Sobre Lista

Mas antes, Conceitos abstratos

Conceito	Descrição	Atributos
Carro	automóvel, meio de transporte terrestre	nro rodas, cor, chasi, nro portas
Celular	aparelho de comunicação	cor, modelo, marca, sistema operacional, cameras, memória, operadora, tamanho da tela
Lista	agrupador de elementos	Tamanho, estabelecer estado, estabelecer ordem

Definição

Lista lineares são estruturas de dados permitem o armazenamento de dados em memória de forma não contínua.

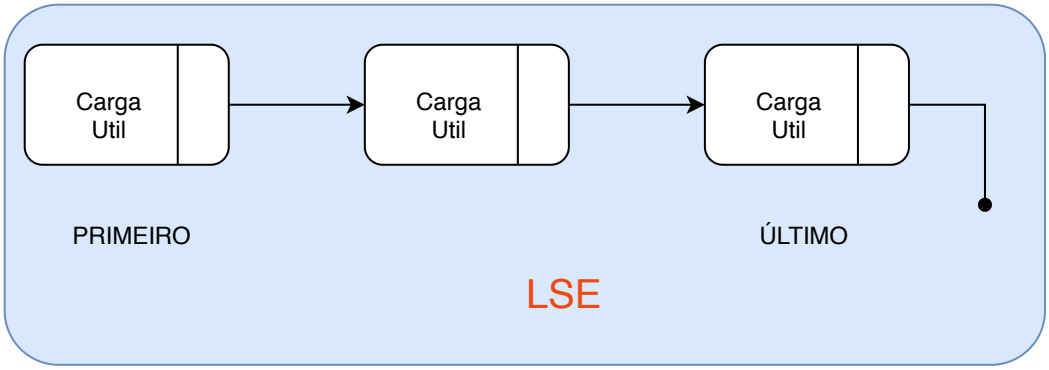
Propriedades

Uma lista L com n elementos tem as seguintes propriedades:

- Seja l_i um elemento de L, então l_{i-1} precede l_i e l_{i+1} sucede l_i;
  - Último elemento de L, l_n não tem sucessor;
  - Primeiro elemento de l, l_1 não tem predecessor.
- Para acessar o elemento l_i é preciso acessar todos os elementos l_j, $j=1,...,i-1$.

Lista Simplemente Encadeada (LSE)

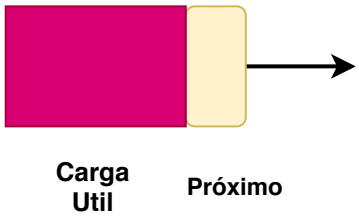
Representação gráfica Codificação



```c
typedef struct elem_lse t_elemento_lse;
typedef struct {
    // end. do primeiro elemento
    t_elemento_lse* primeiro;

    // end. do último elemento
    t_elemento_lse* ultimo;

    // tamanho da lista
    int tamanho;
}t_lse;
```

Elemento da Lista

Representação gráfica Codificação




```c
typedef struct elem_lse{
   // carga útil
   int cargautil;

   // endereço próximo elemento
   struct elem_lse* prox;
} t_elemento_lse;
```

Caso de uso: Playlist de música

1. As últimas músicas tocadas
    - Como a lista será organizada?
    - Onde deve ocorrer as inserções?
2. As músicas mais tocadas
    - Tem um limite?
    - Quem deve sair em uma lista com as 10+ tocadas?

Operação Posicional: Inserir no **inicio**

Permite que um novo elemento seja inserido no -inicio- da Lista L de elementos

1. Instanciar um novo elemento da lista
    - O elemento tem **carga util** e **endereço do sucessor**
2. Redefinir o status de **primeiro** elemento da lista
    - **Novo elemento** deve ser conectado à lista
    - o campo **inicio** da lista deve ser atualizado
3. Atualizar a quantidade de elementos

Operação Posicional: Inserir no **final**

Permite que um novo elemento seja inserido no -final- da Lista L de elementos

1. Instanciar um novo elemento da lista
    - O elemento tem **carga util** e **endereço do sucessor**
2. Redefinir o status de **último** elemento da lista
    - **Novo elemento** deve ser conectado à lista;
    - Identificar o elemento que possui o status de **último** da lista;
    - Atualizar seu campo **prox** com o endereço do novo elemento.
3. Atualizar a quantidade de elementos

Operação Posicional: Acessar

Permite acessar um elemento da Lista L em um determinada posição $i \in {1,...N}$.

1. Garante o acesso em ordem natural $[1,N]$
    - $i = i % N$, se $i != N$;
    - $i = N$, se $i = k*N, k \in [1,...,m]$;
    - $i = 0$ é considerado um valor inválido.
2. Garante o acesso em ordem reversa $[-1,-N]$
    - $i = (i % N) + (N+1)$, se $i != -N$
    - $i = 1$, se $i = k*N, k \in [-1,...,-m]$.

Operação Posicional: Remover do **inicio**

Permite que o elemento com status de -primeiro- da Lista L seja removido

1. Identificar o elemento e armazenar o seu endereço
    - Use informação do campo **inicio** da lista instanciada;
2. Atualizar o campo **inicio** da lista
    - Use a informação no campo **prox** do elemento identificado no passo 1;
3. Remover o elemento fisicamente da memória
    - Copie a informação no campo **cargautil** do elemento identificado no passo 1;

- Use o comando **free()**;

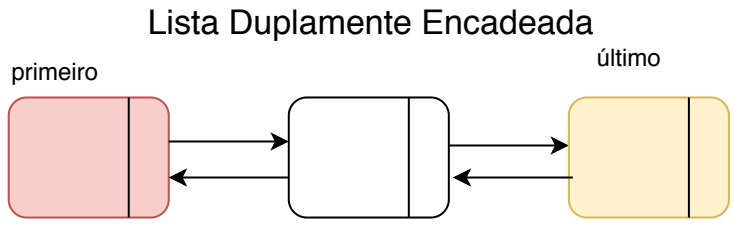
4. Retornar a informação copiada no passo anterior.

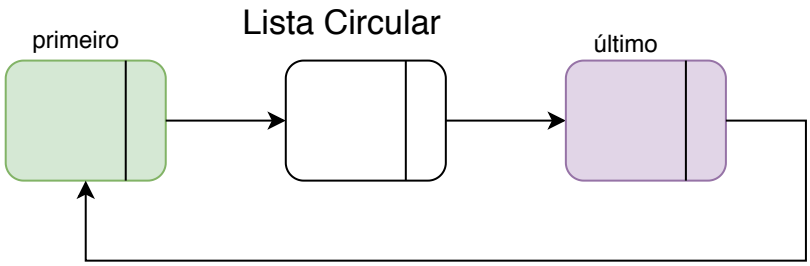
Operação Posicional: Remover do **final**

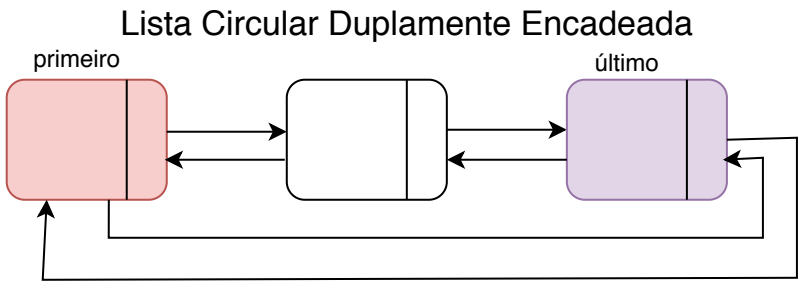
Permite que o elemento com status de -último- da Lista L seja removido

1. Identificar o elemento e armazenar o seu endereço
   - É preciso caminhar pela lista L até o último elemento e idenficar o penúltimo elemento, conjuntamente;
2. Atualizar o status de -último- elemento da lista
   - Atualize a informação no campo **prox** do penúltimo elemento identificado no passo 1;
3. Remover o elemento fisicamente da memória
   - Copie a informação no campo **cargautil** do elemento com status -último- identificado no passo 1;
   - Use o comando **free()**;
4. Retornar a informação copiada no passo anterior.

Outros tipos de Lista encadeada







Acomplamento entre a ED e Tipos

```
A lista criada até agora aceita apenas cargautil de valor inteiro
```

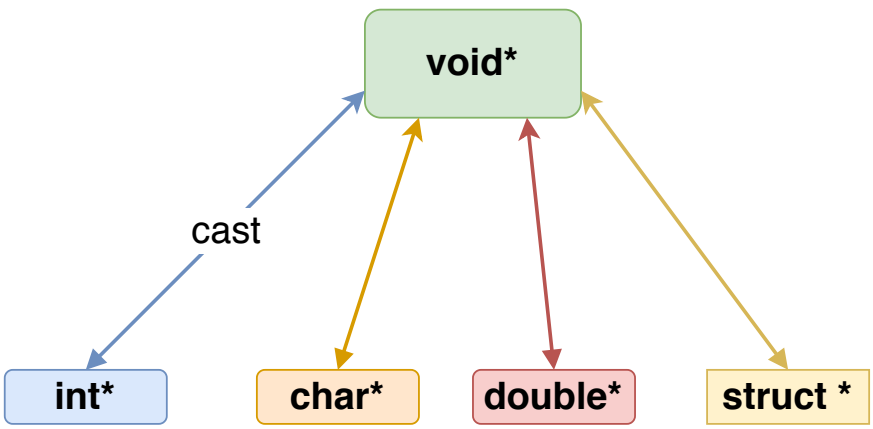
```c
typedef struct elem_lse{
    int cargautil;
    struct elem_lse* prox;// endereco prox
 } t_elemento_lse;

int acessar_lse(t_lse* lse, int pos){
void inserir_inicio_lse(t_lse* lse, int carga);
void inserir_final_lse(t_lse* lse, int carga);
int remover_inicio_lse(t_lse* lse);
int remover_final_lse(t_lse* lse);
```

Neutralização da ED: Por que?

```
Armazenamento de mais informações na lista tem implicações na programação
```

1. Alterar assinaturas das operações;
    - lista de novos itens que definem a **cargautil**
2. Alterar a lógica das operações
    - qual valor retornar entre os muito armazenados?
    - É possível que seja preciso retornar todos.

Neutralização de ED: o Tipo ponteiro void*

Definição	Representação
Permite que a variável declarada armazene endereços de memória de qualquer tipo de dado.	

Neutralização da LSE: Implementação

1. Trocar todos as referências do **int** para **void***;
2. Criar as cargas **no código** de quem está usando a Lista;
    - Especialização fica na solução (**minhasolucao.c**);
3. Qualquer solução deve passar a referência (**endereço memória**) da carga que vai na lista;
    - Implemente uma função (**criar***)para instanciar a carga;
4. Qualquer solução deve saber tratar a referência (**endereço de memória**)
    - Implemente uma função (**imprimir***) para imprimir a carga;

Neutralização da ED LSE: Casos de Uso.

Considere a criação das seguintes Listas:

1. Lista de alunos da disciplina: **matricula**(int), **notafinal**(double), **nro_faltas**(int).
2. Lista de medidas de temperatura: **coordenadaX**(double), **coordenadaY**(double), **temperatura**(double).
3. Lista de eventos: **hora**(int), **minuto**(int), **segundo**(int), **nro_evento**(int) $\in [1,5]$

Operações baseadas em conteúdo

A lista precisa ser manipulada a partir dos valores armazenados.

1. **Inserir** mantendo a ordem estabelecida pelo valor dos elementos;
    - A ordem alfabética de nome de alunos;
    - A ordem temporal (datas) de uma coleta de eventos;
2. **Remover** um elemento considerando alguma informação associada;
    - Matricula do aluno;
    - ID sensor;
3. **Buscar** um elemento considerando alguma informação associada;
    - Nome da música;
    - Matricula do aluno;

Neutralizando a lógica

1. Considere a lógica da operação **buscar**
    - Percorrer a lista;
    - Comparar a informação na **cargautil** com aquele procurada;
    - Retornar a **cargautil** sempre que ela tiver a informação buscada;
2. Como realizar as comparações?
    - o tipo void* não permite operações de acesso especializada;

- Veja o que acontece sempre que você programa usando o **void***;

Ponteiro para função: O que é.

1. Defina uma função a seguinte função:

```
void __trocar(int *nro1, int* nro2){
    int aux;
    aux = *nro1;
    *nro1 = *nro2;
    *nro2 = aux;
    printf("trocar 1\n");
}
```

2. Agora defina uma função **main()** com o seguinte código:

```
int main(){
    int maior=10, menor=20;
    printf("%p",__trocar);
    __trocar(&maior,&menor);
}
```

3. Compile e execute o código 5x e veja o que acontece.

Ponteiro para função: Como definir.

1. Escreva o seguinte código no arquivo **.c**, logo após os **#include**

```
typedef void(*da_funcao_trocar)(int*, int*);
```

2. Na função principal escreva o seguinte código:

```
// declarando uma variável
da_funcao_trocar trocador;

// atribuindo um endereço para a variável
trocador = __trocar;

printf("%p\n", __trocar);
printf("%p\n", trocador);
```

Ponteiro para função: como usar.

1. Escreva o seguinte código na função **main()**, compile, execute e anote o resultado

```
int maior=10;
int menor=20;
__trocar(&maior, &menor);
printf("%d %d\n", maior, menor);
```

2. Altere o código, incluindo

```
trocador = __trocar;

// chamada indireta da função
trocador(&maior, &menor);

printf("%d %d\n", maior, menor);
```

Ponteiro para função: como usar na parametrização.

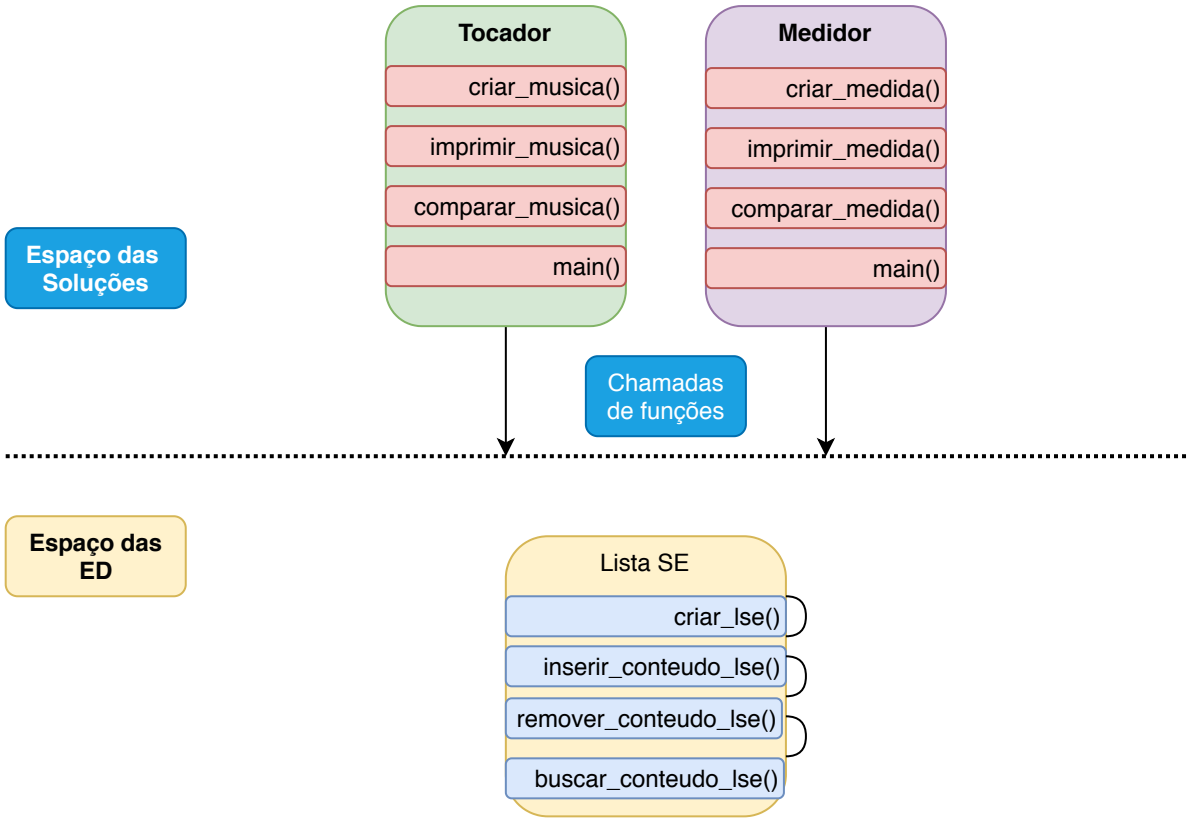
1. Inclua o seguinte código no arquivo **.c**

```
void chamadora(da_funcao_trocar trocador, int nro_1, int nro_2){
    printf("%d %d\n", nro_1, nro_2);
    // chamada indireta da função
    trocador(&nro_1, &nro_2);
    //printf("%d %d\n", nro_1, nro_2);
}
```

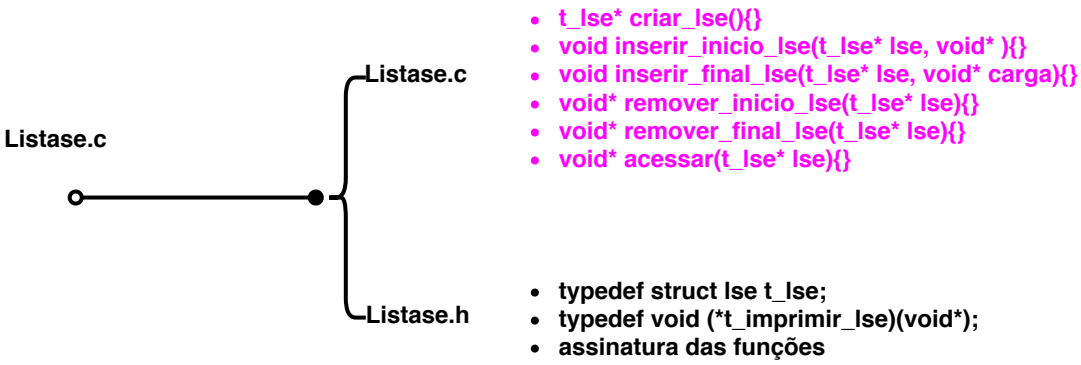
2. Inclua o seguinte código na função **main()**

```
chamadora(__trocar, maior, menor);
chamadora(__trocar2, maior, menor);
```

e na LSE? Visão geral



E na LSE? vamos refatorar.



Refatorar LSE

1. Crie o novo arquivo **listase.h**
    - inclua a assinatura das funções;
    - remova os itens anteriores do arquivo **listase.c**
2. ainda no arquivo **listase.h**, inclua o seguinte código, logo após os **#include**;

```
typedef void(*t_imprimir_lse)(void*);
typedef struct lse t_lse;
```

Vamos refatorar **listase.c**

1. no arquivo **listalse.c**, redefina a LSE incluindo

```
struct lse{
    t_elemento_lse* inicio;
    int tamanho;
    t_imprimir_lse impressora;
};
```

2. redefina a assinatura da função **criar_lse()**

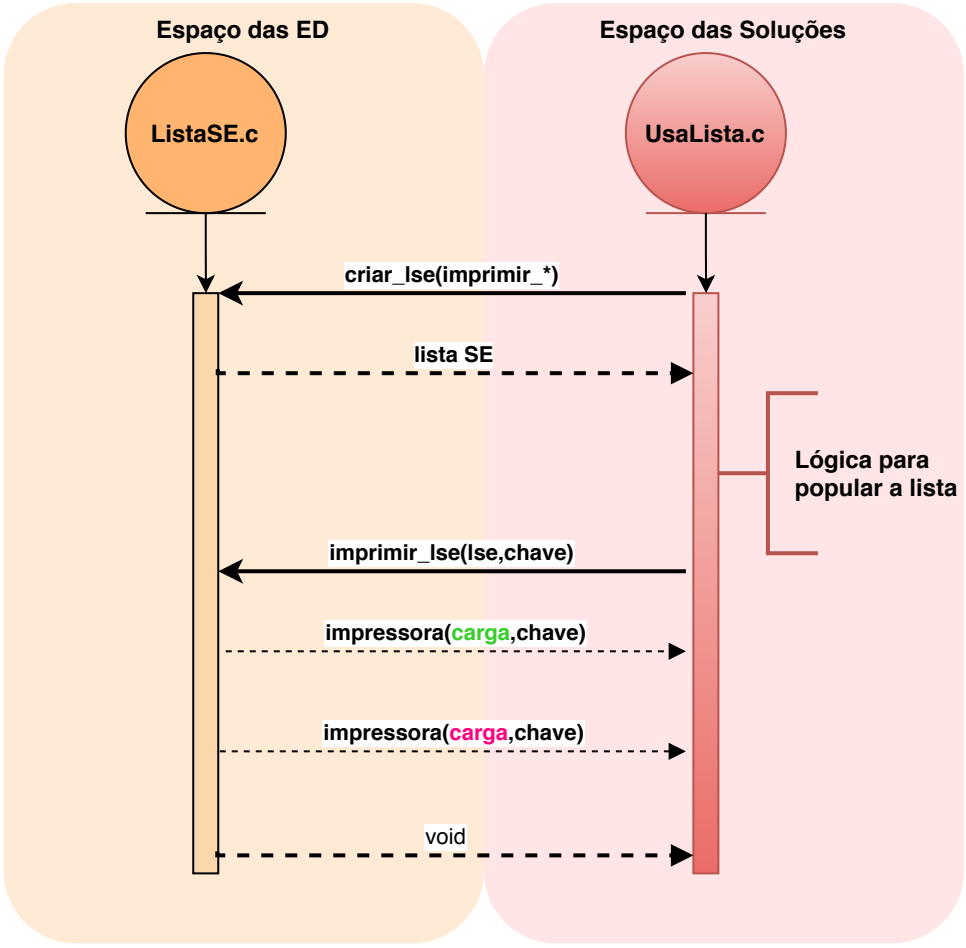
```
    t_lse* criar_lse(t_imprimir_lse imprimir){
        t_lse* nova  = malloc(sizeof(t_lse));
        nova->inicio = NULL;
        nova->impressora = imprimir;
        nova->tamanho=0;

        return nova;
    }
```

Último ajuste, no listase.c

1. Inclua o código que percorre a lista imprimindo os elementos.

```
void imprimir_lse(t_lse *lse){
    t_elemento_lse* cam = lse->inicio;
    while(cam!=NULL){
        lse->impressora(cam->cargautil);
        cam = cam->prox;
    }
}
```

A sequência de chamadas na impressão



Inserir, Remover e Buscar baseado em conteúdo

1. Reconhecer o acomplamento da lógica ao tipo de dados;
2. Criar novas funções neutralizando as lógicas;
3. Criar um novo tipo Ponteiro para função;
4. Reimplementar a função que cria a lista;

Alteração nas declarações (listase.h)

```
typedef int (*t_comparar_lse)(void* carga_na_lista, void* nova_carga);

t_lse* criar_lse(t_imprimir_lse imprimir, t_comparar_lse comparar);
```

Alteração no tipo LSE (listase.c)

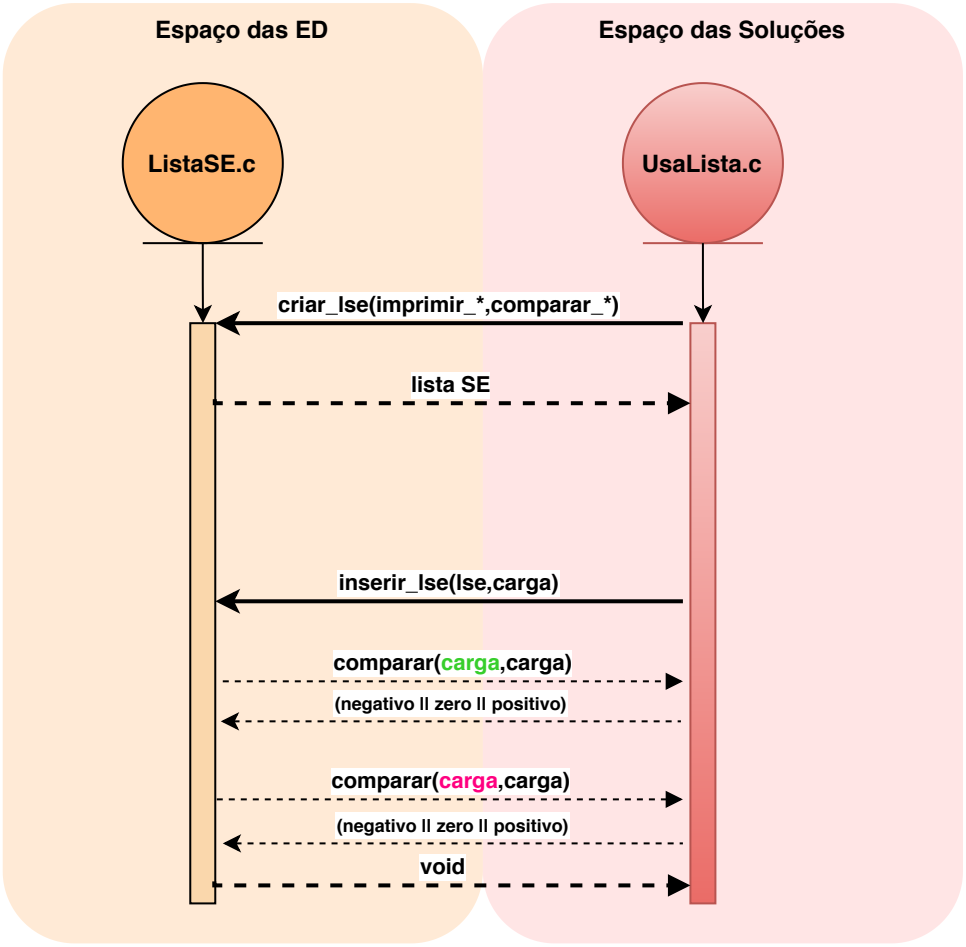
```
struct lse{
    t_elemento_lse* inicio; // Primeiro elemento
    int tamanho;
    t_imprimir_lse impressora;
    t_comparar_lse comparar;
};
```

```
t_lse* criar_lse(t_imprimir_lse impressora, t_comparar_lse comparar){
    t_lse* nova  = malloc(sizeof(t_lse));
    nova->inicio = NULL;
    nova->tamanho=0;
    nova->impressora = impressora;
    nova->comparar = comparar;

    return nova;
}
```

Inserir um novo elemento

1. Encontrar a posição da inserção
2. Caracterizar a posição
    - inicio
    - fim
    - meio
3. Conectar o novo elemento

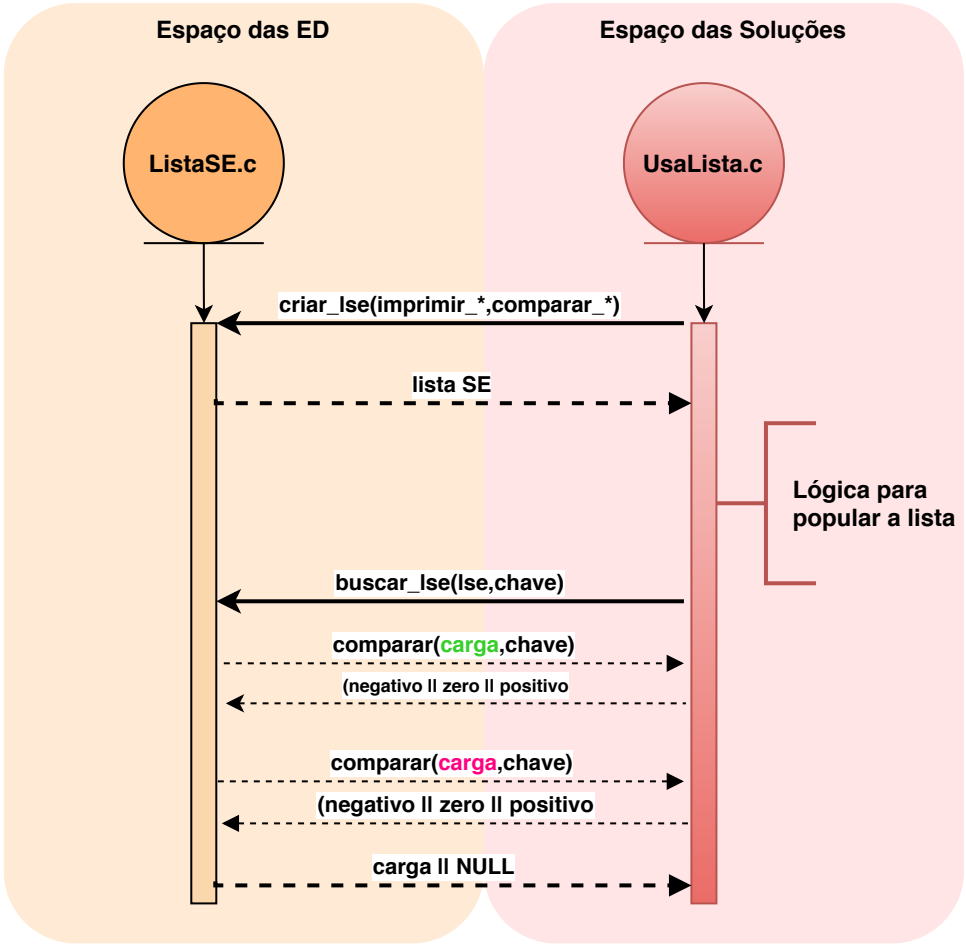
4. Atualizar a lista se for o caso



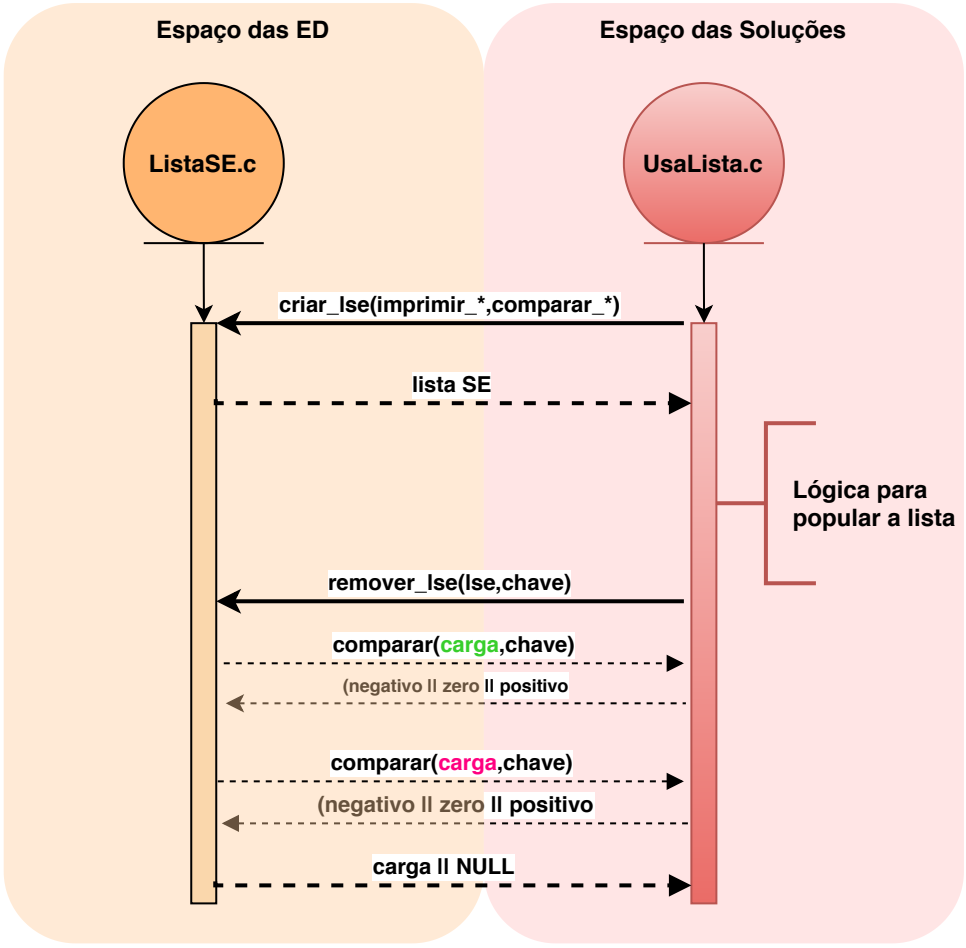
Buscar um novo elemento

1. Percorrer a lista;
2. Dois resultados são possíveis
    - Pertence à lista

- Não pertence à lista



Remover um elemento

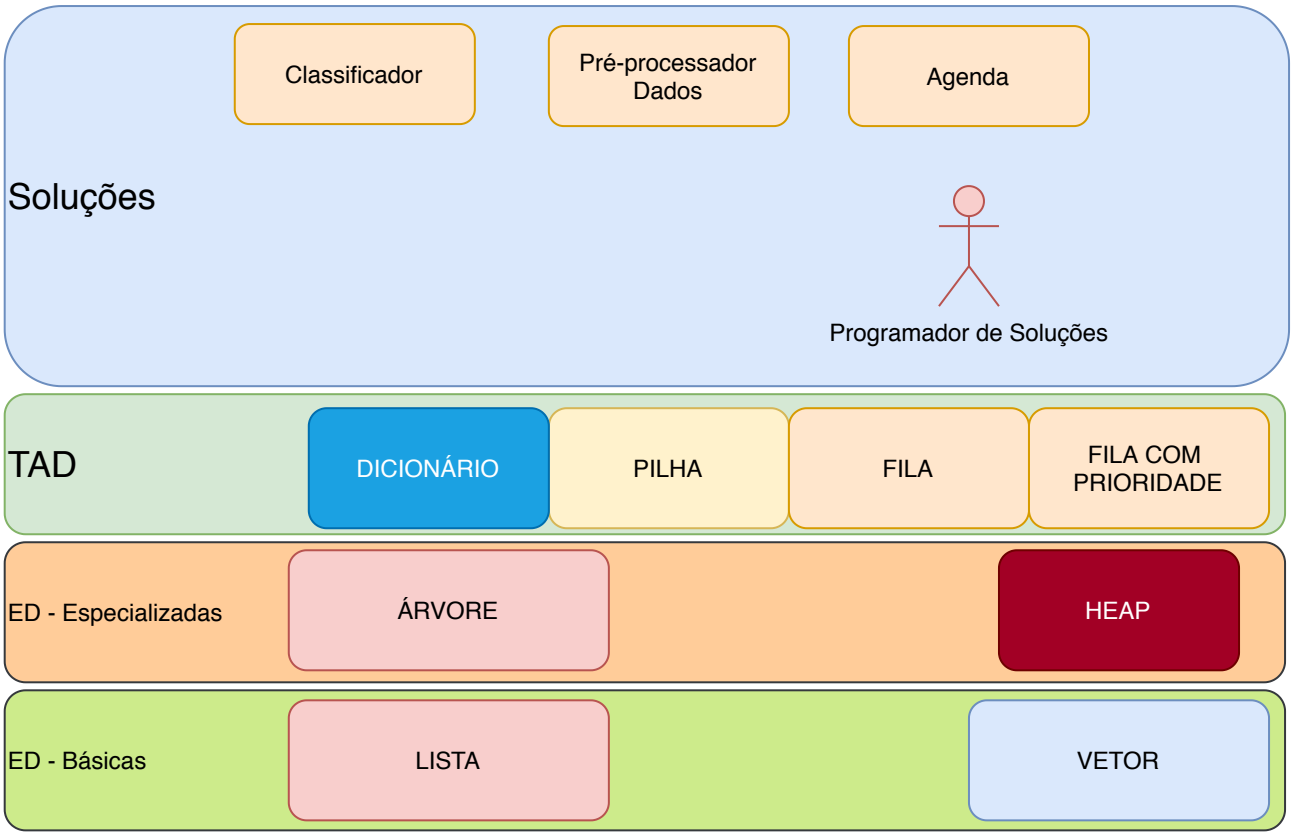


1. Buscar um elemento
2. Caracterizar a sua posição
    - inicio da lista
    - demais elementos
3. Exceções
    - Remover o único elemento da lista;
    - Remover em uma lista vazia

Atividade: criando e manipulando playlists

1. Apresente e implemente o conceito de um música;
    - use o **typedef** para criar o tipo que representará o conceito;
2. Crie um arquivo chamado "playlist.c"
    - Defina o conceito de um playlist +10;
    - Defina o conceito de uma playlist aleatória;
    - Defina o conceito de uma playlist as últimas *k tocadas;

Fechamento



Tipos Abstratos de Dados

---

1. Definição
2. Como são implementados
3. Exemplos de uso
4.
