Bach. em Ciência da Computação 3º Período 12/05/2016
Estrutura de Dados Prof. Hamilton J. Brumatto
Prova 2 Nome:GABARITO Nº: _____

Instruções:
(a) A prova tem a duração de 90 minutos;
(b) A prova é individual sendo proibida qualquer consulta ou o uso de qualquer meio de comunicação;
(c) A interpretação do enunciado é parte integrante da prova;
(d) O total de pontos é proporcional à nota, sendo que a nota equivalente à totalidade de pontos definidos na prova não é menor que 10;
(e) INCLUA O RACIOCÍNIO (ou contas) para chegar à resposta.
(f) Pode indicar o uso de algoritmos de apoio vistos em sala.
(g) Na resposta de uma questão pode ser considerado que a função criada na questão anterior esteja correta.

Questão:	Max	Pontos
Q1	20	
Q2	20	
Q3	20	
Q4	20	
Q5	20	
Total	100	
Nota		

Boa Prova!
Considere nesta prova que a entrada/saída de dados e formato de dados é a melhor forma que sirva ao seu propósito. O foco da prova é o algoritmo e não rotinas de leitura e impressão de dados.

1. Este exercício foi proposto nos nossos encontros de segunda, realizado no dia 6/12. Um estacionamento tem largura para um carro apenas, porém é comprido. Normalmente 5 carros, numerados de 1 a 5 entram e saem ao longo do dia (entram apenas uma vez). Eles sempre entram na mesma ordem: 1, 2, 3, 4, 5. A dúvida é, em que ordem eles saem. Por exemplo, se todos entrarem antes do primeiro sair, a ordem de saída é: 5, 4, 3, 2, 1, porém se cada um que entrar sair antes do próximo entrar, a ordem de saída é 1, 2, 3, 4, 5. Algumas ordens são possíveis, como 2, 1, 5, 4, 3, porém outras são impossíveis, como 2, 1, 5, 3, 4. Pede-se que construa um algoritmo onde, dada uma determinada ordem retorne: `SIM` se for uma possível ordem de saída e `NÃO` se não for possível. Considere que todas as estruturas já estão implementadas: fila: enfileirar, desenfileirar e ehvazia; pilha: empilhar, desempilhar e ehvazia; e árvore binária: insFilhoEsq, insFilhoDir, ...
   Aqui vamos usar uma estrutura de pilha, pois é assim que acontece, o primeiro que entra no estacionamento é o último que sai.
   Para saber se uma ordem é possível vamos fazer o seguinte processo:
   - Empilhamos um carro.
   - Verificamos o topo: Enquanto no topo estiver o carro que queremos que saia, desempilhamos.
   - Se no topo não for o carro que queremos que saia, empilhamos outro.
   - Se no final verificarmos que sairam todos os carros na ordem a resposta é sim, caso contrário, não.

   Entrada: S um vetor com a ordem de saída.
   Saída: Sim, se a ordem for válida, Não se a ordem não for válida.

```
pilha p;
```

```
enum {nao, sim};

int verificaOrdem(int S[5]) {
  int entrada[5] = {1, 2, 3, 4, 5};
  int e=0, s=0;
  int ret = nao;

  while(e < 5) {
    empilhar(&p,entrada[e++]);
    while(!ehvazia(p) && p.itens[p.topo-1] == S[s]) {
      desempilhar(&p);
      s++;
    }
  }

  if(s == 5) ret = sim;
  return ret;
}
```

2. Este exercício é original. Muitos supermercados ainda usam filas individuais em caixas. Então quem vai entrar na fila terá de escolher entre as várias. Vamos implementar um modelo de fila dupla. A função/método `desenfileirar(fila)` → `obj` indica explicitamente a fila que será desenfileirada, mas o método `enfileirar(obj)` irá enfileirar na fila com menos objetos. Implemente esta estrutura e os métodos/funções necessários para operar com ela com segurança (sem erro).
   Este é até simples. Poderíamos construir uma função enfileirar única para realizar toda a função. Mas a partir do momento em que já temos as funções enfileirar, desenfileirar e ehvazia para uma fila simples. Duplicamos esta função, uma para cada fila. As funções que criaremos será simplesmente para escolher a fila. A questão de segurança implica em também criar uma função que verifique se uma ou outra fila está vazia: `ehvazia(fila)`. Vou omitir aqui as funções básicas para uma filha que já foi disponibilizada em material de aula.

```
typedef int obj_t; // O tipo do objeto é genérico.
typedef int boolean; // Verdadeiro ou Falso, conforme o enum abaixo.
enum{falso, verdade};

typedef struct fila {  // A fila
   obj_t itens1[TAMANHOFILA];
   obj_t itens2[TAMANHOFILA];
   int inicio1, fim1, inicio2, fim2;
   boolean cheia1, cheia2, vazia1, vazia2;
} fila;

boolean enfileirar(fila *p, obj_t obj);
obj_t desenfileirar(fila *p, int f);
boolean ehvazia(fila p, int f);

boolean enfileirar1(fila *p, obj_t obj);
boolean enfileirar2(fila *p, obj_t obj);
obj_t desenfileirar1(fila *p);
```

```
obj_t desenfileirar2(fila *p);
boolean ehvazia1(fila p);
boolean ehvazia2(fila p);

boolean enfileirar(fila *p, obj_t obj) {
  boolean ret = falso;
  int tam1, tam2; // tamanho das filas
  tam1 = p->inicio1 <= p->fim1 ? p->fim1 - p->inicio1 :
                                 p->fim1 + TAMANHOFILA - p->inicio1;
  tam2 = p->inicio2 <= p->fim2 ? p->fim2 - p->inicio2 :
                                 p->fim2 + TAMANHOFILA - p->inicio2;
  if(tam1 == 0 && p->vazia1) ret=enfileirar1(p, obj);
  else if(tam2 == 0 && p->vazia2) ret=enfileirar2(p,obj);
  else if(tam1 <= tam2) ret=enfileirar1(p,obj);
  else enfileirar2(p,obj);
  return ret;
}

obj_t desenfileirar(fila *p, int f) {
  obj_t o;

  assert(f == 1 || f == 2);
  if(f == 1) o = desenfileirar1(p);
  else o = desenfileirar2(p);
  return o;
}

boolean ehvazia(fila p, int f){
  boolean ret;

  assert(f == 1 || f == 2);
  if(f == 1) ret = ehvazia1(p);
  else ret = ehvazia2(p);
  return ret;
}
```

3. Exercício 3 da lista 8. Uma proposta de algoritmo foi mencionada em sala de aula quando apresentamos solução para expressões pós-fixa e infixa. Crie um algoritmo para construir uma árvore binária de expressão a partir de uma expressão pré-fixa. (você já possui a biblioteca para manipular uma árvore binária: `inserirFilhoEsq(obj)`, ..., `getInfo()`, ...)
   Este eu respondi em sala de aula:

   criarexpressao()
   leio um símbolo;
   crio uma árvore exp com o símbolo como raiz;
   se simbolo== operador:
   . op1 = criarexpressao();
   . op2 = criarexpressao();
   . insFilhoEsq(exp,op1);

. insFilhoDir(exp,op2);
fim.se
retorna exp.

```
arvore *pegarOperando() {
  char symb;
  arvore *ret=NULL;
  arvore *fe, *fd;

  if(scanf("%c",&symb) == 1) {
    ret = (arvore *) malloc(sizeof(arvore));
    iniciar(ret,symb);
    if(symb=='+' || symb=='-' || symb=='*' || symb=='/' || symb=='^') {
      fe=pegarOperando();
      fd=pegarOperando();
      insFilhoEsq(ret,fe);
      insFilhoDir(ret,fd);
    }
  }
  return ret;
}

int main(int argc, char **args) {
  arvore *exp;

  exp=pegarOperando();
  percOrdem(exp,imprimirInfo,abrePar,fechaPar);  //testando
  printf("\n");
  return 0;
}
```

4. Questão 3 da lista 7. Considere uma árvore ternária (3-ária). Na nossa implementação faltaram as funções/métodos de percurso em pré-ordem e em pós-ordem para esta árvore. Construa estas rotinas.
Questão mais fácil da prova.
PercursoPreOrdem():
visitaInfo();
para i=1 até i=3 PercursoPreOrdem(filhos[i]);

PercursoPosOrdem():
para i=1 até i=3 PercursoPosOrdem(filhos[i]);
visitaInfo();

```
void percPreOrdem(arvore *a, funcao_t *visitaInfo) {
  int i;

  if(a != NULL) {
    visitaInfo(a);
    for(i = 0; i < NARIA; i++)
```

```
      percPreOrdem(a->filhos[i],visitaInfo);
  }
  return;
}


void percPosOrdem(arvore *a, funcao_t *visitaInfo) {
  int i;

  if(a != NULL) {
    for(i = 0; i < NARIA; i++)
      percPosOrdem(a->filhos[i],visitaInfo);
    visitaInfo(a);
  }
  return;
}
```

5. Esta questão é um desafio. Questão extra na prova, e será tratada como tal, mas foi cobrada como exercício 9 da lista 8. Uma proposta de solução deste exercício foi apresentada na nossa aula de exercícios na véspera da prova. Crie uma estrutura de heap para representar uma fila de prioridade e construa as funções: `enfileirar(obj)`, `desenfileirar()` → `obj` e `ehvazia()`.
   A grande dificuldade deste problema é a fila andar. Normalmente numa estrutura de vetor quando a fila anda a fazemos rotativa. Neste caso não, pois a fila é uma árvore binária, se tiramos o raiz outro nó precisa assumir o raiz. Escolher o filho maior não é solução, pois poderíamos por exemplo escolher o filho direito e no final uma folha do filho direito sobe para o raiz e a árvore deixa de ser quase completa, condição necessária para um heap.
   A fila tem de andar, ou seja, o último elemento não estará na posição n mais, e sim na $n-1$. O coringa é então o último elemento, onde ele ficaria de fato. Com o uso do Max-Heapfy fica simples, pegamos o último elemento e colocamos no início da fila e chamamos um Max-Heapfy nele e ele volta a assumir a posição correta no Heap, e não só isso, o início do Heap volta a ficar com o maior elemento.
   Ao inserir um elemento, ele é inserido no final da fila. Para que ele assuma sua posição de direito, basta comparar ele com o pai na estrutura, se ele for maior, ele troca de posição com o pai.

```
typedef int obj_t; // O tipo do objeto é genérico. Neste exemplo usamos int. Basta
typedef int boolean; // Verdadeiro ou Falso, conforme o enum abaixo.
enum{falso, verdade};

typedef struct heapfila {  // A fila
   obj_t itens[TAMANHOFILA];
   int fim; // Marcadores de 1 a n --> vetor de 0 a n-1.
} heapfila; // fim é o último elemento. fim = 0 é fila vazia

boolean enfileirar(heapfila *p, obj_t obj);
obj_t desenfileirar(heapfila *p);
boolean ehvazia(heapfila p);

boolean enfileirar(heapfila *p, obj_t obj) {
```

```
  boolean ret = falso;
  int i;
  obj_t aux;

  if(p->fim != TAMANHOFILA) {
    p->itens[p->fim++]=obj;
    i = p->fim;
    while((i > 1) && (p->itens[i-1] > p->itens[i/2 - 1])) {
      aux = p->itens[i-1];
      p->itens[i-1] = p->itens[i/2 - 1];
      p->itens[i/2 - 1] = aux;
      i = i/2;
    }
    ret = verdade;
  }
  return ret;
}

obj_t desenfileirar(heapfila *p) {
  int i = 1, max=1;
  obj_t o, aux;
  boolean para = falso;
  assert(p->fim != 0);
  o = p->itens[0]; // elemento que sai
  p->itens[0] = p->itens[--p->fim];  // ultimo vai para o começo da fila
  while(!para) {          // max-heapfy
    if(((2*i) <= p->fim) && (p->itens[i-1] < p->itens[2*i-1])) max = 2*i;
    if(((2*i+1) <= p->fim) && (p->itens[max-i] < p->itens[2*i])) max = 2*i+1;
    if(i != max) {
      aux = p->itens[i-1];
      p->itens[i-1] = p->itens[max-1];
      p->itens[max-1] = aux;
      i = max;
    }
    else para = verdade;
  }
  return o;
}

boolean ehvazia(heapfila p) {
  return (p.fim == 0);
}
```