

```

1  /**
2   * file: sort.c
3   *
4   * Created by hengxin on 2023/12/13.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <limits.h>
10 #include <string.h>
11
12 // (since C11)
13 // _Generic ( controlling-expression , association-list )
14 // See Section 9.7 of the textbook
15 #define Print(x, y) _Generic((x), \
16     int *: PrintInts, \
17     const char **: PrintStrs \
18     )((x), (y))
19
20 typedef int (*CompareFunction)(const void *, const void *);
21 typedef int CompFunc(const void *, const void *);
22
23 int CompareInts(const void *left, const void *right);
24 int CompareStrs(const void *left, const void *right);
25 int CompareStrsWrong(const void *left, const void *right);
26
27 void PrintInts(const int *integers, size_t len);
28 void PrintStrs(const char *str[], size_t len);
29
30 int main() {
31     int integers[] = { -2, 99, 0, -743, 2, INT_MIN, 4 };
32     int size_of_integers = sizeof integers / sizeof *integers;
33
34     /**
35      * void qsort( void *ptr, size_t count, size_t size,
36                  int (*comp)(const void *, const void *) );
37      * typedef int _Cmpfun(const void *, const void *);
38      * void qsort( void *ptr, size_t count, size_t size, _Cmpfun *comp);
39      */
40     int (*comp)(const void *, const void *) = CompareInts;
41
42     // CompareFunction comp1 = CompareInts;
43     // CompFunc *comp2 = CompareInts;
44
45     // you should not do this!!!
46     // printf("sizeof comp : %zu\n", sizeof comp);
47     printf("comp : %p\n", comp);
48     printf("*comp : %p\n", *comp);
49     printf("CompareInts : %p\n", CompareInts);
50     printf("&CompareInts : %p\n", &CompareInts);
51
52     qsort(integers, size_of_integers, sizeof *integers, comp);
53     // PrintInts(integers, size_of_integers);

```

```

54     Print(integers, size_of_integers);
55
56     // Call functions indirectly via function pointers.
57     int a = 10;
58     int b = 20;
59     printf("%d %s %d\n", a, comp(&a, &b) > 0 ? ">" : "<=", b);
60
61     const char *names[] = {
62         "Luo Dayou",
63         "Cui Jian",
64         "Dou Wei",
65         "Zhang Chu",
66         "Wan Qing",
67         "Li Zhi",
68         "Yao",
69         "ZuoXiao",
70         "ErShou Rose",
71         "Hu Mage",
72     };
73     size_t size_of_names = sizeof names / sizeof *names;
74
75     comp = CompareStrs;
76     // qsort(names, size_of_names,
77     //         sizeof *names, comp);
78     // PrintStrs(names, size_of_names);
79
80     // comp = CompareStrsWrong;
81     comp = CompareStrs;
82     qsort(names, size_of_names,
83           sizeof *names, comp);
84     // PrintStrs(names, size_of_names);
85     Print(names, size_of_names);
86 }
87
88 int CompareInts(const void *left, const void *right) {
89     int int_left = *(const int *) left;
90     int int_right = *(const int *) right;
91
92     if (int_left < int_right) {
93         return -1;
94     }
95
96     if (int_left > int_right) {
97         return 1;
98     }
99
100    return 0;
101
102    // return (int_left > int_right) - (int_left < int_right);
103    // return int_left - int_right; // erroneous shortcut (fails if
104    INT_MIN is present)
105 }

```

```
106 int CompareStrs(const void *left, const void *right) {
107     const char *const *pp1 = left;
108     const char *const *pp2 = right;
109     return strcmp(*pp1, *pp2);
110 }
111
112 // Why keep the original order???
113 // What are compared???
114 int CompareStrsWrong(const void *left, const void *right) {
115     const char *pp1 = left;
116     const char *pp2 = right;
117     return strcmp(pp1, pp2);
118 }
119
120 void PrintInts(const int *integers, size_t len) {
121     printf("\n");
122     for (int i = 0; i < len; i++) {
123         printf("%d ", integers[i]);
124     }
125     printf("\n");
126 }
127
128 void PrintStrs(const char *str[], size_t len) {
129     printf("\n");
130     for (int i = 0; i < len; i++) {
131         printf("%s\n", str[i]);
132     }
133     printf("\n");
134 }
```