```
1 //
 2 // Created by hengxin on 10/19/22.
 3 //
 5 #include <stdio.h>
 6 #include <stdlib.h>
 8 #define LEN_L 4
9 #define LEN_R 5
11 int *Merge(const int L[], int llen, const int R[], int rlen);
13 int main() {
     int L[LEN_L] = \{2, 4, 6, 8\};
14
15
     int R[LEN_R] = \{1, 3, 5, 7, 9\};
16
17
     int *merge = Merge(L, LEN_L, R, LEN_R);
18
19
     for (int i = 0; i < LEN_L + LEN_R; i++) {
20
       printf("%d ", merge[i]);
21
22
23
     return 0;
24 }
25
26 int *Merge(const int L[], int llen, const int R[], int rlen) {
27
     int *merge = malloc((llen + rlen) * sizeof *merge);
28
29
     int l = 0;
30
     int r = 0;
31
     int m = 0;
32
33
     while (l < llen && r < rlen) {
34
       if (L[l] <= R[r]) {
35
         merge[m++] = L[l];
36
         l++;
       } else { // L[l] > R[r]
37
38
         merge[m++] = R[r];
39
         r++;
40
       }
41
     }
42
43
     while (l < llen) {
44
       merge[m++] = L[l];
45
       l++;
46
     }
47
48
     while (r < rlen) {</pre>
49
       merge[m++] = R[r];
50
       r++;
51
     }
52
53
     return merge;
```

	D:\cpl\cpl-coding-0\2022-CPL\10-double-pointers\merge	С
5	4 }	
"	' '	
1		

```
1 //
 2 // Created by hfwei on 2022/12/8.
3 //
 5 #include <stdio.h>
 6 #include <math.h>
8 #define NUM_OF_PARTITIONS 10000
10 typedef double (*fp)(double);
11
12 double Integrate(double low, double high, double (*fp)(double));
13 double Square(double x);
14
15 int main() {
16
     double low = 0.0;
17
     double high = 1.0;
18
     double integration = 0.0;
19
20
     integration = Integrate(low, high, Square);
21
     printf("Integrate(%f, %f, Square) = %f\n", low, high, integration);
22
23
     integration = Integrate(low, high, sin);
24
     printf("Integrate(%f, %f, sin) = %f\n", low, high, integration);
25
26
     integration = Integrate(low, high, cos);
27
     printf("Integrate(%f, %f, cos) = %f\n", low, high, integration);
28
29
    // double (*functions[3])(double) = {Square, sin, cos};
30
     fp functions[3] = {Square, sin, cos};
31
     for (int i = 0; i < 3; i++) {
32
       integration = Integrate(low, high, functions[i]);
33
       printf("integration = %f\n", integration);
34
     }
35
36
     return 0;
37 }
38
39 double Integrate(double low, double high, double (*fp)(double)) {
40
     double interval = (high - low) / NUM_OF_PARTITIONS;
41
     double sum = 0.0;
42
43
     for (int i = 0; i < NUM_OF_PARTITIONS; i++) {</pre>
44
       double x = low + interval * i;
       double y = fp(x);
45
46
       sum += y * interval;
47
48
49
     return sum;
50 }
51
52 double Square(double x) {
53
     return x * x;
```

File - D:\cpl\cpl-coding-0\2022-CPL\10-c	ouble-pointers\integrate.c	
54 }		

```
1 /**
 2 * file: sort.c
 3 *
 4 * Created by hengxin on 12/01/22.
 5 *
 6 * A nice function pointer example on Riemann integration:
 7 * https://en.wikipedia.org/wiki/Function_pointer
 9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <limits.h>
13 #include <string.h>
14
15 int CompareInts(const void *left, const void *right);
16 void PrintInts(const int *integers, int len);
17
18 int CompareStrs(const void *left, const void *right);
19 void PrintStrs(const char *str[], int len);
20
21 int main() {
22
    // sort an array of integers
23
     int integers[] = {-2, 99, 0, -743, 2, INT_MIN, 4};
24
     int size_of_integers = sizeof integers / sizeof *integers;
25
26
    /**
27
    * void gsort( void *ptr, size_t count, size_t size,
28
               int (*comp)(const void *, const void *) );
29
     */
30
     int (*comp)(const void *, const void *) = CompareInts;
31
     // you should not do this!!!
32
     // printf("sizeof comp : %zu\n", sizeof comp);
33
     printf("comp : %p\n", comp);
34
     printf("*comp : %p\n", *comp);
     printf("CompareInts : %p\n", CompareInts);
35
36
     printf("&CompareInts : %p\n", &CompareInts);
37
38
     gsort(integers, size_of_integers, sizeof *integers, comp);
39
     PrintInts(integers, size_of_integers);
40
41
     // Call functions indirectly via function pointers.
42
     int a = 10;
43
     int b = 20;
44
     printf("%d %s %d\n", a, comp(&a, &b) > 0 ? ">" : "<=", b);
45
46
    // Sorting an array of strings
47
     const char *names[] = {
48
         "Luo Dayou",
49
         "Cui Jian",
50
         "Dou Wei",
51
         "Zhang Chu",
52
         "He Yong",
53
         "Wan Qing",
```

```
54
          "WuTiaoRen",
 55
          "ZuoXiao",
 56
          "Hu Mage",
 57
          "Li Zhi"
 58
      };
 59
     int size_of_names = sizeof names / sizeof *names;
 60
 61
      comp = CompareStrs;
      qsort(names, size_of_names, sizeof *names, comp);
 62
      PrintStrs(names, size_of_names);
 63
 64 }
 65
 66 int CompareInts(const void *left, const void *right) {
      int int_left = *(const int *) left;
 67
 68
      int int_right = *(const int *) right;
 69
 70
      if (int_left < int_right) {</pre>
 71
       return -1;
 72
     }
 73
 74
      if (int_left > int_right) {
 75
      return 1;
 76
     }
 77
 78
    return 0;
 79
 80 // return int_left - int_right; // erroneous shortcut (fails if
    INT_MIN is present)
 81 }
 82
 83 // actual arguments: char *const *
 84 int CompareStrs(const void *left, const void *right) {
     char *const *pp1 = left;
      char *const *pp2 = right;
 87
     return strcmp(*pp1, *pp2);
 88 }
 89
 90 void PrintInts(const int *integers, int len) {
 91
      printf("\n");
     for (int i = 0; i < len; ++i) {
        printf("%d ", integers[i]);
 93
 94
 95
      printf("\n");
 96 }
 97
 98 void PrintStrs(const char *str[], int len) {
      printf("\n");
99
      for (int i = 0; i < len; i++) {
100
        printf("%s\n", str[i]);
101
102
103
     printf("\n");
104 }
```