	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

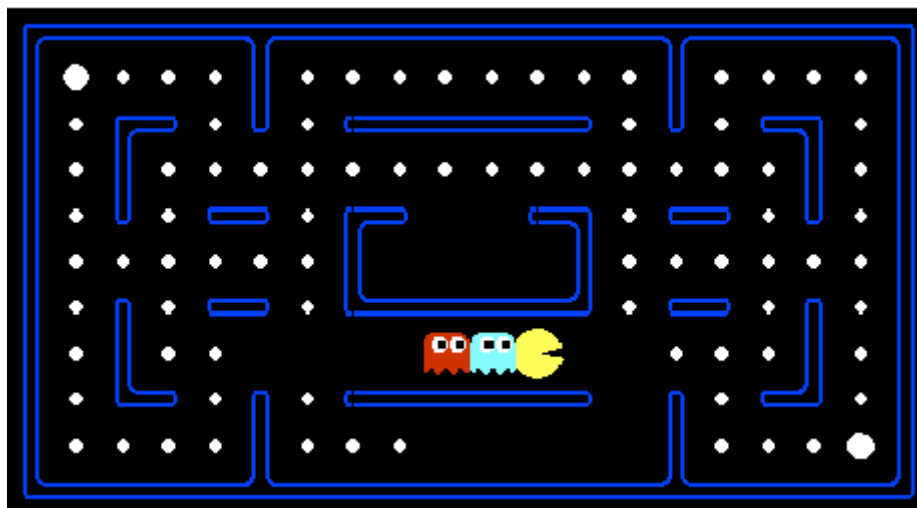
Especificação do Trabalho

Pac-Man

Introdução

Este trabalho tem como objetivo avaliar, de forma prática, os conceitos ensinados no curso de Programação I. A meta deste trabalho é implementar um jogo do gênero *Pac-Man*¹ seguindo as especificações descritas neste documento e utilizando os conceitos ensinados em sala para manter boas práticas de programação.


Pac-Man é um jogo *single-player* no qual o jogador controla um personagem que se move sobre um mapa bidimensional. O objetivo do jogo é obter a maior pontuação possível sem colidir com os fantasmas do jogo.



Descrição do Trabalho

Neste trabalho, você deverá implementar um jogo do gênero *Pac-Man*. O programa deverá rodar no terminal, assim como todos os outros exercícios feitos em sala. De maneira geral, o programa irá iniciar pela linha de comando (terminal) e irá ler o estado inicial do jogo, que consiste na definição do mapa, a posição inicial do personagem e dos fantasmas. Com o jogo inicializado, serão executados movimentos dados pelo usuário na entrada padrão. Durante a execução, o programa apresentará os estados parciais na saída padrão até o jogo terminar, hora em que ele apresentará o resultado final e salvará arquivos do jogo contendo outros resultados.

¹ <https://en.wikipedia.org/wiki/Pac-Man>

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	


Funcionamento Detalhado do Programa

A execução do programa será feita através da linha de comando (*i.e.*, pelo *cmd*, *console*, ou terminal) e permitirá a passagem de parâmetros. As funcionalidades a serem realizadas pelo programa são descritas a seguir: *inicializar jogo*, *realizar jogo*, *gerar resumo de resultado*, *gerar estatísticas*, *gerar ranking*, *gerar trilha* e a *função bônus (para tratar túneis)*. A descrição dos parâmetros de inicialização, da exibição do estado atual do programa, assim como, das operações a serem realizadas pelo programa, são apresentadas em detalhes a seguir.

Inicializar jogo: 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 (exemplo assumindo um programa compilado para o executável *prog*, “*./prog /maquinadoprofessor/diretoriodeteste*”). 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 configuracao nao foi informado”), e finalizar sua execução. Dentro desse diretório, espera-se encontrar o arquivo de definição do mapa do jogo: *mapa.txt*, sendo alguns exemplos fornecidos com esta especificação. O programa deverá ler o arquivo e preparar o ambiente de jogo. Caso o programa não consiga ler o arquivo (*e.g.*, porque ele não existe), ele deverá ser finalizado e imprimir uma mensagem informando que não conseguiu ler os arquivos (OBS: a mensagem deve conter o caminho e nome do arquivo que ele tentou ler). Todas as saídas em forma de arquivo do trabalho devem ser escritas na pasta *saida* dentro do mesmo diretório de teste.

O arquivo *mapa.txt* definirá todos os parâmetros do mapa: seu tamanho e o conteúdo de cada célula, ou seja, o que deve estar em cada posição. Além disso, o arquivo *mapa.txt* define um limite de movimentos que o jogador pode executar sem que acarrete *game over*. Um mapa é definido como um arranjo de N linhas contendo M colunas, semelhante a uma matriz, com tamanho máximo de 40x100. Cada posição dessa matriz (denominada célula) pode ser de diferentes tipos como mostrado na tabela abaixo.

Tipo de célula	Representação	Em caso de colisão
Vazio	(espaço em branco)	Nada acontece
Parede	#	O personagem não se move
Comida	*	Ganha +1 ponto
Posição inicial do jogador	>	–

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

Fantasma	B, P, I, C	Fim de jogo
----------	------------	-------------

Um exemplo desse arquivo de definição do mapa é mostrado a seguir.

mapa.txt

```
6 7 20
#####
#*   *#
#  * B#
#   *#
#>   #
#####
```

O exemplo acima define um mapa com 6 linhas e 7 colunas, limite de 20 movimentos, paredes (#) e quatro comidas (*) espalhadas pelo mapa. A posição inicial do personagem está na quinta linha e segunda coluna (posição do ">"). Um fantasma (B) está na linha 3 e coluna 6. Repare que o arquivo de mapa não precisa ter todos os fantasmas.

Como resultado do passo de *Inicializar o Jogo*, o programa deve gerar o arquivo *inicializacao.txt*. Esse arquivo deve conter o mapa lido. Depois da representação do mapa, deve haver a linha de texto "Pac-Man comecara o jogo na linha @ e coluna #", substituindo @ pela linha e # pela coluna da posição de onde o personagem foi definida no arquivo *mapa.txt*. Um exemplo desse arquivo é mostrado a seguir.

inicializacao.txt

```
#####
#*   *#
#  * B#
#   *#
#>   #
#####
Pac-Man comecara o jogo na linha 5 e coluna 2
```

Realizar jogo: Uma vez preparado o jogo, as jogadas poderão começar a ser processadas. O jogo terminará quando não houver mais comidas pelo mapa, Pac-Man morrer (*i.e.*, quando colidir com um fantasma) ou ser atingido o limite de movimentos. As jogadas deverão ser lidas da entrada padrão de dados, permitindo dessa forma, um redirecionamento a partir de arquivo. O fluxo da jogada segue:

- A jogada será fornecida pelo usuário no formato "@", em que @ é um caractere que define o movimento: "a" para ir para esquerda, "w" para cima, "s" para ir para baixo e "d" para ir para a direita.
- Após ler a jogada do jogador, ela deve ser processada. O movimento que o personagem fará depende da posição em que ele estava e do movimento que o jogador definiu. Com isso os fantasmas também se movimentam, eles seguem uma



Centro Tecnológico
Departamento de Informática

Disciplina: Programação I

Código: INF15927

Trabalho Prático

Prof. Dr. Thiago Oliveira dos Santos

política de movimento simples se movendo em apenas uma direção (vertical ou horizontal) e com um sentido inicial especificado:

Fantasma	Direção	Sentido inicial
B	Horizontal	Para esquerda
P	Vertical	Para cima
I	Vertical	Para baixo
C	Horizontal	Para direita

Um fantasma, ao estar de encontro como uma parede, deve ter seu sentido mudado para o oposto. Não haverá casos com fantasma preso entre duas paredes.

- Após um movimento, o programa deverá exibir na saída padrão (ou seja, na tela) o estado atual do jogo (o mapa com o Pac-Man) e a pontuação até o momento no seguinte padrão:

```
Estado do jogo apos o movimento '#':  
<estado do mapa>  
Pontuacao: @
```

Substituindo “#” pelo movimento feito (“a”, “w”, “s” ou “d”), “<estado do mapa>” pelo estado do mapa e “@” pela pontuação no momento. Exemplo:

```
Estado do jogo apos o movimento 's':  
#####  
#  
#B >#  
#  ##*#  
#      #  
#####  
Pontuacao: 3
```

Um detalhe sobre a impressão do mapa que deve ser considerada: se um fantasma estiver na mesma posição de uma comida, o fantasma é impresso ao invés da comida. Além disso, não haverá casos de teste com a possibilidade de fantasmas se cruzarem no caminho.

- Caso após o movimento executado não haja mais nenhuma comida no mapa, o jogo encerra com a seguinte mensagem:


```
Voce venceu!  
Pontuacao final: #
```

Substituindo “#” pela pontuação final.

Caso ainda haja comida no mapa, exhibe-se a mensagem a seguir:

```
Game over!  
Pontuacao final: #
```

Novamente substituindo “#” pela pontuação final.

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

As condições de *game over* são o personagem colidir com algum dos fantasmas (e nesse caso o estado do mapa a ser impresso não exibirá o Pac-Man) ou o número de movimentos ultrapassar o limite estabelecido no arquivo mapa .txt.

Para facilitar a implementação, será fornecido também, com cada caso de teste, um arquivo de movimentos (denominado jogadas.txt), que poderá ser redirecionado pela entrada padrão para gerar uma saída esperada. Isso evitará ter que digitar todas as jogadas na mão. Veja abaixo um exemplo do conteúdo de um arquivo de movimentos. Considere que os casos dados sempre serão suficientes para finalizar o jogo, com uma vitória ou uma derrota. Cabe ao aluno testar outros casos.

jogadas.txt

```
w
w
w
d
d
d
d
d
s
s
w
a
a
```

Gerar resumo de resultado: O programa deverá também salvar na pasta de saída do caso de teste em questão, um arquivo (denominado resumo.txt) contendo o resumo do resultado do jogo. O arquivo de resumo deverá conter uma descrição do que houve em cada movimento onde houve alguma variação significativa: pegou comida, colidiu com a parede ou colidiu com um fantasma. Cada linha deve descrever a alteração causada por um movimento, sendo que cada alteração possui um padrão específico de saída, como mostrado a seguir.

No caso de comida:


```
Movimento # (@) pegou comida
```

No caso um movimento de colisão com parede (em que Pac-Man não muda de posição):

```
Movimento # (@) colidiu com a parede
```

No caso de fim de jogo por encostar em um fantasma:

```
Movimento # (@) fim de jogo por encostar em um fantasma
```

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

Exemplo completo do arquivo de resumo:

resumo.txt

```
Movimento 3 (w) pegou comida
Movimento 7 (d) pegou comida
Movimento 8 (d) colidiu na parede
Movimento 10 (s) pegou comida
Movimento 13 (a) pegou comida
```

Gerar ranking: O programa deverá também salvar na pasta de saída do caso de teste em questão, um arquivo (denominado ranking.txt) contendo um ranking de melhores movimentos junto de características desses movimentos ao fim do jogo. O melhor movimento ('w', 'a', 's', 'd') é aquele que mais resultou em pegar comida. Em caso de empate usa o movimento que menos colidiu com a parede. Como segundo critério de desempate temos o movimento utilizado mais vezes. Se nesse último critério também houver empate, usa-se a ordem alfabética do da letra representando o movimento. O arquivo de ranking deverá apresentar cada linha, o movimento (@), o número de comidas pegadas com esse movimento (#1), o número de colisões com parede (#2) e o número de movimentos do tipo realizado (#3), seguindo essa mesma ordem.

```
@, #1, #2, #3
@, #1, #2, #3
@, #1, #2, #3
@, #1, #2, #3
```

Exemplo do arquivo de ranking:

ranking.txt do caso simples/3 (primeiras 25 linhas)

```
w, 1, 0, 4
a, 1, 0, 2
s, 1, 0, 2
d, 1, 1, 5
```

Gerar arquivo de estatísticas para análise: Ao final do jogo, o programa deverá também escrever, na pasta de saída do caso de teste em questão, um arquivo (estatisticas.txt) contendo algumas estatísticas básicas sobre a partida. As informações que devem ser coletadas são: número de movimentos, número de movimentos sem pontuar, número de colisões com parede, número de movimentos para baixo, número de movimentos para cima, número de movimentos para a esquerda e número de movimentos para a direita. O padrão desse arquivo segue o formato:



Centro Tecnológico
Departamento de Informática

Disciplina: Programação I

Código: INF15927

Trabalho Prático

Prof. Dr. Thiago Oliveira dos Santos

```
Numero de movimentos: #  
Numero de movimentos sem pontuar: #  
Numero de colisoes com parede: #  
Numero de movimentos para baixo: #  
Numero de movimentos para cima: #  
Numero de movimentos para esquerda: #  
Numero de movimentos para direita: #
```

Um exemplo do arquivo de estatísticas pode ser visto abaixo:

estatisticas.txt

```
Numero de movimentos: 13  
Numero de movimentos sem pontuar: 9  
Numero de colisoes com parede: 1  
Numero de movimentos para baixo: 2  
Numero de movimentos para cima: 4  
Numero de movimentos para esquerda: 2  
Numero de movimentos para direita: 5
```


Gerar trilha: Ao final do jogo, o programa deverá também escrever, na pasta de saída do caso de teste em questão, um arquivo (trilha.txt) contendo o mapa que descreve o número do movimento da última vez com que Pac-Man passou por cada célula. Isso será mostrado como uma matriz, onde o valor na linha i e coluna j denota o número do movimento em que o personagem passou pela linha i e coluna j do mapa pela última vez. A posição inicial é o movimento 0. Caso nunca tenha passado posição do mapa, exibi-se o carácter '#'. Um exemplo do arquivo de *trilha* pode ser visto abaixo:

trilha.txt de exemplo

```
# # # # # #  
# 3 4 5 6 8 #  
# 2 # 13 12 11 #  
# 1 # # # 10 #  
# 0 # # # # #  
# # # # # #
```

Tratar túnel (Bônus): Haverá uma pontuação extra para a funcionalidade de túneis, que será avaliada por um conjunto de casos de testes separados. Nesses casos, haverá mais um tipo de célula: o túnel. Em todos esses casos, haverá duas células de túnel no mapa, representadas pelo caractere "@". Quando Pac-Man colidir com uma dessas células, ele deve ser teleportado para a posição da outra célula. No caso do arquivo de trilha, a última posição dele será escrita nas duas pontas do túnel.

Informações Gerais

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

Alguns arquivos de entrada e respectivos arquivos de saída serão fornecidos para o aluno. O aluno deverá utilizar tais arquivos para testes durante a implementação do trabalho. É de responsabilidade do aluno criar novos arquivos para testar outras possibilidades do programa e garantir seu correto funcionamento. O trabalho será corrigido usando, além dos arquivos dados, outros arquivos (específicos para a correção e não disponibilizados para os alunos) seguindo a formatação descrita neste documento. 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 (sendo atribuído a nota zero). Portanto, siga estritamente o formato estabelecido.


Implementação e Verificação dos Resultados

A implementação deverá seguir os conceitos de modularização vistos 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 corretamente, o código deverá estar bem escrito para que o aluno obtenha nota máxima.

É extremamente recomendado (para não dizer obrigatório) utilizar algum programa para fazer as comparações do resultado final do programa. Isto é, os arquivos de saída gerados, poderão ser comparados com os arquivos de saídas esperadas (fornecidos pelo professor) utilizando o comando *diff*, como visto e feito 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 e é muito útil no desenvolvimento do trabalho. Diferenças na formatação poderão impossibilitar a comparação e consequentemente impossibilitar a correção do trabalho. O programa será considerado correto se gerar a saída esperada idêntica à fornecida com os casos de teste.

Os casos de testes visíveis serão disponibilizados juntamente com esta especificação do trabalho. A pasta com os casos de teste visíveis terá uma subpasta (denominada “Gabarito”) contendo todos os casos fornecidos. Além disso, ela terá outra subpasta (denominada “Testes”) contendo o mesmo conteúdo da pasta Gabarito, porém sem os respectivos arquivos de saída. Caso o programa do aluno esteja funcionando adequadamente, após executá-lo utilizando os casos da pasta de Teste, esta deverá ficar idêntica a pasta Gabarito.

Regras Gerais

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

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. Cada aluno deverá trabalhar independente dos outros, não sendo permitido a cópia ou compartilhamento de código. O professor irá fazer verificação de plágio. Trabalhos identificados como iguais, em termos de programação (por exemplo, mudar nomes de variáveis e funções entre outros não faz dois trabalhos serem diferentes), serão penalizados com a nota zero e submetidos para penalidades adicionais em instâncias superiores. 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.


Entrega do Trabalho

O trabalho deverá ser entregue pelo classroom até a data e hora definidas para tal. O trabalho deve ser entregue todo em um único arquivo “.c”, nomeado com o nome do aluno e sem espaços e sem acentos, por exemplo “**ThiagoOliveiraDosSantos.c**”. Ele será necessariamente corrigido no sistema operacional linux das máquinas do laboratório, qualquer incompatibilidade devido ao desenvolvimento em um sistema operacional diferente é de responsabilidade do aluno. Isso pode inclusive levar a nota zero, no caso de impossibilidade de correção. A correção será feita de forma automática (via *script*), portanto trabalhos não respeitando as regras de geração dos arquivos de saída, ou seja, fora do padrão, poderão impossibilitar a correção. Consequentemente, acarretar na atribuição da nota zero. A pessoa corrigindo não terá a obrigação de adivinhar nome de arquivos, diretórios ou outros. **Siga estritamente o padrão estabelecido!**

Pontuação e Processo de Correção


Pontuação: Trabalhos entregues após o prazo não serão corrigidos (recebendo a 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 o jogo corretamente. Código com falta de legibilidade e modularização pode perder pontos conforme informado na tabela. Erros gerais de funcionamento, lógica ou outros serão descontados como um todo. A pontuação de um item será dada pelo número de casos de testes corretos para aquele item. Haverá o mesmo número de casos visíveis e invisíveis (que serão conhecidos no dia da correção).

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

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

semestre. Prioridade será dada para a nota com maior peso, porém não ultrapassando a nota máxima.

Item	Quesitos	Pont o
<input checked="" type="checkbox"/>	Iniciar jogo Ler o arquivo do mapa Gerar corretamente o arquivo de Inicialização (inicializacao.txt)	1
<input type="checkbox"/>	Realizar jogo Receber corretamente as entradas do jogador Mostrar as informações do jogo como especificado Mostrar o resultado do jogo Usar a saída padrão	3
<input checked="" type="checkbox"/>	Gerar resumo do resultado Gerar arquivo com o resumo dos resultados (resumo.txt)	1
<input type="checkbox"/>	Gerar ranking Gerar arquivo com a ordenação solicitada (ranking.txt)	1
<input checked="" type="checkbox"/>	Gerar estatísticas Gerar corretamente o arquivo com as estatísticas (estatisticas.txt)	1
<input checked="" type="checkbox"/>	Gerar trilha Gerar corretamente o arquivo com a trilha (trilha.txt)	1
<input type="checkbox"/>	Manutenibilidade Permitir modificações surpresas e serem passadas no dia da correção do trabalho pelos alunos.	2
<input type="checkbox"/>	Função bônus Tratar os túneis	1
Legibilidade e Modularização	Falta de: <ul style="list-style-type: none"> • Uso de comentários; • Identação do código; • Uso de funções; • Uso de tipos de dados definidos pelo usuário 	-1

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

Processo de correção do trabalho: O trabalho será corrigido, considerando-se 3 turnos: Primeira Entrega, Tempo de Correção e Manutenção e Entrevista.


A Primeira Entrega ocorrerá na data definida para a 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 as funcionalidades entregues nesta etapa.

O Tempo de Correção e Manutenção ocorrerá no dia indicado no classroom. Nesse dia, o aluno terá a possibilidade, dentro do tempo disponibilizado, de corrigir erros encontrados no trabalho. Funcionalidades novas já especificadas anteriormente não serão contabilizadas nesta etapa, portanto não adianta usar este tempo para implementar o que já havia sido pedido e não foi entregue. Neste mesmo dia, será testada a manutenibilidade do código, sendo exigida a implementação de novas funcionalidades para incorporação de pequenas mudanças no comportamento do jogo. Isso faz parte da pontuação do trabalho. Quanto mais modular estiver o código, mais fácil será de implementar as mudanças. 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 do Tempo de Correção (NTC), em que $NPE \leq NTC \leq 10$. 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 deste tempo de correção, o aluno deverá reenviar o programa corrigido, desta vez na atividade de correção do trabalho, 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 no Tempo de Correção. Portanto, se não quiser ter problemas, faça o trabalho e a entrega com antecedência.

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 levantar o conhecimento dos alunos sobre o próprio trabalho. A nota, NTC, 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 $NTC \times NE$, em que $0 \leq NE \leq 1$. A entrevista avaliará o conhecimento do aluno a respeito do seu próprio programa. 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 seu próprio trabalho (i.e., código).

Considerações Finais

Não utilizar caracteres especiais (como acentos) em lugar nenhum do trabalho.

	Centro Tecnológico Departamento de Informática	
Disciplina: Programação I		Código: INF15927
Trabalho Prático	Prof. Dr. Thiago Oliveira dos Santos	

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.