

```
1 // Created by hengxin on 2024/12/04.
2
3 int main() {
4     char **argv;
5
6     int *names[10];
7
8     int (*musician_score_table)[10];
9
10    char *StrCpyStd(char *dest, const char *src);
11
12    int (*comp)(const void *left, const void *right);
13
14    // see https://en.cppreference.com/w/c/program/atexit
15    int atexit(void (*func)(void));
16
17    // see https://en.cppreference.com/w/c/program/signal
18    void (*signal(int sig, void (*handler)(int)))(int);
19
20    // typedef void (*sighandler_t)(int);
21    // sighandler_t Signal(int sig, sighandler_t handler);
22
23    char ((*func(int num, char *str))[3])(void);
24
25    char ((*arr[3])(void))[5];
26
27    // Refer to https://cdecl.org/ for more practice.
28    // See https://c-faq.com/decl/spiral.anderson.html for
    secrets!!!
29 }
```

```

1 // Created by hengxin on 2024/12/04.
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <limits.h>
6 #include <string.h>
7
8 // (since C11)
9 // _Generic ( controlling-expression , association-list )
10 // See Section 9.7 of the textbook
11 #define Print(x, y) _Generic((x), \
12     int *: PrintInts, \
13     const char **: PrintStrs \
14     )((x), (y))
15
16 typedef int (*CompareFunction)(const void *, const void
    *);
17 typedef int CompFunc(const void *, const void *);
18
19 int CompareInts(const void *left, const void *right);
20 int CompareStrs(const void *left, const void *right);
21 int CompareStrsWrong(const void *left, const void *right);
22
23 int StrCmpStd(const char *s1, const char *s2);
24
25 void PrintInts(const int *integers, size_t len);
26 void PrintStrs(const char *str[], size_t len);
27
28 int main() {
29     int integers[] = {-2, 99, 0, -743, 2, INT_MIN, 4};
30     int size_of_integers = sizeof integers / sizeof *
        integers;
31
32     /**
33      * void qsort( void *ptr, size_t count, size_t size,
34      *             int (*comp)(const void *, const void *) );
35      * typedef int _Cmpfun(const void *, const void *);
36      * void qsort( void *ptr, size_t count, size_t size,
37      *             _Cmpfun *comp);
38      */
39     int (*comp)(const void *, const void *) = CompareInts;
40
41     // CompareFunction comp1 = CompareInts;
42     // CompFunc *comp2 = CompareInts;

```

```
42
43 // you should not do this!!!
44 // printf("sizeof comp : %zu\n", sizeof comp);
45 printf("comp : %p\n", comp);
46 printf("*comp : %p\n", *comp);
47 printf("CompareInts : %p\n", CompareInts);
48 printf("&CompareInts : %p\n", &CompareInts);
49
50 qsort(integers, size_of_integers, sizeof *integers, comp
);
51 // PrintInts(integers, size_of_integers);
52 Print(integers, size_of_integers);
53
54 // Call functions indirectly via function pointers.
55 int a = 10;
56 int b = 20;
57 printf("%d %s %d\n", a, comp(&a, &b) > 0 ? ">" : "<=", b
);
58
59 const char *names[] = {
60     "Luo Dayou",
61     "Cui Jian",
62     "Dou Wei",
63     "Zhang Chu",
64     "Wan Qing",
65     "Li Zhi",
66     "Yao",
67     "ZuoXiao",
68     "ErShou Rose",
69     "Hu Mage",
70 };
71 size_t size_of_names = sizeof names / sizeof *names;
72
73 comp = CompareStrs;
74 // qsort(names, size_of_names,
75 //     sizeof *names, comp);
76 // PrintStrs(names, size_of_names);
77
78 // comp = CompareStrsWrong;
79 comp = CompareStrs;
80 qsort(names, size_of_names,
81     sizeof *names, comp);
82 // PrintStrs(names, size_of_names);
83 Print(names, size_of_names);
```

```
84 }
85
86 int CompareInts(const void *left, const void *right) {
87     int int_left = *(const int *) left;
88     int int_right = *(const int *) right;
89
90     if (int_left < int_right) {
91         return -1;
92     }
93
94     if (int_left > int_right) {
95         return 1;
96     }
97
98     return 0;
99
100     // return (int_left > int_right) - (int_left <
    int_right);
101     // return int_left - int_right; // erroneous shortcut (
    fails if INT_MIN is present)
102 }
103
104 int CompareStrs(const void *left, const void *right) {
105     const char *const *pp1 = left;
106     const char *const *pp2 = right;
107     return strcmp(*pp1, *pp2);
108 }
109
110 // Why keep the original order???
111 // What are compared???
112 int CompareStrsWrong(const void *left, const void *right
    ) {
113     const char *pp1 = left;
114     const char *pp2 = right;
115     return strcmp(pp1, pp2);
116 }
117
118 int StrCmpStd(const char *s1, const char *s2) {
119     for (; *s1 == *s2; s1++, s2++) {
120         if (*s1 == '\0') {
121             return 0;
122         }
123     }
124 }
```

```
125 // just for debug
126 const unsigned char s1_char = *((const unsigned char
    *) s1);
127 const unsigned char s2_char = *((const unsigned char
    *) s2);
128
129 return *((const unsigned char *) s1) -
130         *((const unsigned char *) s2);
131 }
132
133 void PrintInts(const int *integers, size_t len) {
134     printf("\n");
135     for (int i = 0; i < len; i++) {
136         printf("%d ", integers[i]);
137     }
138     printf("\n");
139 }
140
141 void PrintStrs(const char *str[], size_t len) {
142     printf("\n");
143     for (int i = 0; i < len; i++) {
144         printf("%s\n", str[i]);
145     }
146     printf("\n");
147 }
```

```
1 // Created by hfwei on 2024/12/04.
2 // See https://en.cppreference.com/w/c/program/atexit
3
4 #include <stdlib.h>
5 #include <stdio.h>
6
7 void f1(void) {
8     puts("f1");
9 }
10
11 void f2(void) {
12     puts("f2");
13 }
14
15 int main(void) {
16     if (!atexit(f1) && !atexit(f2) && !atexit(f2)) {
17         return EXIT_SUCCESS;
18     }
19
20     // atexit registration failed
21     return EXIT_FAILURE;
22
23 } // <- if registration was successful calls f2, f2, f1
```

```
1 // Created by hfwei on 2024/12/04.
2
3 #include <stdio.h>
4 #include <signal.h>
5
6 void SIGSEGV_Handler(int sig) {
7     printf("SIGSEGV %d is caught.\n", sig);
8 }
9
10 int main(void) {
11     signal(SIGSEGV, SIGSEGV_Handler);
12     // raise(SIGSEGV);
13
14     int *p = NULL;
15     *p = 0;
16
17     return 0;
18 }
```

```
1 // Created by hfwei on 2024/12/04.
2
3 #include <stdio.h>
4
5 // See https://elixir.bootlin.com/linux/latest/source/
   include/linux/types.h#L245
6 typedef int (*cmp_func_t)(const void *a, const void *b);
7
8 // See https://elixir.bootlin.com/linux/latest/source/
   include/linux/bsearch.h#L8
9 void *bsearch(const void *key, const void *base,
10              size_t num, size_t size, cmp_func_t cmp);
11
12 int main(void) {
13
14     return 0;
15 }
16
17 void *bsearch(const void *key, const void *base, size_t
   num, size_t size, cmp_func_t cmp) {
18     const char *pivot;
19     int result;
20
21     while (num > 0) {
22         pivot = base + (num >> 1) * size;
23         result = cmp(key, pivot);
24
25         if (result == 0) {
26             return (void *) pivot;
27         }
28
29         if (result > 0) {
30             base = pivot + size;
31             num--;
32         }
33
34         num >>= 1;
35     }
36
37     return NULL;
38 }
```



```
1 # `11-function-pointers`  
2  
3 ## `integrate.c`  
4  
5 ## `sort.c`  
6  
7 ## `bsearch-gnuc.c`  
8  
9 ## `decl.c`
```

```
1 // Created by hfwei on 2024/12/04.
2 // A nice function pointer example on Riemann integration:
3 // https://en.wikipedia.org/wiki/Function\_pointer
4
5 #include <stdio.h>
6 #include <math.h>
7
8 #define NUM_OF_PARTITIONS 1000000
9
10 double Integrate(double low, double high, double (*func)(
    double));
11
12 double Square(double x);
13
14 int main() {
15     double low = 0.0;
16     double high = 1.0;
17     double integration = 0.0;
18
19     // gcc -pedantic (invalid application of sizeof to a
    function type)
20     // See "Function to pointer conversion" (https://en.
    cppreference.com/w/c/language/conversion)
21     // See also https://en.cppreference.com/w/c/language/
    sizeof
22     printf("sizeof sin: %zu\n", sizeof sin); // not allowed
23     printf("sizeof &sin: %zu\n", sizeof &sin);
24
25     integration = Integrate(low, high, sin);
26     printf("sin(x) from %f to %f is %f\n", low, high,
    integration);
27
28     integration = Integrate(low, high, cos);
29     printf("cos(x) from %f to %f is %f\n", low, high,
    integration);
30
31     integration = Integrate(low, high, Square);
32     printf("Square(x) from %f to %f is %f\n", low, high,
    integration);
33
34     double (*funcs[])(double) = {sin, cos, Square};
35
36     int len = sizeof(funcs) / sizeof(*funcs);
37     for (int i = 0; i < len; ++i) {
```

```
38     printf("integration: %f\n", Integrate(low, high, funcs
    [i]));
39 }
40
41 return 0;
42 }
43
44 double Integrate(double low, double high, double (*func)(
    double)) {
45     double interval = (high - low) / NUM_OF_PARTITIONS;
46
47     double sum = 0.0;
48     for (int i = 0; i < NUM_OF_PARTITIONS; i++) {
49         double xi = low + interval * i;
50         double yi = func(xi);
51         sum += yi * interval;
52     }
53
54     return sum;
55 }
56
57 double Square(double x) {
58     return x * x;
59 }
```

```

1 // Created by hfwei on 2024/12/04.
2 // Question: What if char key_name[] = "Zhang Chu"?
3
4 #include <stdio.h>
5 #include <string.h>
6 #include <stdbool.h>
7
8 // See https://codebrowser.dev/glibc/glibc/stdlib/stdlib.h
  .html#__compar_fn_t
9 // The first is a pointer to the key for the search,
10 // and the second is a pointer to the array element to be
  compared with the key.
11 typedef int (*__compar_fn_t)(const void *, const void *);
12
13 // See https://codebrowser.dev/glibc/glibc/bits/stdlib-
  bsearch.h.html#19
14 void *bsearch(const void *__key, const void *__base,
15               size_t __nmemb, size_t __size,
16               __compar_fn_t __compar);
17 void *bsearch_leftmost(const void *__key, const void *
  __base,
18                        size_t __nmemb, size_t __size,
19                        __compar_fn_t __compar);
20
21 int CompareStrs(const void *left, const void *right);
22 int CompareStrsCI(const void *left, const void *right);
23 int CompareStrsAddress(const void *left, const void *right
  );
24
25 // int (*GetCompareFunction(bool case_sensitive))(const
  void *, const void *);
26 __compar_fn_t GetCompareFunction(bool case_sensitive) {
27     return case_sensitive ? &CompareStrs : &CompareStrsCI;
28 }
29
30 const char *names[] = {
31     "Cui Jian",
32     "Dou Wei",
33     "ErShou Rose",
34     "Hu Mage",
35     "Li Zhi",
36     "Luo Dayou",
37     "Wan Qing",
38     "Yao",

```

```

39     "Zhang Chu",
40     "Zhang Chu",
41     "Zhang Chu",
42     "Zhang Chu",
43     "ZuoXiao",
44 };
45
46 int main(void) {
47     char *key_name = "Zhang Chu";
48
49     // char **name_ptr = bsearch(&key_name, names,
50     //                             sizeof names / sizeof *
51     //                             names,
52     //                             sizeof *names,
53     //                             CompareStrs);
54
55     // char **name_ptr = bsearch(&key_name, names,
56     //                             sizeof names / sizeof *
57     //                             names,
58     //                             sizeof *names,
59     //                             CompareStrsAddress);
60
61     char **name_ptr = bsearch_leftmost(&key_name, names,
62                                         sizeof names / sizeof
63                                         *names,
64                                         sizeof *names,
65                                         CompareStrsAddress);
66
67     if (name_ptr != NULL) {
68         printf("Found %s at index %lld.\n",
69               *name_ptr, name_ptr - (char **) names);
70     } else {
71         printf("Could not find %s.\n", key_name);
72     }
73
74     char *key_name_ci = "zhang chu";
75
76     char **name_ci_ptr = bsearch(&key_name_ci, names,
77                                   sizeof names / sizeof *
78                                   names,
79                                   sizeof *names,
80                                   GetCompareFunction(false));
81
82     if (name_ci_ptr != NULL) {
83         printf("Found %s at index %lld.\n",

```

```

79         *name_ci_ptr,
80         name_ci_ptr - (char **) names);
81     } else {
82         printf("Could not find %s.\n", key_name_ci);
83     }
84
85     return 0;
86 }
87
88 int CompareStrs(const void *left, const void *right) {
89     char *const *pp1 = left;
90     char *const *pp2 = right;
91     return strcmp(*pp1, *pp2);
92 }
93
94 int CompareStrsCI(const void *left, const void *right) {
95     const char *const *pp1 = left;
96     const char *const *pp2 = right;
97     // see https://www.ibm.com/docs/en/zos/2.4.0?topic=functions-strcasecmp-case-insensitive-string-comparison
98     return strcasecmp(*pp1, *pp2);
99 }
100
101 // What is the advantage of this version? (performance ???)
102 // What is the disadvantage of this version? (not flexible???)
103 int CompareStrsAddress(const void *left, const void *right) {
104     const char *pp1 = left;
105     const char *pp2 = right;
106     return strcmp(pp1, pp2);
107 }
108
109 void *bsearch(const void *__key, const void *__base,
110              size_t __nmem, size_t __size,
111              __compar_fn_t __compar) {
112     size_t __l, __u, __idx;
113     const void *__p;
114     int __comparison;
115     __l = 0;
116     __u = __nmem;
117     while (__l < __u) {

```

```

118     __p = (const void *) (((const char *) __base) + (
    __idx * __size));
119     __comparison = (*__compar)(__key, __p);
120     if (__comparison < 0) {
121         __u = __idx;
122     } else if (__comparison > 0) {
123         __l = __idx + 1;
124     } else {
125         return (void *) __p;
126     }
127 }
128
129 return NULL;
130 }
131
132 void *bsearch_leftmost(const void *__key, const void *
    __base,
133                        size_t __nmemb, size_t __size,
134                        __compar_fn_t __compar) {
135     size_t __l, __u, __idx;
136     const void *__p;
137     int __comparison;
138
139     __l = 0;
140     __u = __nmemb;
141     // added by ant
142     void *__index = NULL;
143
144     while (__l < __u) {
145         __idx = (__l + __u) / 2;
146         __p = (const void *) (((const char *) __base) + (
    __idx * __size));
147         __comparison = (*__compar)(__key, __p);
148         if (__comparison < 0) {
149             __u = __idx;
150         } else if (__comparison > 0) {
151             __l = __idx + 1;
152         } else {
153             // added by ant
154             __index = (void *) __p;
155             __u = __idx - 1;
156         }
157     }
158

```

```
159 // added by ant
160 return __index;
161 }
```



```
1 add_executable(integrate integrate.c)
2
3 add_executable(sort sort.c)
4
5 add_executable(bsearch bsearch.c)
6 add_executable(bsearch-gnuc bsearch-gnuc.c)
7
8 add_executable(11-decl decl.c)
9 add_executable(atexit atexit.c)
10 add_executable(signal signal.c)
```