Pesquisa binária: função bsearch

```
void *bsearch(const void *key,
              const void *buf,
                    size_t num,
                    size_t size,
                    int(*compare)(const void *, const void *));
```

Descrição

A função **bsearch()** realiza uma pesquisa binária na matriz ordenada apontada por *buf*, devolvendo um ponteiro para algum elemento (se houver) que coincide com a chave apontada por *key*. Não necessariamente é retornado o endereço do primeiro elemento contendo o valor pesquisado. Caso o valor buscado não seja encontrado, **bsearch** retorna um ponteiro nulo. O número de elementos da matriz é especificado por *num* e o tamanho (em bytes) de cada elemento é descrito por *size*.

A função apontada por *compare* compara um elemento da matriz com a chave. A função deve ter o formato

```
int func_nome(const void *arg1, const void *arg2);
```

Os dois parâmetros da função precisam ser declarados com o modificador `const`. Essa função é idêntica à utilizada com a rotina `qsort()`.

A função de comparação deve devolver os seguintes valores:

- Se *arg1* é menor que *arg2*, devolve um número menor que zero.
- Se *arg1* é igual a *arg2*, devolve zero.
- Se *arg1* é maior que *arg2*, devolve um número maior que zero.

Com isso a matriz deve estar em ordem ascendente, com o endereço mais baixo contendo o menor valor. Se a matriz não contém a chave pesquisada, é devolvido um ponteiro nulo.

Atenção

1) Evite sempre que seu programa apresente *warnings* durante a compilação. Para isso é necessária especial atenção aos tipos de dados recebidos na função do usuário chamada por `bsearch`. Preste atenção aos exemplos apresentados a seguir para saber como proceder.

2) O valor de retorno da função do usuário precisa ser do tipo `int`. Cuidado com situações em que os dados não são inteiros ou que podem estar fora do intervalo previsto para o tipo inteiro com sinal padrão na sua plataforma.

3) Você pode e deve brincar com os parâmetros de `bsearch` e as possibilidades da função do usuário, pois esse recurso é muito rico e permite uma série de possibilidades. Conhecer o uso de ponteiros ajuda muito nessa tarefa.

Exemplos

1) O programa a seguir recebe 8 números inteiros, que devem ser digitados pelo usuário em ordem crescente de valores, e depois faz a busca de uma chave, verificando se aquele valor existe ou não no conjunto de números armazenados no vetor.

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

int comp(const void *, const void *);

int main(void)
{  int vetor[8];
   int c;
   int chave;
   int *x;

   for (c = 0; c <= 7; c++)
   {  printf("\nInforme o número: ");
      scanf("%d", &vetor[c]);
   }

   printf("\n\n\nInforme o valor a pesquisar:");
   scanf("%d", &chave);

   x = (int *)bsearch(&chave, vetor, 8, sizeof(int), comp);

   if (x == NULL)
      printf("\nNAO achou\n\n");
   else
      printf("\nACHOU\n\n");

   return 0;
}


int comp(const void *p1, const void *p2)
{  int *i = (int *)p1, *j = (int *)p2;

   if (*i < *j)
      return -1;
   else
      if (*i == *j)
         return 0;
      else
         return 1;
}
```

2) O programa a seguir é idêntico ao anterior, exceto que ele não precisa que os valores sejam digitados em ordem crescente, pois ele ordena o vetor com a rotina `qsort`. O programa também exibe em que posição do vetor está o valor encontrado, ou seja, o índice do elemento que possui o valor encontrado.

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

int comp(const void *, const void *);

int main(void)
{  int vetor[8];
   int c;
   int chave;
   int *x;

   for (c = 0; c <= 7; c++)
   {  printf("\nInforme o número: ");
      scanf("%d", &vetor[c]);
   }

   printf("\n\n\nInforme o valor a pesquisar:");
   scanf("%d", &chave);

   qsort(vetor, 8, sizeof(int), comp);

   x = (int *)bsearch(&chave, vetor, 8, sizeof(int), comp);

   if (x == NULL)
      printf("\nNAO achou\n\n");
   else
      printf("\nACHOU o número %d no elemento %d do vetor\n\n",
                  chave, x - vetor);

   return 0;
}

int comp(const void *p1, const void *p2)
{  int *i = (int *)p1, *j = (int *)p2;

   if (*i < *j)
      return -1;
   else
      if (*i == *j)
         return 0;
      else
         return 1;
}
```

3) O programa seguinte lê caracteres digitados no teclado (assumindo E/S pelo teclado com *buffer*) e determina se eles pertencem ao alfabeto.

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <conio.h>

char *alpha = "abcdefghijklmnopqrstuvwxyz";

int comp(const void *, const void *);

int main(void)
{  char ch;
   char *p;

   do
   {  printf("Digite um caractere: ");
      ch = getche();
      ch = tolower(ch);

      p = (char *) bsearch(&ch, alpha, 26, 1, comp);

      if (p != NULL)
         printf("\nESTA no alfabeto\n\n");
      else
         printf("\nNAO esta no alfabeto\n\n");

   } while (p != NULL);

   return 0;
}

int comp(const void *ch, const void *s)
{  char *ch = (char *)p1, *s = (char *)p2;

   if (*ch < *s)
      return -1;
   else
      if (*ch == *s)
         return 0;
      else
         return 1;
}
```

Exercícios

1) Faça um programa que recebe 20 números reais informados pelo usuário em ordem crescente e os armazena em um vetor. Em seguida o programa deverá solicitar ao usuário para que informe um número, que deverá ser pesquisado no vetor por meio de `bsearch`. Exibir uma mensagem informando se o valor foi encontrado ou não no vetor. Ao executar o programa, certifique-se de informar os valores corretamente em ordem crescente, pois o programa apenas fará a pesquisa, mas não ordenará o vetor.

2) Faça um programa que recebe 10 números inteiros e os armazena em um vetor, que deverá ser ordenado por meio da função `qsort`. Em seguida solicitar ao usuário que informe um número a ser pesquisado, execute a pesquisa por meio da função `bsearch` e informe se o valor foi ou não encontrado.

3) Adapte o programa anterior para que ele ordene o vetor inteiro mas, ao pesquisar, procure apenas nos 5 primeiros elementos.

4) Adapte o programa 2 para que ele ordene o vetor inteiro mas, ao pesquisar, procure apenas nos 5 últimos elementos.

5) Adapte o programa 2 para que ele ordene o vetor inteiro mas, ao pesquisar, procure apenas nos elementos entre as posições 4 e 8 do vetor.

6) Faça um programa que recebe 10 palavras (com até 15 caracteres cada uma) e as armazena em um vetor. Em seguida, ordenar o vetor por meio de `qsort` e pesquisar por meio de `bsearch` palavras informadas pelo usuário. Encerrar quando o usuário informar a palavra 'FIM', que não deverá ser pesquisada.

7) Adaptar o programa do exercício 6 para que armazene as palavras em maiúsculas e, ao pesquisar, converter também as palavras informadas para maiúsculas antes de testar ou procurar no vetor.

8) O próximo programa recebe um conjunto de códigos e nomes de pessoa, faz sua ordenação alfabética com QuickSort e depois executa uma pesquisa binária de um nome informado pelo usuário, informando o respectivo código, caso exista no vetor. Sua tarefa é adaptá-lo para informar em qual elemento do vetor o nome informado pelo usuário foi encontrado.

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

struct registro {int numero; char nome[11];};


int comp(const void *, const void *);

int main(void)
{  struct registro vetor[10];
   struct registro chavePesq;
   struct registro *Resp;
   int i;

   printf("Informe os valores: ");
```

```
   for (i = 0; i < 10; i++)
   {  printf("\n%d:", i);
      scanf("%d", &vetor[i].numero);
      scanf("%s", vetor[i].nome);
   }

   qsort(vetor, 10, sizeof(struct registro), comp);

   printf("\n\nInforme o nome a procurar:\n");
   scanf("%s", chavePesq.nome);

   Resp = (struct registro *)bsearch(&chavePesq,
                                     vetor,
                                     10,
                                     sizeof(struct registro),
                                     comp);

   if (Resp != NULL)
      printf("\n\nACHOU no codigo: %d\n\n", Resp->numero);
   else
      printf("\n\nNAO achou\n\n");


   return 0;
}

int comp(const void *p1, const void *p2)
{  struct registro *i = (struct registro *)p1, *j = (struct registro *)p2;

   return strcmp(i->nome, j->nome);
}
```

9) `bsearch` não retorna necessariamente o endereço da primeira ocorrência de um dado valor no vetor. Adapte o programa 8 para que seja impresso o índice da primeira ocorrência com o valor procurado. Depois, utilizando essa adaptação, resolva o problema 1025 do URI.