Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento Acadêmico de Informática (DAINF)
Estrutura de Dados I
Professor: Rodrigo Minetto
Lista de exercícios (seleção para revisão)

Exercício 1) Marque verdadeiro ou falso (cada resposta errada anula uma certa):

(**V**) $n^2 = \mathcal{O}(n^3)$

(**V**) $5 \times 2^n + n^6 - 10^{10} = \Theta(2^n)$

(**V**) $2n + 1 = \mathcal{O}(n^2)$

(**F**) $n\log n^3 = \Omega(2^n \log n)$

(**V**) $n^k = \mathcal{O}(c^n)$, com $k > 1$ e $c > 1$ constantes.

(**V**) $4^{\log n} = \Theta(n^2)$

(**V**) $\log n^2 = \mathcal{O}(\log n)$ ($\log_b(x^n) = n\log_b x$.)

(**V**) Se $f(n) = \mathcal{O}(g(n))$ e $g(n) = \mathcal{O}(h(n))$ então $f(n) = \mathcal{O}(h(n))$

(**F**) Seja $f(n) = n^3 + nlgn$ e $h(n) = nlgn$. Então, $f(n) \in \Theta(h(n))$

(**V**) $2^{n+1} = \mathcal{O}(2^n)$

$\star$ **Exercício 2**) Descreva a funcionalidade do seguinte fragmento de código (não faça re-leituras do código em português, o que se espera neste exercício é que você determine o que ele resolve):

```
Queue* misterio1 (Queue *q) {
  Stack *s = create ();
  while (!empty(q)) {
    push(s, dequeue(q));
  }
  while (!empty(s)) {
    enqueue(q, pop(s));
  }
  return q;
}
```

O que acontece com a fila ao ser retornada? Considere que dequeue e pop retornam um inteiro. Considere também que todas as operações atualizam a estrutura de dados da forma correta (não importando se elas são implementadas através de uma lista ou vetor).

Solução: Inverte os elementos da fila.

Exercício 3) Escreva uma função iterativa e uma função recursiva que recebe uma lista **simplesmente** encadeada como entrada e retorna o **maior** elemento armazenado. Por exemplo, se $\ell = \{5, 6, 7, 0, 9, -2, 3, 2, 1, 0\}$ então o retorno da função é 9. A função deve utilizar o seguinte protótipo:

```
typedef struct node {
  int data;
  struct node *next;
} List;
```

Dica: você pode usar uma função externa para comparar dois números, desde que ela seja também apresentada.

Solução iterativa:

```
int max_itr (List *l) {
  int m = -INT_MAX;
  List *t = l; /*temporary*/
  for (t = l; t != NULL; t = t->next) {
    if (t->data > m)
      m = t->data;
  }
  return m;
}
```

Solução recursiva:

```
int maximo (int a, int b) {
   return (a > b ? a : b);
}

int max_rec (List *l) {
  if (l == NULL)
    return -INT_MAX;
  else
    return maximo(l->data, max_rec(l->next));
}
```

Exercício 4) Escreva uma função recursiva que recebe uma lista **simplesmente encadeada** e um inteiro k como entrada e remove todas as ocorrências de k na lista. Por exemplo, se $\ell = \{1, 0, 1, 2, 1\}$ e $k = 1$, então após a remoção $\ell = \{0, 2\}$. A função deve utilizar o seguinte protótipo:

```
typedef struct node {
  int data;
  struct node *next;
} List;
```

Ps. se o elemento não existir então a lista deve permanecer inalterada. Teste os limites da lista para ter certeza que sua função funciona, por exemplo, remover a cabeça, meio e cauda da lista.

Solução:

```
List* remove_all (List *l, int k) {
  if (l != NULL) {
    if (l->data == k) {
      List *n = l;
      l = l->next;
      free(n);
      l = remove_all (l, k);
    }
    else
      l->next = remove_all (l->next, k);
    return l;
  }
  else {
    return NULL;
  }
}
```

Exercício 5) Projete uma estrutura de dados **pilha** baseada em uma lista com encadeamento duplo — considere uma lista com acesso direto a apenas a **cabeça** da lista. Precisamente, codifique as seguintes funções de forma iterativa:

Soluções:

```
typedef struct node {
  int data;
  struct node *next;
  struct node *prev;
} Stack;

Stack* push (Stack *l, int elem) {
  Stack *node = (Stack *)malloc(sizeof(Stack));
  node->data = elem;
  node->next = l;
  node->prev = NULL;
  if (l != NULL)
    l->prev = node;
  return node;
}

Stack* pop (Stack *l) {
  if (l != NULL) {
    Stack *tmp = l;
    if (l->next != NULL)
```

```
      l->next->prev = NULL;
    l = l->next;
    free(tmp);
    return l;
  }
}

int peek (Stack *l) {
  if (l != NULL)
    return l->data;
  else
    return ERROR;
}

int empty (Stack *l) {
  return (l == NULL);
}
```

Exercício 6) Mostre como implementar uma FILA através da utilização de UMA OU MAIS PILHAS. Ou seja, codifique as funções enqueue e dequeue com operações de alto nível de um TAD Pilha (push, pop, ...).

Soluções:

Solução aceita com 100% da nota:

```
void enqueue (Stack *s, int elem) {
  push (s, elem);
}

int dequeue (Stack *s) {
  Stack *tmp = create ();
  while (!empty(s)) {
    push(tmp, pop(s));
  }
  int elem = pop(tmp);
  while (!empty(tmp)) {
    push(s, pop(tmp));
  }
  return elem;
}
```

Solução correta:

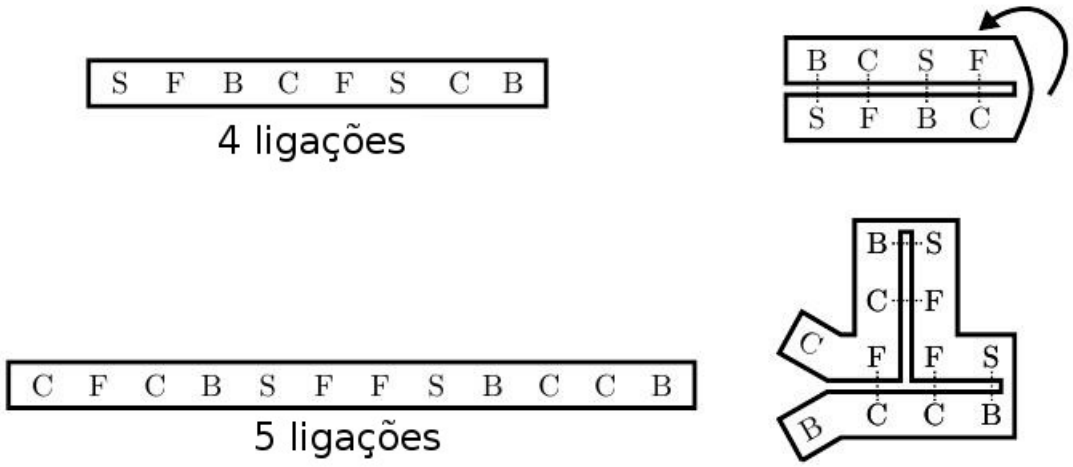
```
void enqueue (Stack **s, int elem) {
  push (s, elem);
}
```

```
int dequeue (Stack **s) {
  Stack *tmp = create ();
  while (!empty(*s)) {
    push(&tmp, pop(s));
  }
  int elem = pop(&tmp);
  while (!empty(tmp)) {
    push(s, pop(&tmp));
  }
  return elem;
}
```

Exercício 7) Foi descoberta uma espécie alienígena de ácido ribonucleico (popularmente conhecido como RNA). Os cientistas, por falta de criatividade, batizaram a descoberta de ácido ribonucleico alienígena (RNAA). Similar ao RNA que conhecemos, o RNAA é uma fita composta de várias bases. As bases são B, C, F, S e podem ligar-se em pares. Os únicos pares possíveis são entre as bases B e S e as bases C e F. Enquanto está ativo, o RNAA dobra vários intervalos da fita sobre si mesma, realizando ligações entre suas bases. Os cientistas perceberam que:

- quando um intervalo da fita de RNAA se dobra, todas as bases neste intervalo se ligam com suas bases correspondentes;
- cada base pode se ligar a apenas uma outra base e elas podem estar em posições consecutivas.
- as dobras ocorrem de forma a maximizar o número de ligações feitas sobre fitas.

A figura abaixo ilustra algumas dobras (mas elas não são únicas, ou seja podem ter dobras diferentes com o mesmo número de ligações).



Escreva um programa em linguagem C para determinar quantas ligações serão realizadas entre as bases se a tira de RNAA ficar ativa. É obrigatório usar um TAD, que faça diferença e seja útil, na solução do exercício. Você não precisa implementar as funções do TAD, basta usar e explicar o motivo.

Solução: uma maneira bem simples de implementar esta solução é utilizando uma estrutura de dados pilha. A cada base que for inserida na pilha, verifica se esta pode

ligar-se com a base do topo da pilha, que em caso positivo, não insere a letra na pilha e ainda remove a base do topo responsável pela ligação. O número de pares removidos é o máximo de ligações que podem ser realizadas.

```
int main ( ) {

  Stack* pilha = create ();

  char *string = "SFBCFSCB";

  int i = 0, ligacoes = 0;
  while (string[i] != '\0') {
    if ( (string[i] == 'B') && (top(pilha) == 'S') ) {
      pop (pilha);
      ligacoes++;
    }
    else if ( (string[i] == 'S') && (top(pilha) == 'B') ) {
      pop (pilha);
      ligacoes++;
    }
    else if ( (string[i] == 'C') && (top(pilha) == 'F') ) {
      pop (pilha);
      ligacoes++;
    }
    else if ( (string[i] == 'F') && (top(pilha) == 'C') ) {
      pop (pilha);
      ligacoes++;
    }
    else {
      push (pilha, string[i]);
    }
    i++;
  }
  printf("Saída: %d\n", ligacoes);
  return 0;
}
```