software produzido atende fielmente às especificações. Para isso, utilizará de testes automatizados (descritos na próxima subseção). Para que estes testes funcionem, é fundamental que as mensagens exibidas pelo seu programa siga os padrões pré-estabelecidos pela empresa. Estes padrões estão descritos na tabela abaixo.

Ao ler a tabela, note que:

- O símbolo ↵ representa uma quebra de linha (`\n` no `printf()`). Quando presente sozinho em uma linha, significa que ali deve ser impressa uma linha em branco;
- Na mensagem número 07, o termo **<#>** deve ser substituído pelo número de tentativas, como no exemplo da seção 1.2;
- Na impressão de números, observe se os mesmos devem ser impressos com um números específico de algarismos, conforme instruções das seções 1.3 e 1.4.

01	Cumprimentando o usuário	`Olá! Seja bem-vindo ao jogo de adivinhação!↵`

02	Apresentando o menu de opções	`↵`
		`Escolha uma das opções abaixo:↵`
		`↵`
		`1. Novo Jogo;↵`
		`2. Ver últimos 10 jogos;↵`
		`3. Ver recordes (5 melhores);↵`
		`4. Limpar histórico (jogos e recordes);↵`
		`5. Sair.↵`
		`↵`
		`Opção:↵`

03	Informando que a opção escolhida não existe	`Opção inexistente. Por favor, escolha dentre as opções do menu, digitando o número correspondente e, em seguida, tecle Enter.↵`
04	Iniciando um novo jogo	`↵` `Tente adivinhar um número de 1 a 100:↵`
05	Informando que o valor digitado está abaixo do número secreto	`Muito baixo! Tente um valor mais alto:↵`
06	Informando que o valor digitado está acima do número secreto	`Muito alto! Tente um valor mais baixo:↵`
07	Informando que o valor digitado é igual ao número secreto	`Parabéns! Você acertou depois de <#> tentativas!↵`
08	Perguntando o nome do usuário	`↵` `Por favor, informe seu nome:↵`
09	Mostrando o histórico quando vazio	`Histórico:↵` `(vazio)↵`
10	Mostrando o histórico quando não vazio	Vide exemplo na seção 1.3.
11	Mostrando os recordes quando vazio	`Recordes:↵` `(vazio)↵`
12	Mostrando os recordes quando não vazio	Vide exemplo na seção 1.4
13	Limpando o histórico	`Histórico zerado.↵`

1.8. Testes automatizados

Como mencionado anteriormente, a empresa conduzirá uma série de testes automatizados para garantir que o programa funciona corretamente. Tais testes consistem em fornecer arquivos de entrada com uma série de comandos e dados supostamente digitados pelo usuário e analisar a saída produzida pelo programa.

Para isso, em primeiro lugar, é necessário pré-estabelecer, também, o nome do programa. A empresa gostaria que o programa se chamasse `adivinha`.

No entanto, dado que o programa gera números secretos aleatoriamente, não é possível montar arquivos de entrada de modo a saber qual é a saída correta do programa. Portanto, seu programa precisa prover uma última funcionalidade, que é a possibilidade de se especificar N números que devem ser utilizados como número secreto nos primeiros N jogos, sendo N uma quantidade qualquer de jogos.

Estes números devem ser especificados na linha de comando no momento da execução do programa. A tabela abaixo mostra alguns exemplos de execução do programa e o que deveria acontecer:

Execução	Resultado esperado
`./adivinha`	O jogo deve usar números aleatórios desde o início.
`./adivinha 30 50 70`	O jogo deve usar como número secreto o número 30 para o primeiro jogo, 50 para o segundo jogo e 70 para o terceiro jogo. A partir do quarto jogo, o número deve ser aleatório.
`./adivinha 1 2 3 4 5 4 3 2 1`	Os primeiros 9 jogos terão seus números aleatórios pré-determinados, seguindo a ordem que aparecem à esquerda. A partir do décimo jogo, o programa deve usar números aleatórios.

Para facilitar o seu trabalho, a empresa irá fornecer alguns arquivos de entrada e saída de testes para que você possa verificar se o seu programa está de acordo com o esperado. No entanto, a empresa reserva o direito de não compartilhar todos os seus arquivos de testes, de modo a garantir que os programas não estarão "viciados" nos arquivos existentes.

1.9. Persistência dos dados (opcional)

Ganhará pontos extra (vide seção 3) o trabalho que implementar a persistência dos dados, que consiste em gravar em um arquivo o conteúdo do histórico e dos recordes ao sair do programa e recuperá-los ao iniciar o programa novamente.

O arquivo deve se chamar obrigatoriamente `adivinha.dat`.

2. Condições de elaboração e entrega

Os trabalhos devem ser feitos obrigatoriamente em dupla. Caso o número de alunos matriculados não seja par, o aluno com a maior nota na primeira prova parcial poderá, a sua escolha, fazer o trabalho sozinho ou juntar-se a uma dupla existente e formar um trio. Em caso de empate da nota mais alta, terá este privilégio apenas o aluno mais velho entre os empatados.

Os alunos devem informar em até 7 dias após a entrega da primeira prova parcial corrigida quais são as duplas de alunos que trabalharão juntos na elaboração do trabalho prático. Caso isso não seja feito no prazo estabelecido, o professor decidirá aleatoriamente as duplas restantes.

Para evitar que um aluno da dupla faça todo o trabalho e o outro não faça nada, alguns alunos poderão ser chamados ao acaso para uma entrevista com o professor, na qual deverão explicar trechos de seus programas. A nota do aluno dependerá desta entrevista.

Dado que existem várias versões de compiladores C, fica determinado o uso das versões instaladas nas máquinas do LCEE como versões de referência para o trabalho prático. Seu trabalho deve compilar e executar corretamente nas máquinas do laboratório.

A entrega do trabalho segue um sistema automatizado de recebimento, descrito a seguir.

2.1. Regras para entrega do trabalho

O código-fonte de sua solução deverá ser compactado e entregue por e-mail (anexo ao e-mail) para o endereço vsouza@ninfa.inf.ufes.br.

> ATENÇÃO: este e-mail foi gentilmente criado pelos membros do laboratório NINFA para ser utilizado apenas para recebimento de trabalhos. Dúvidas devem ser enviadas para o e-mail do professor: vitorsouza@inf.ufes.br.

Serão aceitos trabalhos entregues até as 23h59 da data limite. O assunto do e-mail deverá ser o seguinte:

```
pbcee:trab1:<nome1>:<nome2>:
```

O termo "<nome1>:<nome2>" deverá ser substituído pelos nomes dos alunos em ordem alfabética – somente um nome e um sobrenome de cada aluno, separados por vírgula e sem acentos, til ou cedilha, como no exemplo abaixo:

```
pbcee:trab1:Victorio Carvalho:Vitor Souza:
```

Repare, novamente, que os nomes Victório e Vítor foram escritos em ordem alfabética de primeiro nome e sem acento. É muito importante que o assunto do e-mail esteja correto, do contrário o sistema de correção automática não reconhecerá sua submissão e seu trabalho será considerado como não recebido. Caso um aluno tenha feito o trabalho sozinho ou em trio basta informar, respectivamente, apenas um nome (`prog3:trab1:<nome1>`) ou os três nomes (`prog3:trab1:<nm1>:<nm2>:<nm3>`).

Se tudo correr bem no recebimento do arquivo, você receberá um e-mail de confirmação do recebimento do trabalho. Neste e-mail haverá um *hash* MD5 (https://pt.wikipedia.org/wiki/MD5#Hashes_MD5) do arquivo recebido. Para garantir que o arquivo foi recebido sem ser corrompido, gere o *hash* MD5 do arquivo que você enviou (nas máquinas do laboratório, use o comando `md5 <arquivo>`) e compare com o *hash* recebido na confirmação. Caso você não receba o e-mail de confirmação ou caso o valor do *hash* seja diferente, envie o trabalho novamente. Se o problema persistir, contate o professor.

O arquivo compactado enviado por e-mail deve estar no formato `tar.gz` com o nome `trab1.tar.gz` e conter os arquivos de código-fonte e um *Makefile*. Um *Makefile* é um arquivo que explica como compilar sua aplicação a partir de vários arquivos de código-fonte e é usado pela ferramenta `make` para compilação automatizada. O assunto será abordado na aula sobre bibliotecas de função e exercitado em laboratório, porém o aluno interessado pode encontrar vários tutoriais na Internet sobre `make` e *Makefile*.

Os arquivos fonte podem estar organizados da forma que você achar melhor, desde que o comando `make` produza o programa compilado. Para permitir os testes automáticos, seu programa compilado deverá chamar-se `adivinha`.

É essencial que sejam seguidas à risca as instruções acima, caso contrário a correção automatizada não funcionará. Trabalhos que não passarem pela correção automatizada passarão por correção manual, porém sofrerão penalidade de 2 pontos.

3. Critérios de avaliação

Os trabalhos serão avaliados primeiramente pela execução correta, seguindo os seguintes critérios objetivos:

Critério	Nota
Trabalhos que não compilarem.	De 0 a 3
Trabalhos que compilarem mas não gerarem resultados corretamente.	De 3 a 7
Trabalhos que compilarem e gerarem os relatórios corretamente.	De 6 a 10
Trabalhos entregues fora do prazo.	-1 por dia de atraso
Trabalhos que não seguirem os padrões de mensagem estabelecidos na seção 1.7.	-0,1 por erro
Trabalhos que não permitirem os testes automatizados descritos nas seções 1.8 e 2.1.	-2 pontos
Trabalhos que utilizarem tipos abstratos de dados (TADs) para as estruturas de dados.	Recupera até 2 pontos.*
Trabalhos que implementarem a persistência de dados opcional descrita na seção 1.9.	Até +2 pontos extras.

Em segundo lugar, os trabalhos serão avaliados segundo critérios subjetivos definidos pelo professor, variando a nota dentro da faixa de notas definida pelo critério anterior. Alguns dos critérios subjetivos avaliados serão:

- Legibilidade (nomes de variáveis bem escolhidos, código bem formatado, uso de comentários quando necessário, etc.);
- Consistência (utilização de um mesmo padrão de código);
- Eficiência (sem exageros, tentar evitar grandes desperdícios de recursos);
- Uso de estruturas de dados adequadas. Espera-se, por exemplo, que o histórico e os recordes sejam representados por vetores de ponteiros que se referem a estruturas de dados compostas que representam os jogos.

4. Conceitos a aprender

No momento da primeira versão deste documento, somente a parte básica de programação em C foi coberta pelo curso. Com esta parte básica, é possível atender aos itens 1.1 e 1.2 da descrição do problema.

Os conceitos abaixo serão necessários para que o aluno possa implementar as demais funcionalidades do programa:

---

* Recuperar pontos significa que os pontos ganhos com este item não poderão ultrapassar a nota máxima 10. Ganhar pontos extras significa que a nota poderá ultrapassar 10 se for o caso.

- Para armazenar o histórico e os recordes, o aluno deve aprender o conceito de vetores (variáveis indexadas) e de estruturas de dados compostas;
- Em particular para o histórico, será preciso aprender o conceito de vetores circulares (vetores que armazenam os últimos N elementos inseridos);
- Em particular para os recordes, será preciso aprender a manter um vetor ordenado;
- Para imprimir números inteiros com um número específico de algarismos será necessário aprender sobre os códigos de formatação do `printf()`;
- Para os testes automatizados, será necessário aprender a ler os parâmetros passados pela linha de comando.

5. Observações finais

Caso haja algum erro neste documento, serão publicadas novas versões e divulgadas erratas em sala de aula. É responsabilidade do aluno manter-se informado, frequentando as aulas ou acompanhando as novidades na página da disciplina na Internet.