UNIVERSIDADE ESTADUAL DE SANTA CRUZ
COLEGIADO DE CIÊNCIA DA COMPUTAÇÃO

PROJETOS DE CURSO

CET641 - Linguagens de Programação II

Prof. Dany Sanchez Dominguez

ILHÉUS-BA, BRASIL
DEZEMBRO DE 2015

Sumário

1 Instruções para codificação do projeto

Esta seção apresenta um resumo das regras de estilo que devem ser seguidas ao construir seus programas, este texto foi adaptado do material de aulas do professor Ulysses de Oliveira da Universidade Federal da Paraíba.

Em geral, é crucial que você apresente não apenas um programa que simplesmente funciona, mas leve ainda em consideração os critérios de legibilidade, manutenibilidade e portabilidade, que classifiquem o produto final como um programa de qualidade. Um bom programa reflete um bom projeto de solução para o respectivo problema. Portanto, pense bastante sobre a linha de solução (algoritmo) a ser seguida antes de começar a codificar seu programa. As recomendações de estilo compreendem: uso de comentários, regras de endentação, uso de espaços em branco, escolha de identificadores e interface de usuário.

Comentários devem ser utilizados para clarificar o programa e devem consistir de sentenças claras, escritas em bom e conciso Português. Idealmente, os comentários devem ser escritos simultaneamente com a construção do programa. Se você é daqueles que terminam de escrever o programa para depois comentá-lo, experimente o inverso: escreva os comentários primeiro e depois escreva o programa. O uso de comentários deve considerar

- Blocos de comentários no inicio do programa, informando a finalidade do programa e dados do autor, por exemplo

```
/****
*
* Título:
*
* Autor:
*
* Data de Criação:
* Última modificação:
*
* Descrição: [i.e., o que e como o programa faz]
*
* Entrada: [i.e., o tipo de entrada esperada; coloque um exemplo]
*
* Saída: [i.e., a saída esperada; um exemplo pode ser recomendável]
*
****/
```

- Blocos de comentários precedendo a definição de uma função, descreve a função e como utilizá-la, por exemplo

```
/****
*
* Título:
*
* Autor:
*
* Data de Criação:
* Última modificação:
*
* Descrição: [i.e., o que e como o a função faz]
*
* Parâmetros: ... (entrada): [descrição do parâmetro]
* ... (saída): [descrição do parâmetro]
* ... (entrada/saída): [descrição do parâmetro]
*
* Valor de retorno: [descrição do valor retornado pela função]
*
****/
```

- Comentários na declaração das variáveis, cada variável que tem papel significativo na função ou programa deve ter um comentário associado à sua declaração que clarifica o papel da variável. Variáveis que não sejam significativas não precisam ser comentadas. Exemplo de comentário em declaração de variável

```
float pesoDoAluno; /* O peso do aluno em quilogramas */
```

- Comentários para seções de código, cada seção (sequência de instruções) de uma função que executa uma dada tarefa, deve ser precedida por um bloco de comentário explicando o propósito da seção e o algoritmo utilizado pela mesma (se este não for trivial).

- Comentários de fim de bloco, este tipo de comentário é necessário para indicar a quem pertence um dado fecha-chaves (“}”) em blocos longos ou aninhados. Estes comentários são dispensáveis quando um bloco é suficientemente curto.

Comentários devem ser escritos por programadores para programadores proficientes na linguagem utilizada na codificação do programa e têm o objetivo de documentar e esclarecer pontos que de outra forma seriam ambíguos ou difíceis de entender. Isto significa, entre outras coisas, que comentários não precisam serem didáticos ou escritos para leigos (como ocorre nos livros e manuais de programação). Portanto, não comente

instruções que tenham significado óbvio. Por último, o uso de comentários deve ser feito com uma boa dose de bom senso.

A endentação visa melhorar a legibilidade do código. Fixe o espaço de tabulação em quatro para endentação (esta medida não é aleatória, mas sim resultado de pesquisa). Espaços menores do que este podem não ser muito legíveis, enquanto que espaços maiores o farão chegar rapidamente ao final da linha na tela. Endentação deve ser utilizada para indicar que as instruções endentadas estão sob controle da instrução anterior não-endentada. O uso consistente de endentação é essencial para legibilidade do programa. Alguns exemplos:

```
if (x) {
    ...
}
else if (y){
    ...
}
else {
    ...
} /* Final do if */

while (x) {
    ...
} /* Final do while */

for (i = 0; i < 9; i++){
    ...
} /* Final do for */
```

O uso judicioso de espaços em branco é essencial para a legibilidade do programa. Isto inclui espaços horizontais em torno de operandos e operadores em expressões, em torno de comentários, para alinhar identificadores sendo declarados, etc. Espaços verticais também são importantes para separar funções, blocos, conjuntos de instruções com alguma afinidade lógica dentro de um bloco, etc. Entretanto, não exagere no uso de espaços em branco, pois o uso exagerado de espaços em branco prejudica a legibilidade, ao invés de melhorá-la.

O uso consistente de convenções para a criação de **identificadores** para as diversas entidades que compõem um programa facilita a leitura do programa. A seguir, são resumidas algumas sugestões gerais para criação de identificadores.

- Nomes de variáveis. Comece com letra minúscula; se o nome da variável for composto utilize letra maiúscula no início de cada palavra seguinte, inclusive palavras de ligação (e.g., preposições e conjunções); não utilize sublinha.
- Nome de tipos. Siga a regra acima para nomes de variáveis, mas comece com a letra "`t`"ou termine com "`_t`". Exemplo, `tListaEncadeada` (ou `listaEncadeada_t`).
- Nomes de funções. Utilize a mesma regra para nomes de variáveis, mas comece com letra maiúscula.

Um elemento importante a ser considerado na escolha de um identificador é sua representatividade, a seguir alguns coment[arios sobre a representatividade de um identificador

- Identificadores que exercem papeis importantes no programa devem ter nomes que sejam significativos com relação aos respectivos papeis exercidos (e.g., `nomeDoAluno` é muito melhor do que simplesmente `x`).
- Identificadores com importância menor, não precisam ter nomes significativos. Por exemplo, uma variável utilizada apenas como variável de controle num laço for pode ser denominada `i` (i.e., não precisa ser denominada contador, por exemplo).
- Não siga como exemplos as nomenclaturas de funções de biblioteca de C, pois elas são horríveis em termos de legibilidade e coerência.
- Não utilize identificadores nem muito longos nem muito abreviados: encontre um meio-termo que seja sensato e legível.

A construção da interface de usuário é um problema complexo que será abordado em outras disciplinas do curso, entretanto, algumas recomendações minimas devem ser atendidas:

- Todo programa interativo deve iniciar sua execução apresentando ao usuário informações sobre o que o programa faz, seu autor, versão e qualquer outra informação pertinente para o usuário do programa.
- Toda entrada de dados deve ser precedida por uma indicação (prompt) informando ao usuário o tipo de entrada que o programa espera que o usuário introduza e o formato destes dados.
- Se for o caso, o programa deve ainda informar ao usuário que ação ele deve executar numa certa entrada de dados, se esta ação não for óbvia. Por exemplo, você não precisa dizer ao usuário para pressionar uma tecla para introduzir um

caractere, basta dizer digite um caractere. Mas, você precisa informar ao usuário como executar uma operação não usual, tal como pressione simultaneamente as teclas ALT, SHIFT e A, ao invés de digite ALT-SHIFT-A.

- Toda saída de dados deve ser compreensível para o usuário.
- O programa deve informar ao usuário o que ele deve fazer para encerrar o programa ou uma dada iteração.
- Tenha sempre em mente que o programa deverá ser usado por usuários comuns e não por profissionais de computação. Portanto, não faça suposições sobre o nível intelectual dos usuários (a não ser que isto seja explicitamente pressuposto no enunciado do problema).

2 Critérios de avaliação

A avaliação do projeto compreende a avaliação do programa construído, e a defesa pública do trabalho perante o professore da disciplina. Cada uma destas atividades receberá uma nota de 0 a 10 pontos. A nota final do projeto será a média ponderada, tendo a defesa do projeto peso 40, e o código peso 60.

A avaliação do código contempla a funcionalidade do programa e o estilo de programação. A funcionalidade do programa, envolve oferecer a saída solicitada para um conjunto de dados de entrada. Ademais a construção do algoritmo e os dados utilizados, devem refletir os conceitos e boas práticas de programação discutidos em sala de aulas, dentre delas, uso eficiente da memória considerando uso de ponteiros e alocação dinâmica, uso correto das estruturas de controle, uso de técnicas de modularização e construção de um algoritmo correto e enxuto.

Na tabela 1 são oferecidos alguns critérios que ilustram as notas que podem ser atribuídos a alguns programas.

O estilo de programação envolve os critérios descritos na seção anterior sob uso de comentários, endentação e escolha de identificadores dentre outros. Alguns exemplos das notas que podem ser atribuídas em relação ao estilo de programação são oferecidos na Tabela 2

A defesa do projeto será feita mediante uma discussão/apresentação entre o aluno e o professor. O aluno deverá utilizar seu programa para resolver alguns exemplos que serão apresentados pelo professor. O discente deverá comentar a linha de raciocínio utilizada na solução do problema, e **detalhar** o algoritmos e as estruturas de dados utilizadas. Caso o discente não consiga explicar exaustivamente o código que foi implementado será considerada uma **solução fraudulenta** e receberá **nota zero**.

Tabela 1 – Critérios de notas para a funcionalidade do programa.

Pontuação	**Características do Programa**
10	Absolutamente (ou convincentemente) correto e robusto. Uso adequado das técnicas e recursos da linguagem.
8-9	Quase correto; contém alguns bugs triviais que não comprometem o funcionamento do programa como um todo; não prevê todas as entradas possíveis e pode quebrar em situações não usuais. Alguns pequenos problemas no uso de recursos da linguagem como uso incorreto de funções, ou algoritmo ineficiente.
6-8	Basicamente correto, mas contém alguns erros comprometedores; quebra com relativa facilidade. Problemas no uso dos recursos da linguagem.
3-6	Abordagem de solução do problema é razoável, mas contém muitos erros; quebra com muita facilidade. Uso inadequado dos recursos da linguagem, por exemplo, não utilizar funções auxiliares.
1-3	Basicamente, apenas sintaticamente correto; o programa funciona, mas os resultados são errados.
0	Não consegue sequer ser compilado devido a erros de sintaxe.

Tabela 2 – Critérios de notas para o estilo de programação.

Pontuação	**Características do Programa**
10	Estilo perfeito; uma obra de arte.
8-9	Algoritmos e estruturas de dados adequados; boa interface do usuário (em termos de apresentação e facilidade de uso); bem documentado; tão simples quanto possível; identificadores bem escolhidos; fácil de ler e entender.
6-8	Algoritmos e estruturas de dados não foram bem escolhidos; interface do usuário rudimentar; documentação e clareza descuidadas; mais complexo do que deveria; identificadores não são representativos; razoavelmente fácil de ler e entender.
3-6	Algoritmos e estruturas de dados ruins; interface do usuário que apenas o próprio programador sabe usar; documentação ruim ou ausente; difícil de ser entendido.
1-3	Entende-se com muito sacrifício.
0	Ilegível; não é um trabalho digno de um profissional de Computação.

Programas em que forem constatados algum tipo de fraude receberão nota 0,0 (zero). Na prática, isto significa que, se o seu programa for semelhante a algum outro além da casualidade, ambos receberão nota zero, independentemente do fato de você ter copiado ou ter sido copiado. Você é responsável por sua proteção. Portanto, proteja-se!

Você pode incluir em seus exercícios de programação, porções de código (mas, não programas inteiros!) encontradas em livros e em outras fontes. Se este for o caso,

você deve incluir na documentação (i.e., nos comentários) do programa sua fonte de inspiração. Se você não proceder desta maneira e sua fonte for descoberta, seu programa será considerado fraudulento e receberá a devida avaliação.

O prazo para entregar o projeto será divulgado pelo professor na sala de aulas, os alunos que não se apresentarem poderão solicitar a segunda chamada no DCET e remarcar à avaliação. Entretanto, independentemente da justificativa, cada dia (ou fração de dia) de atraso valerá o desconto de um ponto na nota atribuída ao programa.

3 Problemas

Problema 1. Torres de Hanoi

O problema das Torres de Hanoi é um problema clássico em Ciências da Computação, geralmente utilizado para ilustrar os conceitos de programação recursiva. O problema foi criado pelo matemático francês Edouard Lucas, e estabelece, que dada três hastes $(A, B,$ e $C)$, existem n discos de vários tamanhos no haste A, colocados em ordem decrescente segundo o tamanho. Os discos deverão ser movimentados do haste A para o haste B respeitando as seguintes restrições:

- nunca colocar um disco maior sobre um disco menor,
- pode-se mover um único disco por vez,
- nunca colocar um disco em outro lugar que não numa das três hastes.

Construa um programa que resolva o problema das Torres de Hanoi para um número arbitrário de discos, seu programa deve mostrar a sequencia de movimentos de forma gráfica, por exemplo para $n = 3$ a saída do programa dever ser semelhante a Figura 1. Para valores muito grandes de n ocorrerá o desbordamento da tela, seu programa deve funcionar corretamente para $n \leq 10$. Adicionalmente, seu programa deverá criar um arquivo texto mostrando a saída de programa.

Problema 2. Labirinto do Minotauro

Seja um labirinto descrito através de uma matriz quadrada de inteiros com dimensão n, onde cada posição com valor igual a verdadeiro (1) corresponde a uma passagem livre e uma posição falsa (0) representa uma obstrução, adicionalmente se uma posição tiver o valor 2, significa que nessa posição se encontra o Minotauro. Escreva um programa que encontre um caminho que leve um guerreiro de uma posição inicial qualquer a uma saída do labirinto, caso exista. Uma saída é uma posição livre na borda da matriz que define o labirinto, se durante o percurso o guerreiro encontra o Minotauro

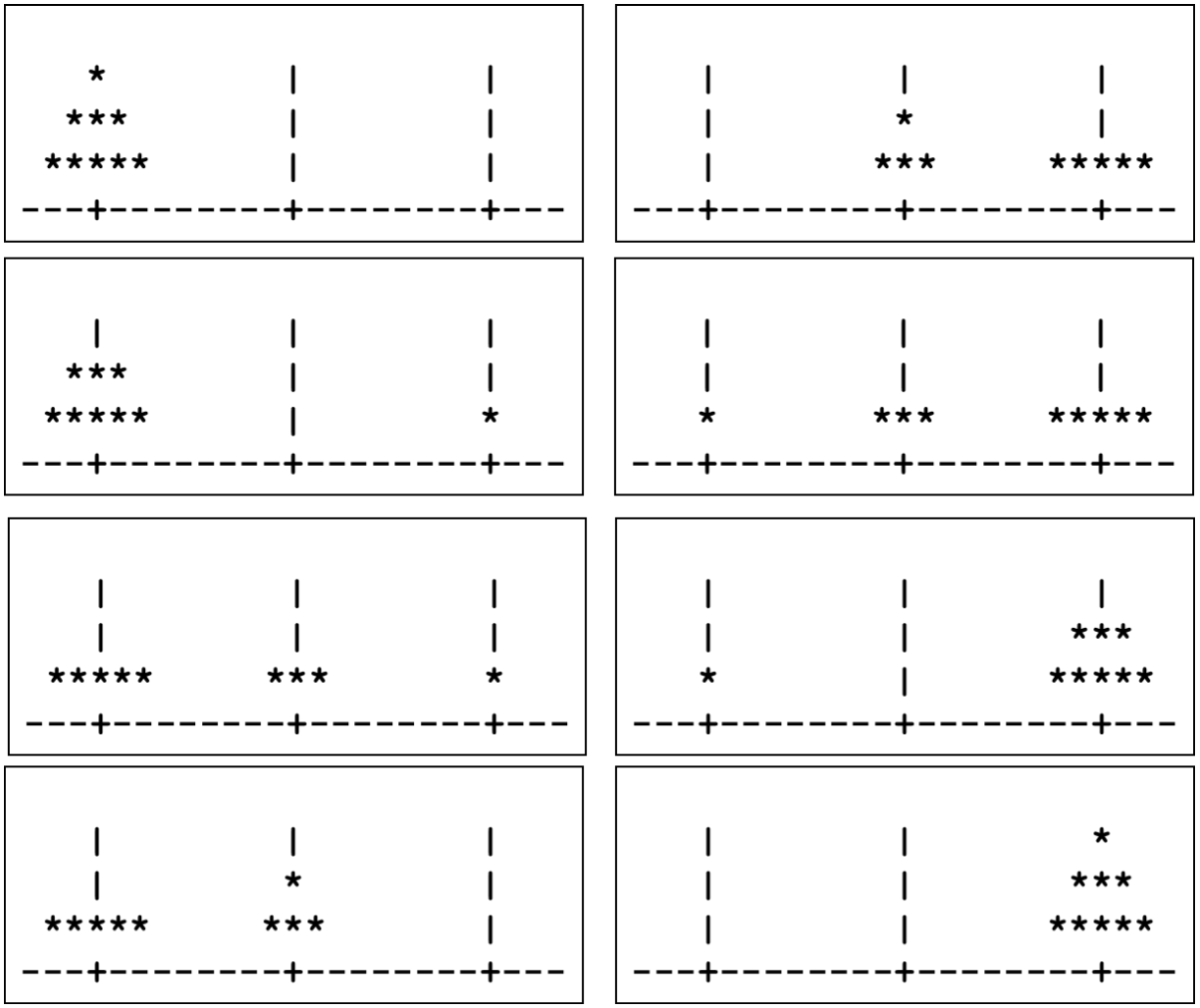

Figura 1 – Saída do programa que resolve as Torres de Hanoi para $n = 3$.

a caminhada termina com a morte do guerreiro. Considere como entrada um arquivo texto que na primeira linha contem a posição inicial do guerreiro no labirinto (x_g, y_g), na segunda linha a posição do Minotauro (x_m, y_m), na terceira a dimensão do labirinto n, e logo a matriz de inteiros com a estrutura do labirinto. Seu programa deverá criar um arquivo de saída mostrando o percurso realizado, o caminho até a saída, ou uma mensagem informando que o caminho não existe, ou que o guerreiro morreu.

Existe um algoritmo simples para atravessar um labirinto que garante a localização da saída (admitindo que há uma saída). Se não houver uma saída, você chegará novamente ao local de partida. Coloque sua mão direita na parede de sua direita e comece a andar para a frente. Nunca tire sua mão direita da parede, se o labirinto dobrar para a direita, siga a parede da direita. Contanto que sua mão direita não seja retirada da parede, você acabará chegando à saída. Pode haver um caminho mais curto, mas essa estrategia garante sua saída do labirinto. Veja duas instancias do problema do labirinto e sua solução na Figura 2.

Instancia 1: O guerreiro morreu.

Entrada

```
2 6
4 8
12
1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 0 0 0 0 0 0 1
0 0 1 0 1 0 1 1 1 1 0 1
1 1 1 0 1 0 0 0 0 1 0 1
1 0 0 0 0 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0 1 0 1
1 0 0 1 0 1 0 1 0 1 0 1
1 1 0 2 0 1 0 1 0 1 0 1
1 0 0 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 0 1 1 1 0 1
1 0 0 0 0 0 0 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1
```

Saída

```
1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 0 0 0 0 0 0 1
0 0 1 0 1 0 1 1 1 1 0 1
1 1 1 0 1 0 0 0 0 1 0 1
1 0 0 0 0 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0 1 0 1
1 0 0 1 0 1 0 1 0 1 0 1
1 1 0 2 0 1 0 1 0 1 0 1
1 0 0 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 0 1 1 1 0 1
1 0 0 0 0 0 0 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1
```

Instancia 2: O guerreiro achou a saída.

Entrada

```
7 3
4 7
12
1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 0 0 0 0 0 0 1
0 0 1 0 1 0 1 1 1 1 0 1
1 1 1 0 1 0 0 0 0 1 0 1
1 0 0 0 0 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0 1 0 1
1 0 0 2 0 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 0 1 0 1
1 0 0 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 0 1 1 1 0 1
1 0 0 0 0 0 0 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1
```

Saída

```
1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 0 0 0 0 0 0 1
0 0 1 0 1 0 1 1 1 1 0 1
1 1 1 0 1 0 0 0 0 1 0 1
1 0 0 0 0 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0 1 0 1
1 0 0 2 0 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 0 1 0 1
1 0 0 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 0 1 1 1 0 1
1 0 0 0 0 0 0 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1
```

Figura 2 – Exemplos para o problema de travessia no labirinto.

Problema 3. O Código do Batman

O Batman precisa criar um mecanismo para envio e recebimento de mensagens entre ele e Robin, entretanto, se o conteúdo das mensagens chegarem as mãos de Curinga, a cidade de Gotham esta perdida. Para resolver o problema o mordomo Alfred teve uma brilhante ideia propondo, utilizar um algoritmo de criptografia de transposição por coluna. A regra de encriptação é bem simples os caracteres da mensagem

omitindo os espaços em branco são dispostos em uma matriz de k colunas, e em seguida a matriz é transposta (a posição m_{ij} converte-se na posição m^t_{ji}), a mensagem cifrada aparece nas linhas da nova matriz. A regra de decodificação segue o processo inverso. Veja o exemplo da Figura 3 onde a mensagem "O ataque sera ao amanhecer da sexta" é criptografada e depcriptografada considerando $k = 5$.

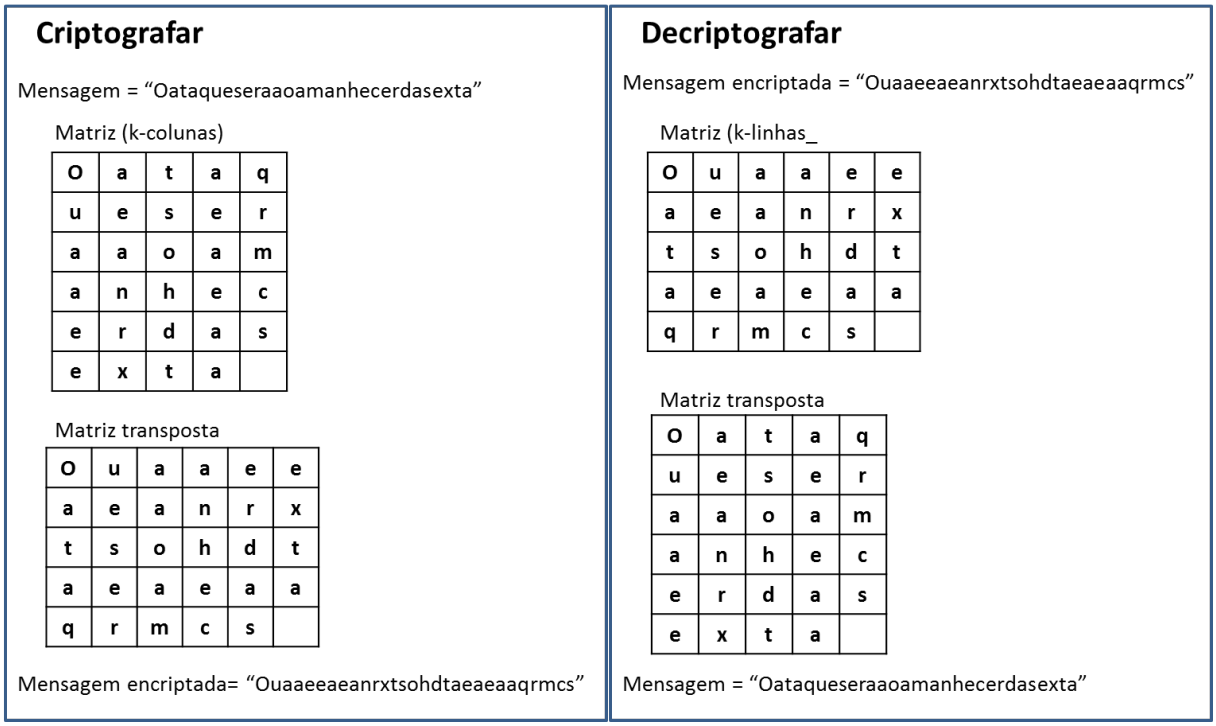

Figura 3 – Mensagem criptografada e decriptografada com transposição de coluna.

Escreva um programa que permita criptografar e descriptografar mensagens para intercambiar de noticias entre Batman e Robin. Seu programa deve mostrar um menu com as opções (i) Criptografar, (ii) Descriptografar e (iii) Sair. Para as opções (i) e (ii) deve ler nome do arquivo de entrada contendo a mensagem, e a chave k (numero de colunas). Em seguida deve executar a operação solicitada, mostrar a mensagem de saída na tela, e criar um arquivo texto de saída com a mensagem. Seu programa deve funcionar para mensagens de quaisquer tamanhos.

Problema 4. Gráfico de Tartaruga

A linguagem Logo, que é particularmente popular entre os usuários de computadores pessoais, tornou famoso o conceito dos gráficos da tartaruga. Imagine uma tartaruga mecânica que percorra uma sala sob o controle de um programa em C. A tartaruga possui uma caneta em uma de duas posições, para cima ou para baixo. Quando a caneta está para baixo, a tartaruga desenha figuras à medida que se move; quando a caneta está para cima, a tartaruga se move livremente sem desenhar nada. Neste problema,

você simulará a operação da tartaruga e criará também um esquema computadorizado de movimento.

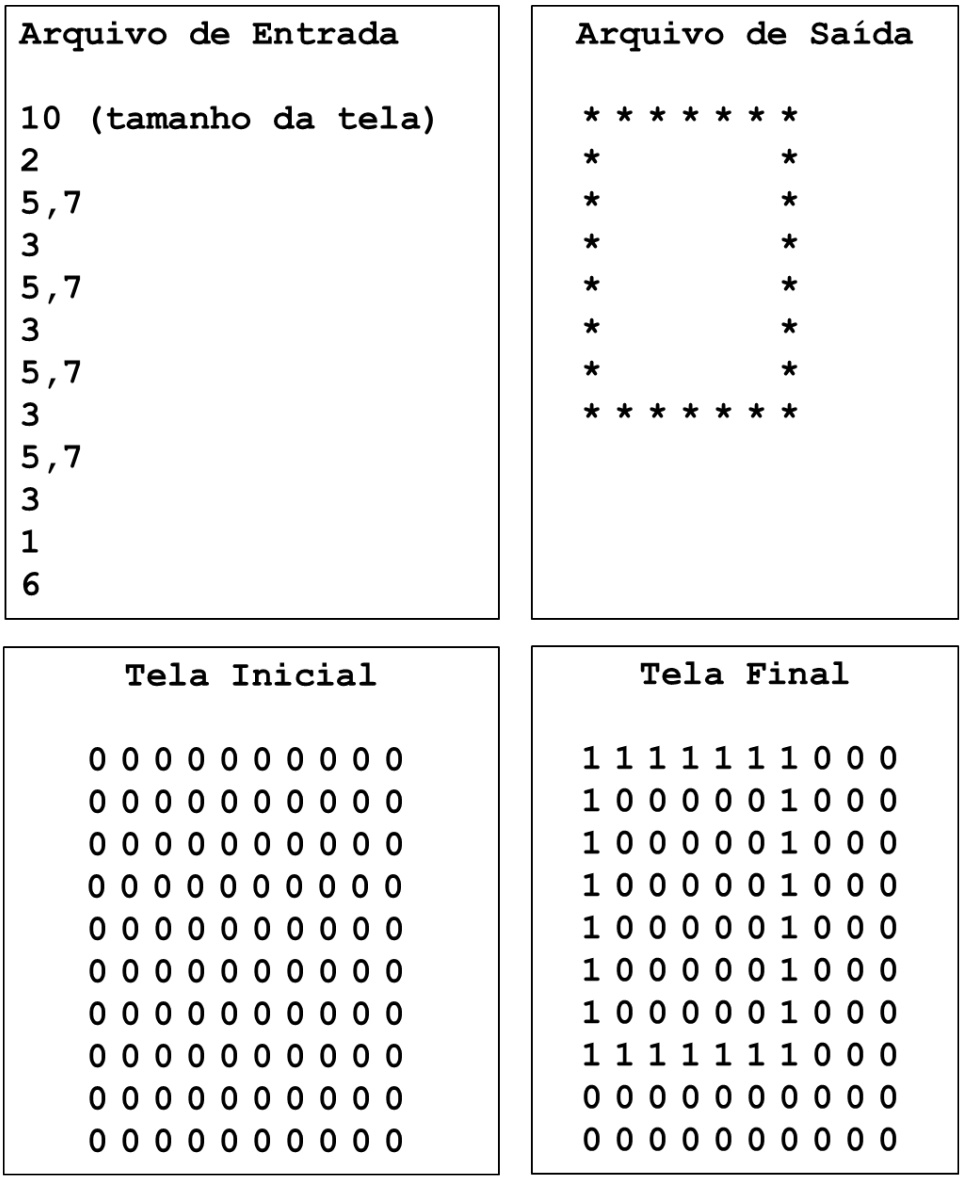

Figura 4 – Exemplo para o problema do gráfico da tartaruga.

Use uma matriz bidimensional A de $n \times n$ inicializada com zeros para representar a tela de desenho da tartaruga. Leia comandos operacionais da tartaruga de um arquivo de entrada. Controle sempre a posição atual da tartaruga e se a caneta está para cima ou para baixo. Admita que a tartaruga sempre começa na posição 0,0 do plano com sua caneta para cima. O conjunto de comandos da tartaruga que seu programa deve processar são os seguintes:

Comando	**Significado**
1	Caneta para cima
2	Caneta para baixo
3	Giro para a direita
4	Giro para a esquerda
5,k	Movimenta k espaços a frente
9	Fim dos dados (sentinela)

Na medida que a tartaruga se move com a caneta para baixo, um desenho aparece na tela da tartaruga, defina os elementos apropriados da matriz A como 1. Quando imprimir o desenho onde houver um 1 em A exiba um asterisco. Sempre que houver um zero, exiba um espaço em branco. Escreva um programa em C para implementar os recursos do gráfico da tartaruga descritos aqui. Escreva vários programas de gráficos da tartaruga para desenhar formas interessantes. Considere o exemplo da Figura 4. O gráfico de tartaruga correspondente deve aparecer na tela do computador e um arquivo texto de saída.

Problema 5. Terreno do novo aeroporto

Para construir a pista do novo aeroporto de Ilhéus, é preciso uma faixa de terreno o mais plana que for possível, minimizando as movimentações de terra e os custos da construção. O que se pretende, então, é determinar o maior retângulo (o retângulo com maior área) na zona do futuro aeroporto, tal que a diferença de altitude entre o ponto mais alto e o ponto mais baixo desse retângulo seja inferior ou igual a k metros. Para realizar os cálculos dispomos de altitudes de uma zona retangular com M quilômetros quadrados (c_t quilômetros de comprimento por l_t de largura sendo $M = l \times c$). Essa zona foi dividida em quadrados de 100 metros de lado, e para cada um desses quadrados registrou-se a altitude (média). Há, portanto, n números a processar, sendo $n = (M \times 100)$. Escreva um programa que receba um arquivo de entrada contendo os dados do problema e a matriz de alturas, e informe as coordenadas (linha inicial, coluna inicial e linha final, coluna final) da faixa de terreno solicitada. Seu programa deve verificar se os dados do arquivo são consistentes. Utilize simbolos e crie um arquivo texto de saída onde se mostra de forma “grafica” a saída do problema. Veja um exemplo de arquivo de entrada no figura 5.

```
5    #Diferença de altura em metros
1,2 #Área total do terreno em km²
1,0  #Comprimento do terreno em Km
1,2  #Largura do terreno em Km
#Matriz de alturas
72  78  65  71  90  65  79  79  83  75
66  66  62  88  62  72  86  63  88  67
82  85  63  64  83  83  82  87  69  66
78  67  72  79  70  83  78  80  86  85
84  82  63  64  64  75  82  62  84  89
78  75  83  88  88  80  84  82  80  89
77  66  81  73  79  77  71  63  65  89
87  76  81  70  71  81  67  82  64  69
70  84  85  65  67  81  77  87  72  89
85  70  87  74  78  89  69  73  70  72
64  66  80  79  85  61  88  87  84  67
72  66  75  83  61  70  78  64  88  73
```

Figura 5 – Exemplo para o problema do gráfico da tartaruga.

4 Distribuição de problemas vs Discentes

Discente	**Problema**
ADSON SANTOS CARDOSO (201420368)	3
ANDRÉ DE SOUZA ALVES (201511472)	4
DIONNES MARINHO SANTOS (201211091)	5
EBERTY ALVES DA SILVA (201511455)	1
EGLOM SOSTENES PEREIRA DA SILVA (201511456)	2
ELYSSON ARAÚJO GUEDES SILVA (201512354)	3
FELIPE OLIVEIRA CARDOSO (201511470)	4
GABRIEL FIGUEIREDO GÓES (201420373)	5
HIVAN CORDEIRO DOS SANTOS JÚNIOR (201511476)	1
IURI CARVALHO PASSOS (201411236)	2
JEFSON ALVES MATOS (201511651)	3
JÔNATAS MOREIRA DE CARVALHO (201512355)	4
MARCOS ROGERIO PINHEIRO SOUSA (201512356)	5
PAULO CÉSAR LESSA BEZERRA (201511462)	1
PRISCILLA TRINDADE ESTRÊLA (201320058)	2
RAÍ SALES PEREIRA BIZERRA (201511463)	3
RAMON SENA MATOS SANTOS (201511481)	4
RICARDO FERREIRA NEVES SILVA (201511471)	5
THALES AUGUSTO RAMOS VIEIRA (201511465)	1
UESLEI SALES VIEIRA MAGALHÃES (201420394)	2
JOENDERSON CHAVES	3