Ordenação: função qsort

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

Descrição

A função **`qsort()`** ordena a matriz apontada por *buf*, usando o método Quicksort. O Quicksort geralmente é considerado o melhor algoritmo de ordenação de uso geral. Ao terminar, a matriz indicada estará ordenada. O número de elementos na matriz é especificado por *num* e o tamanho (em bytes) de cada elemento é descrito em *size*.

A função apontada por *compare* compara dois elementos da matriz, verificando se eles estão fora de ordem ou não. 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 `bsearch()`.

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.

Dessa forma a matriz é classificada em ordem ascendente, com o endereço mais baixo contendo o menor valor.

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 `qsort`. 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 `qsort` 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 seguinte ordena uma lista de inteiros e mostra o resultado.

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

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

int main(void)
{  int i, num[10] = {2, 1, 6, 5, 8, 7, 9, 6, 2, 0};

   printf("Matriz original: ");
   for (i = 0; i < 10; i++)
      printf("%d ", num[i]);

   qsort(num, 10, sizeof(int), comp);

   printf("\n\nMatriz ordenada: ");
   for (i = 0; i < 10; i++)
      printf("%d ", num[i]);

   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 seguinte ordena numericamente um vetor de dados definidos em uma `struct`.

```
#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];
   int i;

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

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

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

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

   return 0;
}

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

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

3) 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.

```
#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\nNAO achou\n\n");
   else
      printf("\n\nACHOU no codigo: %d\n\n", Resp->numero);


   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);
}
```

Exercícios

1) Faça um programa que recebe 20 números reais informados pelo usuário e os armazena em um vetor. Depois o programa deve imprimir todo o conteúdo do vetor, na ordem original digitada pelo usuário. Ordenar os dados em sentido crescente por meio da função `qsort` e imprimir o novo conteúdo do vetor.

2) 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 exibir o seu conteúdo na tela.

3) Adapte o programa a seguir para que ele utilize `qsort` para colocar os dados em ordem decrescente.

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

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

int main(void)
{  int i, num[10] = {2, 1, 6, 5, 8, 7, 9, 6, 2, 0};

   printf("Matriz original: ");
   for (i = 0; i < 10; i++)
      printf("%d ", num[i]);

   qsort(num, 10, sizeof(int), comp);

   printf("\n\nMatriz ordenada: ");
   for (i = 0; i < 10; i++)
      printf("%d ", num[i]);

   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;
}
```

4) Adapte o programa anterior para ele ordenar com `qsort` (em ordem crescente) apenas os valores dos 5 primeiros elementos do vetor. Os demais elementos devem ser ignorados. Com isso, o resultado final a ser exibido seria:

```
1 2 5 6 8 7 9 6 2 0
```

5) Adapte o programa do exercício 3 para ele ordenar com `qsort` (em ordem crescente) apenas os valores do vetor que estejam nos elementos com índices acima de 2. Os demais elementos devem ser ignorados. Com isso, o resultado final a ser exibido seria:

```
2 1 6 0 2 5 6 7 8 9
```

6) Adapte o programa do exercício 3 para ele ordenar com `qsort` (em ordem crescente) apenas os valores do vetor que estejam nos elementos com índices de 3 a 8. Os demais elementos devem ser ignorados. Com isso, o resultado final a ser exibido seria:

```
2 1 6 2 5 6 7 8 9 0
```

7) Adapte o programa a seguir para que ele utilize `qsort` para ordenar os dados da seguinte forma: os alunos de maior nota devem aparecer primeiro e, para aqueles que tiverem a mesma nota, mostrar os alunos em ordem alfabética do nome. Fazer uma única chamada de `qsort` para executar a ordenação.

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

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

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

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

   for (i = 0; i < 10; i++)
   {  scanf("%d", &vetor[i].numero);
      scanf("%s", vetor[i].nome);
      scanf("%f", &vetor[i].nota);
   }

   printf("Matriz original:\n");
   for (i = 0; i < 10; i++)
   {  printf("%d\t", vetor[i].numero);
      printf("%s\t", vetor[i].nome);
      printf("%.2f\n", vetor[i].nota);
   }

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

   printf("\n\nMatriz ordenada:\n");
   for (i = 0; i < 10; i++)
   {  printf("%d\t", vetor[i].numero);
      printf("%s\t", vetor[i].nome);
      printf("%.2f\n", vetor[i].nota);
   }

   return 0;
}
int comp(const void *p1, const void *p2)
{     ?????????????
}
```