Pilhas em listas ligadas, bibliotecas e interfaces

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

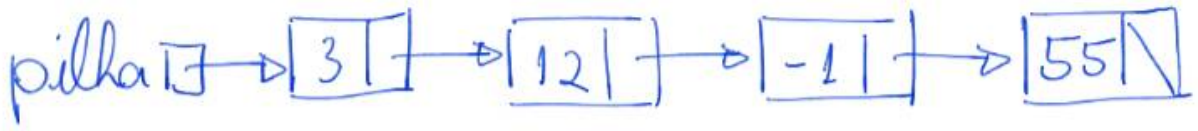
- é um modelo matemático para tipos de dados
    - definido em termos de seu comportamento
        - pelo ponto de vista do usuário,
    - e não de sua implementação.

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

- utilizando outra estrutura de dados que já conhecemos,
    - i.e., listas ligadas.
- Ao longo da aula, vamos comparar as diferentes implementações,
    - para analisar suas vantagens e limitações.

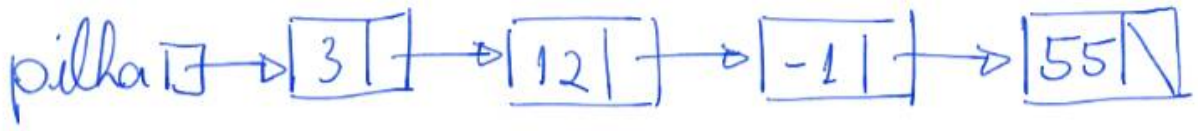
Antes de começar a implementação,

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



- Quiz1: 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.

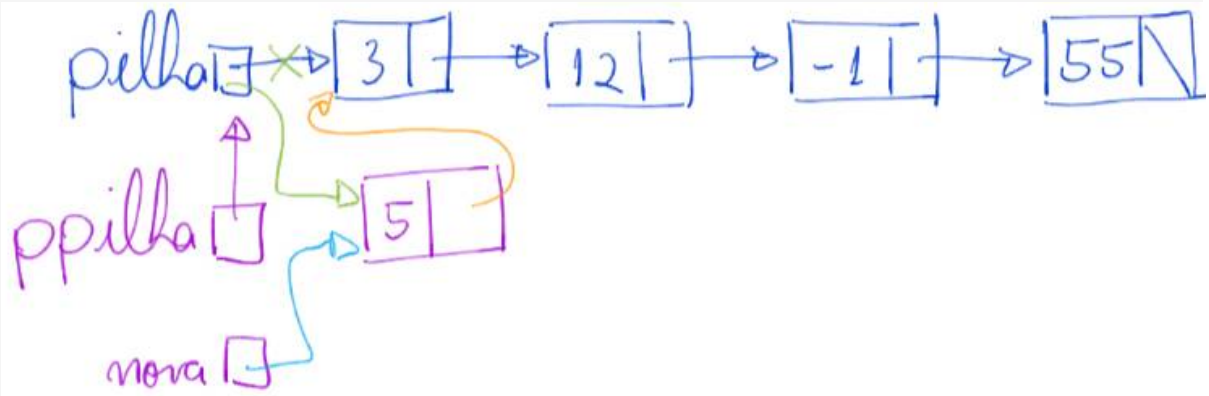
Funções para manipulação de pilha implementada em lista sem nó cabeça



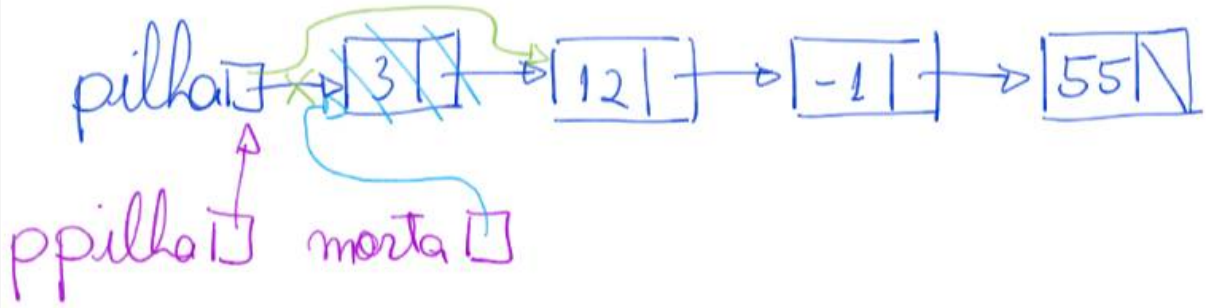

```c
#include <stdio.h>
#include <stdlib.h>

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

```c
Celula *criaPilha() {
    return NULL;
}
```




```c
void empilha(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 desempilha(Celula **ppilha) {
    char valor;
    Celula *morta;
    morta = *ppilha;
    valor = morta->conteudo;
    *ppilha = morta->prox;
    free(morta);
    morta = NULL;
    return valor;
}

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

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

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

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

- Quiz2: Como deixar essa operação mais eficiente?
    - O que deve mudar nas outras operações para a solução funcionar?

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

- Quiz3: Por que preciso do apontador morta na função liberaPilha?

```c
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];

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

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

    imprimePilha(pilha); /* imprime pilha */

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

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

    imprimePilha(pilha); /* imprime pilha */

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

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

    printf("%s eh bem formada? %d\n", str, bemFormada(str));
    return 0;
}
```

- Quiz4: Qual a eficiência de tempo e espaço de cada função?

```
// 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 = criaPilha();
    int sol;
    for (int i = 0; str[i] != '\0'; ++i) {
        char c;
        switch (str[i]) {
        case ')':
            if (pilhaVazia(pilha))
                return 0;
            c = desempilha(&pilha);
            if (c != '(')
                return 0;
            break;
        case ']':
            if (pilhaVazia(pilha))
                return 0;
            c = desempilha(&pilha);
            if (c != '[')
                return 0;
            break;
        default:
            empilha(&pilha, str[i]);
        }
    }
    sol = pilhaVazia(pilha);
    pilha = liberaPilha(pilha);
    return sol;
}
```

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

- Quiz5: implementar as operações em listas com nó cabeça.

Notem que a função bemFormada usando pilha com listas ligadas

    - ficou muito parecida com a função bemFormada usando vetores.
- Será que dá pra esconder esses detalhes de implementação do usuário?

Bibliotecas e interfaces

Uma biblioteca é uma coleção de funções que podem ser utilizadas por programas.

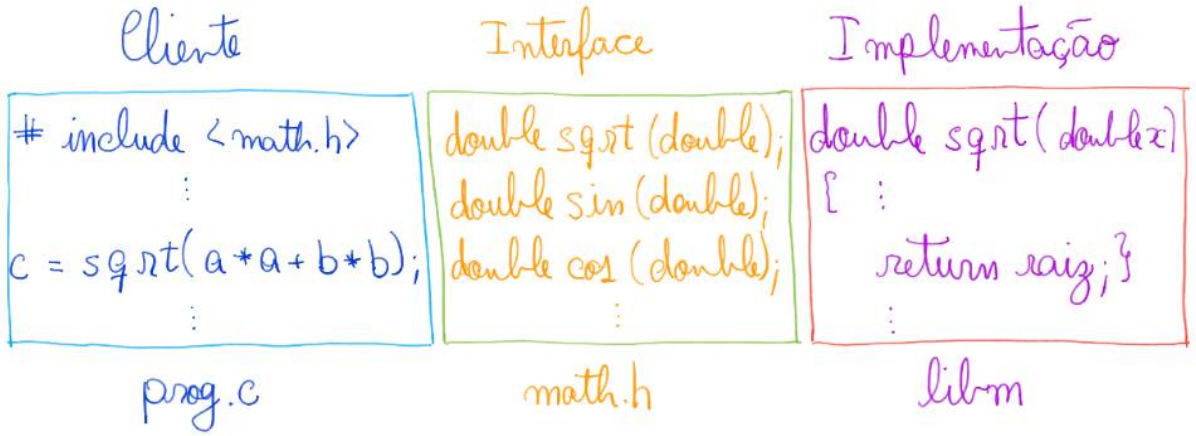
- Um cliente é um programa que utiliza alguma função de uma biblioteca.

Uma interface é a fronteira entre

- a implementação de uma biblioteca e os clientes que a utilizam.

Para cada função na biblioteca,

- o cliente precisa conhecer sua assinatura, i.e.,
  - o nome da função,
  - seus argumentos (e tipos),
  - e o tipo do resultado devolvido.
- Já os detalhes de implementação não são relevantes para o cliente.



Algumas decisões de projeto envolvendo bibliotecas.

- Interface: Quais as funções oferecidas?
- Ocultação: Quais informações são públicas e quais são privadas?
- Recursos: Quem é responsável por gerenciar memória e outros recursos?
- Erros: Quem é responsável por detectar e reportar erros?

Vamos ver como implementar nossa própria biblioteca para pilha.

- Segue o código da interface pilha.h:

```c
typedef struct pilha Pilha;

Pilha *criaPilha();
void empilha(Pilha *s, char x);
char desempilha(Pilha *s);
char consultaTopo(Pilha *s);
int pilhaVazia(Pilha *s);
void imprimePilha(Pilha *s);
int tamPilha(Pilha *s);
Pilha *liberaPilha(Pilha *s);
```

Para usar essa biblioteca,

- é necessário incluir uma chamada para ela no início do seu programa.

```c
#include "pilha.h"
```

Biblioteca implementada usando lista ligada

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

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

struct pilha {
    Celula *lst;
    int tam;
};

Pilha *criaPilha() {
    Pilha *s = (Pilha *)malloc(sizeof(Pilha));
    s->lst = NULL;
    s->tam = 0;
    return s;
}

void empilha(Pilha *s, char x) {
    Celula *nova = malloc(sizeof(Celula));
    nova->conteudo = x;
    nova->prox = s->lst;
    s->lst = nova;
    s->tam++;
}

char desempilha(Pilha *s) {
    char x;
    Celula *morta = s->lst;
    x = morta->conteudo;
    s->lst = morta->prox;
    free(morta);
    morta = NULL;
    (s->tam)--;
    return x;
}
```

```c
char consultaTopo(Pilha *s) {
    return s->lst->conteudo;
}

int pilhaVazia(Pilha *s) {
    return s->lst == NULL;
}

int pilhaCheia(Pilha *s) {
    return 0;
}

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

int tamPilha(Pilha *s) {
    return s->tam;
}

Pilha *liberaPilha(Pilha *s) {
    Celula *p, *morta;
    p = s->lst;
    while (p != NULL) {
        morta = p;
        p = p->prox;
        free(morta);
    }
    free(s);
    return NULL;
}
```

Compilando biblioteca

Para implementar e compilar um programa que usa nossa biblioteca,

- primeiro incluímos uma chamada para ela no início do programa,

```
#include "pilha.h"
```

- então compilamos a biblioteca em um programa objeto
  “gcc -c pilha.c” ou
  “gcc -Wall -O2 -pedantic -Wno-unused-result -c pilha.c”
- e, finalmente, compilamos o programa principal usando esse programa objeto
  “gcc pilha.o usaPilha.c -o usaPilha” ou
  “gcc -Wall -O2 -pedantic -Wno-unused-result pilha.o usaPilha.c -o usaPilha”

Também podemos compilar o programa principal em um programa objeto
“gcc -c usaPilha.c” ou
“gcc -Wall -O2 -pedantic -Wno-unused-result -c usaPilha.c”

- e então compilar os dois programas objetos no executável
  “gcc pilha.o usaPilha.o -o usaPilha”

Ou, no extremo oposto, compilar tudo diretamente, sem usar programas objeto
“gcc pilha.c usaPilha.c -o usaPilha” ou
“gcc -Wall -O2 -pedantic -Wno-unused-result pilha.c usaPilha.c -o usaPilha”

Observe que, como nossas duas implementações de pilha usam a mesma interface,

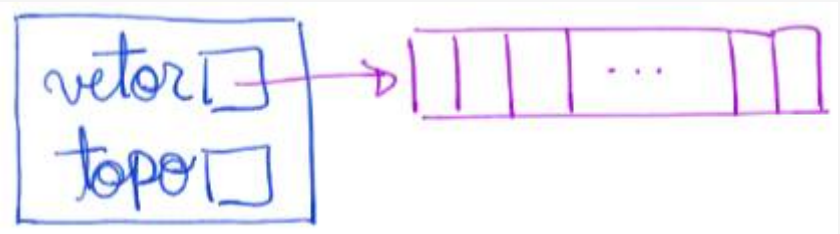
- podemos trocar a biblioteca sem afetar o funcionamento do programa.

Bônus: Biblioteca implementada usando vetor

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

#define N 100

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




```c
Pilha *criaPilha(int size) {
    Pilha *s;
    s = (Pilha *)malloc(sizeof(Pilha));
```

```
    s->vetor = (char *)malloc(size * sizeof(char));
    s->topo = 0;
    return s;
}

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

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

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

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

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

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

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

- Quiz6: Qual a eficiência de tempo e espaço de cada função?