data . .
S T Q Q S S D

# Projeto Exercito

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

struct Node {
    int data;
    struct Node* next;
};

struct Lista {
    struct Node* head;
    int size;
};

void initialize (struct Lista* lista) {
    lista -> head = NULL;
    lista -> size = 0;
}

bool isEmpty (struct Lista* lista) {
    return (lista -> size == 0);
}

int getSize (struct Lista* lista) {
    struct Node* current = lista -> head;
    printf ("Elementos na lista: ");
    While (current != NULL) {
        printf ("%d", current -> data);
        current = current -> next;
    }
    printf ("\n");
}
```

Jandaia

data . .

S T Q Q S S D

```
bool consultar ( struct Lista* lista, int pos, int* value) {
    if ( pos < 0 || pos >= lista -> size )
        return false;
    struct Node* current = lista -> head;
    for ( int i = 0; i < pos; i++ ) {
        current = current -> next;
    }
    *value = current -> data ;
    return true;
}
bool inserir ( struct Lista* lista, int value, int pos ) {
    if ( pos < 0 || pos > lista -> size )
        return false;
    struct Node* new_node = ( struct Node* ) malloc ( size ( struct Node ));
    new_node -> data = value;
    if ( pos == 0 ) {
        new_node -> next = lista -> head;
        lista -> head = new_node;
    } else {
        struct Node* current = lista -> head;
        for ( int i = 0; i < pos - 1; i++ ) {
            current = current -> next;
        }

        new_node -> next = current -> next;
        current -> next = new_node;
    }
    lista -> size ++;
    return true;

}
```

data

S T Q Q S S D

```
bool alterar ( struct Lista* lista, int value, int pos) {
    if ( pos < 0 || pos >= lista -> size)
        return false;
    struct Node* current = lista -> head;
    for ( int i = 0; i < pos; i++) {
        current = current -> next;
    }
    current -> data = value;
    return true;
}

bool removerElement ( struct Lista* lista, int pos) {
    if ( pos < 0 || pos >= lista -> size)
        return false;
    struct Node* temp;
    if ( pos == 0 ) {
        temp = lista -> head;
        lista -> head = temp -> next;
    } else {
        struct Node* current = lista -> head;
        for ( int i = 0; i < pos - 1; i++) {
            current = current -> next;
        }
        temp = current -> next;
        current -> next = temp -> next;
    }
    free (temp);
    lista -> size -- ;
    return true;

}
```

data

S T Q Q S S D

```
bool salvar (struct Lista* lista) {
    FILE* file = fopen ("lista.txt", "w");
    if (file == NULL) {
        return false;
    }
    struct Node* current = lista -> head;
    while (current != NULL) {
        fprintf (file, "%d\n", current -> data);
        current = current -> next;
    }
    fclose (file);
    return true;
}

bool carregar (struct Lista* lista) {
    FILE* file = fopen ("lista.txt", "r");
    if (file == NULL) {
        return false;
    }

    inicializa (lista);

    int value;
    while (fscanf (file, "%d", &value) != EOF) {
        insert (lista, value, lista -> size);
    }

    fclose (file);
    return true;

}
```

```
void reinicializar (struct Lista* lista) {
    struct Node* current = lista -> head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current -> next;
        free (temp);
    }
    inicializa (lista);
}


int main () {
    struct Lista lista;
    inicializa (&lista);
    int opcao, elemento, posicao;
    while (1) {
        printf ("\nEscolha uma operação:\n");
        printf ("1. Verificar Tamanho da ED\n");
        printf ("2. Exibir Elementos da ED\n");
        printf ("3. consultar Elementos na ED\n");
        printf ("4. Inserir Elemento na ED\n");
        printf ("5. Alterar Elemento na ED\n");
        printf ("6. Excluir Elemento na ED\n");
        printf ("7. Salvar ED\n");
        printf ("8. Carregar ED\n");
        printf ("9. Reinicializar ED\n");
        printf ("10. Sair\n");
        printf ("Opção: ");
        scanf ("%d", &opção);

        switch (opção) {
            case 1:
                printf ("Tamanho da ED: %d\n", getSize (&lista));
                break;
```

data

Ⓢ Ⓣ Ⓠ Ⓠ Ⓢ Ⓢ Ⓓ

```
Case 2:
    display(&lista);
    break;
Case 3:
    printf("Digite a posição a ser consultada: ");
    scanf("%d", &posicao);
    if(consultar(&lista, posicao, &elemento)){
        printf("Elemento na posição %d: %d\n", posicao, elemento);
    } else {
        print("Posição inválida ou lista vazia.\n");
    }
    break;
Case 4:
    printf("Digite o elemento a ser inserido: ");
    scanf("%d, &elemento);
    printf("Digite a posição a inserção: ");
    scanf("%d", &posicao);
    if(inserir(&lista, elemento, posicao)){
        print("elemento inserido com sucesso.\n");
    } else {
        printf("Falha ao inserir elemento.\n");
    }
    break;
Case 5:
    print("Digite o novo valor: ");
    scanf("%d", &elemento);
    printf("Digite a posição a ser alterada: ");
    scanf("%d", &posicao);
    if(alterar(&lista, elemento, posicao)){
        printf("Elemento alterado com sucesso.\n");
    } else {
        printf("Falha ao alterar elemento.\n");
    }
    break
```

```
Case 6:
    printf("Digite a posição a ser excluída: ");
    scanf("%d", &posicão);
    if( removerElement(&lista, posicao)) {
        printf("Elemento excluído com sucesso.\n");
    } else {
        printf("Falha ao excluir elemento.\n");
    }
    break;
Case 7:
    if(salvar (&lista)) {
        printf("ED salva com sucesso.\n");
    } else {
        printf("Falha ao salvar ED.\n");
    }
Case 8:
    if(Carregar (&lista)) {
        print("ED carregada com sucesso.\n");
    } else {
        printf("Falha ao carregar ED.\n");
    }

  }

 }

}
```