

Universidade Federal de Goiás – UFG  
Instituto de Informática – INF  
Bacharelados (Núcleo Básico Comum)

Algoritmos e Estruturas de Dados 1 – 2019/2

Lista de Exercícios nº 09 – Lista Extra

Turmas: INF0286, INF0011, INF0061 – Prof. Wanderley de Souza Alencar)

## **Sumário**



## 1 Preenchedor de cheque



(+++)

Apesar do uso de cartões eletrônicos ter, nos últimos cinco anos, ampliado sua participação como forma de pagamento de transações financeiras, segundo pesquisa realizada pelo Serviço de Proteção ao Crédito (SPC Brasil) e pela Confederação Nacional de Dirigentes Lojistas (CNDL), em 2018, 8% ainda utilizam cheque pré-datado para realizar compras, ou seja, 16,62 milhões de pessoas. Em média, uma pessoa utiliza cheque nove vezes por ano e os produtos mais adquiridos são: alimentos em supermercados (34%), materiais de construção (20%) e móveis (18%).

Um dos problemas que surge neste contexto é realizar o “preenchimento” dos cheques de maneira automática, ou seja, utilizando uma máquina “preenchedora de cheques” (veja a figura no cabeçalho desta questão) ou um programa de computador que realize esta tarefa por meio do uso de uma impressora.

A empresa em que você trabalha adotou a segunda opção e você está na equipe responsável por desenvolver a aplicação “PreencheCheque”. Sua tarefa é implementar um programa que seja capaz de receber como entrada um número com duas casas decimais e gerar, como saída, o valor daquele número “por extenso”.

Por exemplo, se o programa receber o valor 1986,32, ele deverá imprimir:

UM MIL NOVECENTOS E OITENTA E SEIS REAIS E TRINTA E DOIS CENTAVOS

### Entrada

A primeira linha da entrada contém um número inteiro  $t \in \mathbb{N}^*$ ,  $0 < t \leq 50$ , que corresponde ao número de casos de teste.

As  $t$  linhas seguintes possuem, cada um, um número  $x$  com duas casas decimais, sendo que  $0,01 \leq x \leq 1.000.000,00$ .

### Saída

O programa deverá imprimir, em cada linha saída, o valor por extenso do caso de teste correspondente.

**Observação 01:** A saída deve sempre ser com letras maiúsculas, sem uso de acentuação gráfica e com

um único espaço entre as palavras que a formam.

**Observação 02:** O número “14” deve ser escrito como ”CATORZE”.

### Exemplos

Entrada	Saída
4	UM CENTAVO
0,01	UM REAL
1,00	UM REAL E UM CENTAVO
1,01	UM REAL E ONZE CENTAVOS
1,11	

Entrada	Saída
5	DOIS REAIS E SESSENTA E CINCO CENTAVOS
2,65	SETE REAIS E CINQUENTA E SEIS CENTAVOS
7,56	TRES REAIS E TRINTA E TRES CENTAVOS
3,33	DOZE REAIS E QUARENTA E SETE CENTAVOS
12,47	CENTO E VINTE E OITO REAIS E SESSENTA E UM CENTAVOS
128,61	

Entrada	Saída
5	UM MIL E UM REAIS E UM CENTAVO
1001,01	UM MIL E UM REAIS E ONZE CENTAVOS
1001,11	UM MIL E DEZ REAIS E UM CENTAVO
1010,01	UM MIL E DEZ REAIS E ONZE CENTAVOS
1010,11	UM MIL CENTO E ONZE REAIS E ONZE CENTAVOS
1111,11	

<b>Entrada</b>	<b>Saída</b>
5	NOVECIENTOS E NOVENTA E NOVE MIL E NOVECIENTOS E NOVENTA E
999999,00	NOVE REAIS
999999,11	NOVECIENTOS E NOVENTA E NOVE MIL E NOVECIENTOS E NOVENTA E
999999,33	NOVE REAIS E ONZE CENTAVOS
999999,99	NOVECIENTOS E NOVENTA E NOVE MIL E NOVECIENTOS E NOVENTA E
1000000,00	NOVE REAIS E TRINTA E TRES CENTAVOS
	NOVECIENTOS E NOVENTA E NOVE MIL E NOVECIENTOS E NOVENTA E
	NOVE REAIS E NOVENTA E NOVE CENTAVOS
	UM MILHAO DE REAIS

47	53	59	61	67	71	73	79	83	89	97	101	103	107
109	113	127	131	137	139	149	151	157	163	167	173	179	181
191	193	197	199	211	223	227	229	233	239	241	251	257	263
269	271	277	281	283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409	419	421	431	433
439	443	449	457	461	463	467	479	487	491	499	503	509	521
523	541	547	557	563	569	571	577	587	593	599	601	607	613
617	619	631	641	643	647	653	659	661	673	677	683	691	701
709	719	727	733	739	743	751	757	761	769	773	787	797	809
811	821	823	827	829	839	853	857	859	863	877	881	883	887

## 2 Números Primos (1)



(+++)

Alana, yma estudante de Computação é fascinada pelos *números primos* e, por isso, ela resolveu desenvolver um programa para fazer o seguinte:

Primeiramente o programa recebe um número natural  $n$  e gera uma lista com todos os números primos no intervalo de 2 a  $n$ , inclusive extremos. Considere haja  $p$  números primos neste intervalo.

Em seguida, o programa recebe dois números,  $x$  e  $y$ , ambos no intervalo de 1 a  $p$ , inclusive extremos e com  $x \leq y$ , e imprime todos os números primos que ocupam da posição  $x$  até a posição  $y$ , inclusive estas, na lista anteriormente gerada.

### Entrada

A primeira linha de entrada contém o número natural  $n$ ,  $1 \leq n \leq 10.000$ .

A segunda linha contém os números  $x$  e  $y$ , nesta ordem e separados por um único espaço em branco entre eles.

### Saída

O programa deve apresentar, numa única linha, todos os números primos que ocupam da posição  $x$  à posição  $y$  na lista de números primos que está entre 2 e  $n$ , inclusive extremos.

### Exemplos

Entrada	Saída
100 5 11	11 13 17 19 23 29 31

Entrada	Saída
1000 29 41	109 113 127 131 137 139 149 151 157 163 167 173 179

<b>Entrada</b>	<b>Saída</b>
1000 131 137	739 743 751 757 761 769 773

<b>Entrada</b>	<b>Saída</b>
10000 1118 1127	9001 9007 9011 9013 9029 9041 9043 9049 9059 9067

<b>Entrada</b>	<b>Saída</b>
10000 169 174	1009 1013 1019 1021 1031 1033



### 3 Números Primos (2)



(++)

Alana, a nossa estudante de Computação, continuando seus estudos a respeito dos números primos, resolveu desta vez escrever um programa  $\mathbb{C}$  para determinar qual é a quantidade de números primos existentes entre dois números naturais dados, digamos  $x$  e  $y$ .

#### Entrada

A única linha contém um par de números naturais,  $x$  e  $y$ , separados por um único espaço em branco e com  $1 \leq x \leq y \leq 10.000$ .

#### Saída

Seu programa deve imprimir uma linha contendo a quantidade de números primos existentes entre  $x$  e  $y$ , considerando inclusive estes, se eles forem primos.

**Exemplos**

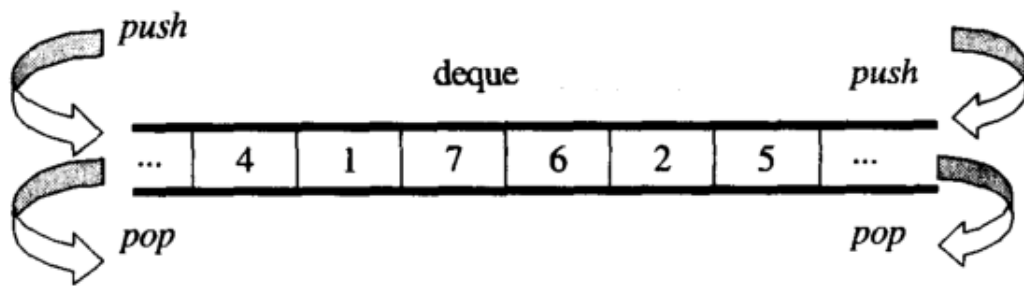
Entrada		Saída
1	100	25

Entrada		Saída
1	1000	168

Entrada		Saída
9000	10000	112



<b>Entrada</b>		<b>Saída</b>
1	10000	1229



## 4 Deque



(++++)

Um *deque* é um conjunto de elementos (itens, objetos, etc.) que são mantidos em ordem linear, onde as operações de *inserção* (ou *push*) e de *remoção* (ou *pop*) podem ser efetivadas em qualquer uma de suas extremidades.

Para facilitar denominemos as extremidades de esquerda e direita e, por suas vez, as operações de *pushEsquerda*, *pushDireita*, *popEsquerda* e *popDireita*.

É também possível “listar” os elementos existentes no *deque* por meio das operações *listEsquerda* e *listDireita*, sendo que a primeira mostra os elementos da esquerda para a direita e a segunda da direita para a esquerda.

Você está implementando, em  $\mathbb{C}$ , um programa para manipular *deques* de números naturais e, por isso, deve implementar as seguintes operações:

- (1) `int criarDeque(d)` – cria o deque *d*, inicialmente vazio. Se tiver sucesso, retorna SUCESSO. Do contrário, retorna FALHA.
- (2) `int pushEsquerda(d, elem)` – insere na extremidade esquerda do deque *d* o elemento *elem*. Se tiver sucesso, retorna SUCESSO. Do contrário, retorna FALHA.
- (3) `int pushDireita(d, elem)` – insere na extremidade direita do deque *d* o elemento *elem*. Se tiver sucesso, retorna SUCESSO. Do contrário, retorna FALHA.
- (4) `int popEsquerda(d)` – remove, e retorna, o elemento que está na extremidade esquerda do deque *d*. Se o deque estiver vazio, retorna FALHA.
- (5) `int popDireita(d, elem)` – remove, e retorna, o elemento que está na extremidade direita do deque *d*. Se o deque estiver vazio, retorna FALHA.
- (6) `int listEsquerda(d)` – apresenta, numa única linha da saída, os elementos presentes no deque *d*. A apresentação é realizada da extremidade esquerda para a direita. Se o deque estiver vazio, mostra apenas a palavra “vazio” (em letras minúsculas). Se tiver sucesso, retorna SUCESSO. Do contrário, retorna FALHA.
- (7) `int listDireita(d)` – apresenta, numa única linha da saída, os elementos presentes no deque *d*. A apresentação é realizada da extremidade direita para a esquerda. Se o deque estiver vazio, mostra apenas a palavra “vazio” (em letras minúsculas). Se tiver sucesso, retorna SUCESSO. Do contrário, retorna FALHA.

**Observação:** A constante FALHA vale -1 (menos um) e a SUCESSO VALE 1 (um).

## Entrada

A entrada será uma sequência de operações a serem realizadas sobre um único deque  $d$ , sendo que cada operação aparecerá numa linha.

A primeira linha conterá, sempre, a operação de criação do deque  $d$  – `criarDeque(d)` – e a última linha terá a palavra “fim” (escrita em letras minúsculas).

A partir da segunda linha, demais operações podem ser realizadas em qualquer ordem que o usuário desejar.

## Saída

A saída apresentará os resultados das operações `listEsquerda(d)` e `listDireita(d)`, sempre uma por linha.

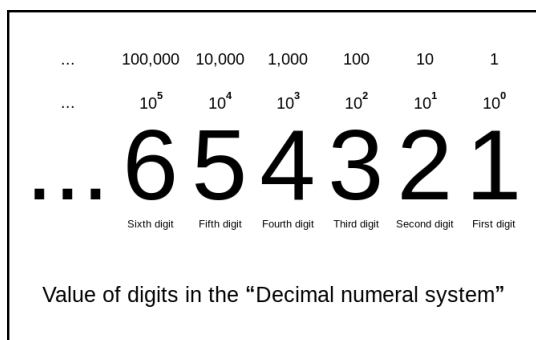
Deverá também apresentar a palavra “falha” (em letras minúsculas) se qualquer uma das operações retornar FALHA. Neste caso, o programa deverá ser interrompido imediatamente.

## Exemplos

Entrada	Saída
<code>criarDeque(d)</code> <code>pushEsquerda(d, 3)</code> <code>pushEsquerda(d, 7)</code> <code>pushDireita(d, 8)</code> <code>pushEsquerda(d, 5)</code> <code>listEsquerda(d)</code> <code>popEsquerda(d)</code> <code>listEsquerda(d)</code> <code>popEsquerda(d)</code> <code>listDireita(d)</code> <code>fim</code>	<code>5 7 3 8</code> <code>7 3 8</code> <code>8 3</code>         

Entrada	Saída
<code>criarDeque(d)</code> <code>pushEsquerda(d, 3)</code> <code>pushDireita(d, 7)</code> <code>pushEsquerda(d, 8)</code> <code>pushDireita(d, 5)</code> <code>listEsquerda(d)</code> <code>popEsquerda(d)</code> <code>listEsquerda(d)</code> <code>popEsquerda(d)</code> <code>listDireita(d)</code> <code>popEsquerda(d)</code> <code>popEsquerda(d)</code> <code>listEsquerda(d)</code> <code>fim</code>	<code>8 3 7 5</code> <code>3 7 5</code> <code>5 7</code> <code>vazio</code>            

Entrada	Saída
criarDeque(d) popEsquerda(d) pushDireita(d, 8) fim	falha



## 5 Contador de dígitos



(++)

Escreva, em  $\mathbb{C}$ , uma função recursiva que seja capaz de determinar a quantidade de dígitos iguais a  $k$ , com  $k \in \{0, 1, 2, \dots, 8, 9\}$ , contidos na representação decimal do número  $n$  fornecido como entrada.

### Entrada

A primeira linha contém um número natural  $t$ ,  $1 \leq t \leq 50$ , que corresponde ao número de casos de teste. A segunda linha contém o dígito  $k$ . A terceira linha contém os  $t$  números naturais separados por um único espaço em branco entre eles.

### Saída

Uma única linha contendo o número de ocorrências do dígito  $k$  em cada um dos casos de teste, sempre separados por um único espaço em branco entre eles.

### Exemplos

Entrada	Saída
2 9 192 127	1 0

Entrada	Saída
5 8 10 30 48 820 880	0 0 1 1 2

Entrada	Saída
10 1 111 121 717 818 1881 701 818 991 1001 10	3 2 1 1 2 1 1 1 2 1

**Observação:** Apenas a implementação que utilize *recursividade* será considerada válida como resposta, ou seja, uma implementação iterativa, mesmo que com todas as saídas corretas, não será aceita.



## 6 Busca Binária



(+++)

Considere que seja dado um vetor  $v$  de números naturais considerados como *chaves primárias*. Sobre este vetor deseja-se realizar a operação de pesquisa (ou busca) por uma determinada destas chaves. Um método eficiente para realização desta pesquisa é denominado de “Busca Binária”. Entretanto ele exige que o vetor esteja ordenado.

Seguindo as especificações a seguir, você deve implementar um algoritmo *recursivo* para realizar a busca binária em  $v$ .

### Entrada

Na primeira linha será fornecido o número natural  $t$ ,  $1 \leq t \leq 10$ , que representa o número de casos de testes.

Cada caso de teste possui duas linhas de entrada: (1) a primeira contém o valor da chave,  $k$ , a ser pesquisado no vetor  $v$ ; (2) a segunda possui os elementos de  $v$ , da primeira posição em direção à última. Para indicar a finalização da entrada dos dados do vetor, o valor  $-1$  (menos um) é utilizado como *flag*.

Tanto os elementos de  $v$  quanto a chave de pesquisa  $k$  variam no intervalo de 1 a  $(2^{32} - 1)$ , inclusive extremos.

### Saída

Seu programa deve imprimir, para cada caso de teste, uma linha contendo a posição em que a chave  $k$  foi encontrada no vetor  $v$  ou, no caso de sua ausência, a palavra “ausente” (com todas as letras minúsculas).

### Exemplos

Entrada	Saída
3	6
8	1
1 7 9 4 2 8 3 5 6 10 -1	ausente
1	
1 7 9 4 2 8 3 5 6 10 -1	
14	
1 7 9 4 2 8 3 5 6 10 -1	

Entrada	Saída
3	6
81	10
18 71 93 44 25 81 13 15 61 100 -1	ausente
108	
101 711 19 224 235 889 389 511 619	
108 -1	
141	
11 17 91 24 652 18 31 587 601 110 -1	

**Observação:** Apenas a implementação que utilize *recursividade* será considerada válida como resposta, ou seja, uma implementação iterativa, mesmo que com todas as saídas corretas, não será aceita.



Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(\log(n))$
Mergesort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$
Timsort	$O(n)$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$
Heapsort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(1)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Shell Sort	$O(n)$	$O((n \log(n))^2)$	$O((n \log(n))^2)$	$O(1)$
Bucket Sort	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$

## 7 Ordenação



(+++)

A operação de *ordenar* dados é muito frequente em computação e, por isso, conhecer os principais algoritmos de ordenação é fundamental para qualquer profissional envolvido com a programação de computadores.

Há um grande número de algoritmos de ordenação *interna* bastante conhecidos, dentre eles citam-se:

- selection sort;
- insertion sort;
- bubble sort;
- quick sort;
- merge sort;
- counting sort;
- radix sort;
- bucket sort.

Escolha um deles para, de acordo com as especificações a seguir, resolver esta questão.

### Entrada

A entrada é formada por uma sequência de cadeias de caracteres, sendo uma por linha.

Para indicar a finalização da entrada de cadeias de caracteres utiliza-se a palavra “ASC” ou a palavra “DESC”. A palavra “ASC” indica que a saída deverá apresentar as cadeias em ordem “lexicográfica ascendente” e a palavra “DESC” em ordem “lexicográfica descendente”.

### Saída

A saída apresenta as cadeias originalmente fornecidas em ordem lexicográfica “ascendente” ou “descendente”, conforme indicado para última linha da entrada.

## Exemplos

Entrada	Saída
Agora eu era o herói E o meu cavalo só falava inglês A noiva do cowboy Era você além das outras três Eu enfrentava os batalhões Os alemães e seus canhões Guardava o meu bodoque E ensaiava o rock para as matinês ASC	A noiva do cowboy Agora eu era o herói E ensaiava o rock para as matinês E o meu cavalo só falava inglês Era você além das outras três Eu enfrentava os batalhões Guardava o meu bodoque Os alemães e seus canhões

Entrada	Saída
Agora eu era o herói E o meu cavalo só falava inglês A noiva do cowboy Era você além das outras três Eu enfrentava os batalhões Os alemães e seus canhões Guardava o meu bodoque E ensaiava o rock para as matinês DESC	Os alemães e seus canhões Guardava o meu bodoque Eu enfrentava os batalhões Era você além das outras três E o meu cavalo só falava inglês E ensaiava o rock para as matinês Agora eu era o herói A noiva do cowboy

Entrada	Saída
Agora eu era o rei Era o bedel e era também juiz E pela minha lei A gente era obrigado a ser feliz E você era a princesa que eu fiz coroar E era tão linda de se admirar Que andava nua pelo meu país ASC	A gente era obrigado a ser feliz Agora eu era o rei E era tão linda de se admirar E pela minha lei E você era a princesa que eu fiz coroar Era o bedel e era também juiz Que andava nua pelo meu país

**Observação:** Apesar os exemplos apresentarem acentuação gráfica, os casos de teste no *Sharif Judge System* do INF/UFG foram fornecidos sem qualquer acentuação, inclusive sem o uso do “ç” (c com cedilha).

8	11	14	1
13	2	7	12
3	16	9	6
10	5	4	15

## 8 *Magic Square Matrix*



(+++)

Uma *Magic Square Matrix* (MSM) é uma matriz quadrada de números inteiros, de ordem  $n \times n$ ,  $n \in \mathbb{N}^*$ , com linhas e colunas numeradas a partir de 1, que apresenta as seguintes propriedades:

- (1) Todos os números no intervalo de 1 a  $n^2$  aparecem na matriz e o fazem uma única vez;
- (2) A soma dos elementos de cada linha, coluna ou diagonal da matriz é idêntica.

Você deve desenvolver um programa que seja capaz de, a partir de  $n$ , gerar e imprimir a MSM de ordem  $n \times n$ , exceto quando  $n = 2$ , quando se deve imprimir a mensagem “nao existe”.

### Entrada

A única linha da entrada contém  $n \in \mathbb{N}^*$ , a ordem da matriz, com  $1 \leq n \leq 50$ .

### Saída

A saída é uma das possíveis *magic squares* de ordem  $n \times n$ , conforme mostram os exemplos.

### Exemplos

Entrada	Saída
1	1
Entrada	Saída
2	nao existe

**Observação-01:** Note que não existe MSM de ordem  $2 \times 2$  e, por isso, neste caso a saída deve ser a mensagem “não existe” (em letras minúsculas, mas sem acentuação).

**Observação-02:** Há apenas uma MSM de ordem  $3 \times 3$ , desde de que se desconsidere as operações de *rotação* e *reflexão* aplicadas da matriz. Há 880 MSMs de ordem  $4 \times 4$  e 275.305.224 MSMs de ordem  $5 \times 5$ . Por isso, para esta questão, os “casos de teste” no *Sharif Judge System* do INF/UFG são apenas exemplos e correspondem aos aqui apresentados.

Entrada	Saída
3	2 7 6 9 5 1 4 3 8

Entrada	Saída
4	2 16 13 3 11 5 8 10 7 9 12 6 14 4 1 15

Entrada	Saída
5	17 24 1 8 15 23 5 7 14 16 4 6 13 20 22 10 12 19 21 3 11 18 25 2 9



## 9 Desemprego



(++++)

Um dos enormes problemas brasileiros na atualidade é a gigantesca “fila de desempregados” e, na tentativa de contribuir para sua redução, uma ONG de cadastra, diariamente, um grupo de  $N$  pessoas desempregadas por meio de um sistema computacional denominado de *SisEmprego*. Nele, cada pessoa desempregada é associada a um número único  $n_i$ , com  $1 \leq n_i \leq N$ , e, portanto, aquela pessoa fica identificada de maneira unívoca no sistema naquele dia.

O objetivo do *SisEmprego* é possibilitar que as pessoas recebam, de maneira gratuita, cursos de capacitação profissional e, por consequência, aumentem suas chances de obter uma colocação no mercado de trabalho.

Para ser “justo”, o programa que determina a sequência em que as pessoas serão encaminhadas para a capacitação gratuita aplicando a seguinte estratégia:

- (1) Todas as pessoas são, virtualmente, colocadas num grande *círculo* em que a contagem é realizada no sentido horário, ou seja, a pessoa 2 está à direita da pessoa 1, a pessoa 3 está à direita da pessoa 2 e, assim, sucessivamente, formando um “relógio” quanto aos números de identificação de cada uma das  $N$  pessoas no círculo.
- (2) Um número  $k$  é arbitrariamente definido e, a partir da pessoa de número 1, faz-se a contagem, no sentido horário, até atingir a  $k$ -ésima pessoa. A pessoa identificada é *retirada* do círculo.
- (3) Simultaneamente, um outro número,  $m$ , é também arbitrariamente definido e, faz-se a contagem a partir da pessoa de número  $N$ , mas no sentido anti-horário. A pessoa indenticada é *retirada* do círculo. Se ambas contagens resultarem na mesma pessoa, apenas ela é retirada do círculo.
- (4) Na sequência, ambas contagens são reiniciadas a partir da próxima pessoa disponível nos sentidos horário e anti-horário, e o processo continua até que ninguém mais reste no círculo.

Perceba que as duas pessoas escolhidas para sair deixam o círculo simultaneamente, assim é possível que uma das contagens inclua uma pessoa que já havia sido escolhida para sair devido à outra contagem.

Escreva um programa que leia, nesta ordem, os valores de  $N$ ,  $k$  e  $m$ , sabendo que  $0 < k, m < e$  que  $0 <$

$N < 60$ , e mostre em que ordem as pessoas são retiradas do círculo, pois é nesta ordem que elas serão encaminhadas para

### Entrada

A primeira linha da entrada conterá, nesta ordem, os valores de  $N$ ,  $k$  e  $m$ , separados por um único espaço em branco entre eles.

### Saída

Seu programa deve imprimir, numa única linha, a ordem em que as pessoas serão encaminhadas para a capacitação. As pessoas são identificadas por meio de seu número  $n_i$

### Exemplos

Entrada	Saída
10 4 3	4 8 9 5 3 1 2 6 10 7

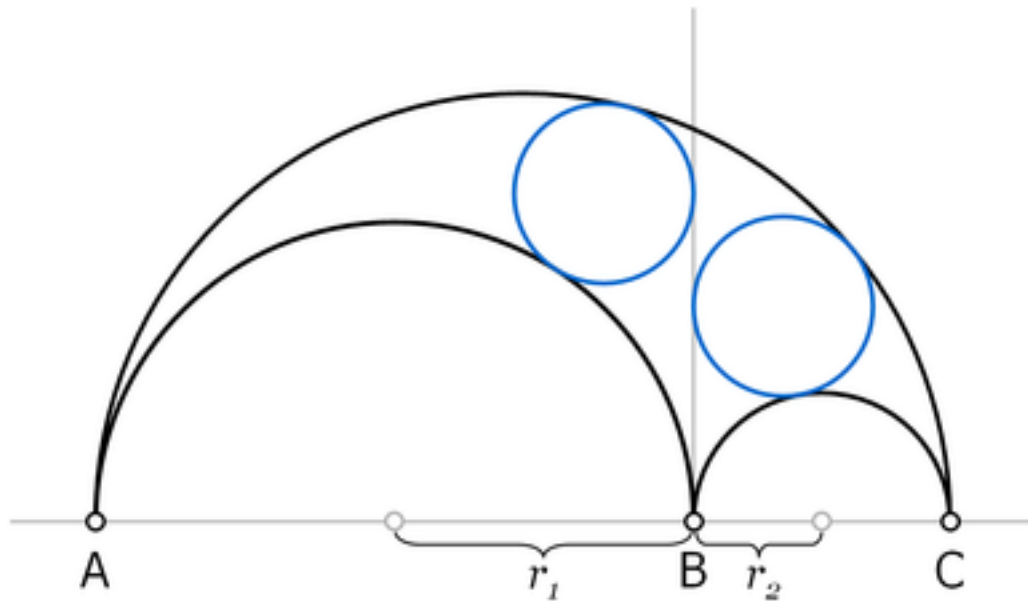
Detalhando o primeiro exemplo:

- A primeira linha define que  $N = 10$ ,  $k = 4$  e  $m = 3$ . Assim, a cada vez, vai-se contar até 4 no sentido horário, até 3 no sentido anti-horário e há 10 pessoas no círculo.
- Na primeira contagem, a pessoa de número 4 é a escolhida no sentido horário e a pessoa de número 8 é escolhida no sentido anti-horário pois, lembre-se, a numeração no círculo é realizada no sentido horário.
- Na segunda contagem, a pessoa de número 9 é a escolhida no sentido horário, pois a contagem começou na pessoa de número 5: a primeira pessoa, no sentido horário, depois daquela que havia sido retirada na primeira rodada. No sentido anti-horário, a escolhida é a pessoa de número 5, pois a contagem iniciou na pessoa de número 7: a primeira pessoa, no sentido anti-horário, depois da que havia sido retirada na primeira rodada.
- Na terceira contagem, a pessoa de número 3 é a escolhida no sentido horário, pois a contagem começou na pessoa de número 10. No sentido anti-horário, a escolhida é a pessoa de número 1, pois a contagem iniciou na pessoa de número 3. Lembre-se: numa rodada, ambas pessoas escolhidas são retiradas simultaneamente.
- Na quarta contagem, a pessoa de número 2 é a escolhida no sentido horário, pois a contagem começou na pessoa de número 6. No sentido anti-horário, a escolhida é a pessoa de número 6, pois a contagem iniciou na pessoa de número 10. Novamente: numa rodada, ambas pessoas escolhidas são retiradas simultaneamente.
- Na quinta contagem, a pessoa de número 10 é a escolhida no sentido horário, pois a contagem começou na pessoa de número 7. No sentido anti-horário, a escolhida é a pessoa de número 10, pois a contagem iniciou na pessoa de número 10, ou seja, uma única pessoa deixará o círculo desta vez, a pessoa de número 10.
- Na sexta e última contagem, resta apenas a pessoa número 7, que sai do círculo.
- Por fim, perceba que as saídas são sempre mostrando primeiro a pessoa que foi retirada pelo sentido horário e, depois, a pessoa que foi retirada pelo sentido anti-horário, embora, como dito, elas saiam simultaneamente.

Entrada	Saída
20 4 3	4 18 8 15 12 17 9 2 5 10 20 16 14 6 7 1 19 3 13 11

Entrada	Saída
10 2 2	2 9 4 7 6 5 10 1 8 3

Entrada	Saída
20 3 5	3 16 6 11 9 5 13 19 17 12 1 2 8 10 18 15 7 14 20 4



## 10 Círculos, círculos, círculos



(+++)

Uma programadora  $\mathbb{C}$  está desenvolvendo uma aplicação de computação gráfica em 2D e, para isso, precisa manipular círculos. Elanão deseja utilizar bibliotecas de *software*, ou seja: ela mesma quer construir sua *biblioteca* para manipular *circulos*.

Você gostou da ideia e quer contribuir. Depois de uma reunião de duas horas, vocês decidiram que haveria a necessidade de definir um *TAD* (Tipo Abstrato de Dados), e o fizeram assim:

```
typedef struct Circle*   CirclePointer;

struct Circle {
    double      x;        // coordenada x do centro do círculo
    double      y;        // coordenada y do centro do círculo
    double      radius;    // raio
    unsigned int lineColor; // código RGB da cor da linha
    unsigned int areaColor; // código RGB da cor de preenchimento
    bool        visible;   // O círculo é, ou não, visível?
};
```

Vocês também especificaram várias funções capazes de manipular círculos e, a você, coube implementar as seguintes:

```
(1) int createCircle (CirclePointer* c, Circle p);
(2) int destroyCircle (CirclePointer* c);
(3) int putLineColor (CirclePointer* c, unsigned int color);
(4) int putFillColor (CirclePointer* c, unsigned int color);
(5) int turnVisible (CirclePointer* c);
(6) int turnInvisible (CirclePointer* c);
```



```

(7) int getCircleCenter (CirclePointer c,
                        double *x, double *y);
(8) int getLineColor (CirclePointer c, unsigned int *color);
(9) int getAreaColor (CirclePointer c, unsigned int *color);
(10) int showCircle (CirclePointer c);
(11) int getRelativePosition (CirclePointer c1, CirclePointer c2,
                            unsigned int * position);

```

Cada uma das funções deverá ter o seguinte comportamento:

- (1) Cria, de maneira dinâmica, o círculo *c* a partir dos dados fornecidos na variável *p*, ou seja, centro (coordenadas *x* e *y*), raio, cor da linha, cor de preenchimento e visibilidade.
- (2) Destrói o círculo *c*.
- (3) Associa a cor *color* para a cor da linha do círculo *c*.
- (4) Associa a cor *color* para a cor de preenchimento do círculo *c*.
- (5) Torna o círculo *c* visível. Se ele já estava visível, continuará visível.
- (6) Torna o círculo *c* invisível. Se ele já estava invisível, continuará invisível.
- (7) Retorna, nos parâmetros *x* e *y*, os valores das respectivas coordenadas do círculo *c*.
- (8) Retorna, no parâmetro *color*, qual é a cor da linha do círculo *c*.
- (9) Retorna, no parâmetro *color*, qual é a cor de preenchimento do círculo *c*.
- (10) Mostra, no dispositivo de saída do computador (normalmente o monitor de vídeo), as informações referentes ao círculo *c*, no seguinte formato:  
Circulo(*x*; *y*; raio; cor da linha; cor de preenchimento; visibilidade)  
Por exemplo, uma impressão será:  
Circulo(2,826574; 7,272811; 1,298734; 172; 13656160; 255; true)  
para indicar um círculo cujo centro é (2,826574; 7,272811) e de raio igual a 1,298734. A cor de sua linha é a de código 13656160 (um certo tom de vermelho) e o preenchimento é da cor de código 255 (azul, puro) e, neste momento, ele está *visível*.
- (11) Retorna, no parâmetro *position*, o código da posição relativa do círculo *c1* em relação ao círculo *c2*. O código pode ser:
  - (a) **INSCRITO** – o círculo *c1* está *inscrito* no círculo *c2*.
  - (b) **CIRCUNSCRITO** – o círculo *c1* está *circunscrito* ao círculo *c2*.
  - (c) **COINCIDENTE** – o círculo *c1* é *coincidente* como o círculo *c2*, ou seja, todos os pontos de *c1* e *c2* são coincidentes.
  - (d) **TANGENTE** – o círculo *c1* é *tangente* ao círculo *c2*, ou seja, há um único ponto comum entre as circunferências de ambos.
  - (e) **SECANTE** – o círculo *c1* está *secante* ao círculo *c2*, ou seja, há dois pontos comuns entre as circunferências de ambos. **DISJUNTOS** – o círculo *c1* está *disjunto* em relação ao círculo *c2*, ou seja, não há pontos comuns entre ambos.

**Observações:**

- (a) Todas as funções devem retornar a constante `SUCCESS` (valor igual a 1 – um) quando for terminada com *sucesso* e a constante `FAILURE` (valor igual a -1) quando ocorrer qualquer tipo de erro. Assim o programa principal conseguirá identificar, pelo retorno da função, se a operação foi realizada corretamente.
- (b) Considere que as *cores* são expressas por um código numérico correspondente à tabela RGB (*red-green-blue*). Veja mais detalhes em [https://www.rapidtables.com/web/color/RGB\\_Color.html](https://www.rapidtables.com/web/color/RGB_Color.html).
- (c) As variáveis do tipo `double` devem ser impressas, sempre, com seis casas decimais de precisão.

A partir da definição, e implementação, deste *TAD*, vocês deverão elaborar um programa `C` para realizar os testes de mesa e validar se todas as operações estão corretamente implementadas.

### **Entrada**

De acordo com a descrição anterior, não há entrada de teste, pois o programa principal deverá ser elaborado para testar a implementação das funções definidas.

Sugere-se que um programa do tipo *menu driven* seja elaborado.

### **Saída**

De acordo com a descrição anterior, não há saída de teste, pois o programa principal deverá ser elaborado para testar a implementação das funções definidas.

Sugere-se que um programa do tipo *menu driven* seja elaborado.

### **Exemplos**

Não há exemplos para este exercício e, portanto, você deverá apenas submeter seu código-fonte (completo) para o *Sharif Judge System* do INF/UFG.