a) Ordenação simples

Esse é um problema bem simples! Ordene um conjunto de números lidos usando o Algoritmo de Ordenação por Inserção ou por Seleção ou por Bolha. Não use funções prontas: desenvolva a sua.

Entrada
A entrada possui um único caso de teste com uma quantidade arbitrária de números, a entrada termina quando o arquivo terminar (EOF). Os números cabem em um inteiro de 32 bits. Sabemos que cada caso de teste não possui mais que 1000 elementos.

Saída
Imprima os mesmos números ordenados de forma não decrescente. Os números devem ser separados por espaço e não deve sobrar espaço após o último número que deve ter uma quebra de linha.

Exemplo
Exemplo de Entrada
7 3 2 5 4 3
Saída para o exemplo de entrada acima
2 3 3 4 5 7

```c
#include <stdio.h>
#include <stdlib.h>

void sel (int *v, int tam) {

    int i, j;
    int min, x;


    for (i = 0; i < tam - 1; ++i)
    {
        min = i;

        for (j = i+1; j < tam; ++j)
        {
            if (v[j] < v[min]) min = j;
        }

        x = v[min];
        v[min] = v[i];
        v[i] = x;
    }

}

void printSel (int *v, int tam) {

    for (int i = 0; i < tam; i++)  printf("%d ", v[i]);
    printf("\n");
}

int main (void) {

    int v[1000], tam = 0, num;

    while (scanf("%d", &num) != EOF) {
        v[tam] = num;
        tam++;
    }

    sel(v, tam);
    printSel(v, tam);

    return 0;
}
```

b) Ordenação sem laço

Ordenação Esse é um problema bem simples, mas vou te pedir uma solução desafiadora! Ordene um conjunto de n números armazenados num vetor v usando o algoritmo de ordenação por inserção ou por seleção. Mas você deve submeter apenas sua função de ordenação com o seguinte protótipo:

void ordena (int *v, int n);

Não use funções prontas: desenvolva a sua. Ah! Sua função não pode usar laços.

```c
#include <stdio.h>
#include <stdlib.h>

void ordena (int *v, int n) {

   if (n <= 1) return "";

   ordena(v, n - 1);

   int ultimo = v[n-1];
   int j = n - 2;

   while (j >= 0 && v[j] > ultimo){
      v[j+1] = v[j];
      j--;
   }
   v[j+1] = ultimo;

}
```

c)Busca Binaria

O algoritmo de busca binária é um algoritmo clássico que busca que é usado como uma alternativa rápida para fazer buscas num conjunto de dados. Para que o algoritmo funcione, o conjunto deve estar ordenado.
Para fixar ideias, sua tarefa, nesse exercício, é ler um conjunto de N números inteiros e, em seguida, ler M números inteiros que devem ser buscados no conjunto de dados.
Dado um inteiro x, seu algoritmo deve retornar um índice j tal que $v[j-1] < x \leqslant v[j]$.

Entrada
A entrada é composta M + N + 1 linhas. A primeira linha contém o valor de N e M, respectivamente ($1 \leqslant N, M \leqslant 109$). As N linhas seguintes contém números inteiros (que cabem num inteiro de 32 bits) que compõem o conjunto de dados de interesse de busca.
Os N números são dados em ordem não decrescente. As M linhas seguintes contêm os inteiros que devem ser procurados no conjunto de dados.

Saída
Para cada inteiro x dado, você deve imprimir j tal que: **$v[j-1] < x \leqslant v[j]$**.

Exemplo Entrada
5 4
1
3
4
7
9
0
15
3
5

Saída
0
5
1
3

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char const *argv[])
{
    int n, m, i;
    scanf("%d %d", &n, &m);
    int *N = malloc(n * sizeof(int)), *M = malloc(m * sizeof(int));
    for (i = 0; i < n; i++)
    {
        scanf("%d", &N[i]);
    }
    for (i = 0; i < m; i++)
    {
        scanf("%d", &M[i]);
    }

    int d, e, meio;

    for (i = 0; i < m; i++)
    {
        d = n - 1;
        e = 0;

        if (M[i] > N[n - 1])
        {
            printf("%d\n", n);
        }
        else
        {
            if (M[i] < N[0])
            {
                printf("0\n");
            }
            else
            {
                do
                {

                    meio = (e + d) / 2;
                    if (N[meio] < M[i] && M[i] <= N[meio + 1])
                    {
                        printf("%d\n", meio + 1);
                        break;
                    }
                    else
                    {
                        if (N[meio] < M[i])
                        {
                            e = meio + 1;
                        }
                        else
                        {
                            d = meio - 1;
                        }
                    }
                } while (e <= d || e == d);
            }
        }
    }

    return 0;
}
```

d) Busca geral num conjunto não ordenado

Sua tarefa, neste exercício, é ler um conjunto de N número inteiros e depois verificar se M elementos pertencem ou não ao conjunto. Se pertencerem, você deve imprimir a posição que ocupam. Se não, você deve imprimir -1.
M é um valor muito grande, por isso, você deve elaborar um algoritmo eficiente para efetuar as buscas!

Entrada
A entrada é composta M + N + 1 linhas. A primeira linha contém o valor de N e M, respectivamente ($1 \leqslant N, M \leqslant 10^9$).
As N linhas seguintes contém números inteiros (que cabem num inteiro de 32 bits) que compõem o conjunto de dados de interesse de busca.
As M linhas seguintes contêm os inteiros que devem ser procurados no conjunto de dados.

Saída
Para cada inteiro x dado, você deve imprimir a posição j tal que v[j] = x, ou −1 se x não pertencer a v.
Exemplo Entrada
6 4
7
3
4
9
1
5
0
3
15
5

Saída
-1
1
-1
5

```c
#include <stdio.h>
#include <stdlib.h>

int andar(int *casas, int *encomendas, int tam_casas, int tam_encomendas){
        int encomenda;
    int achou;
    int pos =0;
    int nova_pos = 0;
        int passos = 0;
        int diferenca_casas;

        int *vet = malloc(tam_encomendas*sizeof(int));
        for(int i=0;i<tam_encomendas;i++){
                for(int j=0;j<tam_casas;j++){
                        if(casas[j]==encomendas[i]){
                                nova_pos=j;
                                diferenca_casas=nova_pos - pos;
                                passos+=abs(diferenca_casas);
                                pos=j;
                        }
                }
        }
        return passos;
}


int main(void) {
  int N_casas, N_encomendas, passos=0;
        int *casas, *entregas;
        scanf("%d %d", &N_casas, &N_encomendas);

        casas = malloc(N_casas*sizeof(int));
        entregas = malloc(N_encomendas*sizeof(int));

        //Inserindo as casas
        for(int i=0;i<N_casas;i++){
                scanf("%d", &casas[i]);
        }

        //Formando as encomendas
        for(int i=0;i<N_encomendas;i++){
                scanf("%d", &entregas[i]);
        }
        passos = andar(casas, entregas, N_casas, N_encomendas);
        printf("%d\n", passos);

        return 0;
}
```

f) Tiro ao alvo

Recentemente Juquinha ganhou de aniversário um joguinho bem clássico: Tiro ao Alvo. Ele arrumou um ótimo lugar em seu quarto para se divertir com o jogo, porém após ler todas as regras do jogo ele percebeu que precisa da sua ajuda para calcular a pontuação obtida.
Segundo as regras, o alvo do jogo é composto por C círculos, todos centrados na origem (0, 0). Juquinha atira T vezes e após cada tiro informa suas coordenadas. A pontuação de cada tiro é feita da seguinte forma: para cada círculo em que o tiro estiver contido Juquinha recebe um ponto.
Considere por exemplo a figura abaixo. O tiro marcado com a letra A recebe zero pontos, pois não está contido por nenhum círculo. O tiro marcado com a letra B recebe um ponto, pois está contido por um círculo (o mais externo). O tiro marcado com a letra C recebe dois pontos, pois está contido por dois círculos (note que este caso mostra que tiros exatamente na borda de um círculo são considerados como contidos pelo círculo). Já o tiro marcado com a letra D recebe três pontos, pois está contido pelos três círculos. Considerando todos os pontos, a pontuação total de Juquinha é de 13 pontos.
Dados os raios de C círculos centrados na origem e as coordenadas dos T tiros realizados por Juquinha, escreva um programa que calcula o total de pontos que Juquinha obteve.

Entrada
A primeira linha da entrada contém dois inteiros positivos, C e T, que representam, respectivamente, o número de círculos do alvo e o número de tiros.
Cada uma das C linhas seguintes contém um inteiro positivo. O i-ésimo inteiro Ri representa o raio do i-ésimo círculo.
Os raios Ri são fornecidos em ordem crescente.
Cada uma das T linhas seguintes contém um par (X, Y) de inteiros, separados por espaço, que representam as coordenadas de cada tiro.

Saída
Seu programa deve imprimir uma única linha, contendo apenas um inteiro, o total de pontos obtidos por Juquinha.

Restrições
- $1 \leqslant C \leqslant 10^5$
,
- $1 \leqslant Ri \leqslant 10^6$
, para $1 \leqslant i \leqslant C$,
- $Ri > Ri-1$, para $1 < i \leqslant C$
- $1 \leqslant T \leqslant 10^5$
- $-10^5 \leqslant X, Y \leqslant 10^5$

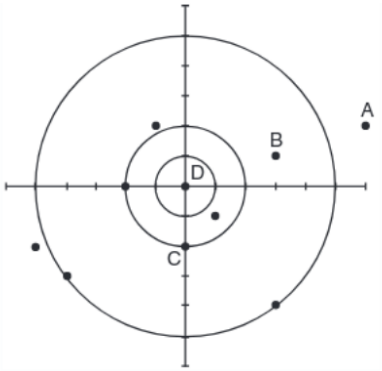
Exemplo de Entrada 1
3 10
1
2
5
0 0
-2 0
0 -2
3 -4
-4 -3
3 1
6 2
-1 2
-5 -2
1 -1

Exemplo de Saída 1
13

Exemplo de Entrada 2
3 6
1
2
5
1 0
0 3
-5 0
0 0
-3 -3
1 1

Exemplo de Saída 2
11

```c
#include <stdio.h>

enum{MAXN=100010};
enum{MAXM=100010};
int n,m;
long long int r[MAXN];

int bbin(long long int val)
{
        int inicio=1, fim=n;

        if(val>r[n]) return 0;

        while(inicio<fim)
        {
                int meio=(inicio+fim)/2;

                if(r[meio]>=val) fim=meio;
                else inicio=meio+1;
        }
        return n+1-fim;
}
int main()
{
        scanf("%d %d",&n, &m);
        for(int i=1;i<=n;i++)
        {
                scanf("%lld",&r[i]);
                r[i]=r[i]*r[i];
        }
        long long int resposta=0;
        for(int i=1;i<=m;i++)
        {
                long long int x,y;
                scanf("%lld %lld",&x,&y);
                resposta+=bbin(x*x+y*y);
        }
        printf("%lld\n",resposta);
        return 0;
}
```



e) Carteiro

Um carteiro é o responsável por entregar as encomendas na rua de Joãozinho. Por política da empresa, as encomendas devem ser entregues na mesma ordem que foram enviadas, mesmo que essa não seja a forma mais rápida. Cansado de subir
e descer aquela rua tantas vezes, nosso amigo quer mostrar à empresa quanto tempo ele leva para entregar as encomendas,
na tentativa de derrubar essa política.
A rua de Joãozinho tem N casas. Todavia, a rua de Joãozinho é meio estranha, e os números das casas não são ordenados. Como as casas possuem aproximadamente o mesmo tamanho, você pode assumir que o carteiro leva uma unidade de tempo para caminhar de uma casa até a casa imediatamente vizinha.
Há M encomendas para essa rua, que devem ser entregues na mesma ordem em que chegaram. Cada encomenda contém
o número da casa onde deve ser entregue.
Escreva um programa que determine quanto tempo o carteiro levará para entregar todas as encomendas, assumindo que quando o tempo começa a contar, ele está na primeira casa (a casa mais à esquerda da lista de casas), e o tempo termina de contar quando todas as encomendas foram entregues (mesmo que o carteiro não esteja de volta na primeira casa). Você
pode desprezar o tempo para colocar a encomenda na caixa de correio (ou seja, se ele só tiver uma encomenda, para a primeira casa, a resposta para o problema é zero).

```c
#include <stdio.h>

typedef struct num {
        int dado;
        int position_antiga;
} num;
void ordendaDados (num *v, int n){
   int i, j, min, x, aux;
   for (i=0; i<n; i++){
      min = i;
   for (j=i; j<n; j++){
         if (v[j].dado < v[min].dado)
            min = j;
      }
   x = v[min].dado; aux = v[min].position_antiga;
   v[min].dado = v[i].dado; v[min].position_antiga = v[i].position_antiga;
   v[i].dado = x; v[i].position_antiga = aux;
   }
}
int buscaBinaria(num *v, int x, int y) {
    int right = x;
    int left = -1;
    int medium;

    while( left < right-1) {
        medium = ((right + left)/2);
        if(v[medium].dado < y) {
            left = medium;
        }
        else {
            right = medium;
        }
    }
    return right;
}

int main() {
    int N, M, NUM;
    scanf("%d %d", &N, &M);

    num v[N];

    for( int i = 0; i < N; i++ ) {
        scanf("%d", &v[i].dado);
        v[i].position_antiga = i;
    }

    ordendaDados(v, N);

    int w = 0;
    int j;

    while(w < M) {
    scanf("%d", &NUM);

    j = buscaBinaria(v, N, NUM);

    if(NUM == v[j].dado){
        printf("%d\n", v[j].position_antiga);
    }
    else {
        printf("%d\n", -1);
    }
    w++;
    }
return 0;
}
```

Entrada
A primeira linha contém dois inteiros, N e M, respectivamente o número de casas e o número de encomendas. A segunda linha contém N inteiros indicando os números das casas na ordem em que aparecem na rua, não necessariamente ordenados numericamente. A terceira linha contém M inteiros indicando os números das casas onde as encomendas devem ser entregues, na ordem dada na entrada.

Saída
Seu programa deve produzir uma única linha, contendo um único inteiro, o tempo que o carteiro levará para entregar todas as encomendas na ordem correta, assumindo que ele começa na casa de menor número.

Restrições
• $1 \leqslant N, M \leqslant 45.000$,
• O número de cada casa é um
inteiro entre 1 e 1.000.000.000

Exemplo de Entrada 1
5 5
1 40 5 20 10
10 20 10 40 1

Exemplo de Saída 1
10

Exemplo de Entrada 2
3 4
100 50 80
80 80 100 50

Exemplo de Saída 2
5

Listas encadeadas - inserção

Considere uma lista encadeada com nó cabeça `le` definida por células

```
typedef struct celula {
   int dado;
   struct celula *prox;
} celula;
```

Sua tarefa nesse exercício é implementar a operação de **inserção** na lista encadeada encabeçada por `le`. Para tanto, você deve submeter um arquivo contendo apenas:

1. Os `#include` necessários para execução das instruções utilizadas no seu código.
2. A definição da `struct celula`.
3. Uma função que insere um elemento x no início da lista encadeada, cujo protótipo deve ser:

```
void insere_inicio (celula *le, int x);
```

4. Uma função que insere um elemento x imediatamente antes da primeira ocorrência de um elemento y na lista encadeada. Se y não estiver na lista encadeada, x deve ser inserido ao final. O protótipo dessa função deve ser

```
void insere_antes (celula *le, int x, int y);
```

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct celula {
    int dado;
    struct celula *prox;
} celula;

void insere_inicio (celula *le, int x) {
    celula *new = malloc(sizeof(celula));
    new->dado=x;
    new->prox=le->prox;
    le->prox=new;
}

void insere_antes (celula *le, int x, int y){
    celula *new=malloc(sizeof(celula));
    new ->dado=x;

    celula *temp;
    for(temp=le;temp!=NULL;temp=temp->prox){
        if(temp->prox==NULL){
            new->prox=temp->prox;
            temp->prox=new;
            break;
        }
        if(temp->prox->dado==y){
            new->prox=temp->prox;
            temp->prox=new;
            break;
            }
    }
    le=temp;
}
```

Listas encadeadas - remoção

Considere uma lista encadeada com nó cabeça `le` definida por células

```
typedef struct celula {
   int dado;
   struct celula *prox;
} celula;
```

Sua tarefa nesse exercício é implementar a operação de **remoção** da lista encadeada encabeçada por `le`. Para tanto, você deve submeter um arquivo contendo apenas:

1. Os `#include` necessários para execução das instruções utilizadas no seu código.
2. A definição da `struct celula`.
3. Uma função que remove o elemento imediatamente seguinte do ponteiro p, com protótipo

```
int remove_depois (celula *p);
```

Sua função deve ser capaz de lidar com o(s) caso(s) em que não seja possível remover o elemento seguinte a p.

4. Uma função que remove a primeira ocorrência de x da lista encadeada, cujo protótipo é

```
void remove_elemento (celula *le, int x);
```

4. Uma função que remove todas as ocorrências de x da lista encadeada, cujo protótipo é

```
void remove_todos_elementos (celula *le, int x);
```

```c
#include<stdio.h>
#include<stdlib.h>

typedef struct celula {
   int dado;
   struct celula *prox;
} celula;

int remove_depois (celula *p){
    if(p->prox==NULL) return 1;
    else{
        celula *lixo;
        lixo=p->prox;
        p->prox=lixo->prox;
        return 0;
    }
}


void remove_elemento (celula *le, int x){
    if (le->prox!=NULL){
        if (le->prox->dado!=x) remove_elemento(le->prox,x);
        else le->prox=le->prox->prox;
    }
}

void remove_todos_elementos (celula *le, int x){
    if (le->prox!=NULL){
        if (le->prox->dado!=x) remove_todos_elementos(le->prox,x);
        else{
        remove_todos_elementos(le->prox,x);
        le->prox=le->prox->prox;
        }
    }
}
```

Listas encadeadas - busca

Considere uma lista encadeada com nó cabeça `le` definida por células

```
typedef struct celula {
   int dado;
   struct celula *prox;
} celula;
```

Sua tarefa nesse exercício é implementar a operação de **busca** na lista encadeada encabeçada por `le`. Para tanto, você deve submeter um arquivo contendo apenas:

1. Os `#include` necessários para execução das instruções utilizadas no seu código.
2. A definição da `struct celula`.
3. Uma função que procura pela primeira ocorrência do elemento x na lista encadeada e devolve um ponteiro para a célula que o contém. O protótipo desta função deve ser:

```
celula *busca (celula *le, int x);
```

4. Uma função que faz o mesmo que o item **3**, mas *recursiva*, com protótipo

```
celula *busca_rec (celula *le, int x);
```

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct celula {
    int dado;
    struct celula *prox;
} celula;

celula *busca (celula *le, int x){
    celula *elem;
    elem=le;
    while (elem != NULL && elem->dado != x)
        elem = elem->prox;
        return elem;
}

celula *busca_rec (celula *le, int x){
    if (le == NULL)  return NULL;
        if (le->dado == x)  return le;
        return busca_rec ( le->prox,x);
    }
```

Listas encadeadas - impressão

Considere uma lista encadeada com nó cabeça `le` definida por células

```
typedef struct celula {
   int dado;
   struct celula *prox;
} celula;
```

Sua tarefa nesse exercício é implementar a operação de **impressão** da lista encadeada encabeçada por `le`. Para tanto, você deve submeter um arquivo contendo apenas:

1. Os `#include` necessários para execução das instruções utilizadas no seu código.
2. A definição da `struct celula`.
3. Duas funções (uma iterativa e outra recursiva) que imprimem a lista encadeada. Os protótipos devem ser

```
void imprime (celula *le);
void imprime_rec (celula *le);
```

Exemplos

Se a lista estiver vazia, sua função deve imprimir

```
NULL
```

Se não estiver, os elementos devem ser impressos antes do `NULL` e separados por `->`, da seguinte forma: suponha uma lista com os elementos 1, 2 e 3:

```
1 -> 2 -> 3 -> NULL
```

Atenção: Não deve haver espaço depois do `NULL`.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct celula {
    int dado;
    struct celula *prox;
} celula;

void imprime (celula *le) {
    for (celula *elem = le->prox; elem != NULL; elem = elem->prox)     printf ("%d -> ", elem->dado);
    printf ("NULL\n");
}

void imprime_rec (celula *le){
    if(le->prox==NULL)    printf ("NULL\n");
    else{
    printf("%d -> ",le->prox->dado);
    imprime_rec(le->prox);
    }
}
```

Mesclar listas encadeadas

Considere uma lista encadeada com nó cabeça `le` definida por células

```
typedef struct celula {
   int dado;
   struct celula *prox;
} celula;
```

Faça uma função

```
void mescla_listas (celula *l1, celula *l2, celula *l3);
```

que recebe duas listas encadeadas, encabeçadas por `l1` e `l2`, cujo conteúdo está ordenado em ordem não decrescente, e gere uma nova lista encabeçada por `l3` que contém os elementos de `l1` e `l2` ordenados.

Observações

1. Você não deve alocar nenhuma nova célula na sua função, apenas manipular os ponteiros dos nós de `l1` e `l2` para que estejam em `l3`.
2. Você deve considerar que o nó cabeça `l3` já foi alocado antes da chamada para a função `mescla_listas`.
3. As listas encabeçadas por `l1` e `l2` não precisam estar intactas após a chamada à sua função.

Exemplos

Suponha, por exemplo, que a lista `l1` seja

`l1 -> 1 -> 7 -> 9 -> 10 -> NULL`

e a lista `l2` seja

`l2 -> 2 -> 3 -> 8 -> NULL`

Sua função deve montar a lista `l3` da seguinte forma

`l3 -> 1 -> 2 -> 3 -> 7 -> 8 -> 9 -> 10 -> NULL`

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct celula {
   int dado;
   struct celula *prox;
} celula;

void mescla_listas(celula *l1, celula *l2, celula *l3)
{
    if (l2->prox == NULL && l1->prox == NULL) return;

    if (l1->prox == NULL){
        l3->prox = l2->prox;
        l2->prox = l2->prox->prox;
        l3->prox->prox = NULL;
        mescla_listas(l1, l2, l3->prox);
        return;
    }

    if (l2->prox == NULL){
        l3->prox = l1->prox;
        l1->prox = l1->prox->prox;
        l3->prox->prox = NULL;
        mescla_listas(l1, l2, l3->prox);
        return;
    }

    if (l1->prox->dado < l2->prox->dado){
        l3->prox = l1->prox;
        l1->prox = l1->prox->prox;
        l3->prox->prox = NULL;
        mescla_listas(l1, l2, l3->prox);
    }
    else{
        l3->prox = l2->prox;
        l2->prox = l2->prox->prox;
        l3->prox->prox = NULL;
        mescla_listas(l1, l2, l3->prox);
    }
}
```

Dividir listas encadeadas

Considere uma lista encadeada com nó cabeça `le` definida por células

```
typedef struct celula {
   int dado;
   struct celula *prox;
} celula;
```

Faça uma função

```
void divide_lista (celula *l, celula *l1, celula *l2);
```

que recebe uma lista encadeada encabeçada por `l` e a divide em duas listas `l1` e `l2` de forma que `l1` contenha todos os *numeros ímpares* de `l` (na ordem em que aparecem em `l`) e `l2` todos os *números pares* de `l` (na ordem em que aparecem em `l`).

Observações

1. Você não deve alocar nenhuma nova célula na sua função, apenas manipular os ponteiros dos nós de `l` para que estejam em `l1` ou `l2`.
2. Você deve considerar que os nós cabeça `l1` e `l2` já foram alocados antes da chamada para a função `divide_lista`.
3. Como consequência, a lista encabeçada por `l` não estará intacta após a chamada à sua função.

Exemplo

Suponha, por exemplo, que a lista `l` seja

`l -> 10 -> 4 -> -9 -> 2 -> 7 -> 10 -> NULL`

Sua função deve devolver as listas

`l1 -> -9 -> 7 -> NULL`

e

`l2 -> 10 -> 4 -> 2 -> 10 -> NULL`

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct celula {
    int dado;
    struct celula *prox;
} celula;

void divide_lista (celula *l, celula *l1, celula *l2){
    if(l->prox==NULL)   return;

    if(l->prox->dado%2!=0){
        l1->prox=l->prox;
        l->prox=l->prox->prox;
        l1->prox->prox=NULL;
        divide_lista(l,l1->prox,l2);
    }
    else{
        l2->prox=l->prox;
        l->prox=l->prox->prox;
        l2->prox->prox=NULL;
        divide_lista(l,l1,l2->prox);
    }
}
```