INF 1620- Estruturas de Dados
Primeira Prova - 20/09/00

1. *Resolvendo um exercício, um aluno escreveu uma função que não funciona como esperado. A função `SP` deveria calcular a soma e o produto dos números inteiros `a` e `b`, e devolver seus valores em `s` e `p`, respectivamente.*

```
void SP(int a, int b, int s, int p) {
   s=a+b;   /* s recebe a soma de a e b */
   p=a*b;   /* p recebe o produto de a e b */
}
```

(a) Explique o funcionamento da função `SP`, mostrando o que acontece quando é executado o trecho de código a seguir:

```
...
int n1, n2, soma, prod;
n1=2;
n2=3;
soma=0;
prod=0;
SP(n1,n2,soma,prod);
...
```

(b) Mostre como a função `SP` pode ser consertada, para que `soma` e `prod` recebam os valores esperados (`n1+n2` e `n1*n2`, respectivamente).

Problema: As variáveis `s` e `p` da função `SP` vão receber os valores corretos, mas estes valores não serão passados de volta para as variáveis `soma` e `prod`, que aparecem na chamada. Isto acontece porque a passagem de parâmetros em C é por valor: o parâmetro recebe o *valor* do argumento no instante da chamada, mas nenhum valor é passado de volta ao final da execução da função.

Solução: Em vez de passar o valor das variáveis `soma` e `prod`, passamos seus *endereços*. Várias modificações são necessárias:

a) no cabeçalho (protótipo) da função precisamos alterar os tipos dos dois últimos parâmetros (`s` e `p`). Para deixar a situação bem clara, vamos trocar também seus nomes para `ps` e `pp`, indicando que agora temos *ponteiros* (ou apontador) para os antigos `s` e `p`.
b) no corpo da função `SP`, vamos usar `*ps` e `*pp` (os conteúdos de `ps` e `pp`), para que esses valores sejam alterados.
c) Na chamada da função `SP`, vamos passar os endereços de `soma` e `prod`.

O resultado final é

```
void SP(int a, int b, int *ps, int *pp) {
   *ps=a+b;   /* *ps recebe a soma de a e b */
   *pp=a*b;   /* *pp recebe o produto de a e b */
}
...
SP(n1,n2,&soma,&prod);
...
```

2. *Escreva uma função que receba como entrada uma cadeia de caracteres `s` e substitua todos os dígitos (caracteres `'0'`, `'1'`, … `'9'`), contidos na cadeia, pelo caracter `'*'`. A função deve ter como protótipo*
   `void troca_digitos(char *s);`

Por exemplo, se tivermos

`t="abs 12df345 def"`,

após a chamada

`troca_digitos(t);`

teremos

`t="abs **df*** def"`.

A função deve percorrer toda a cadeia, fazendo as substituições necessárias.

```
void troca_digitos(char *s) {
   int i;
   for (i=0; s[i]!='\0'; i++)
      if ((s[i]>='0') && (s[i]<='9'))
         s[i]='*';
}
```

Uma solução um pouco mais rápida guarda o valor de `s[i]` em uma variável `c`.

```
void troca_digitos(char *s) {
   int i;
   char c;
   for (i=0;(c=s[i])!='\0'; i++)
      if ((c>='0') && (c<='9'))
         s[i]='*';
}
```

Outra solução pode usar o comprimento da cadeia, obtido por `strlen`.

```
#include <string.h>
void troca_digitos(char *s) {
   int i;
   int l=strlen(s);
   for (i=0; i<l; i++)
      if ((s[i]>='0') && (s[i]<='9'))
         s[i]='*';
}
```

3. *Escreva um programa que recebe como entrada uma cadeia de caracteres `s` e um inteiro `n`, e, em seguida imprime o "sufixo" da cadeia `s` obtido retirando da cadeia os primeiros `n` caracteres. Se a cadeia não tiver pelo menos `n` caracteres, deve ser impressa a mensagem `"erro"`.*

Por exemplo, se `s= "abcdefghi"` e `n=3`, então deve ser impresso o sufixo `"defghi"`; com a mesma cadeia `s` e `n=17`, deve ser impresso `"erro"`.

```
#include <stdio.h>
#include <string.h>
int main(void) {
   int l, i;
   char s[120];    /* chega? */
   printf("cadeia de entrada >");
   scanf("%[^\n]",s);
   l=strlen(s);
   if (n>l)
      printf("erro");
   else
      for (i=n; i<l; i++)
         printf("%c",s[i]);
   printf("\n");
   return 0;
}
```

Outra solução escreve o sufixo da cadeia como sendo a cadeia que se inicia em `s[n]`. O `for` é substituído por

```
printf("%s",&s[n]);
```

4. *Escreva uma função com protótipo*
   `char *dupl(char *s);`

que recebe uma cadeia de caracteres `s` e retorna a cadeia duplicada. (O espaço para a nova cadeia deve ser alocado pela função.)
Por exemplo, se tivermos `s="abcd"`, o resultado da função deve ser `"abcdabcd"`.

```
#include <string.h>
char *dupl(char *s) {
   int l=strlen(s);
   char *t=(char *)malloc(2*l+1);
   strcpy(t,s);
   strcat(t,s);
   return t;
}
```

uma solução mais curta, porém menos clara, dispensa a necessidade de declarar `l` e `t`:

```
#include <string.h>
char *dupl(char *s) {
   return strcat(strcpy(
      (char *)malloc(2*strlen(s)+1),s),s);
}
```

5. *Uma pilha de inteiros é descrita pelas seguintes declarações:*

```
#define MAX 100
int vet[MAX];        /* vetor com os elementos da pilha */
int num;             /* número de elementos na pilha */
void init(void);     /* inicia a pilha */
void push(int elem); /* empilha elem */
int pop(void);       /* desempilha um elemento, que e' o
                        resultado */
int vazia(void);     /* testa se pilha esta' vazia */
```

```
void show(void);     /* imprime os elementos da pilha,
                        entre colchetes, em uma única linha,
                        em ordem, a partir do topo */
```

Terminada a implementação, o seguinte programa é executado:

```
int main(void){
   int a,b;
   init();
   show();
   push(3); push(4); push(5); push(6);
   show();
   a=pop(); b=pop(); push(a+b);
   show();
   a=pop(); push(a*pop());
   show();
   printf("%d", pop());
   show();
   return 0;
}
```

Mostre qual será o efeito deste programa, indicando qual será a saída impressa pelo programa.

Comando	Conteúdo da pilha: num, vet	Saída impressa
`init();`	`0, [? ? ? ? ? ... ? ]`	
`show();`		`[ ]`
`push(3);`	`1, [3 ? ? ? ? ... ? ]`	
`push(4);`	`2, [3 4 ? ? ? ... ? ]`	
`push(5);`	`3, [3 4 5 ? ? ... ? ]`	
`push(6);`	`4, [3 4 5 6 ? ... ? ]`	
`show();`		`[ 6 5 4 3 ]`
`a=pop();`	`3, [3 4 5 6 ? ... ? ]` (a=6)	
`b=pop();`	`2, [3 4 5 6 ? ... ? ]` (b=5)	
`push(a+b);`	`3, [3 4 11 6 ? ... ? ]`	
`show();`		`[ 11 4 3 ]`
`a=pop();`	`2, [3 4 11 6 ? ... ? ]` (a=11)	
`push(a*pop());`	`2, [3 44 11 6 ? ... ? ]` (pop tem resultado 4)	
`show();`		`[ 44 3 ]`
`printf("%d", pop())`	`1, [3 44 11 6 ? ... ? ]` (pop tem resultado 44)	`44`
`show()`		`[ 3 ]`