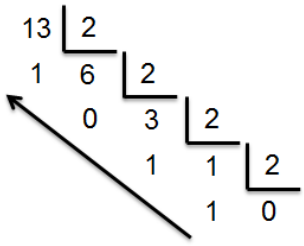
Universidade do Estado de Santa Catarina (UDESC)
Curso: Tecnologia em Análise e Desenvolvimento de Sistemas
Disciplina: Estruturas de Dados (EDA0001)
Prof. Rui J. Tramontin Jr.

Lista de Exercícios 3 (Pilhas)

1) Escreva um programa em C que converta um número decimal para binário. Dado um número inteiro n, a conversão é feita de seguinte forma: divida sucessivamente n por 2, guardando os restos das divisões, até obter um quociente igual a 0. Os restos dessas divisões representam os dígitos (0 ou 1) do número binário gerado. No entanto, tais dígitos são gerados na ordem inversa (do menos para o mais significativo). Portanto, o programa deve usar uma pilha para guardar os restos das divisões que, quando desempilhados e mostrados na tela, representam corretamente o número binário gerado.

   Exemplo:



$(13)_{10} \rightarrow (1101)_2$

2) Implemente uma função que receba duas pilhas *p1* e *p2*, e passe todos os elementos de *p2* para o topo de *p1*. Essa função deve obedecer ao protótipo:

```
void concatena( Pilha *p1, Pilha *p2 );
```

3) Escreva uma função (em nível de aplicação, ou seja, no arquivo principal do programa) que utiliza uma pilha para inverter um *string*. Para isso, basta empilhar todos os seus caracteres em uma pilha e, em seguida, desempilhar de volta na *string*.

4) Uma palavra é dita **palíndrome** se a sequência de letras que a forma é a mesma, seja ela lida da esquerda para a direita ou vice-versa. Exemplos: *arara*, *rairar*, *hanah*. Escreva uma função (definida no mesmo arquivo do programa principal) que indique se uma dada *string* é ou não palíndrome. Para isso, é preciso inverter a *string* empilhando todos os seus caracteres em uma pilha e, em seguida, desempilhando-os numa nova *string*. Por fim, basta comparar a *string* invertida com a original. Protótipo da função:

```
int eh_palindrome( char palavra[] );
```

5) Escreva uma função que inverta a ordem das letras de cada palavra de uma *string*, preservando a ordem das palavras. Suponha que as palavras da *string* são separadas por espaços. A aplicação da operação à *string* "AMU MEGASNEM ATERCES", por exemplo, deve produzir "UMA MENSAGEM SECRETA".

6) Faça um programa que verifique se uma expressão aritmética contém os parênteses organizados aos pares (utilizando apenas as operações da pilha). O programa lê uma *string*, por exemplo `"((2 + 4) * 6) / 3"`, e segue o seguinte algoritmo:

```
Para cada caractere lido na expressão:
  Se encontrar um abre parênteses:
    empilhe-o;
  Se encontrar um fecha parênteses:
    Se pilha estiver vazia:
      imprima erro: "Fecha parênteses sem abre parênteses (posição i)." ... fim do programa.
    Senão:
      desempilhe o abre parênteses já empilhado;
Ao final da leitura da expressão:
  Se pilha não é vazia, imprima erro: "Há parênteses abertos que não foram fechados."
  Senão, expressão OK!
```

7) É possível definir um TAD que armazene **duas pilhas** num único vetor. A pilha 1 começa na posição 0 e a pilha 2 começa na posição *capacidade*-1 (onde *capacidade* é o tamanho do vetor). São necessários dois índices para o topo, sendo que para a pilha 2 ele deve ser decrementado sempre que um novo valor for empilhado. No exemplo a seguir, assumindo um vetor de inteiros e que *capacidade* = 12, temos o topo da pilha 1 no índice 4 e o topo da pilha 2 no índice 8:

4	3	9	27	81	-	-	-	7	25	36	8
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Dadas as definições abaixo de uma **pilha dupla**, implemente as funções *inicializa*, *pilha1_vazia*, *pilha2_vazia*, *pilhas_cheias*, *empilha1*, *empilha2*, *desempilha1* e *desempilha2*.

```
typedef struct{
    int *dados;
    int topo1, topo2;
    int capacidade;
} PilhaDupla;
```