AED1 - Aula 14
Pilhas em listas encadeadas
(com e sem nó cabeça)

Já estudamos o funcionamento do tipo abstrato de dados “pilha”,

- vimos como implementá-las usando vetores
- e as utilizamos para resolver problemas.

Um tipo abstrato de dados

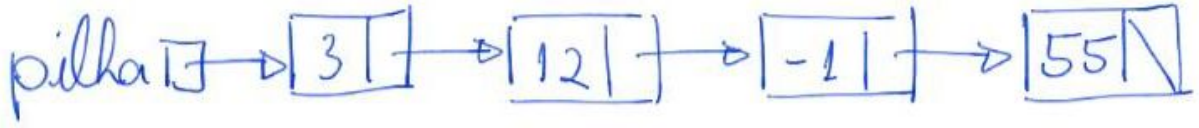
- é um modelo matemático para tipos de dados
    - definido em termos de seu comportamento
        - pelo ponto de vista do usuário,
    - e não de sua implementação.

Hoje, veremos como implementar o tipo abstrato de dados “pilha”

- utilizando outra estrutura de dados que já conhecemos,
    - i.e., listas ligadas (com e sem nó cabeça).
- Ao final da aula, vamos comparar as diferentes implementações,
    - para analisar suas vantagens e limitações.

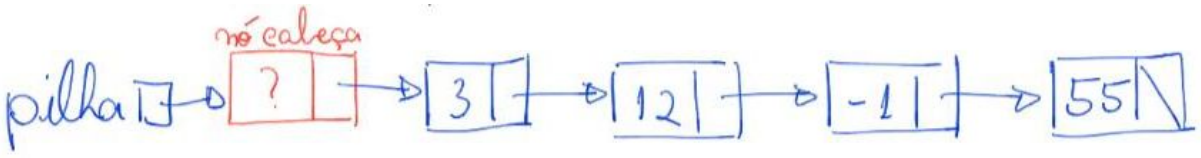
Antes de começar a implementação,

- uma importante decisão de projeto deve ser tomada.
- Considere a seguinte lista encadeada,
    - que representa uma pilha.

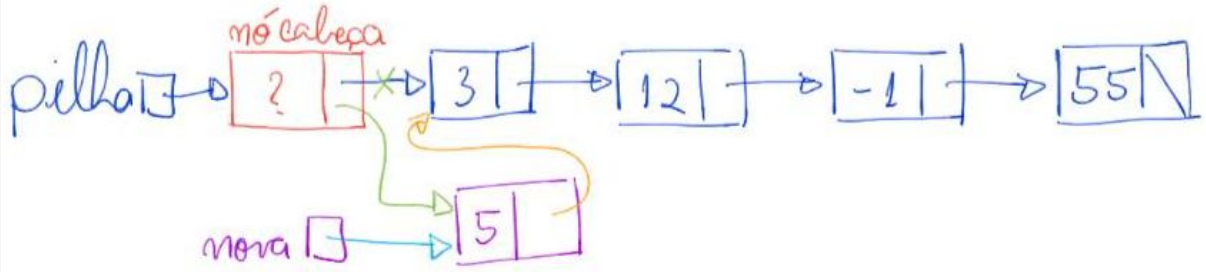


- Se desejamos empilhar o elemento 5,
    - onde ele deve ser inserido?
        - No início da fila (logo após p),
        - ou no final desta (logo após 55)?
- Considere a eficiência da operação empilhar
    - e também da operação desempilhar,
        - de acordo com sua escolha.

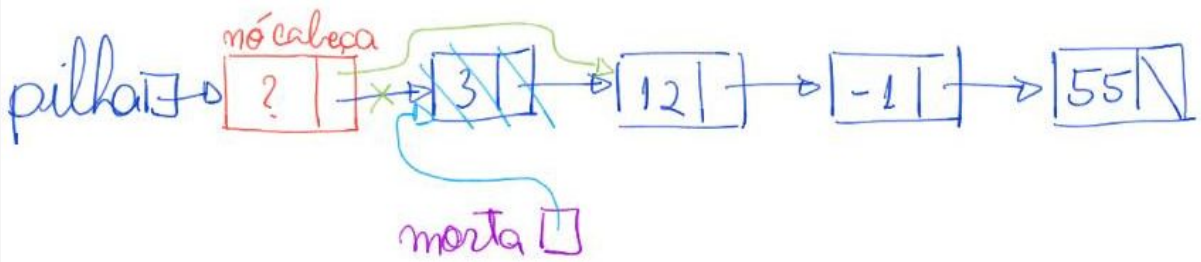
Funções para manipulação de pilha implementada em lista com nó cabeça



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

typedef struct celula
{
    char conteudo;
    struct celula *prox;
} Celula;

Celula *criaPilhaCNC()
{
    Celula *pilha;
    pilha = (Celula *)malloc(sizeof(Celula));
    pilha->prox = NULL;
    return pilha;
}
```




```c
void empilhaCNC(Celula *pilha, char x)
{
    Celula *nova;
    nova = malloc(sizeof(Celula));
    nova->conteudo = x;
    nova->prox = pilha->prox;
    pilha->prox = nova;
}
```




```c
// supõe que a pilha não está vazia
char desempilhaCNC(Celula *pilha)
{
    char valor;
    Celula *morta;
    morta = pilha->prox;
    valor = morta->conteudo;
    pilha->prox = morta->prox;
    free(morta);
    morta = NULL;
    return valor;
}

// supõe que a pilha não está vazia
char consultaTopoCNC(Celula *pilha)
{
    return pilha->prox->conteudo;
}

// devolve 1 se a pilha está vazia e 0 caso contrário
int pilhaVaziaCNC(Celula *pilha)
{
    return pilha->prox == NULL;
}

void imprimePilhaCNC(Celula *pilha)
{
    Celula *p;
    p = pilha->prox; // pula o nó cabeça
    while (p != NULL)
```

```
    {
        printf("%c ", p->conteudo);
        p = p->prox;
    }
    printf("\n");
}

int tamPilhaCNC(Celula *pilha)
{
    Celula *p;
    int tam = 0;
    p = pilha->prox; // pula o nó cabeça
    while (p != NULL)
    {
        tam++;
        p = p->prox;
    }
    return tam;
}

Celula *liberaPilhaCNC(Celula *pilha)
{
    Celula *p, *morta;
    p = pilha;
    while (p != NULL)
    {
        morta = p;
        p = p->prox;
        free(morta);
    }
    return NULL;
}

// Esta função devolve 1 se a string ASCII s
// contém uma sequência bem-formada de
```

```
// parênteses e colchetes e devolve 0 se
// a sequência é malformada.
int bemFormada(char str[])
{
    Celula *pilha;
    pilha = criaPilhaCNC();
    int sol;
    for (int i = 0; str[i] != '\0'; ++i)
    {
        char c;
        switch (str[i])
        {
        case ')':
            if (pilhaVaziaCNC(pilha))
                return 0;
            c = desempilhaCNC(pilha);
            if (c != '(')
                return 0;
            break;
        case ']':
            if (pilhaVaziaCNC(pilha))
                return 0;
            c = desempilhaCNC(pilha);
            if (c != '[')
                return 0;
            break;
        default:
            empilhaCNC(pilha, str[i]);
        }
    }
    sol = pilhaVaziaCNC(pilha);
    pilha = liberaPilhaCNC(pilha);
    return sol;
}
```

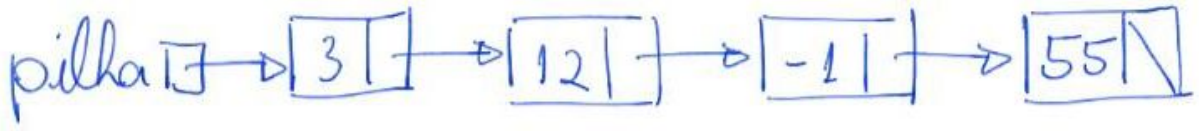
- Note que a pilha deveria ser liberada antes de cada return.

```c
int main(int argc, char *argv[])
{
    char *str;
    Celula *pilha;
    char aux;

    if (argc != 2)
    {
        printf("Numero incorreto de parametros! Ex.: .\\pilhaCNC
\"(()[()])\"");
        return 0;
    }
    str = argv[1];

    /* inicializa a pilha */
    pilha = criaPilhaCNC();

    /* empilha abc */
    empilhaCNC(pilha, 'a');
    empilhaCNC(pilha, 'b');
    empilhaCNC(pilha, 'c');

    /* imprime pilha */
    imprimePilhaCNC(pilha);

    /* desempilha e armazena em x */
    aux = desempilhaCNC(pilha);
    printf("%c\n", aux);

    /* consulta topo da pilha */
    printf("%c\n", consultaTopoCNC(pilha));

    /* imprime pilha */
    imprimePilhaCNC(pilha);
```

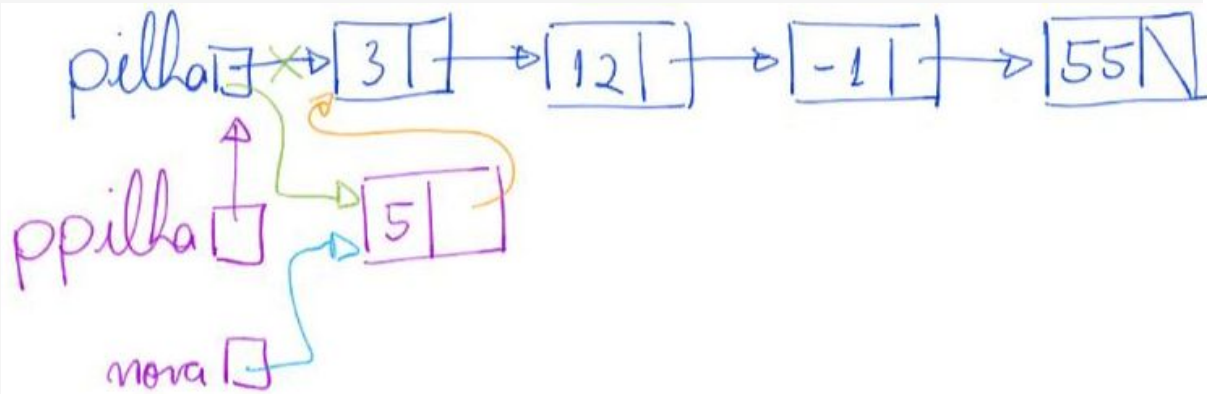
```c
    /* tamanho da lista */
    printf("%d\n", tamPilhaCNC(pilha));

    /* libera a pilha */
    pilha = liberaPilhaCNC(pilha);

    printf("%s eh bem formada? %d\n", str, bemFormada(str));

    return 0;
}
```

- Qual a eficiência de tempo e espaço de cada função?

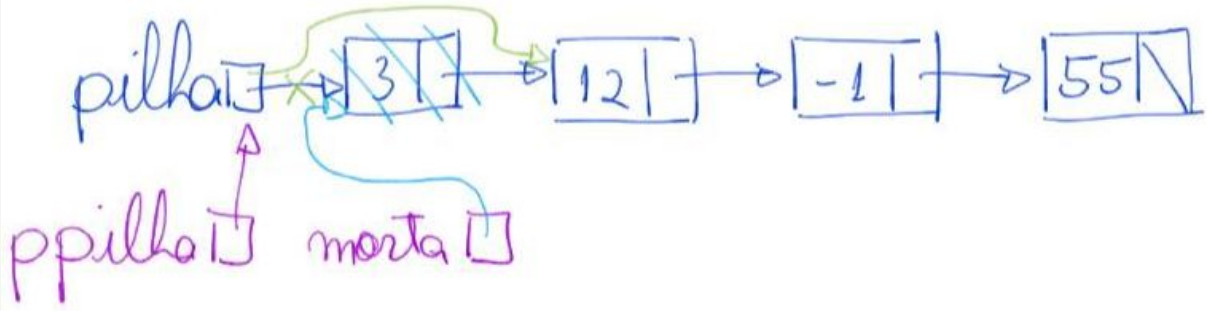
Funções para manipulação de pilha implementada em lista sem nó cabeça




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

typedef struct celula
{
    char conteudo;
    struct celula *prox;
} Celula;

Celula *criaPilhaSNC()
{
    return NULL;
}
```




```c
void empilhaSNC(Celula **ppilha, char valor)
{
    Celula *nova;
    nova = malloc(sizeof(Celula));
    nova->conteudo = valor;
    nova->prox = *ppilha;
    *ppilha = nova;
}
```




```c
// supõe que a pilha não está vazia
char desempilhaSNC(Celula **ppilha)
{
    char valor;
    Celula *morta;
    morta = *ppilha;
    valor = morta->conteudo;
    *ppilha = morta->prox;
    free(morta);
    morta = NULL;
    return valor;
}
```

```c
// supõe que a pilha não está vazia
char consultaTopoSNC(Celula *pilha)
{
    return pilha->conteudo;
}

// devolve 1 se a pilha está vazia e 0 caso contrário
int pilhaVaziaSNC(Celula *pilha)
{
    return pilha == NULL;
}

void imprimePilhaSNC(Celula *pilha)
{
    Celula *p;
    p = pilha;
    while (p != NULL)
    {
        printf("%c ", p->conteudo);
        p = p->prox;
    }
    printf("\n");
}

int tamPilhaSNC(Celula *pilha)
{
    Celula *p;
    int tam = 0;
    p = pilha;
    while (p != NULL)
    {
        tam++;
        p = p->prox;
    }
```

```
    return tam;
}

Celula *liberaPilhaSNC(Celula *pilha)
{
    Celula *p, *morta;
    p = pilha;
    while (p != NULL)
    {
        morta = p;
        p = p->prox;
        free(morta);
    }
    return NULL;
}

// Esta função devolve 1 se a string ASCII s
// contém uma sequência bem-formada de
// parênteses e colchetes e devolve 0 se
// a sequência é malformada.
int bemFormada(char str[])
{
    Celula *pilha;
    pilha = criaPilhaSNC();
    int sol;
    for (int i = 0; str[i] != '\0'; ++i)
    {
        char c;
        switch (str[i])
        {
        case ')':
            if (pilhaVaziaSNC(pilha))
                return 0;
            c = desempilhaSNC(&pilha);
            if (c != '(')
```

```
                return 0;
            break;
        case ']':
            if (pilhaVaziaSNC(pilha))
                return 0;
            c = desempilhaSNC(&pilha);
            if (c != '[')
                return 0;
            break;
        default:
            empilhaSNC(&pilha, str[i]);
        }
    }
    sol = pilhaVaziaSNC(pilha);
    pilha = liberaPilhaSNC(pilha);
    return sol;
}
```

- Note que a pilha deveria ser liberada antes de cada return.

```
int main(int argc, char *argv[])
{
    char *str;
    Celula *pilha;
    char aux;

    if (argc != 2)
    {
        printf("Numero incorreto de parametros! Ex.: .\\pilhaSNC
\"(()[()])\"");
        return 0;
    }
    str = argv[1];

    /* inicializa a pilha */
    pilha = criaPilhaSNC();
```

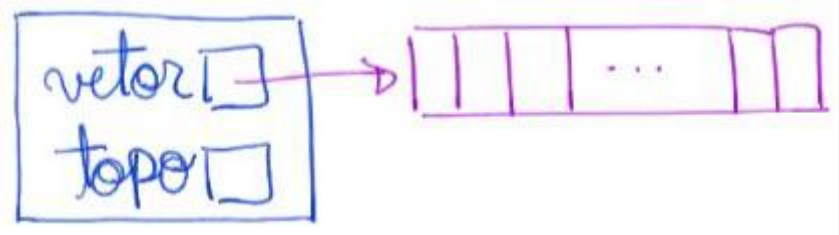
```
    /* empilha abc */
    empilhaSNC(&pilha, 'a');
    empilhaSNC(&pilha, 'b');
    empilhaSNC(&pilha, 'c');

    /* imprime pilha */
    imprimePilhaSNC(pilha);

    /* desempilha e armazena em x */
    aux = desempilhaSNC(&pilha);
    printf("%c\n", aux);

    /* consulta topo da pilha */
    printf("%c\n", consultaTopoSNC(pilha));

    /* imprime pilha */
    imprimePilhaSNC(pilha);

    /* tamanho da lista */
    printf("%d\n", tamPilhaSNC(pilha));

    /* libera a pilha */
    pilha = liberaPilhaSNC(pilha);

    printf("%s eh bem formada? %d\n", str, bemFormada(str));

    return 0;
}
```

- Qual a eficiência de tempo e espaço de cada função?

Funções para manipulação de pilha implementada em vetor

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

```c
#define N 100
```




```c
typedef struct pilha
{
    char *vetor;
    int topo;
} Pilha;

Pilha *criaPilhaVetor(int size)
{
    Pilha *s;
    s = (Pilha *)malloc(sizeof(Pilha));
    s->vetor = (char *)malloc(size * sizeof(char));
    s->topo = 0;
    return s;
}

// supõe que a pilha não está cheia
void empilhaVetor(Pilha *s, char x)
{
    s->vetor[s->topo] = x;
    (s->topo)++;
}

// supõe que a pilha não está vazia
char desempilhaVetor(Pilha *s)
{
    (s->topo)--;
    return s->vetor[s->topo];
```

```c
}

// supõe que a pilha não está vazia
char consultaTopoVetor(Pilha *s)
{
    return s->vetor[(s->topo) - 1];
}

// devolve 1 se a pilha está vazia e 0 caso contrário
int pilhaVaziaVetor(Pilha *s)
{
    return s->topo <= 0;
}

void imprimePilhaVetor(Pilha *s)
{
    for (int i = (s->topo) - 1; i >= 0; i--)
        printf("%c ", s->vetor[i]);
    printf("\n");
}

int tamPilhaVetor(Pilha *s)
{
    return s->topo;
}

Pilha *liberaPilhaVetor(Pilha *s)
{
    free(s->vetor);
    free(s);
    return NULL;
}

// Esta função devolve 1 se a string ASCII s
// contém uma sequência bem-formada de
```

```c
// parênteses e colchetes e devolve 0 se
// a sequência é malformada.
int bemFormada(char str[])
{
    Pilha *s;
    s = criaPilhaVetor(N);
    int sol;
    for (int i = 0; str[i] != '\0'; ++i)
    {
        char c;
        switch (str[i])
        {
        case ')':
            if (pilhaVaziaVetor(s))
                return 0;
            c = desempilhaVetor(s);
            if (c != '(')
                return 0;
            break;
        case ']':
            if (pilhaVaziaVetor(s))
                return 0;
            c = desempilhaVetor(s);
            if (c != '[')
                return 0;
            break;
        default:
            empilhaVetor(s, str[i]);
        }
    }
    sol = pilhaVaziaVetor(s);
    s = liberaPilhaVetor(s);
    return sol;
}
```

- Note que a pilha deveria ser liberada antes de cada return.

```c
int main(int argc, char *argv[])
{
    char *str;
    Pilha *s;
    char x;

    if (argc != 2)
    {
        printf("Numero incorreto de parametros! Ex.: .\\pilhaVetor
\"(()[()])\"");
        return 0;
    }
    str = argv[1];

    /* inicializa a pilha */
    s = criaPilhaVetor(N);

    /* empilha abc */
    empilhaVetor(s, 'a');
    empilhaVetor(s, 'b');
    empilhaVetor(s, 'c');

    /* imprime pilha */
    imprimePilhaVetor(s);

    /* desempilha e armazena em x */
    x = desempilhaVetor(s);
    printf("%c\n", x);

    /* consulta topo da pilha */
    printf("%c\n", consultaTopoVetor(s));

    /* imprime pilha */
    imprimePilhaVetor(s);
```

```
    /* tamanho da lista */
    printf("%d\n", tamPilhaVetor(s));

    /* libera a pilha */
    s = liberaPilhaVetor(s);

    printf("%s eh bem formada? %d\n", str, bemFormada(str));

    return 0;
}
```

- Qual a eficiência de tempo e espaço de cada função?

Compare as implementações de pilha

- em vetor e em lista ligada (com e sem nó cabeça), segundo:
    - eficiência de tempo das operações,
        - tanto das principais (empilha, desempilha, consulta topo)
        - quanto das demais operações,
    - eficiência de espaço, i.e., uso de memória,
    - limitações de tamanho.
- Podemos melhorar a eficiência de alguma operação,
    - modificando/aumentando a estrutura utilizada?