AED1 - Aula 16
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 encadeadas (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 *s;
   s = (Celula *)malloc(sizeof(Celula));
   s->prox = NULL;
   return s;
}

void empilhaCNC(Celula *s, char x)
{
   Celula *nova;

   nova = malloc(sizeof(Celula));
   nova->conteudo = x;
   nova->prox = s->prox;
   s->prox = nova;
}

char desempilhaCNC(Celula *s)
{
   char x;
   Celula *morta;

   morta = s->prox;
   x = morta->conteudo;
   s->prox = morta->prox;
   free(morta);
   morta = NULL;
   return x;
}

char consultaTopoCNC(Celula *s)
{
   return s->prox->conteudo;
}

int pilhaVaziaCNC(Celula *s)
{
   return s->prox == NULL;
}

void imprimePilhaCNC(Celula *s)
{
   Celula *p;
```

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

int tamPilhaCNC(Celula *s)
{
    Celula *p;
    int tam = 0;

    p = s->prox;
    while (p != NULL)
    {
        p = p->prox;
        tam++;
    }
    return tam;
}

Celula *liberaPilhaCNC(Celula *s)
{
    Celula *p, *morta;

    p = s;
    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 *s;
    s = criaPilhaCNC();
    int sol;
    for (int i = 0; str[i] != '\0'; ++i)
```

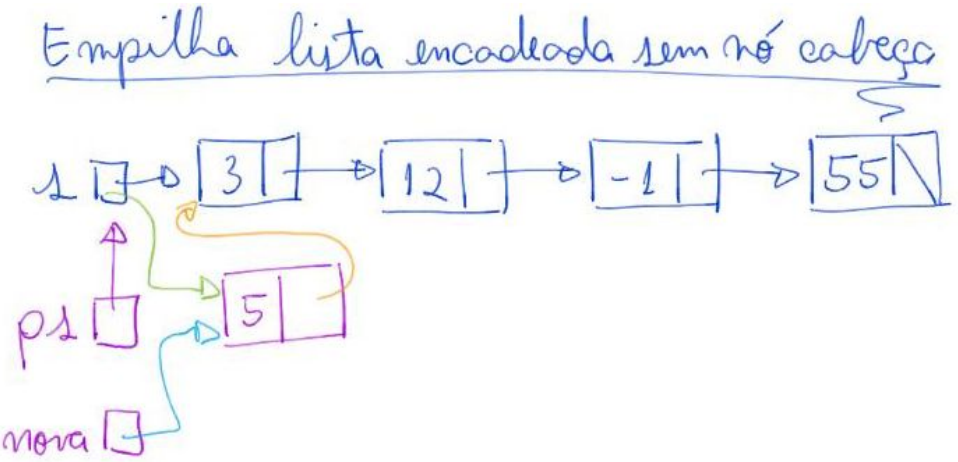
```
    {
        char c;
        switch (str[i])
        {
        case ')':
            if (pilhaVaziaCNC(s))
                return 0;
            c = desempilhaCNC(s);
            if (c != '(')
                return 0;
            break;
        case ']':
            if (pilhaVaziaCNC(s))
                return 0;
            c = desempilhaCNC(s);
            if (c != '[')
                return 0;
            break;
        default:
            empilhaCNC(s, str[i]);
        }
    }
    sol = pilhaVaziaCNC(s);
    s = liberaPilhaCNC(s);
    return sol;
}

int main(int argc, char *argv[])
{
    int i;
    char *str;
    Celula *s;
    char x;

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

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

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

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

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

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

    /* imprime pilha */
    imprimePilhaCNC(s);

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

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

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

    return 0;
}
```

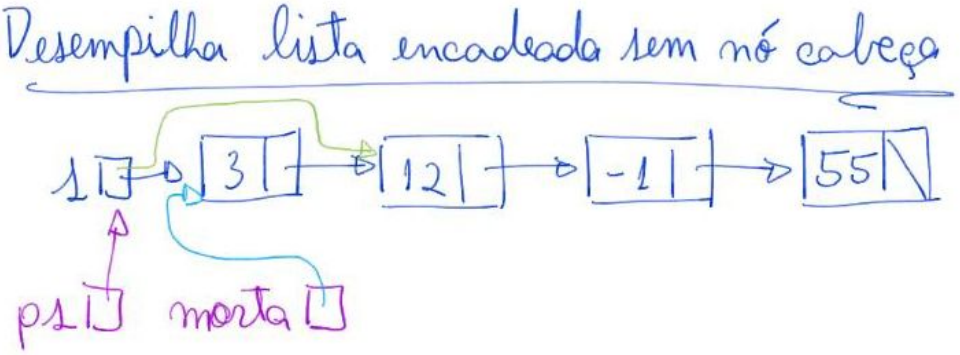
Funções para manipulação de pilha implementada em lista sem nó cabeça

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

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

Celula *criaPilhaSNC()
{
    return NULL;
}

void empilhaSNC(Celula **ps, char x)
{
    Celula *nova;

    nova = malloc(sizeof(Celula));
    nova->conteudo = x;
    nova->prox = *ps;
```

```
    *ps = nova;
}
```




```
char desempilhaSNC(Celula **ps)
{
    char x;
    Celula *morta;

    morta = *ps;
    x = morta->conteudo;
    *ps = morta->prox;
    free(morta);
    morta = NULL;
    return x;
}
```




```
char consultaTopoSNC(Celula *s)
{
    return s->conteudo;
}

int pilhaVaziaSNC(Celula *s)
{
    return s == NULL;
}

void imprimePilhaSNC(Celula *s)
```

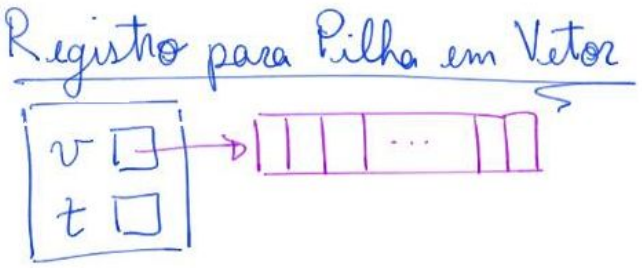
```
{
   Celula *p;

   p = s;
   while (p != NULL)
   {
       printf("%c ", p->conteudo);
       p = p->prox;
   }
   printf("\n");
}

int tamPilhaSNC(Celula *s){
   Celula *p;
   int tam = 0;

   p = s;
   while (p != NULL){
       p = p->prox;
       tam++;
   }
   return tam;
}

Celula *liberaPilhaSNC(Celula *s)
{
   Celula *p, *morta;

   p = s;
   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 *s;
   s = criaPilhaSNC();
   int sol;
   for (int i = 0; str[i] != '\0'; ++i)
```

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

int main(int argc, char *argv[])
{
    int i;
    char *str;
    Celula *s;
    char x;

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

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

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

```c
    /* imprime pilha */
    imprimePilhaSNC(s);

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

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

    /* imprime pilha */
    imprimePilhaSNC(s);

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

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

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

    return 0;
}
```

Funções para manipulação de pilha implementada em vetor

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

#define N 100

typedef struct pilha
{
    char *v;
    int t;
} Pilha;
```




```c
Pilha *criaPilhaVetor(int size)
{
    Pilha *s;
```

```
    s = (Pilha *)malloc(sizeof(Pilha));
    s->v = (char *)malloc(size * sizeof(char));
    s->t = 0;
    return s;
}

void empilhaVetor(Pilha *s, char x)
{
    s->v[s->t] = x;
    (s->t)++;
}

char desempilhaVetor(Pilha *s)
{
    (s->t) -= 1;
    return s->v[s->t];
}

char consultaTopoVetor(Pilha *s)
{
    return s->v[s->t - 1];
}

int pilhaVaziaVetor(Pilha *s)
{
    return s->t <= 0;
}

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

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

Pilha *liberaPilhaVetor(Pilha *s)
{
    free(s->v);
    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;
}

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

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

```c
    /* 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;
}
```

Compare as implementações de pilha

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