Teste de Programação Imperativa

LCC/MIEF/MIEI

30 de Maio de 2017

Parte A

Considere as seguintes definições de tipos:

```
typedef struct slist {
   int valor;
   struct slist *prox;
} *LInt;
```

```
typedef struct nodo {
   int valor;
   struct nodo *esq, *dir;
} *ABin;
```

1. Defina uma função `int limpaEspacos (char t[])` que elimina repetições sucessivas de espaços por um único espaço. A função deve retornar o comprimento da string resultante.

2. Defina uma função `void transposta (int N, float m [N][N])` que transforma uma matriz na sua transposta.

3. Apresente uma definição da função `LInt cloneL (LInt)` que cria uma nova lista ligada com os elementos pela ordem em que aparecem na lista argumento.

4. Defina uma função `int nivelV (ABin a, int n, int v[])` que preenche o vector `v` com os elementos de `a` que se encontram no nível `n`.

   Considere que a raíz da árvore se encontra no nível 1.

   A função deverá retornar o número de posições preenchidas do array.

5. Defina uma função `void removeMaiorA (ABin *)` que remove o maior elemento de uma árvore binária de procura.

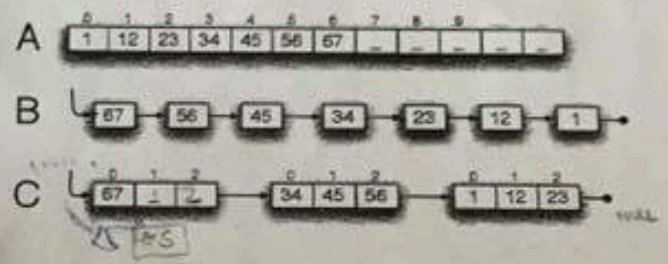
Parte B

Uma *stack* é um caso particular de um *buffer* em que os elementos são removidos pela ordem inversa da que foram inseridos (LIFO). As alternativas habituais de implementação de *stacks* são:

- (A) Usar um *array* e um inteiro (que indica quantos elementos o *array* contém) inserindo cada novo elemento no final do *array*.
- (B) Usar uma lista ligada onde as inserções (e remoções) são feitas no início da lista.

Uma forma de combinar as vantagens destas duas alternativas consiste em:

- (C) usar uma lista ligada de *arrays* (todos do mesmo tamanho – `MAXc`). A inserção de um novo elemento faz-se no final do primeiro *array* da lista se ainda não estiver cheio. Quando este se encontra cheio, é criado um novo *array* que é inserido no início da lista.



Na figura mostram-se as três alternativas (A, B e C) que representam a *stack* (de inteiros) que resulta de, na *stack* vazia se acrescentarem os elementos 1, 12, 23, 34, 45, 56 e 67, por esta ordem. Note que nesta última representação apenas temos que guardar o número de elementos do primeiro dos *arrays* da lista uma vez que todos os outros se encontram completos. Por isso definimos a `struct StackC` que guarda este número, juntamente com a lista dos *arrays*.

Considere o tipo `stackC` definido ao lado e defina as seguintes funções.

```
typedef struct chunk {
   int vs [MAXc];
   struct chunk *prox;
} *CList;
typedef struct stackC {
   CList valores;
   int sp;
} StackC;
```

1. `int push (StackC *s, int x)` que acrescenta um elemento `x` a `s`. Retorna 0 em caso de sucesso.
2. `int pop (StackC *s, int *x)` que remove o elemento do topo da stack, colocando-o em `*x`. Retorna 0 em caso de sucesso.
3. `int size(Stack s)` que calcula o comprimento (número de elementos) de `s`.
4. Usando as funções `push` e `pop` acima defina a função `void reverse (Stack *s)` que inverte a ordem dos elementos de `s`.

   Apresente o resultado de aplicar essa função à stack apresentada como exemplo.
5. Apresente uma definição alternativa da função `reverse` da alínea anterior que reutiliza as células da lista, i.e., que não faz quaisquer `push` (`malloc`) ou `pop` (`free`).