INF 1620- Estruturas de Dados
Soluções das questões da P2 - 08/11/00

1. *Queremos programar uma pilha em que a cada elemento devem corresponder duas informações:*
   ```
   int matric;
   char *nome;
   ```

Escreva os tipos necessários para esta implementação, e as funções para empilhar (`push`) e desempilhar (`pop`).
Explique que limitações tem a sua implementação.

As duas soluções a seguir são exemplos da variedade de soluções admissíveis.

Solução 1: lista encadeada com informação do tipo `INFO`

```
struct s_info {
   int matric;
   char *nome;
};
typedef struct s_info INFO;

typedef struct no *PT;  /* PT é o tipo pilha */
struct no {
   INFO info;
   PT prox;
};

/* auxiliares */
INFO fazinfo(int m, char *n) {
   INFO i;
   i.matric=m;
   i.nome=copia(n);  /* como sempre */
   return i;
}

PT fazno(INFO i, PT px) {
   PT p=(PT)malloc(sizeof(struct no));
   p->info=i;
   p->prox=px;
   return p;
}

void push(PT *p, INFO i) {
   *p=fazno(i, *p);
}
```

```
INFO pop(PT *p) {
   PT q=*p;
   INFO i;
   assert (q!=NULL);   /* precisa? */
   i=q->info;
   *p=q->prox;
   free(q);
   return i;
}
```

Esta solução tem como desvantagem que várias estruturas `INFO` são copiadas, o que pode ser ineficiente no caso de estruturas grandes.

Para empilhar um registro `(5, "Maria")` na pilha `p1`, o comando seria

```
push(&p1,fazinfo(5,"Maria"));
```

Solução 2: vetor de ponteiros para `INFO`.

```
#define MAX 100
struct s_info {
   int matric;
   char *nome;
};
typedef struct s_info *PINFO;

typedef struct s_pilha PILHA;

struct s_pilha {
   PINFO vet[MAX];
   int num;     /* pilha vazia, num=0 */
};

/* auxiliar */
PINFO fazinfo(int m, char *n) {
   PINFO pi=(PINF) malloc(sizeof(struct s_info));
   pi->matric=m;
   pi->nome=copia(n);   /* como sempre */
   return pi;
}

void push(PILHA *p, int m, char *n) {
   assert(p->num<MAX-1);   /* precisa? */
   p->vet[p->num++]=fazinfo(i,n);
}

void pop(PILHA *p, int *m, char **n) {
   PINFO pi;
   assert(p->num>0);
   pi->p->vet[--p->num];
   *m=pi->matric;
   *n=pi->nome;
   free(pi);
}
```

Esta solução tem a limitação de que apenas `MAX=100` elementos podem ser empilhados. Note que estas soluções são apenas dois exemplos das soluções possíveis para esta questão.

2. *Considere uma árvore binária definida pelos tipos*

```
typedef struct no *BIN;
struct no {
   INFO *s;
   struct no *esq, *dir;
} ;
```

Escreva uma função com protótipo

```
int alt (BIN b);
```

para calcular a altura de uma árvore binária.
Sugestão: Como a altura de uma árvore é definida como o maior comprimento de caminho da raiz até algum nó, a altura de uma árvore vazia deve ser –1.

```
/* auxiliar */
int max2(int i, int j) {
   if (i>j)
      return i;
   return j;
}

int alt(BIN b) {
   if (b==NULL)
      return –1;
   return 1+max2(alt(b->esq),alt(b->dir));
}
```

3. *Uma lista de dados duplamente encadeada é dada pelos seguintes tipos:*

```
typedef struct no *PT;
struct no {
   int info;
   PT ant, suc; /* antecessor e sucessor */
} ;
struct s_lista2 {
   PT prim, ult;/* primeiro e ultimo da lista */
} ;
typedef struct s_lista2 *LISTA2;
```

Escreva uma função `copia`*, com protótipo*

```
LISTA2 copia(LISTA2 l);
```

cujo resultado seja uma cópia da lista completamente independente da lista original.

```
/* auxiliar */
PT fazno(int i) {
   PT q = (PT) malloc(sizeof(struct no));
   q->info = i;
   q->ant  = NULL;
   q->suc  = NULL;
   return q;
}

LISTA2 copia(LISTA2 l) {
   LISTA2 l2 = (LISTA2) malloc(sizeof (struct s_lista2));
   PT p; /* percorre a lista l */
   PT q; /* copia de p */
   PT r; /* antecessor de q */
   if (l->prim==NULL) {   /* ult tambem é NULL */
       l2->prim=NULL;
       l2->ult=NULL;
       return l2;
   }
   /* há nós para serem copiados */
   p=l->prim;  /* primeiro nó para ser copiado */
   r=NULL;
   while(p!=NULL){
      q=fazno(p->info);
      if (r==NULL) /* q é o primeiro nó da cópia */
        l2->prim=q;
      else
        r->suc = q;
      q->ant=r;
      r=q;
      p=p->suc;
   }
   l2->ult=q;
   return l2;
}
```

4. *Uma árvore binária de busca é definida com a propriedade de que em qualquer nó n, todos os elementos da sub-árvore esquerda de n têm valores menores que o de n, e todos os elementos da sub-árvore direita têm valores maiores ou iguais que o de n. Os tipos correspondentes são*

```
typedef struct no *ARV;
struct no {
   int val;
   struct no *esq, *dir;
} ;
```

Escreva uma função com protótipo

```
void showmenor (ARV a, int x);
```

que imprime todos os valores nos nós da árvore a que sejam menores que x, em ordem crescente.

```
void showmenor(ARV a, int x) {
   if (a==NULL)
      return;
   showmenor(a->esq);
   if (a->val<x) {
      printf ("%d ", a->val);
      showmenor(a->dir);
   }
}
```

Note que a sub-árvore direita não precisa ser visitada se `a->val>=x`.

5. *Escreva uma função `max` que visita uma árvore e determina qual é o grau máximo de seus nós, isto é, quantos filhos tem o nó que tem mais filhos na árvore. A função `max` tem como protótipo:*

```
int max(ARV a);
```

onde `ARV` é o tipo da árvore:

```
typedef struct no *ARV;
struct no {
   char val;
   struct no *prim, *prox; /* lista de filhos */
} ;
```

Suponha que a árvore não é vazia.

```
/* ver max2 na questão 2 */

int max(ARV a) {
   PT p;      /* p percorre a lista de filhos */
   int m=0;   /* máximo número de filhos dos filhos */
   int n=0;   /* número de filhos de a */
   if (a->prim==NULL)
      return 0;
   for(p=a->prim; p!=NULL; p=p->prox) {
      n++;
      m=max2(m,max(p));
   }
   return max2(m,n);
}
```

O teste `a->prim==NULL` pode ser omitido, porque neste caso o valor do resultado será `max2(0,0)=0`.