Lista de exercícios
– Listas encadeadas –

1. A função a seguir possui como tarefa verificar se uma lista é ou não acíclica. Porém, há um caso particular onde o código falha. Encontre-o e o descreva.

```
bool tst_lst_aciclica(list* start)
{
    for (list* p = start; p != NULL; p = p->next)
    {
        if (p == NULL) return true;

        for (list* q = start; q != p; q = q->next)
            if (q == p->next) return false;
    }

    return true;
}
```

2. O que será impresso pelo programa a seguir?

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

typedef struct node* link;
struct node {
    int key;
    link next;
};

int main(void) {
  link x, y, t;
  x = malloc(sizeof *x);
  y = malloc(sizeof *y);
  x->next = y; x->key = 1;
  y->next = x; y->key = 1;

  for (t = x; t->key < 100; t = t->next)
    t->key = x->key + y->key;

  printf("%d\n", t->key);
  return 0;
}
```

Para os exercícios a seguir, indique qual a funcionalidade do programa.

3.

```c
int exer3(struct node* head, int i) {
  struct node* c = head;
  int j = 0;

  while (c != NULL) {
    if (c->data == i) j++;
      c = c->next;
  }

  return j;
}
```

4.

```c
int exer4(struct node* head, int index) {
  struct node* current = head;
  int count = 0;

  while (current != NULL) {
    if (count == index)
      return(current->data);

    count++;
    current = current->next;
  }
}
```

5.

```c
int exer5(struct node** headRef) {
  struct node* head;
  int result;

  head = *headRef;
  assert(head != NULL);
  result = head->data;
  *headRef = head->next;

  free(head);
  return(result);
}
```

Nos exercícios a seguir, preencha as lacunas.

6.

```c
void deletar_lista(struct node** headRef) {
  struct node* current = --------;
  struct node* next;

  while (current != NULL) {
    next = --------;
    free(current);
    current = next;
  }

  *headRef = NULL;
}
```

7.

```
void inserir_n(struct node** headRef, int index, int data) {
  if (index == 0)
    ______(headRef, data);
  else {
    struct node* current = *headRef;
    int i;

    for (i=0; i < index-1; i++) {
      assert(current != NULL);
      current = ______;
    }

    assert(current != NULL);
    ______(&(current->next), data);
  }
}
```

8.

```
void acrescentar(struct node** aRef, struct node** bRef) {
  struct node* current;

  if (*aRef == NULL) {
    *aRef = *bRef;
  }
  else {
    current = ______;

    while (current->next != NULL) {
      current = current->next;
    }

    ______ = *bRef;
  }

  ______ = NULL;
}
```

Nos exercícios a seguir, corrija os erros.

9.

```
void mover_node(struct node** destRef, struct node** srcRef) {
  struct node* newNode = *srcRef;
  assert(newNode != NULL);

  *srcRef = newNode->next;
  newNode = *destRef;
  *destRef = newNode;
}
```

10.

```
void inverter(struct node** headRef) {
  struct node* result = NULL;
  struct node* current = *headRef;
  struct node* next;

  while (current != NULL) {
    next = current;
    current = result;
    result = current;
    current = next;
  }

  *headRef = result;
}
```

11.

```
void inverter_recursivamente(struct node** headRef) {
  struct node* first;
  struct node* rest;

  if (*headRef == NULL)
    return;

  first = *headRef;
  rest = first;

  if (rest == NULL)
    return;

  inverter_recursivamente(rest);

  first->next = first;
  first->next = NULL;
  *headRef = rest;
}
```

Respostas:
Exercício 1:
O código falha quando a lista de entrada é um auto-loop (laço), isto é, quando start->next = start.

Exercício 2:
144

Exercício 3:
Retorna a quantidade de termos iguais ao **i** dentro da lista.

Exercício 4:
Retorna o valor do enésimo termo da lista.

Exercício 5:
Função Pop: retorna o valor do termo no topo (cabeça - head) e remove-o da lista.

Exercício 6:
Linha 2: *headRef
Linha 6: current->next

Exercício 7:
Linha 3: Push
Linha 10: current->next
Linha 14: Push

Exercício 8:
Linha 8: *aRef
Linha 14: current->next
Linha 17: *bRef

Exercício 9:
Linha 6: newNode->next = *destRef;

Exercício 10:
Linha 7: next = current->next;
Linha 8: current->next = result;

Exercício 11:
Linha 9: rest = first->next;
Linha 14: inverter_recursivamente(&rest);
Linha 16: first->next->next = first;