Exercício de fixação e aprendizagem

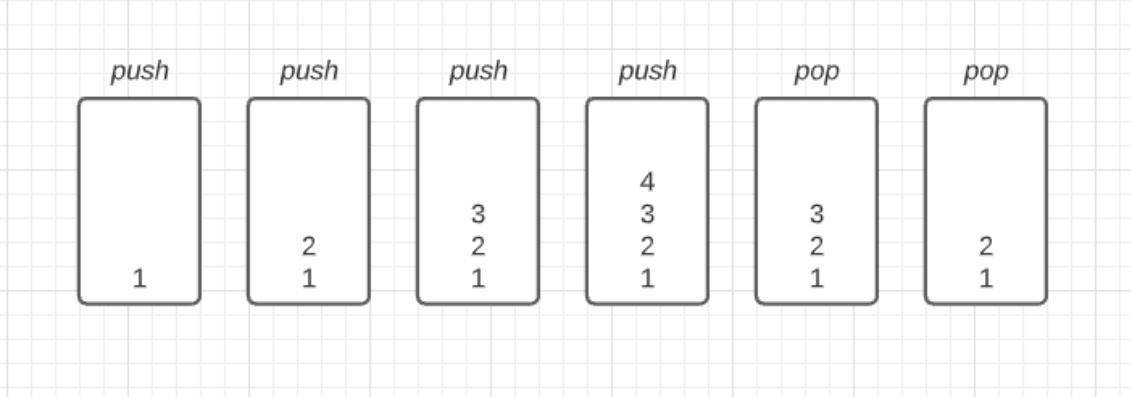
Aluno: José Messias Marinho Olimpio

Matrícula: 20190033377

Primeira Questão

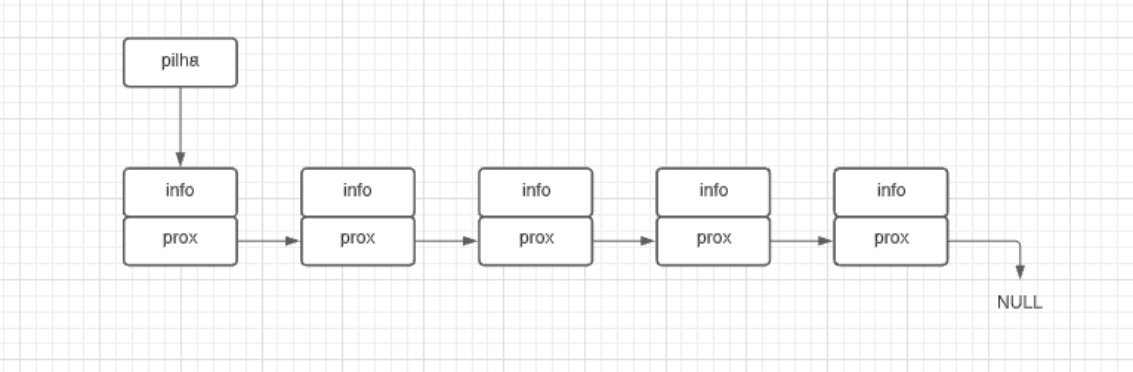
▼ Letra **a)**

- A pilha é uma das estruturas de dados mais simples, e uma das mais usadas na programação. Os elementos inseridos são colocados no topo, e a variável, a qual apontará para a pilha, apontará para o elemento que está no seu topo.
- Portanto, pode-se dizer que a estrutura de dados pilha funciona como uma pilha de tijolos empilhados um a um em uma única coluna. Nessa pilha de tijolos não possível remover o primeiro tijolo, que é a base, nem tampouco o tijolo que está no meio, pois nesse caso a pilha de tijolos irá cair. Assim, o acesso a pilha deve ser feito somente através do **topo**, respeitando sua estrutura e, por esse motivo, segue a regra *LIFO (last in, first out)*, o último a entrar será o primeiro a sair.
- As operações de inserir e remover são comumente referidas como:
  - *push*
    - Insere um novo elemento no topo da pilha
  - *pop*
    - Remove o elemento que esta no topo
- Diagrama demonstrativo de uma pilha:



- No diagrama acima os valores são inseridos e removidos no topo da pilha.

- Implementação da pilha usando lista encadeada



▼ Letra **b)**

- *main.c*

```
#include <stdio.h>
#include <string.h>
#include "pilha.h"

int main(void) {
  Pilha *pilha = NULL;

  pilha = pilha_cria();

  if(pilha_push(pilha, 'o')) {
    printf("Valor inserido com sucesso!\n");
  } else {
    printf("Algo de errado aconteceu!\n");
  }

  if(pilha_push(pilha, 'v')) {
```

```
    printf("Valor inserido com sucesso!\n");
  } else {
    printf("Algo de errado aconteceu!\n");
  }

  if(pilha_push(pilha, 'o')) {
    printf("Valor inserido com sucesso!\n");
  } else {
    printf("Algo de errado aconteceu!\n");
  }

  printf("\n%c", pilha_pop(pilha));
  printf("%c", pilha_pop(pilha));
  printf("%c\n", pilha_pop(pilha));

  if(pilha_vazia(pilha)) {
    printf("A pilha esta vazia!\n");
  } else {
    printf("A pilha nao esta vazia!\n");
  }

  pilha_libera(&pilha);

  if(!pilha) printf("\nPilha limpa com sucesso!\n");
}
```

- *pilha.h*

```
#ifndef _PILHA_H
#define _PILHA_H

typedef struct pilha Pilha;
typedef struct lista Lista;

Pilha* pilha_cria(void);
int pilha_push(Pilha *pilha, char caractere);
char pilha_pop(Pilha *pilha);
int pilha_vazia(Pilha *pilha);
void pilha_libera(Pilha **pilha);

#endif
```

- *pilha.c*

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

```
#ifndef _PILHA_C
#define _PILHA_C

struct lista {
  char info;
  Lista *prox;
};

struct pilha {
  Lista *topo;
};

Pilha* pilha_cria(void) {
  Pilha *nova_pilha = NULL;

  nova_pilha = (Pilha*) malloc(sizeof(Pilha));
  if(!nova_pilha) return 0;

  nova_pilha->topo = NULL;

  return nova_pilha;
}

int pilha_push(Pilha *pilha, char caractere) {
  if(!pilha) return 0;

  Lista *novo_elemento = NULL;

  novo_elemento = (Lista*) malloc(sizeof(Lista));
  if(novo_elemento) {
    novo_elemento->info = caractere;

    // O novo elemento ira apontar para o elemento que esta no topo
    novo_elemento->prox = pilha->topo;

    // O topo ira apontar para o novo elemento
    pilha->topo = novo_elemento;

    return 1;
  }

  return 0;
}

char pilha_pop(Pilha *pilha) {
  Lista *aux = NULL;

  char elemento = pilha->topo->info;

  aux = pilha->topo;
  pilha->topo = aux->prox;

  free(aux);
```

```
  return elemento;
}

// 1 - Lista esta vazia (1 equivale a true)
// 0 - Lista nao esta vazia (0 equivale a false)
int pilha_vazia(Pilha *pilha) {
  return (pilha == NULL);
}

void pilha_libera(Pilha **pilha) {
  Pilha *aux = *pilha;
  Lista *lista = aux->topo;
  Lista *elemento_pilha = NULL;

  while(lista) {
    elemento_pilha = lista->prox;

    free(lista);
    lista = elemento_pilha;
  }

  free(aux);

  *pilha = NULL;
}

#endif
```

▼ Letra **c)**

- *pilha_balanceamento (função)*

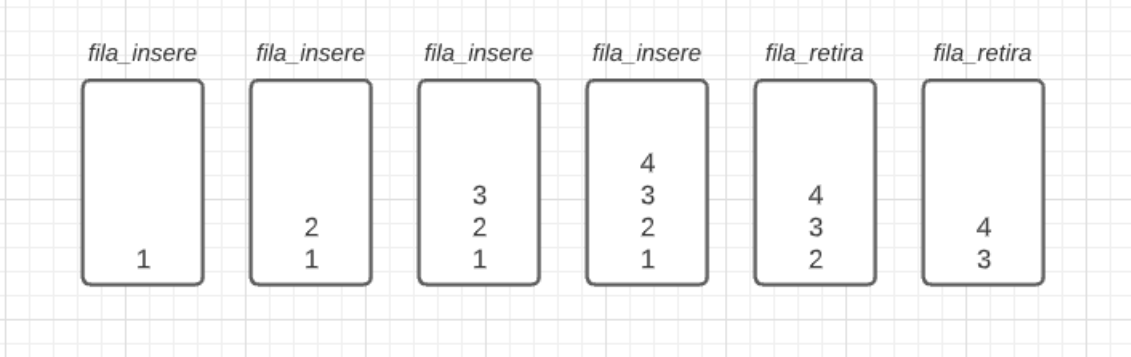
```
int pilha_balanceada(Pilha *pilha, char *expressao) {
  int chaves = 0, colchetes = 0, parenteses = 0;
  char elemento;

  // Ira empilhar a expressao
  for (int i = 0; i < strlen(expressao); i++) {
    pilha_push(pilha, expressao[i]);
  }

  printf("%s\n", expressao);

  // Ira desempilhar e analisar os caracteres
  // A cada fechamento e somado um ao valor do caractere
  // A cada fechamento ira subtrair 1
  // Ao final de tudo todas as variaves terao que ser iguais a 0
  while (pilha->topo) {
    elemento = pilha_pop(pilha);
```

```
    if(elemento == '}') chaves++;
    else if(elemento == ']') colchetes++;
    else if(elemento == ')') parenteses++;
    else if(elemento == '{') chaves--;
    else if(elemento == '[') colchetes--;
    else if(elemento == '(') parenteses--;

    // Retorna caso algum valor negativo
    // O valor sera negativo se houve uma abertura antes do fechamento
    if(chaves < 0 || colchetes < 0 || parenteses< 0) return 0;
  }

  return 1;
}
```

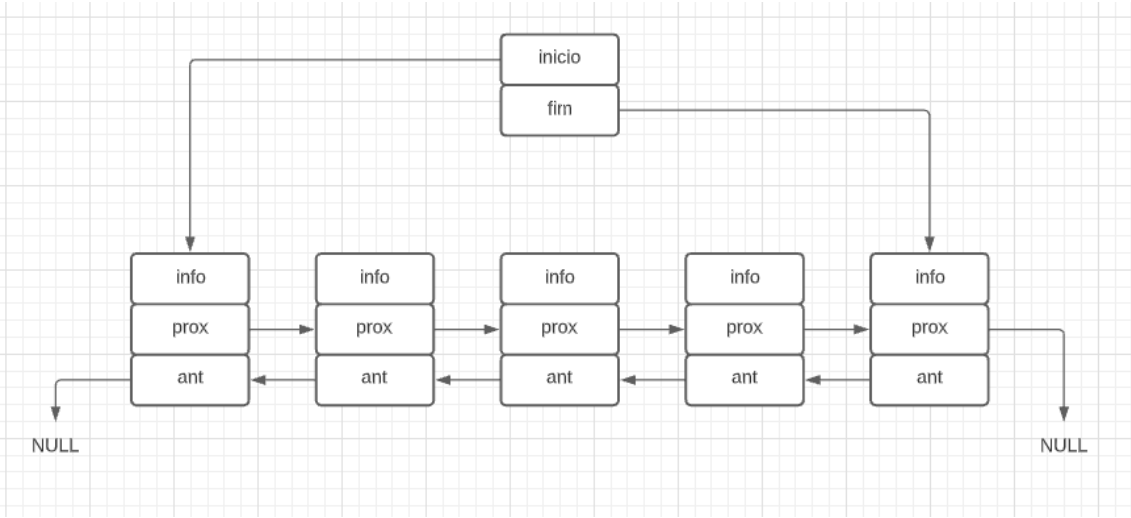
Segunda Questão

▼ Letra **a)**

- A fila, assim como a pilha, é uma estrutura de dados que também é bastante utilizada. Para explicar seu funcionamento basta utilizar o conceito comumente conhecido de fila por exemplo, a fila de um caixa de supermercado: o primeiro que entra na fila é o primeiro a ser atendido e consequentemente, o último a chegar será o último a sair. Logo, a estrutura de dados fila funciona da mesma forma, em outras palavras, a estrutura segue a regra *FIFO (first in, first out)* o primeiro a entrar será o primeiro a sair.
- Desse modo, é necessário que a inserção seja no fim, porém, a retirada seja no início. Portanto, o TAD da estrutura deverá conter duas operações básicas:
  - Inserção no fim
  - Remoção no início
- Diagrama demonstrativo de uma fila:



- Implementação da fila com lista duplamente encadeada



▼ Letra **b)**

- *main.c*

```
#include <stdio.h>
#include "fila.h"

int main(void) {
  Fila *fila_principal = NULL;

  fila_principal = fila_cria();

  if(fila_insere(fila_principal, '1')) {
    printf("Valor inserido com sucesso!\n");
  } else {
    printf("Algo de errado aconteceu!\n");
  }

  if(fila_insere(fila_principal, '2')) {
```

```
    printf("Valor inserido com sucesso!\n");
  } else {
    printf("Algo de errado aconteceu!\n");
  }

  if(fila_insere(fila_principal, '3')) {
    printf("Valor inserido com sucesso!\n");
  } else {
    printf("Algo de errado aconteceu!\n");
  }

  if(fila_insere(fila_principal, '4')) {
    printf("Valor inserido com sucesso!\n");
  } else {
    printf("Algo de errado aconteceu!\n");
  }

  if(fila_insere(fila_principal, '5')) {
    printf("Valor inserido com sucesso!\n");
  } else {
    printf("Algo de errado aconteceu!\n");
  }

  if(fila_vazia(fila_principal)) {
    printf("A fila esta vazia!\n");
  } else {
    printf("A fila nao esta vazia!\n");
  }

  printf("\n%c", fila_retira(fila_principal));
  printf("%c", fila_retira(fila_principal));
  printf("%c", fila_retira(fila_principal));
  printf("%c", fila_retira(fila_principal));
  printf("%c\n", fila_retira(fila_principal));

  if(fila_vazia(fila_principal)) {
    printf("A fila esta vazia!\n");
  } else {
    printf("A fila nao esta vazia!\n");
  }

  fila_libera(&fila_principal);

  if(!fila_principal) printf("A fila foi limpa!\n");
}
```

- *fila.c*

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

```
#ifndef _FILA_C
#define _FILA_C

struct no {
  char info;
  NoLista* ant;
  NoLista* prox;
};

struct fila {
  NoLista* prim;
  NoLista* ult;
};

Fila* fila_cria(void) {
  Fila *nova_fila = NULL;

  nova_fila = (Fila*) malloc(sizeof(Fila));
  if(!nova_fila) return 0;

  nova_fila->prim = NULL;
  nova_fila->ult = NULL;

  return nova_fila;
}

int fila_insere(Fila* fila_main, char caractere) {
  if(!fila_main) return 0;

  NoLista *novo_elemento = NULL;

  novo_elemento = (NoLista*) malloc(sizeof(NoLista));
  if(novo_elemento) {
    // Caso onde ta sendo inserido o primeiro elemento na fila
    if(!fila_main->prim && !fila_main->ult) {
      novo_elemento->info = caractere;
      novo_elemento->ant = fila_main->ult;
      novo_elemento->prox = fila_main->prim;

      fila_main->prim = novo_elemento;
      fila_main->ult = novo_elemento;
    } else {
      novo_elemento->info = caractere;
      novo_elemento->ant = fila_main->ult;
      novo_elemento->prox = NULL;

      fila_main->ult->prox = novo_elemento;
      fila_main->ult = novo_elemento;
    }

    return 1;
  }
```

```
  return 0;
}

char fila_retira(Fila* fila_main) {
  NoLista *aux = fila_main->prim;
  char elemento = aux->info;

  fila_main->prim = aux->prox;

  // Caso onde a fila possui um unico elemento e ele sera removido
  if(fila_main->prim) {
    fila_main->prim->ant = NULL;
  } else {
    fila_main->ult = NULL;
  }

  free(aux);

  return elemento;
}

int fila_vazia(Fila* fila_main) {
  return (fila_main->prim == NULL && fila_main->ult == NULL);
}

void fila_libera(Fila** fila_main) {
  Fila *aux_fila = *fila_main;
  NoLista *aux_no = aux_fila->prim;
  NoLista *elemento_lista = NULL;

  while(aux_no) {
    elemento_lista = aux_no->prox;

    free(aux_no);
    aux_no = elemento_lista;
  }

  free(aux_fila);

  *fila_main = NULL;
}

#endif
```

- *fila.h*

```
#ifndef _FILA_H
#define _FILA_H

typedef struct no NoLista;
typedef struct fila Fila;
```

```
Fila* fila_cria(void);
int fila_insere(Fila* fila_main, char caractere);
char fila_retira(Fila* fila_main);
int fila_vazia(Fila* fila_main);
void fila_libera(Fila** fila_main);

#endif
```

▼ Letra **c)**

- *verifica_palindroma (programa)*

```
#include <stdio.h>
#include <string.h>
#include "./includes/fila.h"
#include "./includes/pilha.h"

struct fila {
  NoLista* prim;
  NoLista* ult;
};

struct pilha {
  Lista *topo;
};

int verifica_palindroma(Pilha *pilha_main, Fila *fila_main, char *frase);

int main(void) {
  Pilha *pilha_principal = NULL;
  Fila *fila_principal = NULL;

  pilha_principal = pilha_cria();
  fila_principal = fila_cria();

  if(verifica_palindroma(pilha_principal, fila_principal, "subi no onibus")) {
    printf("A palavra e palindroma!\n");
  } else {
    printf("A palavra nao e palindroma!\n");
  }

  pilha_libera(&pilha_principal);
  fila_libera(&fila_principal);

  if(!pilha_principal && !fila_principal) {
    printf("As estruturas foram liberadas com sucesso!\n");
  }

  return 0;
}
```

```
int verifica_palindroma(Pilha *pilha_main, Fila *fila_main, char *frase) {
  for (int i = 0; i < strlen(frase); i++) {
    // Pula os espacos em branco e insere sem eles
    if(frase[i] == ' ') {
      pilha_push(pilha_main, frase[i]);
      fila_insere(fila_main, frase[i]);
    }
  }

  // Verifica se algum caractere foi inserido
  if(!pilha_main->topo && !fila_main->prim) return 0;

  // Retira todos os elementos da pilha e da fila ate que estejam vazios
  while(pilha_main->topo && fila_main->prim) {
    if(pilha_pop(pilha_main) != fila_retira(fila_main)) return 0;
  }

  return 1;
}
```

Terceira Questão

▼ Letra **a)**

- *main.c*

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

int maior_valor(int *vetor, size_t tamanho);

int main(void) {
  int vetor_numeros[10] = {2, 1, 4, 6, 5, 10, 3, 7, 9, 8};

  // Ira dividir o tamanho do vetor pelo tamanho do primeiro elemento
  // Nesse caso o tamanho de um int e 4 bytes, o vetor tem 40 bytes
  // 40 / 10 = 4
  size_t tamanho_vetor = sizeof(vetor_numeros) / sizeof(vetor_numeros[0]);

  printf("Maior numero -> %d\n", maior_valor(vetor_numeros, tamanho_vetor));
}

int maior_valor(int *vetor, size_t tamanho) {
  int maior_numero = 0;

  for (int i = 0; i < tamanho; i++) {
    if(vetor[i] > maior_numero) maior_numero = vetor[i];
```

```
  }

  return maior_numero;
}
```

▼ Letra **b)**

- Função

$$f(n) = n$$

  - Essa função será linear uma vez que o código terá apenas um ***for*** que irá iterar de acordo com o tamanho do vetor que será recebido, junto com o vetor, como parâmetro na função, ou seja, quanto maior o vetor maior será o ***n***.

▼ Letra **c)**

- Complexidade *Big O*

$$n \leq c \cdot n$$

$$\frac{n}{n} \leq c$$

$$c \geq 1$$

  - Logo, a afirmação acima é valida para um $c = 20$ e $n_0 = 1$
  - Então, é possível afirmar que $f(n) = O(n)$

- Complexidade Ômega

$$n \geq c \cdot n$$

$$\frac{n}{n} \geq c$$

$$c \leq 1$$

  - Logo, a afirmação acima é valida para um $c = 0$ e $n_0 = 0$
  - Então, é possível afirmar que $f(n) = \omega(n)$

- Complexidade Theta
  - Uma vez conhecida as complexidades *Ômega* e *Big O* utilizaremos como constantes:$C_1 g(n) = 0, C_2 g(n) = 20$
  - Sabendo que o tamanho do *vetor* usado na **letra a)** é igual a 10, então:

$$\theta(g(n)) = \{c_1 \cdot n \leq n \leq c_2 \cdot n\}$$

$$\theta(g(n)) = \{0n \leq n \leq 20n\}$$

$$\theta(g(n)) = \{0 \leq 10 \leq 200\}$$

  - Então, é possível afirmar que $f(n) = \theta(n)$
- Logo, o algoritmo será válido para todos os casos em que n será maior ou igual a 0.

Quarta Questão

▼ Algoritmo de Euclides

- *main.c*

```
#include <stdio.h>

int mdc(int a, int b);

int main(void) {
  printf(" %d ", mdc(12, -5));
}

// Irá verificar o mdc dos valores recebidos através de recursao
// A cada chamada ira realizar algumas verificacoes e chamar de novo a funcao
// Passando o valor de b no primeiro argumento e o resto da divisao de
// a por b como segundo argumento ate que o segundo argumento seja 0
// Apos isso ira retornar o valor de a
int mdc(int a, int b) {
  if(b == 0) return a;
  else if(b < 0) return mdc(a, -b);

  return mdc(b, a % b);
}
```

Bibliografia

- Waldemar Celes. Introdução a Estruturas de Dados - Com Técnicas de Programação em C. São Paulo: Grupo GEN, 2016. 9788595156654. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595156654/. Acesso em: 2021 set. 12.