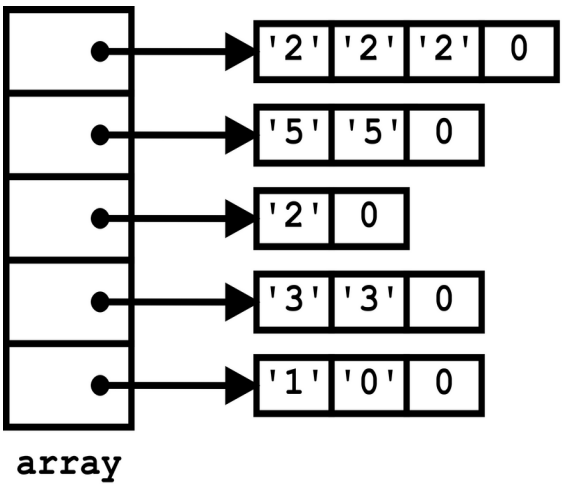
Ordenar uma sequência de *strings* por ordem alfabética .

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

void swap(char **a, char **b) {
      char *aux = *a;
      *a = *b;
      *b = aux;
}

void sort(char *array[], int size) {
      for (size_t i = 0; i < size - 1; ++i)
            for (size_t j = 0; j < size - i - 1; ++j)
                  if (strcmp(array[j], array[j + 1]) > 0)
                        swap(&array[j], &array[j + 1]);
}

void print(char *array[], size_t size) {
      putchar('\n');
      for (size_t i = 0; i < size; ++i)
            printf("%s\n", array[i]);
}

char *values[] = {
      "10",
      "33",
      "2",
      "55",
      "222"
};

int main() {
      print(values, ARRAY_SIZE(values));
      sort(values, ARRAY_SIZE(values));
      print(values, ARRAY_SIZE(values));
}
```



Ordenar uma sequência de *strings* pelo valor numérico que representam.

```
void sort(char *array[], int size) {
      for (size_t i = 0; i < size - 1; ++i)
            for (size_t j = 0; j < size - i - 1; ++j)
                  if (atoi(array[j]) > atoi(array[j + 1]))
                        swap(&array[j], &array[j + 1]);
}
```

Como programar uma função de ordenação cujo código sirva para os dois casos?

Parametrizar a operação de comparação! Como?

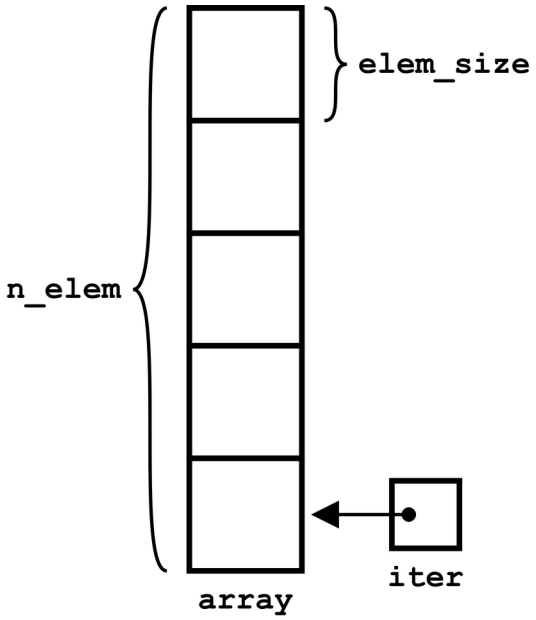
Programar a operação de comparação numa função. Fornecer esta função como argumento à função de ordenação, usando ponteiro para função.

(Na linguagem C não existem variáveis do tipo função mas existem variáveis do tipo ponteiro para função.)

```
void sort(char *array[], int size, int (*compare)(char *, char *)) {
      for (size_t i = 0; i < size - 1; ++i)
            for (size_t j = 0; j < size - i - 1; ++j)
                  if (compare(array[j], array[j + 1]) > 0)
                        swap(&array[j], &array[j + 1]);
}

int compare_lexical(char *a, char *b) {
      return strcmp(a, b);
}

int compare_numeric(char *a, char *b) {
      if (atoi(a) > atoi(b))
            return 1;
      else if (atoi(a) < atoi(b))
            return -1;
      else
            return 0;
}

#ifdef NUMERIC

int main() {
      print(values, ARRAY_SIZE(values));
      sort(values, ARRAY_SIZE(values), compare_numeric);
      print(values, ARRAY_SIZE(values));
}

#else

int main() {
      print(values, ARRAY_SIZE(values));
      sort(values, ARRAY_SIZE(values), compare_lexical);
      print(values, ARRAY_SIZE(values));
}

#endif
```

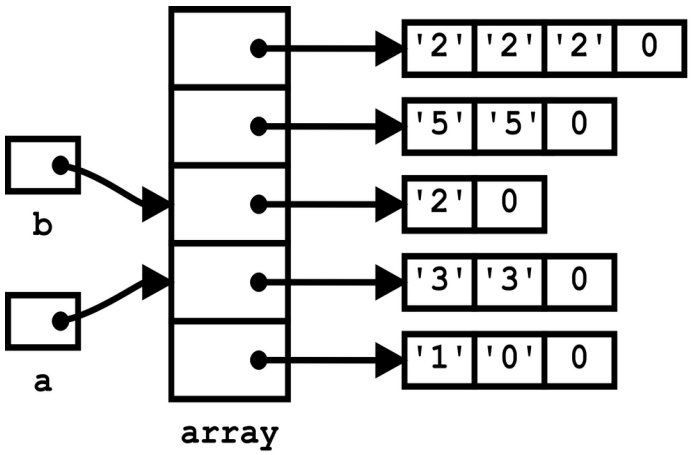
Pode levar- se a generalização mais longe, tornando o algoritmo independente do tipo dos elementos do *array*.

```
void swap(void *a, void *b, size_t elem_size) {
    char tmp[elem_size];
    memcpy(tmp, a, elem_size);
    memcpy(a, b, elem_size);
    memcpy(b, tmp, elem_size);
}

void sort(void *array, int n_elem, int elem_size, int (*compare)(void *, void *))
{
      for (size_t i = 0; i < n_elem - 1; ++i) {
            void *iter = array;
            for (size_t j = 0; j < n_elem - i - 1; ++j, iter += elem_size)
                  if (compare(iter, iter + elem_size) > 0)
                        swap(iter, iter + elem_size, elem_size);
      }
}
```



As funções de comparação não são genéricas, são específicas para o tipo dos elementos do *array*.

```
char *values[] = {
      "10",
      "33",
      "2",
      "55",
      "222"
};

int compare_lexical(void *a, void *b) {
      return strcmp(*(char **)a, *(char **)b);
}

int compare_numeric(void *a, void *b) {
      if (atoi(*(char **)a) > atoi(*(char **)b))
            return 1;
      else if (atoi(*(char **)a) < atoi(*(char **)b))
            return -1;
      else
            return 0;
}

int main() {
      print(values, ARRAY_SIZE(values));
      sort(values, ARRAY_SIZE(values), sizeof values[0], compare_numeric);
      print(values, ARRAY_SIZE(values));
}
```



A biblioteca normalizada da linguagem C possui duas funções genéricas para operar sobre *arrays* - **`qsort`** e **`bsearch`**.

```
void qsort(void *base, size_t num, size_t size,
           int (*compar)(const void *, const void *)) {
    if (num == 0)
        return;
    void *last = base + size * (num - 1);
    void *middle = base;
    for (void *iter = base + size; iter <= last; iter += size)
        if ((*compar)(iter, base) < 0)
            memswap(middle += size, iter, size);
    memswap(base, middle, size);
    qsort(base, (middle - base) / size, size, compar);
    qsort(middle + size, (last - middle) / size, size, compar);
}
```

```
const void* bsearch(const void* key, const void* base,
                    size_t num, size_t size,
                    int (*compar)(const void*,const void*)) {
    const void *left = base;
    const void *right = base + num * size;
    const void *middle = base + (num / 2) * size;
    while (left < right) {
        int cmp_result = compar(key, middle);
        if (cmp_result > 0)
            left = middle + size;
        else if (cmp_result < 0)
            right = middle;
        else
            return middle;
        middle = left + ((right - left) / size / 2) * size;
    }
    return NULL;
}
```

Exemplo 1

Ordenar um *array* de ponteiros para *string* utilizando a função **`qsort`** da biblioteca da linguagem C.

```
char *values[] = {
      "10",
      "33",
      "2",
      "55",
      "222"
};

int compare_lexical(const void *a, const void *b) {
      return strcmp(*(char **)a, *(char **)b);
}

int compare_numeric(const void *a, const void *b) {
      if (atoi(*(char **)a) > atoi(*(char **)b))
```

```
            return 1;
      else if (atoi(*(char **)a) < atoi(*(char **)b))
            return -1;
      else
            return 0;
}

int main() {
      qsort(values, ARRAY_SIZE(values), sizeof values[0], compare_lexical);
}
```

Exemplo 2

Ordenar um conjunto de *strings* armazenadas num *array* bidimensional.

```
char values[][5] = {
      "10",
      "33",
      "2",
      "55",
      "222"
};

int compare_lexical(const void *a, const void *b) {
      return strcmp((char *)a, (char *)b);
}

int compare_numeric(const void *a, const void *b) {
      if (atoi((char *)a) > atoi((char *)b))
            return 1;
      else if (atoi((char *)a) < atoi((char *)b))
            return -1;
      else
            return 0;
}

int main() {
      qsort(values, ARRAY_SIZE(values), sizeof values[0], compare_lexical);
}
```

Exemplo 3

Ordenar um *array* de **`struct person`**, segundo a ordem alfabética dos nomes.

```
typedef struct person {
      char name[20];
      int age;
      int weight;
      int height;
} Person;

Person people[] = {
      {"Manuel", 20, 70, 165},
      {"Xavier", 24, 68, 176},
      {"David", 22, 81, 182}
```

```
};

int compare_name(const void *a, const void *b) {
      return strcmp(((Person *)a)->name, ((Person *)b)->name);
}

int main() {
      qsort(people, ARRAY_SIZE(people), sizeof people[0], compare_name);
}
```

Exemplo 4

Ordenar um array de ponteiros para **`struct person`**, segundo a ordem alfabética dos nomes.

```
Person person1 = {"Manuel", 20, 70, 165};
Person person2 = {"Xavier", 24, 68, 176};
Person person3 = {"David", 22, 81, 182};

Person *people[] = {
      &person1,
      &person2,
      &person3
};

int compare_name(const void *a, const void *b) {
      return strcmp((*(Person * *)a)->name, (*(Person **)b)->name);
}

Person person = {"Ezequiel"};

Person *pperson = &person;

int main() {
      qsort(people, ARRAY_SIZE(people), sizeof people[0], compare_name);
}
```

Exercícios

1. Procurar uma dada *string* num *array* de ponteiros para *string*.
2. Procurar uma dada *string* num conjunto de *strings* armazenadas num *array* bidimensional.
3. Procurar uma dada pessoa num *array* de **`struct person`**, ordenado pela ordem alfabética dos nomes.
4. Procurar uma dada pessoa num *array* de ponteiros para **`struct person`**, ordenado pela ordem alfabética dos nomes.