Estruturas de Quebra Seqüencial

break

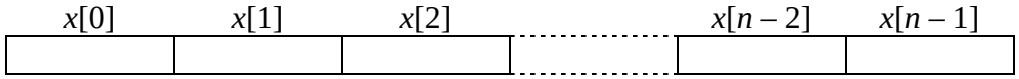
- Quando executado em qualquer ponto de uma estrutura de repetição (`for`, `while` ou `do while`), causa seu término imediato. O controle do programa passa então imediatamente para o código seguinte ao loop
- *AVISO: o uso desta estrutura deve ser evitado por dificultar a identificação de falhas no código.*

continue

- Força a próxima iteração de um loop pulando os códigos que estiverem depois do *continue.*
- *AVISO: o uso desta estrutura deve ser evitado por dificultar a identificação de falhas no código.*

Vetor

- Conjunto de locais para armazenamento de dados de um mesmo tipo e com um mesmo nome.
- *Elemento do vetor* – cada local de armazenamento do vetor.
  - Em um vetor *x* de *n* elementos, o primeiro é o *x*[0] e o último é o *x*[*n* – 1]



- Sintaxe de declaração: `<tipo do vetor> <nome do vetor>[<tamanho do vetor>];`
- Exemplos de declaração:

```
float notas_alunos[4];
int ra_alunos[4];
```

- Exemplos de atribuição de dados ao vetor:

```
ra_alunos[0] = 980001;
ra_alunos[1] = 980002;
ra_alunos[2] = 980003;
ra_alunos[3] = 980004;
notas_alunos[0] = 10.0;
notas_alunos[1] = 5.5;
notas_alunos[2] = 7.3;
notas_alunos[3] = 9.8;
```

- Exemplos de acesso aos dados do vetor:

```
for (int i = 0; i < 4; i++)
{
   printf("O aluno %d ficou com nota %f", ra_alunos[i], notas_alunos[i]);
}
```

- Vetores podem ser declarados e inicializados como no exemplo a seguir.

```
char notas_musicais[7] = {‘C’, ‘D’, ‘E’, ‘F’, ‘G’, ‘A’, ‘B’};
```

Exercício: Fazer um programa em linguagem C para cada um dos seguintes problemas:

a) *Primeira ocorrência* – Receber um inteiro positivo não nulo i < 20 correspondente ao tamanho da lista de números a ser fornecida pelo usuário e alocar todos os números desta lista em um vetor. Em seguida, receber outro valor e encontrar sua primeira ocorrência no vetor, imprimindo na tela esta posição. Caso não exista nenhuma ocorrência, avisar o usuário.

```
#include <stdio.h>

int main()
{
   int vetor[19];
   int tamanho_vetor, num_procurado, passo;

   do {
      printf("Digite o número de elementos do vetor: ");
      scanf("%d", &tamanho_vetor);
   } while ((tamanho_vetor >= 20) || (tamanho_vetor <= 0));

   printf("Digite o(s) %2d elemento(s) do vetor: ", tamanho_vetor);
   for (passo = 0; passo < tamanho_vetor; passo++)
      scanf("%d", &vetor[passo]);

   printf("Digite o elemento a ser procurado: ", tamanho_vetor);
   scanf("%d", &num_procurado);
   passo = 0;
   while ((passo < tamanho_vetor) && (vetor[passo] != num_procurado))
      passo++;

   if (passo == tamanho_vetor)
      printf("\nO número %d não foi encontrado.", num_procurado);
   else
      printf("\nNúmero %d encotrado na posição %d.", num_procurado, passo);

   return 0;
}
```

b) *Pares e ímpares* – Receber um inteiro positivo não nulo *i* < 20 correspondente ao tamanho da lista de números a ser fornecida pelo usuário e alocar todos os números desta lista em um vetor. Em seguida, preencher dois outros vetores, um com os números pares desta lista e outro com os números ímpares. Imprimir estes últimos dois vetores na tela.

```
#include <stdio.h>

int main()
{
   int vet_todos[19], vet_impares[19], vet_pares[19];
   int tam_vet_todos, tam_vet_impares, tam_vet_pares, passo;

   do {
      printf("Digite o número de elementos do vetor: ");
      scanf("%d", &tam_vet_todos);
   } while ( (tam_vet_todos >= 20) || (tam_vet_todos <= 0) );

   tam_vet_impares = 0;
   tam_vet_pares = 0;
   printf("Digite o(s) %2d elemento(s) do vetor: ", tam_vet_todos);
   for (passo = 0; passo < tam_vet_todos; passo++) {
      scanf("%d", &vet_todos[passo] );
      if ( (vet_todos[passo] % 2) != 0 ) {
         vet_impares[tam_vet_impares] = vet_todos[passo];
         tam_vet_impares++;
      }
      else {
         vet_pares[tam_vet_pares] = vet_todos[passo];
         tam_vet_pares++;
      }
   }

   printf("\nNúmeros ímpares: ");
   for (passo = 0; passo < tam_vet_impares; passo++)
        printf("%d ", vet_impares[passo]);

   printf("\nNúmeros pares: ");
   for (passo = 0; passo < tam_vet_pares; passo++)
        printf("%d ", vet_pares[passo]);

   return 0;
}
```

c) *Todas as ocorrências* – Equivalente ao problema *Primeira ocorrência*, porém retornando todas as ocorrências encontradas no vetor para o valor digitado pelo usuário.

d) *Rotação* – Receber um inteiro positivo não nulo $i < 20$ correspondente ao tamanho da lista de números a ser fornecida pelo usuário e alocar todos os números desta lista em um vetor. Em seguida, rotacionar à direita todos estes valores como no exemplo a seguir, imprimindo o vetor resultante (não use dois vetores):

Vetor original:	23	43	3	766	8
Vetor resultante:	8	23	43	3	766

e) *Intercalação* – Receber dois inteiros positivos não nulos *i* e *j* tal que ambos são menores que 20 e correspondem aos tamanhos das duas listas de números a serem fornecidas pelo usuário e alocar todos os números de cada lista em um vetor independente. Ao fornecer estas duas listas, considera-se que o usuário irá digitá-las em ordem crescente. Em seguida, copiar os elementos dos dois vetores para um terceiro também em ordem crescente e imprimir este vetor (note que, como os dois primeiros vetores já estão ordenados, basta intercalar ordenadamente os seus elementos para construir o terceiro vetor ordenado). O resultado na tela deve ser como no exemplo a seguir:

```
Digite o número de elementos do vetor A: 4
Digite o número de elementos do vetor B: 3
Digite cada elemento do vetor A........: 3 7 88 91
Digite cada elemento do vetor B........: 2 4 90
Elementos ordenados no vetor C.........: 2 3 4 7 88 90 91
```