INSTITUTO FEDERAL
Brasília

Instituto Federal de Educação, Ciência e Tecnologia de Brasília – Câmpus Taguatinga
Ciência da Computação – Estruturas de Dados e Algoritmos
Prova II – 1°/2017 – Listas, Filas e Pilhas
Prof. Daniel Saad Nogueira Nunes

Aluno: _____
Matrícula: _____
Data: 21 de junho de 2017

Duração da prova: 150 minutos

Tabela de notas (uso exclusivo do professor)

Questão	Pontos	Nota
1	3	
2	3	
3	3	
4	3	
Total	12	

Observações

- Esta prova tem o total de 4 páginas (incluindo a capa) e 4 questões.
- O número total de pontos é 12.
- Certifique-se de assinar todas as folhas de resposta bem como a capa da prova.
- Leia atentamente todas as questões da prova. A interpretação do problema é crucial para o desenvolvimento correto da resposta.
- Resoluções sem justificativa não serão consideradas.
- É vedado o uso de equipamentos eletrônicos, como celulares, notebooks entre outros.
- A prova será **anulada** e medidas disciplinares serão tomadas para os alunos que "colarem" durante a avaliação.

⋆ Certifique-se de assinar todas as folhas de resposta.

Questão 1 (3 pontos)

De acordo com listas encadeadas simples:

(a) (1.5 pontos) Implemente a função `list_t* reverse(list_t* l)`, que recebe uma lista e retorna uma nova lista correspondendo a original invertida.

(b) (1.5 pontos) Implemente uma função que dada uma lista com dados do tipo `int*` e um parâmetro inteiro k, retorne verdadeiro se k está presente na lista e falso caso contrário. Sua função deve possuir a seguinte assinatura:

```
int find(list_t* l,int k);
```

Para esta questão, assuma que qualquer função básica sobre as listas está pronta e também assuma que os tipos a serem utilizados correspondem a:

```
typedef struct list_node_t{
    void* data; /*Ponteiro para um dado genérico de qualquer tipo*/
    struct list_node_t* next; /*ponteiro para o próximo elemento*/
}list_node_t;

typedef struct list_t{
    list_node_t* head; /*Cabeça da Lista*/
    list_node_t* tail; /*Cauda da Lista*/
    list_node_constructor_fn constructor; /*Função para construir o objeto*/
    list_node_destructor_fn destructor; /*Função para destruir o objeto*/
    size_t size; /*tamanho da lista*/
}list_t;
```

Questão 2 (3 pontos)

De acordo com listas **duplamente** encadeadas:

(a) (1.5 pontos) Implemente a função `dlist_append`, que adiciona um elemento na cauda da lista. Sua função deverá possuir a seguinte assinatura:

```
void dlist_append(dlist_t* l, void* data);
```

(b) (1.5 pontos) Implemente a função `dlist_remove_head`, que remove a cabeça da lista. Sua função deverá possuir a seguinte assinatura:

```
void dlist_remove_head(dlist_t* l);
```

Assuma para esta questão os seguintes tipos:

```
typedef struct dlist_node_t{
    void* data; /*Ponteiro para um dado genérico de qualquer tipo*/
    struct dlist_node_t* next; /*ponteiro para o próximo elemento*/
    struct dlist_node_t* prev; /*Ponteiro para o elemento anterior*/
}dlist_node_t;
```

⋆ Certifique-se de assinar todas as folhas de resposta.

```
typedef struct dlist_t{
     dlist_node_t* head; /*Cabeça da dlista*/
     dlist_node_t* tail; /*Cauda da dlista*/
     dlist_node_constructor_fn constructor; /*Função para construir o objeto*/
     dlist_node_destructor_fn destructor; /*Função para destruir o objeto*/
     size_t size; /*tamanho da dlista*/
}dlist_t;
```

Considere que as seguintes funções já estão implementadas:

```
size_t dlist_empty(dlist_t* l);
size_t dlist_size(dlist_t* l);
```

Questão 3 (3 pontos)

Considerando pilhas e filas:

(a) (1.5 pontos) Implemente a função `stack_push`, que insere um elemento no topo da pilha. Ela deverá possuir a seguinte assinatura:

```
void stack_push(stack_t* s, void* data);
```

(b) (1.5 pontos) Implemente a função `queue_pop`, que retira o elemento do início da fila. Ela deverá possuir a seguinte assinatura:

```
void queue_pop(queue_t* q);
```

Para esta questão, considere os seguintes tipos:

```
typedef struct queue_node_t{
    void* data; /*Ponteiro para um dado genérico de qualquer tipo*/
    struct queue_node_t* next; /*ponteiro para o próximo elemento*/
}queue_node_t;

typedef struct queue_t{
    queue_node_t* front; /*Frente da fila*/
    queue_node_t* back; /*Traseira da fila*/
    queue_node_constructor_fn constructor; /*Função para construir o objeto*/
    queue_node_destructor_fn destructor; /*Função para destruir o objeto*/
    size_t size; /*tamanho da pilha*/
}queue_t;

typedef struct stack_node_t{
    void* data; /*Ponteiro para um dado genérico de qualquer tipo*/
    struct stack_node_t* next; /*ponteiro para o próximo elemento*/
}stack_node_t;

typedef struct stack_t{
    stack_node_t* top; /*Topo da pilha*/
    stack_node_constructor_fn constructor; /*Função para construir o objeto*/
```

⋆ Certifique-se de assinar todas as folhas de resposta.

```
    stack_node_destructor_fn destructor; /*Função para destruir o objeto*/
    size_t size; /*tamanho da pilha*/
}stack_t;
```

Questão 4 (3 pontos)

Uma *string* possui a propriedade "semi-invertida" quando a metade da direita da *string* corresponde ao inverso da metade da esquerda. Por exemplo, a string:

```
abracadabraarbadacarba
```

é uma *string* "semi-invertida". Obviamente, para possuir esta propriedade, a *string* deve ter tamanho par.

Implemente um programa completo que leia uma *string* (máximo de 100 caracteres) e imprima **Sim**, caso ela seja "semi-reversa", e **Não**, caso contrário.

Obrigatoriamente deverão ser utilizadas uma fila e uma pilha para resolver este problema. No máximo uma leitura da esquerda para direita pode ser efetuada na *string* e uma única chamada à função `strlen` pode ser realizada.

Assuma que as funções sobre filas e pilhas estão prontas e os tipos das estruturas de dados são os mesmos da **Questão 3**.

OBS: não se esqueça de inicializar as estruturas e liberá-las após sua utilização.

> Computer Science is a science of abstraction — creating the right model for thinking about a problem and devising the appropriate mechanizable techniques to solve it
>
> A. Aho and J. Ullman

⋆ Certifique-se de assinar todas as folhas de resposta.