UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
Bacharelado em Ciência da Computação e
Engenharia da Computação
INF 01203 – Estruturas de Dados
Profa. Renata Galante (galante@inf.ufrgs.br)

01. ***(Revisão Ponteiros)*** Considere a estrutura de dados lista duplamente encadeada **NÃO** circular conforme descrição a seguir. Para resolver esta questão, siga a ordem de chamadas de funções da função `main()` que está na próxima página.

> ***Atenção:*** o código desta questão **NÃO é** um exemplo de boas práticas de programação. O código foi construído apenas para estudar o uso de ponteiros. ***Não tente repetir em casa ou no trabalho!!!*** ☺

```
Struct PtNo {
      char palavra[20];
      PtNo *ant;
      PtNo *prox;
      };

PtNo* insere(PtNo *PtLista, char Dado)
{ PtNo *Pt;
  Pt = (PtNo*) malloc(sizeof(PtNo));
  Pt->palavra = Dado;
  Pt->ant = NULL;
  Pt->prox = PtLista;
  if (PtLista != NULL)
     PtLista->ant = Pt;
  PtLista = Pt;
  return PtLista;}

PtNo*  confusao_ponteiros (PtNo *PtLista, char Dado)
{   PtNo   *novo,*aux1,*aux2,*aux3,*aux4,*aux5,*aux6,*aux7,*aux8,
*aux9;
  novo = (PtNo*) malloc(sizeof(PtNo));
  novo->palavra = Dado;
  aux1 = PtLista->prox;
  aux2 = aux1->prox;
  aux3 = aux2->prox;
  aux4 = aux3->prox;
  aux5 = aux4->prox;
  aux6 = aux3->ant;
  aux7 = aux3->prox;
// Responda a questão (B).
  PtLista = aux2->ant;
  aux2->prox = novo;
  novo->ant = aux6;
// Responda a questão (C).
  aux4->ant = novo;
  novo->prox = aux7;
  free(aux3);
  aux8 = novo;
// Responda a questão (D).
  PtLista = aux1->ant;
  return PtLista;}
```

```
int main{
   PtNo *lis;
   lis = NULL;
   lis = insere(lis, "divertido");
   lis = insere(lis, "pouco");
   lis = insere(lis, "é");
   lis = insere(lis, "ponteiros");
   lis = insere(lis, "estudar");
// Responda a questão (A).
   lis = confusao_ponteiros(lis, "muito");
// Responda a questão (E).
   return(0);}
```

(A) O que será exibido na tela se executarmos uma função para exibir o campo `palavra` de todos os elementos da lista iniciando no ponteiro `lis` até o final da lista? Se der erro, indicar o ponto da lista em que a aplicação foi abortada.

(B) Qual valor será impresso na tela se executarmos um `printf` no campo `palavra` de cada um dos seguintes ponteiros? Se der erro, indicar se é lixo ou NULL.

`aux1` _____ `aux4`_____ `aux7`_____

`aux2` _____ `aux5`_____ `aux8`_____

`aux3` _____ `aux6` _____ `aux9` _____

(C) O que será exibido na tela se executarmos uma função para exibir o campo `palavra` de todos os elementos da lista iniciando no ponteiro `lis` até o final da lista? ? Se der erro, indicar o ponto da lista em que a aplicação foi abortada.

(D) Qual valor será impresso na tela se executarmos um `printf` no campo `palavra` de cada um dos seguintes ponteiros?

`aux1` _____ `aux4`_____ `aux7`_____

`aux2` _____ `aux5`_____ `aux8`_____

`aux3` _____ `aux6` _____ `aux9` _____

(E) O que será exibido na tela se executarmos uma função para exibir o campo `palavra` de todos os elementos da lista iniciando no ponteiro `lis` até o final da lista? ? Se der erro, indicar o ponto da lista em que a aplicação foi abortada.

02. Assuma que uma lista ***simplesmente encadeada circular*** é constituída de elementos do tipo `PtNodo` com dois campos, `valor` e `prox`, sendo que o campo `prox` é usado para apontar para o próximo nó da lista e o campo valor para guardar a informação de cada nó da lista. A operação `insereNoMeioDaLista(PtNodo* p, TipoInfo info, int pos)` insere um novo nó com o conteúdo `info` na posição indicada em `pos`, sendo que `p` representa o primeiro nó da lista. Observe o extrato de código abaixo, representando a criação de um novo nó. **<trecho de código>** executa a inserção do novo nó no meio da lista (isto é, para este exercício o nó a ser inserido não é primeiro nem o último), encadeando-o corretamente com os demais de forma que o resultado seja uma ***lista simplesmente encadeada circular***.

```
PtNodo* insereNoMeioDaLista(PtNodo* p, TipoInfo info, int pos)
{     PtNodo* novo;
      novo = (PtNodo*) malloc(sizeof(PtNodo));
      novo->valor = info;
      if (p != NULL)
      {  <trecho de código> }
      else ...
      ....}
```

Assinale a implementação em C correta para <**trecho de código**> de acordo com o comportamento acima descrito.

(a)
```
int i = 1;
PtNodo* paux;
while (i < (pos-1)) {
      paux = p->prox;
      paux->prox = novo;
      i++;}
novo->prox = paux;
```

(b)
```
int i = 1;
PtNodo* paux;
while (i < pos) {
      paux = p->prox;
      i++;}
paux->prox = novo;
novo->prox = paux->prox->prox;
```

(c)
```
int i = 1;
PtNodo* paux;
paux = p;
while (i < (pos-1)) {
      paux = paux->prox;
      i++;}
novo->prox = paux->prox;
paux->prox = novo;
```

03. Assuma que uma ***lista duplamente encadeada circular*** é constituída de elementos do tipo `PtNodo` com três campos, `valor`, `ant` e `prox`, sendo que o campo `prox` é usado para apontar para o próximo nó da lista, o campo `ant` é usado para apontar para o nó anterior da lista, e o campo `valor` para guardar a informação de cada nó da lista. A operação `insereNoFimDaLista(PtNodo* p, TipoInfo info)` insere um novo nó com o conteúdo `info` no final da lista, onde `p` representa o primeiro nó da lista. Observe o extrato de código a seguir, representando a criação de um novo nó, e sua inserção em uma lista não vazia. <**trecho de código**> executa a inserção do novo nó no fim da lista, encadeando-o corretamente com os demais de forma que o resultado seja uma ***lista duplamente encadeada circular***.

```
PtNodo* insereNoFimDaLista(PtNodo* p, TipoInfo info)
{     PtNodo* novo;
      novo = (PtNodo*)malloc(sizeof(PtNodo));
      novo->valor = info;
      if (p != NULL)
      {  <trecho de código> }
      else …..
}
```

Assinale a implementação em C correta para <**trecho de código**> de acordo com o comportamento acima descrito.

(a)
```
p->ant = novo;
novo->ant = p;
p->ant->prox = novo;
novo->prox = p;
```

(b)
```
novo->prox = NULL;
p->ant->prox = novo;
novo->ant = p;
p->ant->prox = novo;
```

(c)
```
p->ant->prox = novo;
novo->ant = p->ant;
p->ant = novo;
novo->prox = p;
```

(d)
```
novo->prox = p;
novo->ant = p->ant->ant;
p->ant->prox = novo;
p->ant = novo;
```

04. Escreva uma função em C que encontra o elemento do meio de uma lista duplamente encadeada circular **sem utilizar contadores**. Caso a lista tenha um número par de elementos, qualquer dos dois elementos do meio podem ser retornados. Note que a lista possui apenas um ponteiro `ptLista` que aponta para o primeiro elemento da lista. A função deve retornar o conteúdo do campo de informação do nó.

Exemplos: Entrada: 1 2 3 4 5 Saída esperada: 3
Entradas: 1 2 3 4 5 6 Saída esperada: 3 ou 4

05. O objetivo do jogo troca letras é conseguir descobrir o mais rapidamente possível a palavra cujas letras foram misturadas. Considere que uma palavra é representada por uma lista duplamente encadeada circular (veja a estrutura de dados abaixo), na qual cada letra da palavra fica armazenada em um nó da lista. A função `troca_letras` recebe como entrada dois ponteiros (`letra1` e `letra2`), cada um deles apontando respectivamente para uma letra distinta da lista e não consecutiva, e troca as duas letras de posição. Isto significa que após a troca, a `letra1` fica na posição da `letra2` e a `letra2` fica na posição da `letra1`. A função `troca_letras` troca as duas letras de posição, simplesmente alterando os ponteiros da lista, ou seja, não há alocação nem liberação de memória (apenas troca de ponteiros). Ordene (numere) as linhas do trecho de código a seguir de forma que a troca de posição entre as duas letras ocorra corretamente.

Considere o seguinte exemplo para resolver a questão. A palavra correta é T **R** O **C** A. Porém, a palavra está escrita T **C** O **R** A. As letras C e R serão trocadas de posição.

```
Struct PtNo {
     char info;
     PtNo *ant;
     PtNo *prox;
     };
```

```
          ...
_______   letra2->ant = aux1;
_______   aux1 = letra1->ant;
_______   aux2 = letra2->prox;
_______   letra2->ant->prox = letra1;
_______   letra1->prox = aux2;
_______   letra1->prox->ant = letra2;
_______   letra1->ant = letra2->ant;
_______   aux1->prox = letra2;
_______   aux2->ant = letra1;
_______   letra2->prox = letra1->prox;
          ...
```