UNIVERSIDADE FEDERAL DO CEARÁ
Campus de Quixadá – Ciência da Computação
Prof. Atílio Luiz
QXD0010- Estrutura de Dados

AP2
2025.2

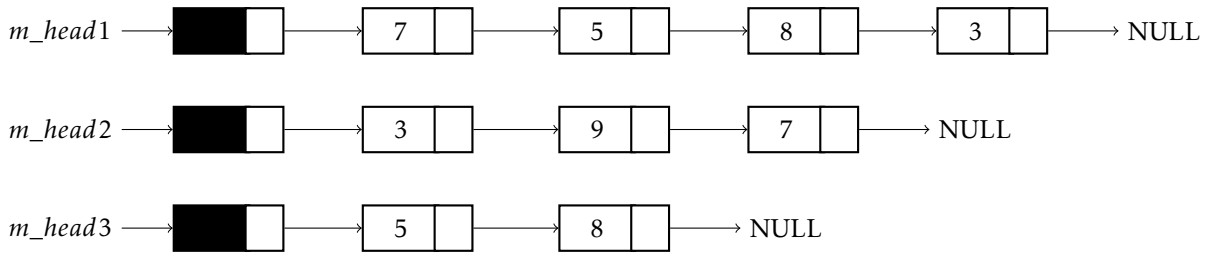
Nome: _____ Matrícula: _____

[2 pontos] 1. Para essa questão, suponha uma **lista simplesmente encadeada com nó sentinela**. Escreva uma função em C que recebe como entrada dois ponteiros para os nós sentinelas de duas listas simplesmente encadeadas e retorna um ponteiro para uma nova lista contendo todos os elementos da primeira lista que não estejam na segunda lista. As duas listas usadas na criação da terceira NÃO devem ser modificadas. Pode supor que numa lista não há elementos repetidos.

O protótipo da função é dado por:

Node* diferença(Node *m_head1, Node *m_head2);

Um exemplo da operação que deve ser realizada entre as duas listas é ilustrado abaixo. Veja que os únicos elementos que estão na primeira lista e que não estão na segunda lista são os números 5 e 8. Os números na terceira lista que será produzida podem estar em qualquer ordem. Atenção: qualquer função adicional que venha a ser utilizada deve ser implementada também.



Você pode usar a seguinte declaração abaixo:

```c
typedef struct node {
  int value;
  struct node *next;
} Node;
```

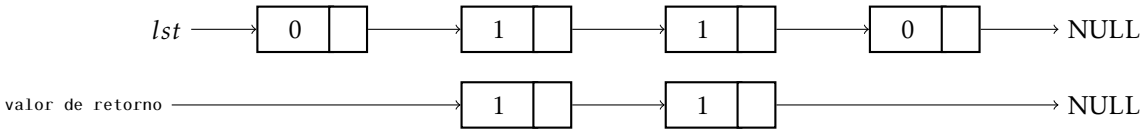
Solução:

```
01. Node* diferenca(Node *mhead1, Node *mhead2) {
02.     Node *mhead3 = (Node*) malloc(sizeof(Node));
03.     mhead3->next = NULL;
04.     for(Node *p1 = mhead1->next; p1 != NULL; p1 = p1->next) {
05.         bool esta_contido = false;
06.         for(Node *p2 = mhead2->next; p2 != NULL; p2 = p2->next) {
07.             if(p1->value == p2->value) {
08.                 esta_contido = true;
09.                 break;
10.             }
11.         }
12.         if(!esta_contido) {
13.             Node *novo = (Node*) malloc(sizeof(Node));
14.             novo->next = mhead3->next;
15.             novo->value = p1->value;
16.             mhead3->next = novo;
17.         }
18.     }
19.     return mhead3;
20. }
```

Boa Prova!

[2 pontos] 2. Escreva uma função que percorre uma lista simplesmente encadeada de inteiros sem nó sentinela e remove (liberando adequadamente a memória) todos os nós que contêm elementos de valor igual a 0. A função retorna um ponteiro para a lista modificada. Utilize a definição de nó dada na questão anterior. Atenção: qualquer função adicional que venha a ser utilizada deve ser implementada também. A função deve seguir o seguinte protótipo:

Node* remove_zeros(Node *lst);



Solução:

```
01. Node* remove_zeros(Node *lst) {
02.     while(lst != NULL && lst->value == 0) {
03.         Node *temp = lst;
04.         lst = lst->next;
05.         free(temp);
06.     }
07.     if(lst == NULL) {
08.         return NULL;
09.     }
10.     Node *aux = lst;
11.     while(aux->next != NULL) {
12.         if(aux->next->value == 0) {
13.             Node *temp = aux->next;
14.             aux->next = temp->next;
15.             free(temp);
16.         } else {
17.              aux = aux->next;
18.         }
19.     }
20.     return lst;
21. }
```

Boa Prova!

[2 pontos] 3. Considerando o tipo Node definido na Questão 1, implemente uma função para testar se duas listas passadas como parâmetro são iguais. Assuma que as listas são **simplesmente encadeadas sem nó sentinela**. Essa função deve obedecer o seguinte protótipo:

bool listas_iguais(Node *l1, Node *l2);

Atenção: qualquer função adicional que venha a ser utilizada deve ser implementada também.

Solução:

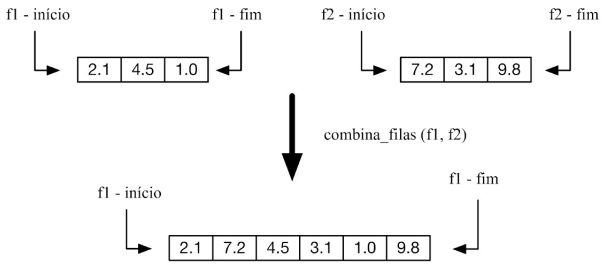
```
01. bool listas_iguais(Node *l1, Node *l2) {
02.     while(l1 != NULL && l2 != NULL) {
03.         if(l1->value != l2->value) {
04.             return false;
05.         }
06.         l1 = l1->next;
07.         l2 = l2->next;
08.     }
09.     return (l1 == NULL && l2 == NULL);
10. }
```

Boa Prova!

[2 pontos] 4. Considere a existência de um tipo abstrato Fila (Queue) de números reais, cuja interface está definida no arquivo "Queue.h" da seguinte forma:

```
typedef struct queue Queue;
Queue* queue_create (void);
void queue_push (Queue *q, float value);
void queue_pop (Queue *q);
float queue_front (Queue *q);
bool queue_empty (Queue *q);
void queue_free (Queue *q);
```

Sem conhecer a representação interna desse tipo abstrato e usando apenas as funções declaradas no arquivo de interface Queue.h, implemente uma função que receba três filas, `f_res`, `f1`, `f2`, e transfira alternadamente os elementos de `f1` e `f2` para `f_res`, conforme ilustrado na figura a seguir:



Note que, ao final dessa função, as filas `f1` e `f2` vão estar vazias e a fila `f_res` vai conter todos os valores que estavam originalmente em `f1` e `f2`. Se uma fila for maior que a outra, os valores excedentes devem ser transferidos para a nova fila no final. Essa função deve obedecer ao seguinte protótipo:

void combina_filas(Queue *f_res, Queue *f1, Queue *f2);

Solution:

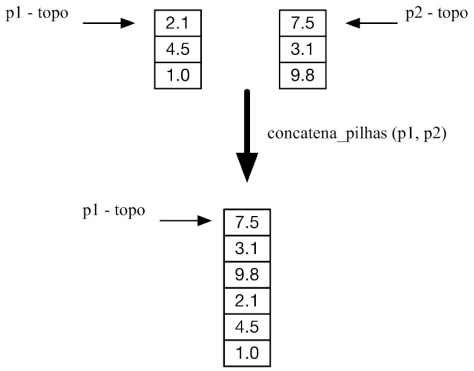
```
01. void combinaFilas(Queue *fres, Queue *f1, Queue *f2) {
02.     while(!queue_empty(f1) && !queue_empty(f2)) {
03.         queue_push(fres, queue_front(f1));
04.         queue_pop(f1);
05.         queue_push(fres, queue_front(f2));
06.         queue_pop(f2);
07.     }
08.     while(!queue_empty(f1)) {
09.          queue_push(fres, queue_front(f1));
10.          queue_pop(f1);
11.     }
12.     while(!queue_empty(f2)) {
13.          queue_push(fres, queue_front(f2));
14.          queue_pop(f2);
15.     }
16. }
```

Boa Prova!

[2 pontos] 5. Considere a existência de um tipo abstrato Pilha (Stack) de números reais, cuja interface está definida no arquivo "Stack.h" da seguinte forma:

```
typedef struct stack Stack;
Stack* stack_create(void);
void stack_push (Stack *f, float value);
void stack_pop (Stack *f);
float stack_top (Stack *f);
bool stack_empty (Stack *f);
void stack_free (Stack* f);
```

Sem conhecer a representação interna desse tipo abstrato e usando apenas as funções declaradas no arquivo de interface, implemente uma função que recebe duas pilhas p1 e p2 e passa todos os elementos da pilha p2 para a pilha p1. A figura a seguir ilustra essa concatenação de pilhas. Note que, ao final dessa função, a pilha p2 vai estar vazia e a pilha p1 conterá todos os elementos das duas pilhas. Note que os elementos da pilha p2 não podem ser colocados de qualquer jeito na pilha p1, eles devem ter a mesma ordem que eles tinha quando estavam na pilha p2. Você pode usar memória adicional para resolver a questão.



A sua função deve obedecer o seguinte protótipo:

void concatena_pilhas(Stack *p1, Stack *p2);

Solution:

```
01. void concatena_pilhas(Stack *p1, Stack *p2) {
02.     Stack *newp = stack_create();
03.     while(!stack_empty(p2)) {
04.         stack_push(newp, stack_top(p2));
05.         stack_pop(p2);
06.     }
07.     while(!stack_empty(newp)) {
08.         stack_push(p1, stack_top(newp));
09.         stack_pop(newp);
10.     }
11.     stack_free(newp);
12. }
```

Boa Prova!