INSTRUÇÕES:

- Site para envio: https://github.com
- Instruções para envio: Google Classroom
- Nome da lista (pasta no github): lista04
  - Enviar ex1.cpp, . . . , exN.cpp, cada um com a solução do N exercícios correspondentes.
- Siga **estritamente** o formato da saída solicitada no enunciado.
- Permitido somente **conteúdo aprendido em aula ou em listas anteriores** a não ser que enunciado explicitamente contrário
- **Plágio será severamente punido**
- Dúvidas através de todos os meios de comunicação disponibilizados
- Não se preocupe com a precisão das respostas **numéricas não inteiras**. **Erros numéricos inferiores a 1% serão desconsiderados**, a não ser que o enunciado indique outra precisão desejada.
- Apenas bibliotecas padrões de C/C++ serão permitidas.

LISTA PRÁTICA

1. (0.5 pontos) Faça uma função

```
int elementok(std::stack<int>& pilha, int k);
```

que receba uma **pilha** e um número inteiro **k**. A função deve retornar o **k**-ésimo elemento par contado a partir do topo da pilha, sem modificá-la. Caso haja menos de **k** pares, retorne **-1**.

EXEMPLO

Entrada

```
#include <iostream>
int main() {
    // INICIALIZAÇÃO DA PILHA
    std::stack<int> pilha;
    for(int e=5; e>=1; e--)
        pilha.push(e);

    // TESTE DE FATO
    std::cout << elementok(pilha, 2) << std::endl;

    // TESTE SE A PILHA ESTÁ IGUAL
    while(not pilha.empty()) {
        std::cout << pilha.top() << " ";
```

```
        pilha.pop();
    }
    std::cout << std::endl;
    return 0;
}
```

Saída

```
4
1 2 3 4 5
```

2. (0.5 pontos) Faça uma função

```
std::string jogo(std::stack<int> cartas, std::queue<std::string> jogadores);
```

que recebe uma **pilha** de **cartas** contendo números naturais e distribui continuamente para a **fila** de **jogadores** até que as cartas acabem. O jogador que receber a menor **soma de cartas** ganha. A função deve retornar o jogador vencedor. Em caso de empate, tem vantagem o que vier antes em ordem alfabética. Para os nomes dos jogadores serão utilizados somente letras minúsculas, sempre sem acento.

EXEMPLO

Entrada

```
#include <iostream>
int main() {
    // INICIALIZAÇÃO DA PILHA
    std::stack<int> cartas;
    for(int e=1; e<=5; e++)
        cartas.push(e);

    // INICIALIZAÇÃO DA FILA
    std::queue<std::string> jogadores;
    jogadores.push("andre");
    jogadores.push("bruno");
    jogadores.push("carla");

    // TESTE DE FATO
    std::cout << jogo(cartas, jogadores) << std::endl;
    return 0;
}
```

Saída

```
carla
```

3. (2 pontos) Lembra dos experimentos de Caixa Preta nos laboratórios de física? Nesse exercício temos uma EDP (Estrutura de Eados Preta) que tem as seguintes operações:

- Adicionar um elemento à EDP: identidificada por ”add x”, sem aspas, onde x é um número itneiro
- Retirar um elemento da EDP: ”pop”, sem aspas

Dada uma sequência de operações e seus resultados, você deve fazer um programa que advinhe a estrutura de dados, imprimindo uma string dentre as seguintes:

- **stack**: a EDP definitivamente é uma Pilha
- **queue**: a EDP definitivamente é uma Fila simples
- **priority**: a EDP definitivamente é uma Fila de Prioridades, onde quanto maior o inteiro, maior a prioridade
- **none**: a EDP definitivamente não é nenhuma das antriores
- **both**: a EDP pode ser mais de uma dentre as anteriures

Cada operação é composta por:

- **add numero**: indica que **numero** foi adicionado à EDP
- **pop numero**: indica que foi aplicada a operação de remoção e o resultado foi **numero**

Veja no teste abaixo por exemplo, poderia ser tanto uma pilha quanto uma fila de prioridades.

EXEMPLO

Entrada	**Saída**

`add 1`
`add 2`
`add 3`
`pop 3`
`pop 2`
`pop 1`
	`both`

4. (1 ponto) Faça uma função

```
void ordena_strings(std::vector<std::string>& vetor);
```

que ordene um vetor de strings com letras minúsculas da menor para a maior. Em caso de duas strings do mesmo tamanho, o desempate deve ordená-las por ordem ao contrário da alfabética.

EXEMPLO

Entrada

```
int main() {
    std::vector<std::string> v;
    v.push_back("aaaaaaaaaaa");
    v.push_back("zzzzzzzzzzz");
    v.push_back("test");
    v.push_back("testando");
    v.push_back("testados");
    v.push_back("teste");
    ordena_strings(v);
    fos(auto s : v)
        std::cout << s << std::endl;
    return 0;
}
```

Saída

```
test
teste
testando
testados
zzzzzzzzzzz
aaaaaaaaaaa
```

5. (0.5 pontos) É muito comum encontrarmos em pesquisa científica matrizes gigantescas, com até milhões de linhas e colunas, mas que na verdade a maioria dos elementos são 0, sendo alguns poucos, de fato não nulos. Essas matrizes são chamadas **esparsas**.

   Perceba que se tentarmos representar uma matriz dessa como vimos em aula (usando array de array), teremos que armazenar da ordem de trilhões de elementos, o que daria uma matriz de mais de 4 TB, algo impraticável para qualquer computador (**NÃO FAÇA ISSO!!**).

   Uma forma mais inteligente de fazer isso é usando mapas (std::map). Nele temos uma chaves do tipo **std::pair¡int, int¿** indicando que o valor correspondente está na linha **i** e coluna **j**. Vamos assumir aqui que existirá uma chave (-1, -1) indicando **n**, que o número de linhas e colunas da matriz.

   Para ficar mais claro, vamos definir o tipo **Matriz**.

   Faça uma função

```
typedef std::map<std::pair<int, int>, int> Matriz;
Matriz soma(const Matriz& A, const Matriz& B);
```

   que receba duas matrizes representadas no formato descrito e retorne outra matriz, representada no mesmo formato que seja a soma das duas recebidas. É garantido que as matrizes fornecidas terão o mesmo tamanho. A ordem das linhas na saída não importa.

   EXEMPLO

Entrada	**Saída**

`#include <iostream>`
`int main() {`
`  Matriz A = { {{-1, -1}, 2}, {{0,0}, 1}, {{1,0}, 5}};`
`  Matriz B = { {{-1, -1}, 2}, {{1,0}, 5}};`
`  Matriz C = soma(A, B);`
 	
`  for(const auto& [k, v] : C)`
`  std::cout << k.first << " " << k.second << " "`
`  << v << std::endl;`
`  return 0;`
`}`
	`-1 -1 2`
	`0 0 1`
	`1 0 5`

6. (0.5 pontos) Faça uma função

```
std::vector<int> retira(std::vector<int> v, int n);
```

   que receba um **vetor** de números inteiros e um número **n**. Retorne uma cópia do vetor de números anterior em ordem crescente e sem repetição, exceto pelo número **x**, caso ele esteja na lista **sem usar estruturas condicionais e sem usar as bibliotecas stdlib.h e algorithm**.

   EXEMPLO

Entrada	Saída

```
#include <iostream>
int main() {
    std::vector<int> v = {5,1,8,4,3,5};
    for(auto e : retira(v, 8) )
        std::cout << e << std::endl;
    return 0;
}
```

```
1
3
4
5
```

7. (0.5 pontos) Faça uma função

```
std::string retira(std::string texto, char c);
```

que receba uma string **texto** e um caractere **c** e retorne uma cópia de **texto** sem nenhuma ocorrencia de **c**.

EXEMPLO

Entrada	Saída

```
#include <iostream>
int main() {
    std::cout << retira("Testes", 'e') << std::endl;
    std::cout << retira("Testes", 'X') << std::endl;
    return 0;
}
```

```
Tsts
Testes
```

8. (1 ponto) Faça uma função

```
typedef std::vector< std::vector<int> > Matriz;
void ordena_par(Matriz& M);
```

que ordene as linhas de uma matriz pela quantidade de números pares na linha, isto é, uma linha com 1 número par deve vir antes de uma linha com dois deles. Em caso de empate, ordene pelos elementos da linha, isto é, pelo primeiro elemento da linha, em caso de empate pelo segundo, em caso de empate pelo terceiro, etc.

EXEMPLO

Entrada	Saída

```
#include <iostream>
int main() {
    int nl, nc, val;
    std::cin >> nl >> nc;
    Matriz m = {{1,2,3},{4,5,6},{7,8,9}};
    ordena_par(m);
    for(int i=0; i<nl; i++) {
        for(int j=0; j<nc; j++)
            std::cout << m[i][j] << " ";
        std::cout << std::endl;
    }
    return 0;
}
```

```
1 2 3
7 8 9
4 5 6
```

9. (1 ponto) Faça uma função

```
std::vector<double> notas(std::string s);
```

que receba uma string **s** no seguinte formato:

nome1,nota1,nome2,nota2,...

E retorne um **vector** com as notas

EXEMPLO

Entrada	Saída

`#include <iostream>`
`int main() {`
`  std::string s = "Andre,10,Smaira,5";`
`  for(double n : notas(s))`
`  std::cout << n << std::endl;`
`  return 0;`
`}`
	10.0
	5.0

10. (1 ponto) Faça uma função

```
void ordena_versoes(std::vector<std::string>& versoes);
```

que receba um vetor de strings representando identificadores de versões de um software e as ordene em ordem crescente. Basicamente um identificador de versões é um conjunto de números inteiros separados por pontos. A ordenação deve ser feita em ordem crescente do valor do primeiro número, em caso de empate, repete-se a emsma comparação apra o segundo e assim por diante. Caso um identificador tenha menos números que outro e tenha dado empate até chegar o fim do menor, esse menor vem antes do outro na ordenação.

EXEMPLO

Entrada	Saída

`int main() {`
`  std::vector<std::string> vs = {"1.2.3", "1.2", "1.2.0", "3.0", "0"};`
`  ordena_versoes(vs);`
`  for(auto v : vs)`
`  std::cout << v << std::endl;`
`  return 0;`
`}`
	0
	1.2
	1.2.0
	1.2.3
	3.0

11. (0.5 pontos) Faça uma classe **Pessoa** cujo construtor receba uma **std::string** no seguinte formato:

«nome» tem **«idade»** anos e mora em **«cidade»**-**«UF»**, **«pais»**

e ao usarmos o **std::cout** para impressão, imprima um dado por linha, conforme o exemplo, **sem usar estruturas de repetição** e **sem usar estruturas condicionais**.

EXEMPLO

Entrada

```
int main() {
    std::cout << Pessoa("André Smaira tem 34 anos e mora em São Carlos-SP, Brasil");
    return 0;
}
```

Saída

```
André Smaira
34
São Carlos
SP
Brasil
```