INF 1620 – P1 - 26/10/01	Questão 1

(a) Escreva uma função em C, chamada `ordena`, para colocar em ordem crescente dois valores reais. Por exemplo, se tivermos duas variáveis a=3.4 e b=2.1 antes da chamada de `ordena`, teremos a=2.1 e b=3.4, após a chamada da função, pois 2.1 é menor que 3.4. Caso contrário, a troca não seria feita.

(b) O programa abaixo deve ilustrar a utilização da função `ordena` e imprimir os valores 2.1 e 3.4, nesta ordem. Mostre como seria a chamada da função `ordena` para que o programa funcione de maneira correta.

```
#include <stdio.>
int main (void) {
   float a = 3.4, b = 2.1;
   /* chama funcao ordena */

   ..........................      ← escreva aqui a chamada de ordena
   printf("%f  %f\n",a,b);
   return 0;
}
```

(a) Aparentemente, o cabeçalho de `ordena` deve permitir dois parâmetros reais (tipo `float`). Mas, como em C parâmetros são passados por valor, os parâmetros não podem ser alterados. Para que a alteração dos valores dos dois números reais seja possível, devemos passar seus *endereços* (tipo `float *`) para a função, que pode ter cabeçalho

```
void ordena(float *pr1, float *pr2);
```

O tipo do resultado da função pode ser `void`, porque o efeito da função deve ser o de trocar os valores reais, sem gerar nenhum resultado. O código da função pode ser

```
void ordena(float *pr1, float *pr2) {
   float temp;
   if (*pr1>*pr2) {
      temp = *pr1;
      *pr1 = *pr2;
      *pr2 = temp;
   }
}
```

Outra possibilidade seria

```
void ordena(float *pr1, float *pr2) {
   float temp;
   if (*pr1<=*pr2)
      return;    /* não precisa trocar, sai da função */
   temp = *pr1;
   *pr1 = *pr2;
   *pr2 = temp;
}
```

(b) A chamada da função deve passar como argumentos os endereços de `a` e `b`. Ou seja, pode ser

```
ordena(&a,&b);
```

INF 1620 – P1 - 26/10/01	Questão 2

(a) Escreva uma função em C que receba como parâmetro um número inteiro não negativo e retorne, como resultado da função, o valor do seu fatorial. A assinatura desta função deve ser:

```
int fat (int n);
```

(b) Escreva um programa que capture um número inteiro entrado pelo usuário via teclado e imprima o valor do fatorial usando a função escrita no item acima. Pode-se assumir que o usuário nunca entrará com um valor menor do que zero.

(a) Uma primeira opção é usar recursão para implementar a função. Podemos usar a propriedade

```
n! = n*(n-1)! para n>0
```

Neste caso, a função seria

```
int fat(int n) {
   if (n==0)
      return 1;
   /* aqui não precisa else! */
   return n*fat(n-1);
}
```

Uma segunda opção seria usar um comando `for` para calcular o produto

```
n! = 1 * 2 * ... * (n-1) * n
```

Neste caso, a função seria

```
int fat(int n) {
   int prod=1; /* para calcular o produto */
   int i;      /* variável do for */
   for (i=1; i<=n; i++)
      prod=prod*i;
   return prod;
}
```

(b) O programa poderia ser composto de duas funções: a função `fat` (por exemplo na versão recursiva), e a *inevitável* função `main`, que leria o número cujo fatorial deve ser calculado, chamaria a função fat para o cálculo, e escreveria o resultado. As funções para ler e imprimir reais são da biblioteca padrão `stdio.h`.

```
#include <stdio.h>
int fat(int n);

int main(void) {
   int n,f;
   printf("escreva o numero > ");
   scanf("%f", &n);   /* deve ser endereço de n */
   f=fat(n);
   printf("O fatorial de %d vale %d\n", n, f);
   return 0;
}
```

INF 1620 – P1 – 26/10/01	Questão 3

Escreva uma função em C para concatenar duas cadeias de caracteres. A função deve receber duas cadeias de caracteres como parâmetros de entrada e retornar uma nova cadeia, cujo espaço de memória deve ser alocado pela função, contendo a concatenação da primeira com a segunda. Por exemplo, se forem passadas as cadeias "PUC" e "Rio", a função deve retornar a cadeia "PUCRio". A assinatura da função deve ser:

```
char* concatena (char* a, char* b);
```

Nossa solução usa funções da biblioteca `string.h`, como `strlen` (comprimento), `strcpy` (cópia) e `strcat` (concatenação). Além disso, vamos usar a função de alocação de memória `malloc`, da biblioteca `stdlib.h`.

Note que `strcat` concatena uma cadeia na outra, mas não faz a cópia da primeira, nem aloca espaço para a cadeia resultado. Temos:

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

char *concatena(char *a, char *b) {
   char *c;
   int la, lb;
   la = strlen(a);
   lb = strlen(b);
   c = (char *)malloc(la+lb+1);
       /* espaço para os símbolos das duas cadeias
          e para o símbolo nulo */
   strcpy(c,a);
   strcat(c,b);
   return c;
}
```

Outra solução, mais trabalhosa, não usa as funções de tratamento de string.

```
#include <stdlib.h>

char *concatena(char *a, char *b) {
   char *c;
   int la, lb, i;

/* calcula o comprimento de a:
   a[la] deve ser o primeiro nulo */
   la = 0;
   while(a[la]!='\0')
      la++;

/* calcula o comprimento de b:
   b[lb] deve ser o primeiro nulo */
   lb = 0;
   while(b[lb]!='\0')
      lb++;

   c = (char *)malloc(la+lb+1);
       /* espaço para os símbolos das duas cadeias
          e para o símbolo nulo */

/* copia a para c, sem o nulo */
for(i=0; i<la; i++)
   c[i]=a[i];

/* copia b para c, com o nulo, defasada de la */
for(i=0; i<=lb; i++)
   c[i+la]=a[i];

   return c;
}
```

INF 1620 – P1 – 26/10/01	Questão 4

Escreva uma função que faça a busca de um elemento num vetor de valores inteiros ordenados em ordem decrescente. A função deve tirar proveito do fato dos elementos do vetor estarem ordenados e retornar 1 (um), se o elemento estiver presente no vetor, ou 0 (zero), se o elemento não estiver presente. A assinatura da função deve ser:

```
int busca (int n, int *vet, int elem);
```

A resposta mais simples usa a ordem apenas para parar a busca de um valor, quando um valor menor que ele for encontrado.

```
int busca(int n, int *vet, int elem) {
   int i;
   for (i=0; i<n; i++) {
      if (vet[i]==elem)
         return 1;    /* achou elem */
      if (vet[i]<elem)
         return 0;    /* achou menor que elem */
   }
   return 0;     /* acabou o vetor */
}
```

Uma solução mais complicada faz a busca binária no vetor. A busca binária é implementada pela função (recursiva) `bin`.

```
int bin(int ini, int fim, int *vet, int elem) {
   int meio;
   if (ini>fim)
      return 0;
   meio=(ini+fim)/2;
   if (elem==vet[meio])
      return 1;
   if (elem>vet[meio])
      return bin(ini,meio-1,vet,elem);
   return bin(meio+1,fim,vet,elem);
}

int busca(int n, int *vet, int elem) {
   return bin(0,n-1,vet,elem);
}
```

INF 1620 – P1 - 26/10/01	Questão 5

O código C abaixo mostra a implementação de uma função para ordenação (ordem crescente) de um vetor de valores reais usando o método conhecido por bubblesort.

```
Linha 1       void bubblesort (int n, float* vet) {
Linha 2          int i, j;
Linha 3          for (i=n-1; i>=1; i--) {
Linha 4             for (j=0; j<i; j++) {
Linha 5               if (v[j] > v[j+1]) { /* troca */
Linha 6                 float temp = v[j];
Linha 7                 v[j] = v[j+1];
Linha 8                 v[j+1] = temp;
Linha 9               }
Linha 10            }
Linha 11         }
Linha 12      }
```

(a) Mostre que alterações são necessárias para que o procedimento acima seja interrompido assim que o vetor ficar ordenado.

(b) Mostre os valores intermediários dos elementos do vetor se este seu algoritmo modificado for aplicado na ordenação do vetor abaixo.

[2 1 4 3 5 8 6 7]

A alteração deve ser feita para garantir que a execução de `bubblesort` se encerra quando uma passagem é feita sem que nenhuma troca de valores seja necessária. Isso pode ser feito usando uma variável `trocou` que recebe o valor 1 quando uma troca é feita. Inicialmente, `trocou` tem o valor 1, para garantir que uma primeira passagem é sempre feita. O processo se encerra no final de uma passagem, quando `trocou` é 0, indicando que não foram necessárias trocas, e que o vetor já se encontra ordenado. (A solução nas notas de aula usa uma variável `acabou`, que tem como valor a negação de `trocou`.)

```
Linha 1       void bubblesort (int n, float* vet) {
Linha 2          int i, j;
                 int trocou;   /* variável nova */
Linha 3          for (i=n-1; i>=1; i--) {
                    trocou=0;
                    /* ainda não trocou, nesta passagem */
Linha 4             for (j=0; j<i; j++) {
Linha 5               if (v[j] > v[j+1]) { /* troca */
Linha 6                 float temp = v[j];
Linha 7                 v[j] = v[j+1];
Linha 8                 v[j+1] = temp;
                        trocou=1;       /* acabou de trocar */
Linha 9               }
Linha 10            }
                    if (!trocou)
                       return;
                    /* uma passagem sem trocar: acabou */
Linha 11         }
Linha 12      }
```

As linhas novas são as linhas sem identificação.

Os valores intermediários são:

```
      [2  1  4  3  5  8  6  7]

Passo i=7:
      [1  2  4  3  5  8  6  7]
      [1  2  3  4  5  8  6  7]
      [1  2  3  4  5  6  8  7]
      [1  2  3  4  5  6  7  8]
```

Não há troca no passo i=6, pois o vetor já se encontra ordenado. Assim, o passo i=5 não chega a ser executado.