

```

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

```

```

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

```

File - D:\cpl\2023-cpl-coding-0\8-pointers-arrays\radius.c

```
106  
107     return 0;  
108 }
```