```c
/**
 * file: sort.c
 *
 * Created by hengxin on 2023/12/13.
 */

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

// (since C11)
// _Generic ( controlling-expression , association-list )
// See Section 9.7 of the textbook
#define Print(x, y) _Generic((x), \
    int *: PrintInts, \
    const char **: PrintStrs \
    )((x), (y))

typedef int (*CompareFunction)(const void *, const void *);
typedef int CompFunc(const void *, const void *);

int CompareInts(const void *left, const void *right);
int CompareStrs(const void *left, const void *right);
int CompareStrsWrong(const void *left, const void *right);

void PrintInts(const int *integers, size_t len);
void PrintStrs(const char *str[], size_t len);

int main() {
  int integers[] = { -2, 99, 0, -743, 2, INT_MIN, 4 };
  int size_of_integers = sizeof integers / sizeof *integers;

  /**
   * void qsort( void *ptr, size_t count, size_t size,
   *          int (*comp)(const void *, const void *) );
   * typedef int _Cmpfun(const void *, const void *);
   * void qsort( void *ptr, size_t count, size_t size, _Cmpfun *comp);
   */
  int (*comp)(const void *, const void *) = CompareInts;

  // CompareFunction comp1 = CompareInts;
  // CompFunc *comp2 = CompareInts;

  // you should not do this!!!
  // printf("sizeof comp : %zu\n", sizeof comp);
  printf("comp : %p\n", comp);
  printf("*comp : %p\n", *comp);
  printf("CompareInts : %p\n", CompareInts);
  printf("&CompareInts : %p\n", &CompareInts);

  qsort(integers, size_of_integers, sizeof *integers, comp);
  // PrintInts(integers, size_of_integers);
```

```c
  Print(integers, size_of_integers);

  // Call functions indirectly via function pointers.
  int a = 10;
  int b = 20;
  printf("%d %s %d\n", a, comp(&a, &b) > 0 ? ">" : "<=", b);

  const char *names[] = {
      "Luo Dayou",
      "Cui Jian",
      "Dou Wei",
      "Zhang Chu",
      "Wan Qing",
      "Li Zhi",
      "Yao",
      "ZuoXiao",
      "ErShou Rose",
      "Hu Mage",
  };
  size_t size_of_names = sizeof names / sizeof *names;

  comp = CompareStrs;
  // qsort(names, size_of_names,
  //       sizeof *names, comp);
  // PrintStrs(names, size_of_names);

  // comp = CompareStrsWrong;
  comp = CompareStrs;
  qsort(names, size_of_names,
        sizeof *names, comp);
  // PrintStrs(names, size_of_names);
  Print(names, size_of_names);
}

int CompareInts(const void *left, const void *right) {
  int int_left = *(const int *) left;
  int int_right = *(const int *) right;

  if (int_left < int_right) {
    return -1;
  }

  if (int_left > int_right) {
    return 1;
  }

  return 0;

  // return (int_left > int_right) - (int_left < int_right);
  // return int_left - int_right; // erroneous shortcut (fails if INT_MIN is present)
}

```

```c
int CompareStrs(const void *left, const void *right) {
  const char *const *pp1 = left;
  const char *const *pp2 = right;
  return strcmp(*pp1, *pp2);
}

// Why keep the original order???
// What are compared???
int CompareStrsWrong(const void *left, const void *right) {
  const char *pp1 = left;
  const char *pp2 = right;
  return strcmp(pp1, pp2);
}

void PrintInts(const int *integers, size_t len) {
  printf("\n");
  for (int i = 0; i < len; i++) {
    printf("%d ", integers[i]);
  }
  printf("\n");
}

void PrintStrs(const char *str[], size_t len) {
  printf("\n");
  for (int i = 0; i < len; i++) {
    printf("%s\n", str[i]);
  }
  printf("\n");
}
```