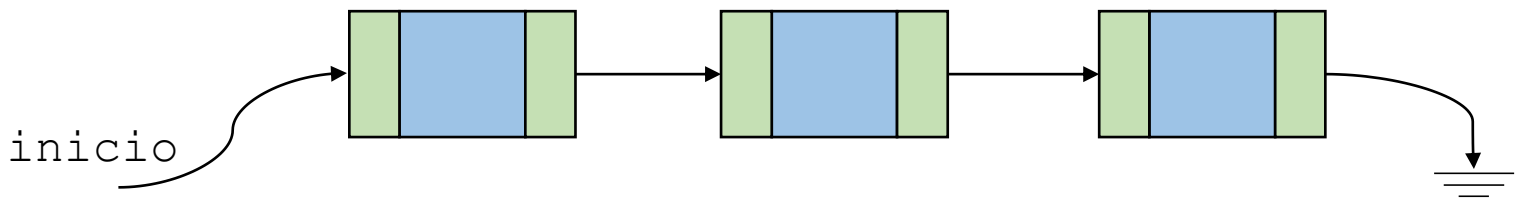
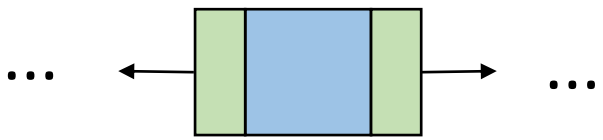
Lista duplamente encadeada

- Uma das desvantagens da lista simples que observamos na aula anterior é que ela apresenta apenas uma direção de movimento (do início para o fim da lista (NULL));
- Uma solução é criar nós com dois ponteiros
  - Um para o próximo nó, como já fazíamos;
  - Um para o nó anterior.



Lista duplamente encadeada

```
1  typedef struct elemento{
2      int info;
3      struct elemento *ant;
4      struct elemento *prox;
5  }No;
```



Universidade Estadual de Londrina

Técnicas de Programação

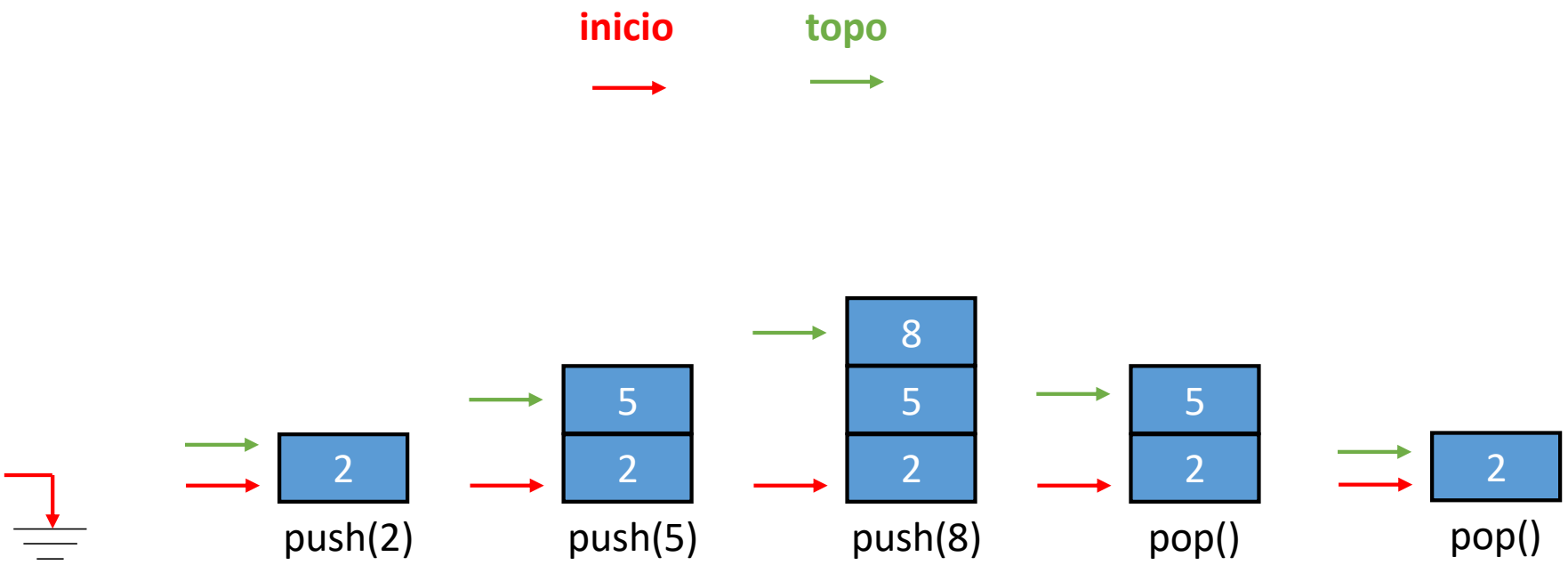
Fila e pilha

Pilha (stack)

- Uma pilha é um caso especial de lista;
- Podemos definir uma pilha restringindo as operações da lista do seguinte modo:
  - Apenas podemos adicionar um novo elemento na última posição da lista (topo);
  - Apenas podemos remover um elemento do final da lista (topo);
- Essas restrições fazem com que uma lista seja também definida como uma lista LIFO (*last-in, first-out*);

Pilha

- As operações de inserção e exclusão de elementos na pilha recebem nomes especiais:
  - **Push:** insere um elemento no topo da pilha;
  - **Pop:** remove o elemento que está no topo da pilha.



Pilha

```
1  typedef struct elemento{
2      int info;
3      struct elemento *prox;
4  }No;
5
6  typedef struct{
7     No *topo;
8     No *inicio;
9  }Pilha;
```

Pilha

```
 1  typedef struct elemento{
 2      int info;
 3      struct elemento *prox;
 4  }No;
 5
 6  typedef struct{
 7     No *topo;
 8     No *inicio;
 9  }Pilha;
10  ...
11
12  int main(){
13     Pilha *p;
14     p = (Pilha *) malloc(sizeof(Pilha));
15
16     ...
17
18     return 0;
19  }
```

Pilha

```
 1  typedef struct elemento{
 2      int info;
 3      struct elemento *prox;
 4  }No;
 5
 6  typedef struct{
 7     No *topo;
 8     No *inicio;
 9  }Pilha;
10  ...
11
12  int main(){
13     Pilha *p;
14     p = (Pilha *) malloc(sizeof(Pilha));
15
16     ...
17
18     return 0;
19  }
```

```
void iniciaPilha(Pilha **p)
```

```
bool vazia(Pilha *p)
```

```
void push(Pilha **p, int x)
```

```
int pop(Pilha **p)
```

```
void imprime(Pilha *p)
```

```
void desaloca(Pilha **p)
```

Pilha

Inicia a pilha fazendo o inicio e o topo apontarem para NULL

```
1  void iniciaPilha(Pilha **p){
2      (*p)->inicio = NULL;
3      (*p)->topo = NULL;
4  }
```

Função que verifica se a pilha está vazia

```
1  bool vazia(Pilha *p){
2      if(p->inicio == NULL)
3          return true;
4      return false;
5  }
```

Pilha

Empilha o inteiro x na pilha

```
 1  void push(Pilha **p, int x){
 2      No *aux;
 3
 4      aux = (No *) malloc(sizeof(No));
 5      aux->info = x;
 6      aux->prox = NULL;
 7
 8      if(vazia(p)){  //Lista vazia
 9          (*p)->inicio = aux;
10          (*p)->topo = (*p)->inicio;
11      }
12      else{
13          (*p)->topo->prox = aux;
14          (*p)->topo = aux;
15      }
16  }
```

Pilha

Elimina o valor que está no topo da pilha o inteiro x na pilha

```
1  void pop(Pilha **p){
2      No *aux;
3      if(vazia(*p)){
4          printf("lista vazia...\n");
5          return;
6      }
7      aux = (*p)->inicio;
8
9      if((*p)->inicio->prox == NULL){ //existe apenas um nó
10        (*p)->inicio = NULL;
11        free(aux);
12     }
13     else{ //existe mais de um nó na pilha
14         while(aux->prox != (*p)->topo)
15             aux = aux->prox;
16
17          free((*p)->topo);
18          (*p)->topo = aux;
19          (*p)->topo->prox = NULL;
20     }
21 }
```

Pilha

Imprime os elementos que estão na pilha, do inicio ao topo

```
1  void imprime(Pilha *p){
2      No *aux;
3
4      aux = p->inicio;
5      while(aux != NULL){
6          printf(" %d ", aux->info);
7          aux = aux->prox;
8      }
9  }
```

Desaloca a memória usada pela pilha

```
1  void desaloca(Pilha **p){
2      No *aux;
3
4      aux = (*p)->inicio;
5
6      while(aux != NULL){
7          (*p)->inicio = (*p)->inicio->prox;
8          free(aux);
9          aux = (*p)->inicio;
10     }
11 }
```

Pilha

Chamando as funções criadas

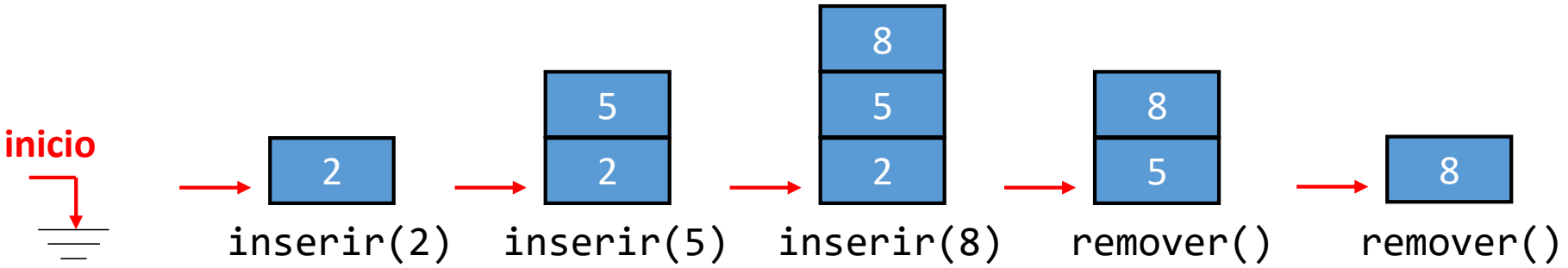
```
 1  int main(){
 2      Pilha *p;
 3      p = (Pilha *) malloc(sizeof(Pilha));
 4
 5      iniciaPilha(&p);
 6      push(&p, 2);
 7      push(&p, 3);
 8      push(&p, -1);
 9      push(&p, 918);
10      pop(&p);
11      imprime(p);
12      desaloca(&p);
13
14      return 0;
15  }
```

Fila

- Uma fila difere de uma pilha na medida em que ela opera na base de uma lista FIFO (*first-in*, *first-out*)
  - Adicionamos novos elementos ao final da lista (fila);
  - Removemos sempre um elemento do início da fila.
- Uma código de uma lista pode ser facilmente convertido para um código de fila. Quais são as alterações necessárias?

Fila

- Uma fila difere de uma pilha na medida em que ela opera na base de uma lista FIFO (*first-in*, *first-out*)
  - Adicionamos novos elementos ao final da lista (fila);
  - Removemos sempre um elemento do início da fila.



Exercícios

1. Faça um algoritmo para converter números decimais em binários usando pilha;
2. Considere o problema de decidir se uma dada sequência de parênteses e colchetes está bem-formada (ou seja, parênteses e colchetes são fechados na ordem inversa àquela em que foram abertos). Por exemplo, a sequência: `(()[()])` está bem-formada, enquanto `([)]` está malformada. Crie um algoritmo usando pilha para verificar se uma sequencia é ou não bem formada.