Prova I

Estrutura de Dados

Professor Eduardo Takeo Ueda
Senac Santo Amaro

Yuri Jorge Carrião Otofuji

01/10/2021

Conteúdo

1. **Implemente (em linguagem C) um algoritmo recursivo que inverta a ordem dos elementos de um vetor de inteiros.**

```c
#include <stdio.h>

void invert(int v[], int s) {

        if (s == 0 || s == 1) return;

        int t = v[0];
        v[0] = v[s-1];
        v[s-1] = t;

        invert(&v[1], s-2);

}

int main() {

        int v[] = { 0, 1, 2, 3, 4, 5 };
        int s = sizeof(v) / sizeof(int);

        for (int i = 0; i < s; i++) printf("%d ", v[i]);
        printf("\n");

        invert(v, s);

        for (int i = 0; i < s; i++) printf("%d ", v[i]);
        printf("\n");

}
```

2. Usando a definição de notação O, prove que $log_{10}n$ é $O(lg(n))$. (Lembre-se que $lg(n)$ denota o logaritmo de n na base 2).

$$\begin{aligned} log(n) &\leq clg(n) \\ log(n) &\leq c\frac{log(n)}{log(2)} \\ n_0 = 2 \quad & c = log(2) \end{aligned}$$

3. Implemente (em linguagem C) uma função que busca/procura, recursivamente, um valor em uma lista simplesmente encadeada/ligada.

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


typedef struct key {
        int value;
        struct key* next;
} KEY;

typedef struct list {
        KEY* head;
} LIST;


void init(LIST* l) {
        l -> head = malloc(sizeof(KEY));
        l -> head = NULL;
}

void show(LIST* l) {
        printf("List: ");
        KEY* k = l -> head;
        while (k != NULL) {
                printf("%d ", k -> value);
                k = k -> next;
        }
        printf("\n");
}

void nsrt(LIST* l, int i) {

        KEY* a = malloc(sizeof(KEY));
        a -> value = i;
        a -> next = NULL;

        KEY* k = l -> head;
        if (k == NULL) {
                l -> head = a;
                return;
        }

        while (k -> next != NULL) k = k -> next;
        k -> next = a;
```

```
}

KEY* find(LIST* l, int v) {
        KEY* k = l -> head;
        do {
                if (k -> value == v) return k;
                k = k -> next;
        } while (k != l -> head);
        return NULL;
}

int fpos(LIST* l, int v) {
        int i = 0;
        KEY* k = l -> head;
        do {
                if (k -> value == v) return i;
                k = k -> next;
                i++;
        } while (k != l -> head);
        return -1;
}


int main() {

        LIST list;
        init(&list);

        show(&list);

        nsrt(&list, 0);
        nsrt(&list, 1);
        nsrt(&list, 2);
        nsrt(&list, 3);
        nsrt(&list, 4);
        nsrt(&list, 5);

        show(&list);

        printf("O valor %d na lista está na posição: %d\n",
                find(&list, 4) -> value, fpos(&list, 4));

}
```

4. **Implemente (em linguagem C), utilizando uma pilha, uma função que recebe uma sequência de caracteres e verifica se ela é um palíndromo, ou seja, se a *string* é escrita da mesma maneira de frente para trás e de trás para frente (ignore espaços e pontos).**

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


typedef struct item {
        char c;
        struct item* down;
} ITEM;

typedef struct stack {
        ITEM* top;
} STACK;


void init(STACK* s) {
        s -> top = malloc(sizeof(ITEM));
        s -> top = NULL;
}

void put(STACK* s, char c) {
        ITEM* i = malloc(sizeof(ITEM));
        i -> c = c;
        i -> down = s -> top;
        s -> top = i;
}

ITEM* pop(STACK* s) {
        ITEM* i = s -> top;
        s -> top = s -> top -> down;
        return i;
}

char srev(ITEM* i) {
        if (i -> down == NULL) {
                return i -> c;
        } else {
                printf("%c ", srev(i -> down));
                return i -> c;
        }
}
```

```
void show(STACK* s, int rev) {
        ITEM* i = s -> top;
        if (i == NULL) {
                printf("\n");
                return;
        }
        if (rev) {
                printf("%c ", srev(s -> top));
        } else {
                while (i != NULL) {
                        printf("%c ", i -> c);
                        i = i -> down;
                }
        }
        printf("\n");
}

STACK invert(STACK* s) {

        STACK stack;
        init(&stack);

        while (s -> top != NULL) put(&stack, pop(s) -> c);
        return stack;

}

typedef struct queue {
        STACK* frnt;
        STACK* back;
} QUEUE;

void qinit(QUEUE* q) {
        q -> frnt = malloc(sizeof(STACK));
        q -> back = malloc(sizeof(STACK));
}

void qshow(QUEUE* q) {
        printf("Front: ");
        show(q -> frnt, 0);
        printf("Back: ");
        show(q -> back, 0);
}

void enqueue(QUEUE* q, char c) {
```

```
        if (q -> frnt -> top == NULL) {
                put(q -> frnt, c);
                put(q -> back, c);
                return;
        }

        put(q -> back, c);

        STACK s;
        init(&s);

        s = *(q -> back);
        *(q -> frnt) = invert(&s);

}

void dequeue(QUEUE* q) {

        if (q -> frnt -> top == NULL) return;

        pop(q -> frnt);

        STACK s;
        init(&s);

        s = *(q -> frnt);
        *(q -> back) = invert(&s);

}

int main() {

        QUEUE queue;
        qinit(&queue);

        enqueue(&queue, 'a');
        enqueue(&queue, 'b');
        enqueue(&queue, 'c');

        qshow(&queue);
        dequeue(&queue);
        qshow(&queue);

        return 0;

}
```

5. **Implemente (em linguagem C) uma fila utilizando duas pilhas. A fila deve realizar as operações usuais de enfileirar (*enqueue*) e desenfileirar (*dequeue*) elementos, assim como as pilhas devem efetuar as operações de empilhar (*push*) e desempilhar (*pop*).**

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


typedef struct item {
        char c;
        struct item* down;
} ITEM;

typedef struct stack {
        ITEM* top;
} STACK;


void init(STACK* s) {
        s -> top = malloc(sizeof(ITEM));
        s -> top = NULL;
}

void put(STACK* s, char c) {
        ITEM* i = malloc(sizeof(ITEM));
        i -> c = c;
        i -> down = s -> top;
        s -> top = i;
}

ITEM* pop(STACK* s) {
        ITEM* i = s -> top;
        s -> top = s -> top -> down;
        return i;
}

char srev(ITEM* i) {
        if (i -> down == NULL) {
                return i -> c;
        } else {
                printf("%c ", srev(i -> down));
                return i -> c;
        }
}
```

```
void show(STACK* s, int rev) {
        ITEM* i = s -> top;
        if (i == NULL) {
                printf("\n");
                return;
        }
        if (rev) {
                printf("%c ", srev(s -> top));
        } else {
                while (i != NULL) {
                        printf("%c ", i -> c);
                        i = i -> down;
                }
        }
        printf("\n");
}

STACK invert(STACK* s) {

        STACK stack;
        init(&stack);

        while (s -> top != NULL) put(&stack, pop(s) -> c);
        return stack;

}

typedef struct queue {
        STACK* frnt;
        STACK* back;
} QUEUE;

void qinit(QUEUE* q) {
        q -> frnt = malloc(sizeof(STACK));
        q -> back = malloc(sizeof(STACK));
}

void qshow(QUEUE* q) {
        printf("Front: ");
        show(q -> frnt, 0);
        printf("Back: ");
        show(q -> back, 0);
}

void enqueue(QUEUE* q, char c) {
```

```
        if (q -> frnt -> top == NULL) {
                put(q -> frnt, c);
                put(q -> back, c);
                return;
        }

        put(q -> back, c);

        STACK s;
        init(&s);

        s = *(q -> back);
        *(q -> frnt) = invert(&s);

}

void dequeue(QUEUE* q) {

        if (q -> frnt -> top == NULL) return;

        pop(q -> frnt);

        STACK s;
        init(&s);

        s = *(q -> frnt);
        *(q -> back) = invert(&s);

}

int main() {

        QUEUE queue;
        qinit(&queue);

        enqueue(&queue, 'a');
        enqueue(&queue, 'b');
        enqueue(&queue, 'c');

        qshow(&queue);
        dequeue(&queue);
        qshow(&queue);

        return 0;

}
```