

```

1  /**
2   * file: pointer.c
3   *
4   * Created by hengxin on 11/28/21.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9
10 int main() {
11     /****** On radius *****/
12     int radius = 100;
13
14     printf("radius = %d\n", radius);
15
16     // every variable has an address
17     // &: address-of operator ("&")
18     printf("The address of radius is %p\n", &radius);
19     // we have already used the address of a variable before
20     // scanf("%d", &radius);
21
22     // radius as a left value; refer to its address (the storage space)
23     radius = 200;
24     // radius as a right value; refer to its value
25     double circumference = 2 * 3.14 * radius;
26     printf("radius = %d; circumference = %f\n", radius, circumference);
27     /****** On radius *****/
28
29     /****** On ptr_radius1 *****/
30     // ptr_radius1 is a variable of type "pointer to int"
31     int *ptr_radius1 = &radius;
32     // ptr_radius1 is a variable: has its value
33     printf("ptr_radius1 = %p\n", ptr_radius1);
34     // ptr_radius1 is a variable: has its address
35     printf("The address of ptr_radius1 is %p\n", &ptr_radius1);
36     /****** On ptr_radius1 *****/
37
38     /****** On *ptr_radius1 *****/
39     // IMPORTANT:
40     // *ptr_radius1: behaves just like radius
41     // type: int; value: the value of radius; address: the address of
42     radius
43     // *: indirection/dereference operator ("&"/"")
44     printf("radius = %d\n", *ptr_radius1);
45     // *ptr_radius1 as a right value
46     circumference = 2 * 3.14 * (*ptr_radius1);
47     // take the address of *ptr_radius1
48     // &*ptr_radius1 is the same as ptr_radius1
49     printf("The address of *ptr_radius1 is %p\n", &*ptr_radius1);
50     // *ptr_radius1 as a left value
51     *ptr_radius1 = 100;
52     printf("radius = %d\n", *ptr_radius1);
53     /****** On *ptr_radius1 *****/

```

```

53
54  /***** On ptr_radius1 again *****/
55  // ptr_radius1 as a left value
56  int radius2 = 200;
57  int *ptr_radius2 = &radius2;
58
59  ptr_radius1 = ptr_radius2;
60  printf("radius = %d\n", *ptr_radius1);
61
62  // ptr_radius1 as a right value
63  ptr_radius2 = ptr_radius1;
64  printf("radius = %d\n", *ptr_radius2);
65  /***** On ptr_radius1 again *****/
66
67  /***** On array names *****/
68  int numbers[5] = {0};
69  // vs. numbers[2] = {2};
70  // numbers++;
71  // numbers = &radius;
72  int *ptr_array = numbers;
73  ptr_array++;
74  /***** On array names *****/
75
76  /***** On malloc/free *****/
77  // undefined behavior
78  // free(numbers);
79  /***** On malloc/free *****/
80
81  /***** On const *****/
82  // const int * and int const *
83  // You cannot modify the value pointed to by ptr_radius3
84  // through the pointer (without casting the constness away).
85  const int *ptr_radius3 = &radius;
86  // *ptr_radius is read-only
87  // *ptr_radius3 = 300;
88  // You are allowed to do this, but you should not do it!
89  int *ptr_radius4 = ptr_radius3;
90  *ptr_radius4 = 400;
91  printf("radius = %d\n", radius);
92
93  // int * const
94  int *const ptr_radius5 = &radius;
95  // ptr_radius5 = ptr_radius3;
96  *ptr_radius5 = 500;
97  printf("radius = %d\n", radius);
98
99  // const int * const
100 const int *const ptr_radius6 = &radius;
101 // ptr_radius6 = ptr_radius3;
102 // *ptr_radius6 = 600;
103 /***** On const *****/
104 }

```

```
1 //
2 // Created by hengxin on 10/19/22.
3 //
4
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 #define LEN_L 4
9 #define LEN_R 5
10
11 int *Merge(const int L[], int llen, const int R[], int rlen);
12
13 int main() {
14     int L[LEN_L] = {2, 4, 6, 8};
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++;
37         } else { // L[l] > R[r]
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) {
49         merge[m++] = R[r];
50         r++;
51     }
52
53     return merge;
```

54 }

```

1 // Created by hfwei on 2022/10/13.
2 //
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 void WrongSwap(int left, int right);
8 void Swap(int *left, int *right);
9 void Print(const int *arr, int len);
10 void SelectionSort(int *arr, int len);
11
12 int main() {
13     int len = 0;
14     printf("Please input the length of the array to sort.\n");
15     scanf("%d", &len);
16
17     // void *: the type for a generic pointer (replacing char *)
18     // include stdlib.h (not malloc.h which is deprecated) for malloc
19     int *numbers = malloc(len * sizeof(*numbers));
20     // null pointer: not the same with any non-null pointers
21     // #define NULL ((void *) 0); but do not rely on it
22     if (numbers == NULL) {
23         printf("Error! Memory Not Allocated!\n");
24         return 0;
25     }
26
27     printf("Please input %d integers.\n", len);
28     for (int i = 0; i < len; i++) {
29         scanf("%d", numbers + i);
30     }
31
32     Print(numbers, len);
33     // numbers: the address of the first element of the `numbers` array
34     // pass by value: the copy of the address of the first element of
    the `numbers` array
35     SelectionSort(numbers, len);
36     Print(numbers, len);
37
38     // avoid memory leak (why memory leak?)
39     free(numbers);
40     // undefined behavior
41     // free(numbers);
42     // numbers[5] = 5;
43 }
44
45 void Print(const int *arr, int len) {
46     printf("\n");
47     for (int i = 0; i < len; i++) {
48         printf("%d ", arr[i]);
49     }
50     printf("\n");
51 }
52

```

```
53 // arr: the (copy of the) address of the first element of the `
    numbers` array
54 void SelectionSort(int *arr, int len) {
55     for (int i = 0; i < len; i++) {
56         // find the minimum of numbers[i .. len - 1]
57         // arr[i] is a syntactic sugar for *(arr + i)
58         int min = *(arr + i);
59         int min_index = i;
60         for (int j = i + 1; j < len; j++) {
61             if (*(arr + j) < min) {
62                 min = *(arr + j);
63                 min_index = j;
64             }
65         }
66
67         // swap arr[i] and arr[min_index]
68         // WrongSwap(arr[i], arr[min_index]);
69         // &*(arr + i))
70         // Swap(&arr[i], &arr[min_index]);
71         // &arr[i] is the same as (arr + i)
72         Swap(arr + i, arr + min_index);
73     }
74 }
75
76 void Swap(int *left, int *right) {
77     int temp = *left;
78     *left = *right;
79     *right = temp;
80 }
81
82 // Wrong WrongSwap: does not work
83 void WrongSwap(int left, int right) {
84     int tmp = left;
85     left = right;
86     right = tmp;
87 }
```