Argumentação - Organização

Azul - Argumento de seleção
Vermelho - Argumento de sequência
Laranja - Argumento de repetição

```
CUB_tpCondRet CUB_buscaPeca(CUB_tpCubo *cubo, int cores[],int numCores){
	CUB_tpPeca *Peca;
	int i,j=2,k=3;
	int cords[12];
```

AE

```
if(cubo==NULL)
	{
		printf("Erro! Cubo inv·lido.");
		return CUB_CondRetPecaNaoExiste;
	}
```

AS
AE

```
	if(numCores == 2)
	{
		if(cores[0] == cores[1])
		{
			printf("Erro! PeÁa inexistente!.");
			return CUB_CondRetPecaNaoExiste;
		}
	}
```

AS

AE

```
	if(numCores == 3)
	{
		if(cores[0] == cores[1] || cores[1] == cores[2] || cores[0] == cores[2])
		{
			printf("Erro! Peca inexistente!.");
			return CUB_CondRetPecaNaoExiste;
		}
	}
```

AS

AE1 - Ponteiro aponta para o nó a ser excluído ou nulo caso a lista esteja vazia.

AE2 - Ponteiro que apontará para a peça da qual será obtida a cor

AE3 - Ponteiro cujo conteúdo conteúdo será atualizado com a cor da peça.

AS1 - Caso o cubo tenha sido preenchido corretamente será criado, caso ele esteja vazio será considerado inválido

AS2 - Caso a peça exista, o conteúdo do ponteiro CorPeca é atualizado com a cor da peça apontada pelo ponteiro pPeca.

AS3 - Caso a peça exista, o conteúdo do ponteiro CorPeca é atualizado com a cor da peça apontada pelo ponteiro pPeca.

```
        Peca = CUB_criaPeca();
        pegaCoordenadas(cubo,cores,cords,numCores);
        preencheCores( Peca, cords, numCores);

AE
if(numCores==2)
        {
                j=2;
                k=3;

AE
                for(i=1; i<6; i=i+4)
                {

AE
                        if(cords[i]== 0 && cords[j] == 1 && cords[k] == 0)
                        {
                                Peca->coordPeca[0] = 1;
                                Peca->coordPeca[1] = 3;
                                Peca->coordPeca[2] = 2;
                        }

AINV
                        else if(cords[i]== 0 && cords[j] == 0 && cords[k] == 1)
                        {
                                Peca->coordPeca[0] = 2;
                                Peca->coordPeca[1] = 3;
                                Peca->coordPeca[2] = 3;
                        }
AINV
                        else if(cords[i]== 0 && cords[j] == 2 && cords[k] == 1)
                        {
                                Peca->coordPeca[0] = 2;
                                Peca->coordPeca[1] = 3;
                                Peca->coordPeca[2] = 1;
                        }
AINV
                        else if(cords[i]== 0 && cords[j] == 1 && cords[k] == 2)
                        {
                                Peca->coordPeca[0] = 3;
                                Peca->coordPeca[1] = 3;
                                Peca->coordPeca[2] = 2;
                        }
AINV
                        else if(cords[i]== 1 && cords[j] == 1 && cords[k] == 0)
                        {
                                Peca->coordPeca[0] = 1;
                                Peca->coordPeca[1] = 2;
                                Peca->coordPeca[2] = 3;
                        }
AINV
                        else if(cords[i]== 1 && cords[j] == 2 && cords[k] == 1)
                        {
                                Peca->coordPeca[0] = 1;
                                Peca->coordPeca[1] = 1;
                                Peca->coordPeca[2] = 2;
                        }
AINV
                        else if(cords[i]== 1 && cords[j] == 1 && cords[k] == 2)
                        {
                                Peca->coordPeca[0] = 1;
                                Peca->coordPeca[1] = 2;
                                Peca->coordPeca[2] = 1;
                        }
AINV
                        else if(cords[i]== 2 && cords[j] == 2 && cords[k] == 1)
```

```
                {
                        Peca->coordPeca[0] = 2;
                        Peca->coordPeca[1] = 1;
                        Peca->coordPeca[2] = 1;
                }
AINV
                else if(cords[i]== 2 && cords[j] == 1 && cords[k] == 2)
                {
                        Peca->coordPeca[0] = 3;
                        Peca->coordPeca[1] = 2;
                        Peca->coordPeca[2] = 1;
                }
AINV
                else if(cords[i]== 3 && cords[j] == 2 && cords[k] == 1)
                {
                        Peca->coordPeca[0] = 3;
                        Peca->coordPeca[1] = 1;
                        Peca->coordPeca[2] = 2;
                }
AINV
                else if(cords[i]== 3 && cords[j] == 1 && cords[k] == 2)
                {
                        Peca->coordPeca[0] = 3;
                        Peca->coordPeca[1] = 2;
                        Peca->coordPeca[2] = 3;
                }
AINV
                else if(cords[i]== 4 && cords[j] == 2 && cords[k] == 1)
                {
                        Peca->coordPeca[0] = 2;
                        Peca->coordPeca[1] = 1;
                        Peca->coordPeca[2] = 3;
                }
AS
                j=j+4;
                k=k+4;
        }
    }
    if(numCores==3)
    {

AE
        for(i= 1; i<10; i=i+4)
        {

AE
                if( cords[i] == 0 && cords[j] == 0 && cords[k] == 0)
                {
                        Peca->coordPeca[0] = 1;
                        Peca->coordPeca[1] = 3;
                        Peca->coordPeca[2] = 3;
                }

AINV
                else if( cords[i] == 0 && cords[j] == 2 && cords[k] == 2)
                {
                        Peca->coordPeca[0] = 3;
                        Peca->coordPeca[1] = 3;
                        Peca->coordPeca[2] = 1;
                }
AINV
                else if( cords[i] == 0 && cords[j] == 0 && cords[k] == 2)
                {
                        Peca->coordPeca[0] = 3;
                        Peca->coordPeca[1] = 3;
                        Peca->coordPeca[2] = 3;
```

```
                        }
AINV
                        else if( cords[i] == 0 && cords[j] == 2 && cords[k] == 0)
                        {
                                Peca->coordPeca[0] = 1;
                                Peca->coordPeca[1] = 3;
                                Peca->coordPeca[2] = 1;
                        }
AINV
                        else if( cords[i] == 1 && cords[j] == 2 && cords[k] == 0)
                        {
                                Peca->coordPeca[0] = 1;
                                Peca->coordPeca[1] = 1;
                                Peca->coordPeca[2] = 3;
                        }
AINV
                        else if( cords[i] == 1 && cords[j] == 2 && cords[k] == 2)
                        {
                                Peca->coordPeca[0] = 1;
                                Peca->coordPeca[1] = 1;
                                Peca->coordPeca[2] = 1;
                        }
AINV
                        else if( cords[i] == 2 && cords[j] == 2 && cords[k] == 2)
                        {
                                Peca->coordPeca[0] = 3;
                                Peca->coordPeca[1] = 1;
                                Peca->coordPeca[2] = 1;
                        }
AINV
                        else if( cords[i] == 3 && cords[j] == 2 && cords[k] == 2)
                        {
                                Peca->coordPeca[0] = 3;
                                Peca->coordPeca[1] = 1;
                                Peca->coordPeca[2] = 3;
                        }
AS
                        j = j+4;
                        k = k+4;
                }
        }
        return CUB_CondRetOK;
}
```

AE1 elemento a ser pesquisado
AE2 - busca de peças de acordo com a entrada do cubo
AINV1 - 1 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.
AINV2 - 2 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV3 - 3 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV2 - 1 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.
AINV2 - 2 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV5 - 3 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV6 - 1 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV7 - 2 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV8 - 3 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV9 - 1 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.
AINV10 - 2 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV11 - 3 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV12 - 1 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.
AINV13- 2 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV14 - 3 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV15 - 1 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.
AINV16 - 2 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV17 - 3 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV18 - 1 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.
AINV19 - 2 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

AINV20 - 3 repetição e para que continue valendo deve garantir que um elemento de “a pesquisar” passe para o conjunto ja foi pesquisado e seja reposicionado.

```
CUB_tpCondRet CUB_giraFrenteEsquerda(CUB_tpCubo *cubo, int n)
{

        int aux;

AE
        if(cubo==NULL)
                return CUB_CondRetFaltouMemoria;

        aux=cubo->faces[0][2][0];
        cubo->faces[0][2][0]=cubo->faces[3][0][0];
```

```
        cubo->faces[3][0][0]=cubo->faces[4][0][2];
        cubo->faces[4][0][2]=cubo->faces[1][2][2];
        cubo->faces[1][2][2]=aux;

        aux=cubo->faces[0][2][1];
        cubo->faces[0][2][1] = cubo->faces[3][1][0];
        cubo->faces[3][1][0] = cubo->faces[4][0][1];
        cubo->faces[4][0][1] = cubo->faces[1][1][2];
        cubo->faces[1][1][2] = aux;

        aux=cubo->faces[0][2][2];
        cubo->faces[0][2][2] = cubo->faces[3][2][0];
        cubo->faces[3][2][0] = cubo->faces[4][0][0];
        cubo->faces[4][0][0] = cubo->faces[1][0][2];
        cubo->faces[1][0][2] = aux;

        aux = cubo->faces[2][0][0];
        cubo->faces[2][0][0] = cubo->faces[2][0][2];
        cubo->faces[2][0][2] = cubo->faces[2][2][2];
        cubo->faces[2][2][2] = cubo->faces[2][2][0];
        cubo->faces[2][2][0] = aux;

        aux = cubo->faces[2][0][1];
        cubo->faces[2][0][1] = cubo->faces[2][1][0];
        cubo->faces[2][1][0] = cubo->faces[2][1][2];
        cubo->faces[2][1][2] = cubo->faces[2][2][1];
        cubo->faces[2][2][1] = aux;

AE
        if(n==2)
        {
                n=n-1;
                CUB_giraFrenteEsquerda(cubo,n);
        }

AS
        return CUB_CondRetOK;
}
```

AE1

- Deve existir um ponteiro por onde será passado por referencia o movimento da face.

AE2 - busca de peças de acordo com a entrada do cubo

AS - Movimento feito indicando qual face o cubo está