UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Avaliação: **Simulado 1** Data: / / Curso: _____

Disciplina: **Estrutura de Dados** Professor: **Adriano Rivolli**

Nome: _____ R.A.: _____ Nota _____

Obs: Não serão aceitas questões parcialmente corretas, em termos de lógica, apenas de sintaxe.

1) (1.5) Considerando o código a seguir relativo as operações padrões de pilha e fila, Informe a sequência numérica que deverá ser impresso na tela.

```
int main() {
  int i;
  stack s;
  queue q;

  newStack(&s);
  newQueue(&q);

  for (i=1; i<4; i++) {
    push(&s, i+i);
    enqueue(&q, i*i);
  }

  enqueue(&q, pop(&s));
  push(&s, dequeue(&q));

  printf(" %d %d", pop(&s), dequeue(&q));
  printf(" %d %d", dequeue(&q), pop(&s));
}
```

R: 1 4 9 4

2) (1.5) Escreva a função PUSH para implementar uma pilha em vetor estático, conforme a estrutura de dados a seguir. Considere que a propriedade top é controlada na primeira posição (índice 0) do vetor VET e que TOP aponta para a próxima posição a ser inserida na pilha. Caso necessite de uma função adicional, você também deverá implementá-la.

```
typedef struct {
   int vet[N];
} stack;

void push(stack *s, int value);
```

R:

```
void push(stack *s, int value) {
   if (s->vet[0] == N) {
     return;
   }
   s->vet[s->vet[0]] = value;
   s->vet[0]++;
}
```

3) (1.5) Escreva o método DEQUEUE de uma fila utilizando a estrutura dinâmica a seguir. Considere que se head é nulo a fila está vazia e se a fila está vazia, retorne o '\0'.

```
typedef struct NodeList {
  char info;
  struct NodeList *prox;
} node;
```

```
typedef struct {
  node *head;
  node *tail;
} queue;
```

```
char dequeue(queue *f);
```

R:

```
char dequeue(queue *f) {
  node *n;
  char aux;

  if (f->head == NULL) {
    return '\0';
  }

  n = f->head;
  f->head = n->prox;
  aux = n->info;
  free(n);

  return aux;
}
```

4) (1.5) Escreva o método para encontrar o penúltimo elemento de uma lista simplesmente encadeada COM CABEÇA. Considere a estrutura 'node' apresentada na atividade anterior. Caso a lista só possua um único elemento, retorno NULO.

```
node *penultimoElemento(node *list);
```

R:

```
node *penultimoElemento(node *list) {
  node *ant = NULL;

  if (list->prox == NULL) {
    return NULL;
  }

  list = list->prox;
  while(list->prox != NULL) {
     ant = list;
     list = list->prox;
  }

  return ant;
}
```

5) (2.0) Considerando uma lista duplamente encadeada, implemente uma função que receba 2 nós como parâmetro e teste se estes nós são vizinhos ou possuem um vizinho em comum. Quando positivo, retorne 1, caso o contrário retorne 0;

```
int vizinho(node* l1, node *l2);
```

R:

```
int vizinho(node* l1, node *l2) {
  return l1->prox == l2 || l2->prox == l1 ||
     (l1->prox == l2->ant && l1->prox != NULL) ||
     (l2->prox == l1->ant && l2->prox != NULL);
}
```

6) (2.0) Dada uma pilha de inteiros formada por uma sequência de números positivos, construa uma função que remova da pilha os valores múltiplos de 3 e mantenha o restante na sua respectiva ordem.

Exemplo:

==> Empilha os elementos: 3 4 5 6 7 9
==> chama a função desenvolvida
==> Desempilha: 7 5 4

Considere que as seguintes funções já estão implementadas:

```
void newStack(stack *s);
int isEmpty(stack *s);
void push(stack *s, int value);
int pop(stack *s);
int top(stack *s);
```

R:

```
void removeMultiplo(stack *p) {
  stack aux;
  newStack(&aux);

  while(!isEmpty(p)) {
    if (top(p) % 3) {
      push(&aux, pop(p));
    } else {
      pop(p);
    }
  }

  while(!isEmpty(&aux)) {
    push(p, pop(&aux));
  }
}
```