

Algoritmos e Estruturas de Dados II

Vinte Perguntas

2^a Avaliação (Programa) - Valor: 25 pontos

Flávio Augusto de Freitas

27 de abril de 2017

Sumário

1	Porque o Jogo de Vinte Perguntas	2
2	O Jogo de Vinte Perguntas	2
3	Detalhes de Implementação	7
4	O Que Deve Ser Entregue	8
5	O Arquivo de Entrada e o de Saída	9
5.1	A Criação de um Arquivo de Entrada	10
6	A Leitura do Arquivo de Entrada e a Gravação no Arquivo de Saída	11
7	Observações	13
8	Aluno ou Grupo de Alunos	13
8.1	Plágio	14
9	Compilação	14
10	Entrega	15

Lista de Códigos Fonte

1	Código no Início do Programa	11
2	Código Para Tratamento do Arquivo de Entrada	11
3	Código Para Tratamento do Arquivo de Saída	12
4	Código Para Tratamento dos Arquivos de Log	12

1 Porque o Jogo de Vinte Perguntas

Este programa centra-se em árvores binárias e recursão. Utilize estruturas como `arvore` e `no`. Você precisará de funções de suporte como `main`, `preOrdem` etc. Coloque-os todos no mesmo arquivo fonte. Caso use orientação a objetos coloque as classes na mesma pasta.

2 O Jogo de Vinte Perguntas

Nesta atribuição você implementará um jogo de adivinhação chamado "20 Perguntas". Cada rodada do jogo começa por você (o jogador humano) pensando em algum objeto, pessoa, animal etc., enfim alguma coisa. O computador tentará adivinhar o que foi pensado fazendo uma série de questões do tipo sim ou não.

Eventualmente o computador terá feito perguntas suficientes que ele "pense" que sabe em que você está pensando. Ele fará uma suposição sobre o que você pensou. Se essa suposição estiver correta, o computador ganha; Se não, você ganha.

O computador acompanha uma árvore binária cujos nós representam perguntas e respostas. (Os dados de cada nó são uma *string* representando o texto da pergunta ou resposta.)

Um nó de "pergunta" contém uma subárvore "sim" esquerda e uma subárvore direita "não".

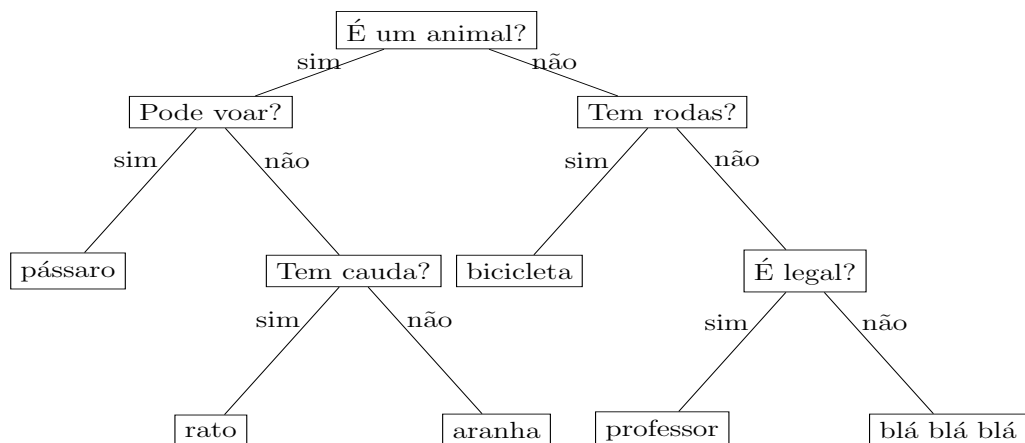
Um nó de "resposta" é uma folha.

A ideia é que esta árvore possa ser percorrida para fazer ao jogador humano uma série de perguntas¹.

¹Embora o jogo seja chamado de "20 Perguntas", nosso jogo não limitará a árvore a uma altura de 20. Qualquer altura é permitida.

Por exemplo, na árvore abaixo, o computador começaria o jogo perguntando ao jogador: "É um animal?" Se o jogador diz "sim", o computador vai para a esquerda para a sub-árvore "sim" e, em seguida, pergunta ao usuário: "Pode voar?" Se o usuário tivesse dito "não", o computador iria direto para a sub-árvore "não" e então perguntaria ao usuário: "Tem rodas?"

Esse padrão continua até que o jogo alcance um nó de "resposta" da folha. Ao alcançar um nó de resposta, o computador pergunta se essa resposta é a resposta correta. Se assim for, o computador ganha.



O seguinte log de saída parcial mostra um jogo sendo jogado na árvore acima:

```

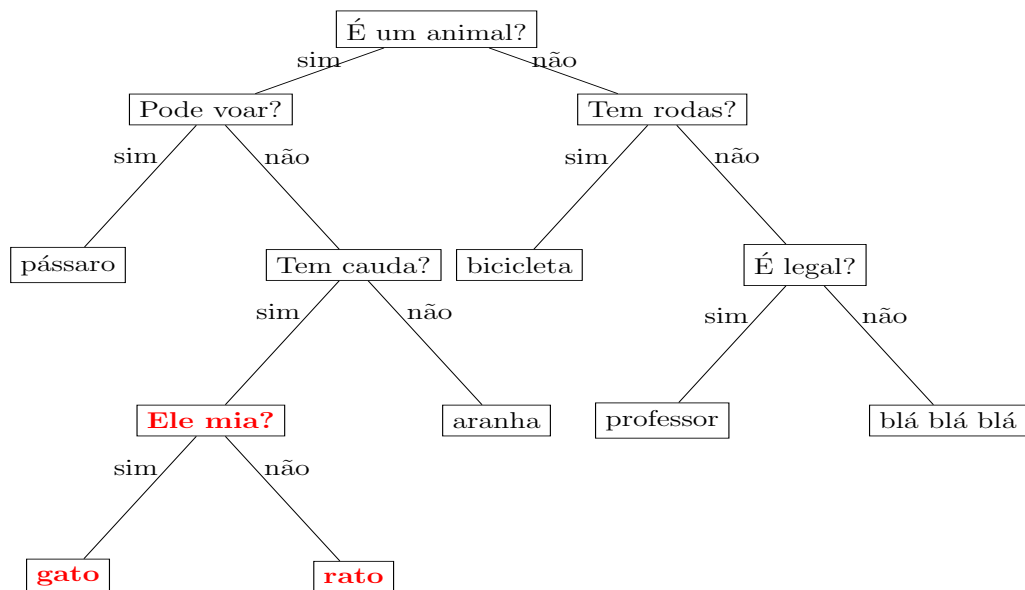
É um animal? S
Pode voar? N
Tem cauda? S
Você pensou em rato? S
Eu ganhei!
  
```

Inicialmente, o computador não é muito inteligente, mas fica mais inteligente cada vez que perde o jogo. Se a resposta do computador está incorreta, você deve dar-lhe uma nova pergunta, o que resultará em jogos melhores no futuro. Por exemplo, suponha que no log anterior o jogador não estava pensando em um rato, mas em um gato. O log do jogo poderia ser assim: (Você

deve criar seu **log** exatamente no formato dos registros nesta especificação. Isso conta para a nota!)

É um animal? **S**
 Pode voar? **N**
 Tem cauda? **S**
 Você pensou em rato? **N**
 Eu perdi. Em que você pensou? **gato**
 Digite uma pergunta sim/não para diferenciar o que você pensou de rato:
Ele mia?
 E qual é a resposta para o que você pensou? **S**

O computador pega as novas informações de um jogo perdido e usa-as para substituir o antigo nó de resposta incorreto por um novo que tem a resposta incorreta antiga e nova resposta correta como seus filhos. Por exemplo, depois do jogo representado pelo **log** anterior, a árvore de jogo geral do computador seria a seguinte:



Nesta atribuição, você criará classes `QuestaoTree` e `QuestaoNo` para representar a árvore do computador de perguntas sim/não e respostas para exe-

cutar jogos de 20 perguntas. Você precisa criar a interação com o usuário e chamar métodos de sua árvore para jogar. Abaixo estão dois **logs** de execução (a entrada do usuário em **negrito**):

LOG 1
Bem-vindo ao jogo de 20 Perguntas! Devo lembrar dos nossos jogos anteriores? N Pense em alguma coisa, e eu vou adivinhar. Você pensou em computador? N Eu perdi. Em que você pensou? gato Digite uma pergunta sim/não para diferenciar de computador: É um animal? E qual é a resposta para a sua pergunta? S Jogar de novo? S É um animal? N Você pensou em computador? N Eu perdi. Em que você pensou? sapato Digite uma pergunta sim/não para diferenciar de computador: Ele vai em seus pés? E qual é a resposta para a sua pergunta? S Jogar de novo? S É um animal? N Ele vai em seus pés? S Você pensou em sapato? S Eu ganhei! Jogar de novo? N Jogos realizados: 3 Eu ganhei: 1 Devo lembrar-me desses jogos? S Qual é o nome do arquivo? questao1.txt

LOG 2

Bem-vindo ao jogo de 20 Perguntas!
Devo lembrar dos nossos jogos anteriores? **S**
Qual o nome do arquivo? **questao2.txt**

Pense em alguma coisa, e eu vou adivinhar.

É um animal? **N**
Tem rodas? **S**
Você pensou em bicicleta? **S**
Eu ganhei! Jogar de novo? **S**

É um animal? **S**
Pode voar? **N**
Tem cauda? **S**
Você pensou em rato? **N**
Eu perdi. Em que você pensou? **gato**
Digite uma pergunta sim/não para diferenciar de rato: **Ele mia?**
E qual é a resposta para a sua pergunta? **S**
Jogar de novo? **S**

É um animal? **S**
Pode voar? **N**
Tem cauda? **S**
Ele mia? **S**
Você pensou em gato? **S**
Eu ganhei!
Jogar de novo? **N**

Jogos realizados: 3
Eu ganhei: 2
Devo lembrar-me desses jogos? **N**

3 Detalhes de Implementação

Você deve armazenar o estado atual da árvore em um arquivo de saída representado pela variável `arvore`. Dessa forma sua árvore de perguntas pode crescer cada vez que o usuário executar o programa. (Você não salva o número de jogos jogados/ganhos.) Uma árvore é especificada por uma sequência de linhas, uma para cada nó. Cada linha deve começar com **Q:** para indicar uma pergunta (Nó); ou **R:** para indicar uma resposta (Folha). Todos os caracteres após estes dois primeiros devem representar o texto para esse nó (uma pergunta ou resposta). Os nós devem aparecer na ordem produzida por um percurso pré-ordem da árvore. Por exemplo, as duas árvores mostradas nos diagramas e/ou logs nas páginas anteriores seriam representadas pelos seguintes conteúdos:

questao1.txt

```
Q: É um animal?
R: gato
Q: Ele vai em seus pés?
R: sapato
R: computador
```

questao2.txt

```
Q: É um animal?
Q: Pode voar?
R: pássaro
Q: Tem cauda?
R: rato
R: aranha
Q: Tem rodas?
R: bicicleta
Q: É legal?
R: blá blá blá
R: professor
```

4 O Que Deve Ser Entregue

- Código fonte do programa em Java, C ou C++ (bem edentado e comentado). *Outras linguagens também podem ser utilizadas desde que tenham prévia autorização do professor.*
- Documentação do trabalho. Entre outras coisas, a documentação deve conter:

Nota: A documentação consiste em um documento $\text{\LaTeX 2}_{\epsilon}$ (.tex) e .pdf correspondente de acordo com os itens abaixo.

- **Introdução:** descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
- **Implementação:** descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos, os quais serão muito bem vindos e serão considerados na avaliação), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
- **Listagem de testes executados:** os testes executados devem ser simplesmente apresentados.
- **Conclusão:** comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
- **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da **Internet** se for o caso.
- **Formato:** mandatoriamente em PDF; e obrigatoriamente em $\text{\LaTeX 2}_{\epsilon}$. O arquivo .tex deve ser entregue também.

Observação 1: Consulte as dicas do Prof. Nívio Ziviani de como deve ser feita uma boa implementação e documentação de um trabalho prático: <http://www.dcc.ufmg.br/~nivio/cursos/aed2/roteiro/>

- **Como deve ser feita a entrega:** A entrega DEVE ser feita pelo email: `flaviocefetrp@gmail.com` na forma de um único arquivo zipado, contendo o código, os arquivos e a documentação.

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- Clareza, edentação e comentários no programa também vão valer pontos;
- **O trabalho pode ser feito individualmente ou em grupo de dois alunos;**
- Trabalhos copiados (e FONTE) terão nota zero;
- Trabalhos entregue em atraso serão aceitos, todavia a nota atribuída ao trabalho será zero;
- Evite discussões inócuas com o professor em tentar postergar a data de entrega do referido trabalho.

5 O Arquivo de Entrada e o de Saída

Abaixo você encontra um exemplo de arquivo de entrada. Teste o seu programa com esse arquivo!

```
Q: É uma planta?
Q: Produz frutos?
R: laranjeira
R: roseira
Q: É uma pessoa?
Q: É homem?
R: Flávio
R: Márcia
Q: É um animal?
Q: É um felino?
R: gato
Q: Uiva?
R: cachorro
R: lobo
Q: É espacial?
R: sol
R: cometa
```

O programa pedirá ao operador o nome do arquivo de entrada e o nome do arquivo de saída. Esta é uma possível interação do operador com o programa:

No início de um jogo:



Devo lembrar dos nossos jogos anteriores? **S**

Qual o nome do arquivo? **questao1.txt**

No fim de um jogo:

Devo lembrar-me desses jogos? **S**

Qual é o nome do arquivo? **questao2.txt**

Se o nome do arquivo de entrada for vazio (isso acontecerá caso o usuário digite a tecla  quando o programa pedir o nome do arquivo de entrada, sem digitar nada antes da tecla ), o programa considerará que a árvore será inicializada vazia). De modo análogo, se o nome do arquivo de saída for vazio, o programa considerará que a árvore não será salva, perdendo-se o aprendizado de máquina.

Se o nome do arquivo de entrada for não-vazio, o programa lerá os dados de entrada do arquivo especificado, preenchendo a árvore com as perguntas e respostas nele contido. Se o nome do arquivo de saída for não-vazio, o programa escreverá os dados de saída no arquivo especificado, sobrescrevendo-o caso já exista e o usuário confirme.

Espera-se tratamento adequado dos problemas de I/O no caso de inexistência do arquivo de entrada e no caso da existência do arquivo de saída.

5.1 A Criação de um Arquivo de Entrada

O arquivo de entrada deve ser um arquivo de texto puro (ASCII), sem informações de formatação. Para criar um arquivo de entrada, use o mesmo editor de texto que você usa para editar programas em C. Não use programas como o Word ou outros editores de texto que incluem no arquivo informações de formatação tais como tipos e tamanhos de fontes, dimensões das margens, distâncias entre linhas, divisão do texto em parágrafos, etc. Certifique-se que o seu arquivo de entrada não contém linhas em branco no final, pois isso pode fazer diferença para o seu programa. Certifique-se também que todas as linhas do arquivo de entrada (inclusive a última) são terminadas por um caractere de fim de linha (`new line`)².

²O caractere de nova linha (`\n`) é equivalente ao caractere de avanço de linha ASCII (0A hexadecimal). Portanto, para arquivos abertos no modo de texto, lidas em pares CR/LF como caracteres de nova linha e os caracteres são gravados como CR/LF. Essas

6 A Leitura do Arquivo de Entrada e a Gravação no Arquivo de Saída

Para fazer a leitura de um arquivo de entrada e a gravação da saída em um arquivo, você precisa primeiro abrir o arquivo de entrada e o arquivo de saída. Faça isso utilizando a seguinte receita no seu programa:

Código Fonte 1: Código no Início do Programa

```
1  \#define MAX_NOME_ARQ 256
2  ...
3
4  /*----- Declarações das variáveis para leitura e gravação em arquivos -----*/
5
6  char buffer[MAX_NOME_ARQ];          /* para ler os nomes dos arquivos */
7
8  char nome_arq_entrada[MAX_NOME_ARQ] = ""; /* nome do arquivo de entrada */
9  char nome_arq_saida[MAX_NOME_ARQ] = "";  /* nome do arquivo de saída */
10 FILE *entrada;                       /* arquivo de entrada aberto */
11 FILE *saida;                          /* arquivo de saída aberto */
12 ...
```

Código Fonte 2: Código Para Tratamento do Arquivo de Entrada

```
1  /* Leitura do nome do arquivo de entrada */
2  printf("Digite o nome do arquivo de entrada: ");
3  fgets(buffer, MAX_NOME_ARQ, stdin);
4  sscanf(buffer, "%s", nome_arq_entrada);
5
6  /* Se o nome do arquivo de entrada for não-vazio,
7  abre o arquivo de entrada para leitura ("r") */
8
9  if (nome_arq_entrada[0] != '\0') {
10     entrada = fopen(nome_arq_entrada, "r");
11     if (entrada == NULL) {
12         fclose(entrada);
13         printf("Arquivo de entrada não encontrado!\n");
14         exit(1);
15     }
16     else {
17         /* Coloque aqui o código para ler e inserir os dados
18         na árvore */
19         fclose(entrada);
20     }
21 }
22 else {
23     arvore = NULL;
24 }
```

informações se aplicam ao STDIN, STDOUT e STDERR, que são abertos no modo de texto por padrão.

Código Fonte 3: Código Para Tratamento do Arquivo de Saída

```
1  /* Leitura do nome do arquivo de saida */
2  printf("Digite o nome do arquivo de saida: ");
3  fgets(buffer, MAX_NOME_ARQ, stdin);
4  sscanf(buffer, "%s", nome_arq_saida);
5
6  /* Se o nome do arquivo de saida for nao-vazio,
7  abre o arquivo de saida para escrita ("w") */
8
9  if (nome_arq_saida[0] != '\0') {
10     saida = fopen(nome_arq_saida, "r");
11     if (saida == NULL) {
12         fclose(saida);
13         saida = fopen(nome_arq_saida, "w");
14         if (saida == NULL) {
15             fclose(saida);
16             printf("Erro na abertura do arquivo de saida!\n");
17             exit(1);
18         }
19     }
20     else {
21         /* Coloque aqui o código para salvar a árvore */
22         fclose(saida);
23     }
24 }
25 else { /* Sobrescrever? */
26     /* Coloque aqui o código para perguntar ao usuário
27     se deseja sobrescrever o arquivo de saída */
28     fclose(saida);
29 }
```

Código Fonte 4: Código Para Tratamento dos Arquivos de Log

```
1  /* Crie código semelhante para os arquivos de log também */
```

Tendo feito isso, a variável `entrada` se referirá ao arquivo de entrada e a variável `saida` se referirá ao arquivo de saída. Note que as variáveis `entrada` e `saida` têm um tipo especial (`FILE *`), que representa um arquivo aberto. É esse o tipo devolvido pela função `fopen`, que abre um arquivo para leitura ("`r`") ou para escrita ("`w`"). Observe, ainda, que esse é o único trecho do programa que usa os nomes dos arquivos. Esses nomes só são usados para abrir os arquivos. Daí para a frente, qualquer referência ao arquivo de entrada será feita por meio da variável `entrada` e qualquer referência ao arquivo de saída será feita por meio da variável `saida`. Desde que não tenham sido fechados pela função `fclose(entrada)` ou `fclose(saida)`.

Durante a leitura do arquivo de entrada utilize a função `fscanf` para ler dados do arquivo de entrada e a função `fprintf` para escrever dados no arquivo de saída. Essas funções (que também estão na biblioteca `<stdio.h>`) funcionam de modo semelhante ao `scanf` e ao `printf`. A única diferença é que elas recebem um parâmetro a mais, que especifica o arquivo do qual o `fscanf` lê os dados ou no qual o `fprintf` escreve os dados. Para informar ao `fscanf` qual é o arquivo de entrada, passe como

parâmetro a variável `entrada`. Para informar ao `fprintf` qual é o arquivo de saída, passe como parâmetro a variável `saida`. Neste trabalho prático o arquivo de entrada deve ser lido linha a linha (utilizando-se o formato de leitura `"%s"`), pois as linhas representam strings de perguntas e respostas. Seu programa deve fechar o arquivo de entrada depois de ler todos os dados. Faça isso com a seguinte chamada:

```
fclose(entrada);
```

Seu programa deve fechar o arquivo de saída depois de gravar todos os dados. Faça isso com a seguinte chamada:

```
fclose(saida);
```

7 Observações

O seu programa deve começar com um cabeçalho (uma sequência de linhas de comentários) como o seguinte:

```
/* **** */
/* Aluno: Fulano de Tal */
/* Matrícula: 9999-17 */
/* Curso: Ciência da Computação */
/* 2º Trabalho Prático -- Vinte Questões */
/* DCC254 -- 2017 -- IFSEMG, turma Especiais/K */
/* Prof. Flávio Augusto de Freitas */
/* Compilador: ... (gcc ou Code::Blocks) versão ... */
/* Sistema Operacional: ... */
/* **** */
```

8 Aluno ou Grupo de Alunos

O trabalho prático é individual, ou no máximo em dupla³.

³**dupla:** *substantivo feminino*. 1. conjunto de duas entidades, seres, objetos etc. de igual natureza. 2. qualquer associação de duas pessoas orientadas para o mesmo propósito (social, profissional, artístico etc.). "d. de alunos"

8.1 Plágio

Veja a política do Departamento Acadêmico de Ciência da Computação para casos de plágio ou cola.

Trabalhos plagiados ou pinçados da Internet receberão nota ZERO.

9 Compilação

Exercícios com erros de sintaxe (ou seja, erros de compilação) receberão nota ZERO. Seu programa deve ser compilável sem erros ou *warnings*, da maneira especificada abaixo (que usa o compilador num modo em que quase todos os *warnings* são emitidos).

Para compilar seu programa, use o gcc ou o Code::Blocks (que na verdade chama o gcc para fazer a compilação). Caso você use diretamente o gcc, passe ao compilador (na linha de comando) as seguintes opções:

```
-Wall -ansi -pedantic -O2 -U_FORTIFY_SOURCE
```

Caso você use o **Code::Blocks**, entre em **Settings** » **Compiler and debugger...** » **Compiler settings** » **Compiler Flags**, selecione as quatro opções correspondentes a **-Wall**, **-ansi**, **-pedantic** e **-O2**, e clique em **OK**. Entre também em **Settings** » **Compiler and debugger...** » **Compiler settings** » **Other options**, digite **-U_FORTIFY_SOURCE** na caixa de texto **Other options** e clique **OK**.

Caso você use a linguagem Java, não inclua firulas, como janelas gráficas e coisas rebuscadas. Atenha-se à formatação exigida neste documento. O programa deverá executar no modo console ou terminal (Linux *flavored*). Seu programa deve estar bem endentado, documentado e organizado. A endentação deve deixar clara a estrutura de subordinação dos comandos. Os comentários devem ser esclarecedores. Toda função deve ser precedida de um comentário que diz o que a função faz. As funções devem ser razoavelmente pequenas, na medida do possível, e cada uma delas deve ter um propósito bem definido. A saída do programa deve ser clara. A avaliação levará em conta todas essas questões! Uma apresentação ruim, ou a falta de clareza do programa ou da saída do programa, poderá prejudicar sua nota.

O PROGRAMA DEVE SER ENTREGUE POR E-MAIL PARA
FLAVIOCEFETRP@GMAIL.COM OU
FLAVIO.FREITAS@IFSUDESTEMG.EDU.BR.

Entregue todos os arquivos zipados num único arquivo com o nome **tp2-<matrícula-aluno>.zip**. Exemplo: Se seu número de matrícula for 9999-17, você deverá entregar um arquivo com o nome **tp2-9999-17.zip**. (Note que não há espaços no nome do arquivo.) Caso tenha feito em dupla, **entregue todos os arquivos zipados num único arquivo com o nome tp2-<matrícula-aluno1_matrícula-aluno2>.zip**. Exemplo: Se seu número de matrícula for 8888-17 e a de seu colega for 9999-17, você deverá entregar um arquivo com o nome **tp2-8888-17_9999-17.zip**. (Note que não há espaços no nome do arquivo.) A primeira versão do programa entregue até o prazo final de entrega será considerada como a entrega do trabalho, não mais sendo aceitas outras versões. Então, só envie quando tiver certeza de que o programa atende a todas as especificações. Encerrado o prazo, não aceitarei mais a entrega do trabalho, desconsiderando qualquer e-mail posterior. **Não deixe para entregar seu programa na última hora!**

Guarde uma cópia do seu programa pelo menos até o final do semestre.

10 Entrega

A entrega do trabalho será no dia **31 de maio de 2017** até 23h:59m:59s (data e hora de envio do e-mail na minha caixa de entrada)⁴, impreterivelmente. **Exercícios enviados com atraso NÃO serão corrigidos e receberão nota ZERO.**

⁴**NÃO HAVERÁ ADIAMENTOS DESTA VEZ. NÃO INSISTAM.**