

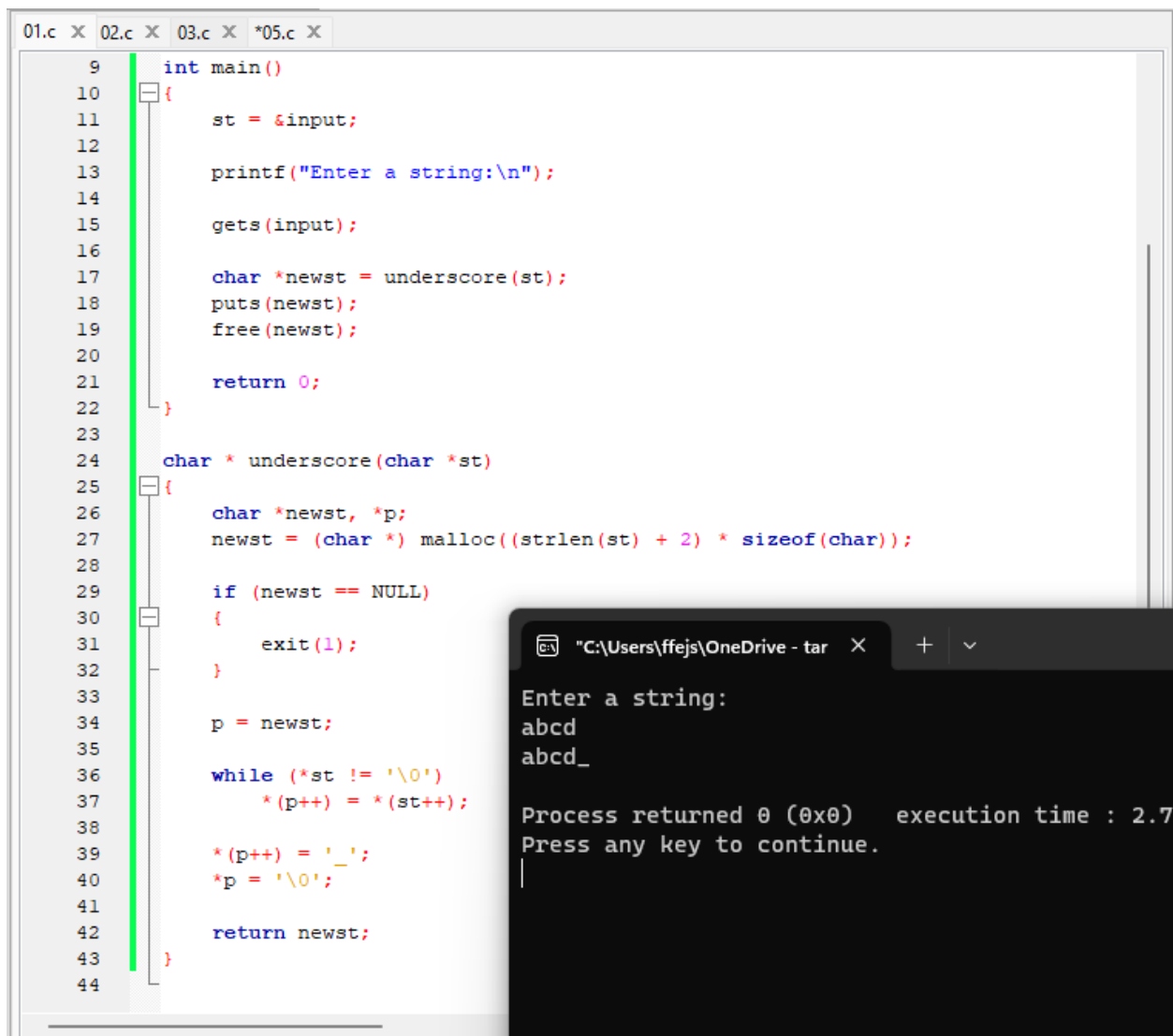
Strings as function arguments, malloc, and structures

1] Write a function **underscore()** that accepts one string **st** as argument.

- The length of **st** is unknown in advance. Use the library function **strlen()** to determine it.
 - Don't forget to include **string.h**!
- Use **malloc()** to allocate enough memory to hold a string that is one character longer than **st**.
- Copy **st** to the block of memory allocated, and add the character **underscore _** at the end.
 - Use a loop to copy character-by-character. What type of loop?
 - Don't forget the string terminator!
- **underscore()** returns a pointer to the new string.

In the main program, ask the user to enter a string, and then print the string returned by **underscore()**; for example, if the user enters **abcd**, the program prints **abcd_**

Remember to liberate the allocated string before ending the program.



```
01.c X 02.c X 03.c X *05.c X
9   int main()
10  {
11      st = &input;
12
13      printf("Enter a string:\n");
14
15      gets(input);
16
17      char *newst = underscore(st);
18      puts(newst);
19      free(newst);
20
21      return 0;
22  }
23
24  char * underscore(char *st)
25  {
26      char *newst, *p;
27      newst = (char *) malloc((strlen(st) + 2) * sizeof(char));
28
29      if (newst == NULL)
30      {
31          exit(1);
32      }
33
34      p = newst;
35
36      while (*st != '\0')
37          *(p++) = *(st++);
38
39      *(p++) = '_';
40      *p = '\0';
41
42      return newst;
43  }
44
```

Enter a string:
abcd
abcd_
Process returned 0 (0x0) execution time : 2.7
Press any key to continue.

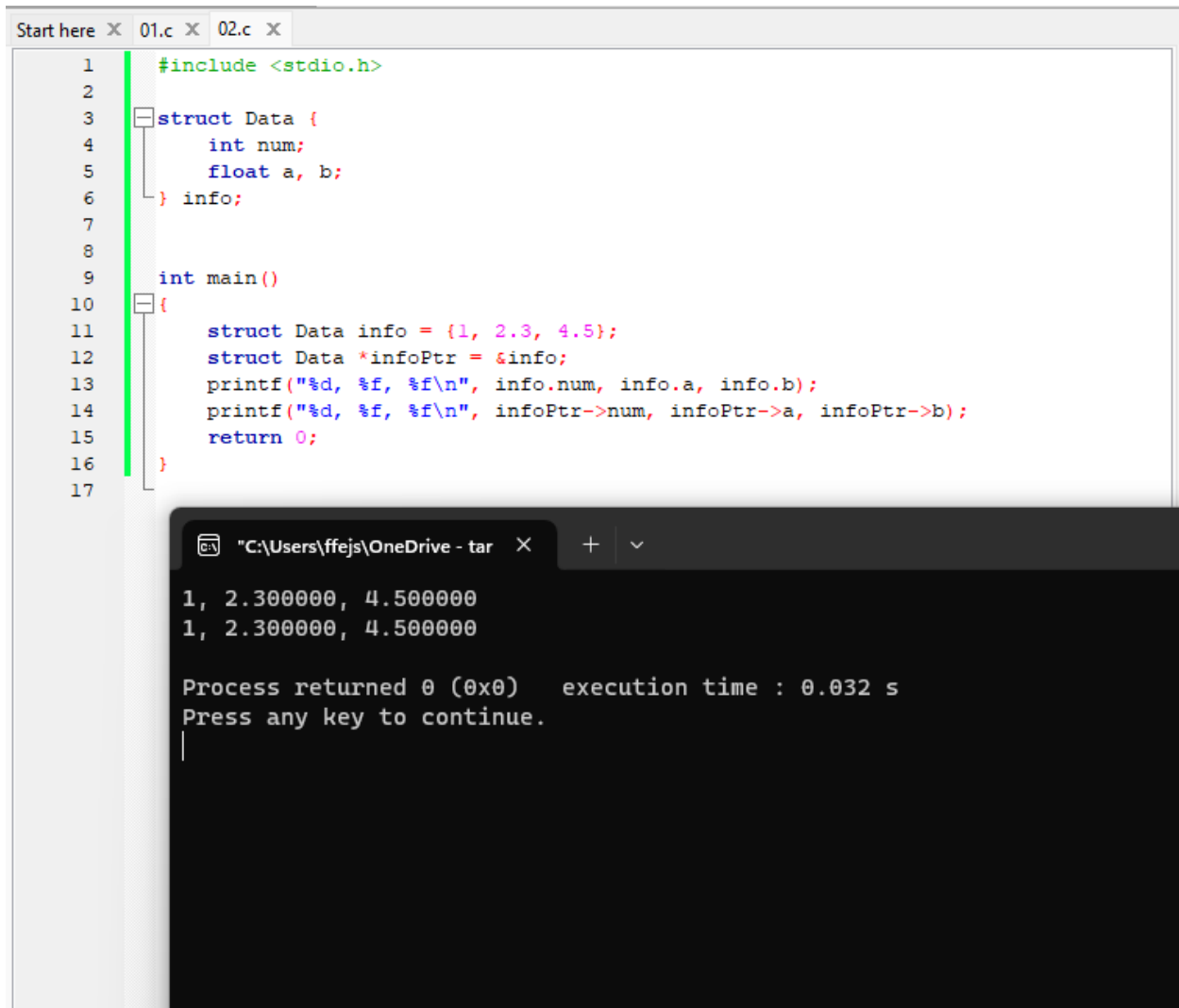
2] Write a program that solves Exercise 2 / p.277:

Write code that performs two tasks: defines a structure named `data` that contains one type `int` member and two type `float` members, and declares an instance of type `data` named `info`.

Initialize `info` with some data.

Then create a pointer **`infoPtr`** to **`info`** and print **`info`** two ways:

- using the variable name **`info`**
- using **`infoPtr`**



The screenshot shows a C program in a code editor and its execution output in a terminal window. The code defines a structure `Data` with an `int` member `num` and two `float` members `a` and `b`. It declares an instance `info` of this structure. In the `main` function, `info` is initialized with values `1`, `2.3`, and `4.5`. A pointer `infoPtr` is created and assigned the address of `info`. The program then prints the values of `info` using both direct access (`info.num`, `info.a`, `info.b`) and pointer access (`infoPtr->num`, `infoPtr->a`, `infoPtr->b`). The output shows that both methods produce the same results: `1, 2.300000, 4.500000`. The process returns `0` and the execution time is `0.032 s`.

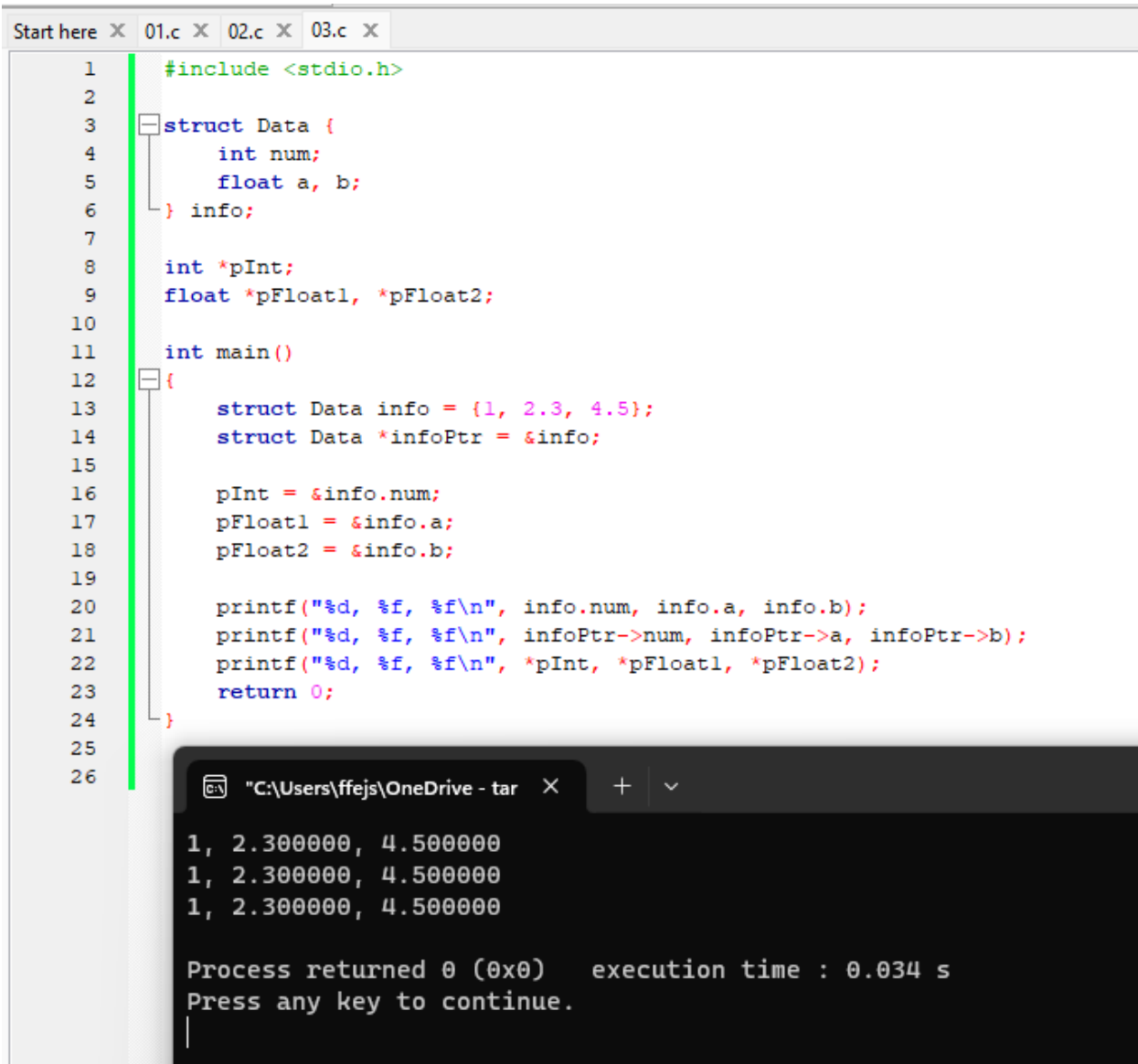
```
1  #include <stdio.h>
2
3  struct Data {
4      int num;
5      float a, b;
6  } info;
7
8
9  int main()
10 {
11     struct Data info = {1, 2.3, 4.5};
12     struct Data *infoPtr = &info;
13     printf("%d, %f, %f\n", info.num, info.a, info.b);
14     printf("%d, %f, %f\n", infoPtr->num, infoPtr->a, infoPtr->b);
15     return 0;
16 }
17
```

```
"C:\Users\ffejs\OneDrive - tar X + v
1, 2.300000, 4.500000
1, 2.300000, 4.500000

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
|
```

3] Using the previous program:

- Declare pInt, pFloat1, and pFloat2 to be pointers to integer and floats.
- Initialize so they will point to the integer and the float members of info.
- Print those values using the pointers.



The image shows a C program editor with a file explorer at the top showing 'Start here', '01.c', '02.c', and '03.c'. The editor displays a C program that defines a struct 'Data' with an integer 'num' and two floats 'a' and 'b'. It then declares pointers 'pInt', 'pFloat1', and 'pFloat2'. In the 'main' function, it initializes a 'Data' struct 'info' with values {1, 2.3, 4.5}, creates a pointer 'infoPtr' to 'info', and assigns the pointers 'pInt', 'pFloat1', and 'pFloat2' to the addresses of 'info.num', 'info.a', and 'info.b' respectively. It then prints the values using 'printf' with three different sets of arguments: the struct members directly, the struct pointer, and the individual pointers. Finally, it returns 0.

```
1  #include <stdio.h>
2
3  struct Data {
4      int num;
5      float a, b;
6  } info;
7
8  int *pInt;
9  float *pFloat1, *pFloat2;
10
11 int main()
12 {
13     struct Data info = {1, 2.3, 4.5};
14     struct Data *infoPtr = &info;
15
16     pInt = &info.num;
17     pFloat1 = &info.a;
18     pFloat2 = &info.b;
19
20     printf("%d, %f, %f\n", info.num, info.a, info.b);
21     printf("%d, %f, %f\n", infoPtr->num, infoPtr->a, infoPtr->b);
22     printf("%d, %f, %f\n", *pInt, *pFloat1, *pFloat2);
23     return 0;
24 }
25
26
```

The output window shows the execution results:

```
"C:\Users\ffejs\OneDrive - tar  X + v
1, 2.300000, 4.500000
1, 2.300000, 4.500000
1, 2.300000, 4.500000

Process returned 0 (0x0)  execution time : 0.034 s
Press any key to continue.
|
```

4] Circle all that apply:

A structure is a data type in which:

A. each element must have the same data type.

☒ C. each element may have a different data type

B. each element must have a pointer type only

☒ D. No element is defined

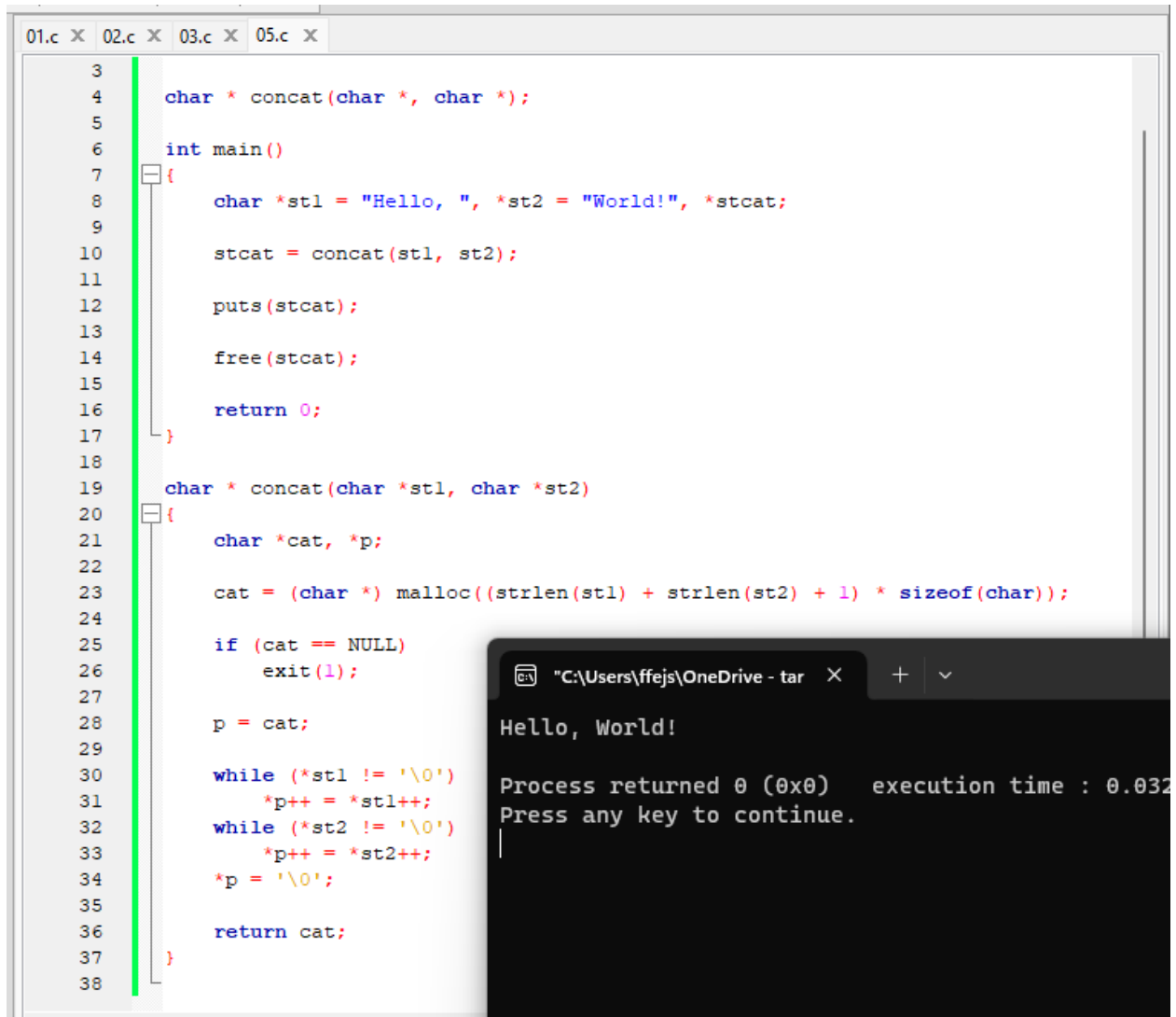
Problems for the lab report

For all problems marked with ►, take screenshots of both code and output, paste the screenshots in a report PDF file, and submit it to Canvas.

5] ► Solve Exercise 7 at the end of Ch.10:

Write a function that accepts 2 strings. Use **malloc()** to allocate enough memory to hold the 2 strings concatenated. Copy the strings to the block of memory allocated. Return a pointer to the new string. Print the new string in the main program.

Hint: Use **strlen()** to determine the lengths of the string arguments. (Don't forget to include **string.h**!)



The image shows a C program in a code editor and its execution output in a terminal window. The code defines a `concat` function that takes two character pointers and returns a new pointer to a concatenated string. The `main` function calls `concat` with "Hello, " and "World!", prints the result, and then frees the memory. The terminal output shows "Hello, World!" and the process return information.

```
01.c X 02.c X 03.c X 05.c X
3
4 char * concat(char *, char *);
5
6 int main()
7 {
8     char *st1 = "Hello, ", *st2 = "World!", *stcat;
9
10    stcat = concat(st1, st2);
11
12    puts(stcat);
13
14    free(stcat);
15
16    return 0;
17 }
18
19 char * concat(char *st1, char *st2)
20 {
21     char *cat, *p;
22
23     cat = (char *) malloc((strlen(st1) + strlen(st2) + 1) * sizeof(char));
24
25     if (cat == NULL)
26         exit(1);
27
28     p = cat;
29
30     while (*st1 != '\0')
31         *p++ = *st1++;
32     while (*st2 != '\0')
33         *p++ = *st2++;
34     *p = '\0';
35
36     return cat;
37 }
38
```

```
"C:\Users\ffejs\OneDrive - tar X + v
Hello, World!
Process returned 0 (0x0)   execution time : 0.032
Press any key to continue.
```

6] ► Write a program that creates the structure from
QUIZ 5 / p.276:

Create a second variable **youraddress** of the same
structure type, and initialize it with data of your choice.

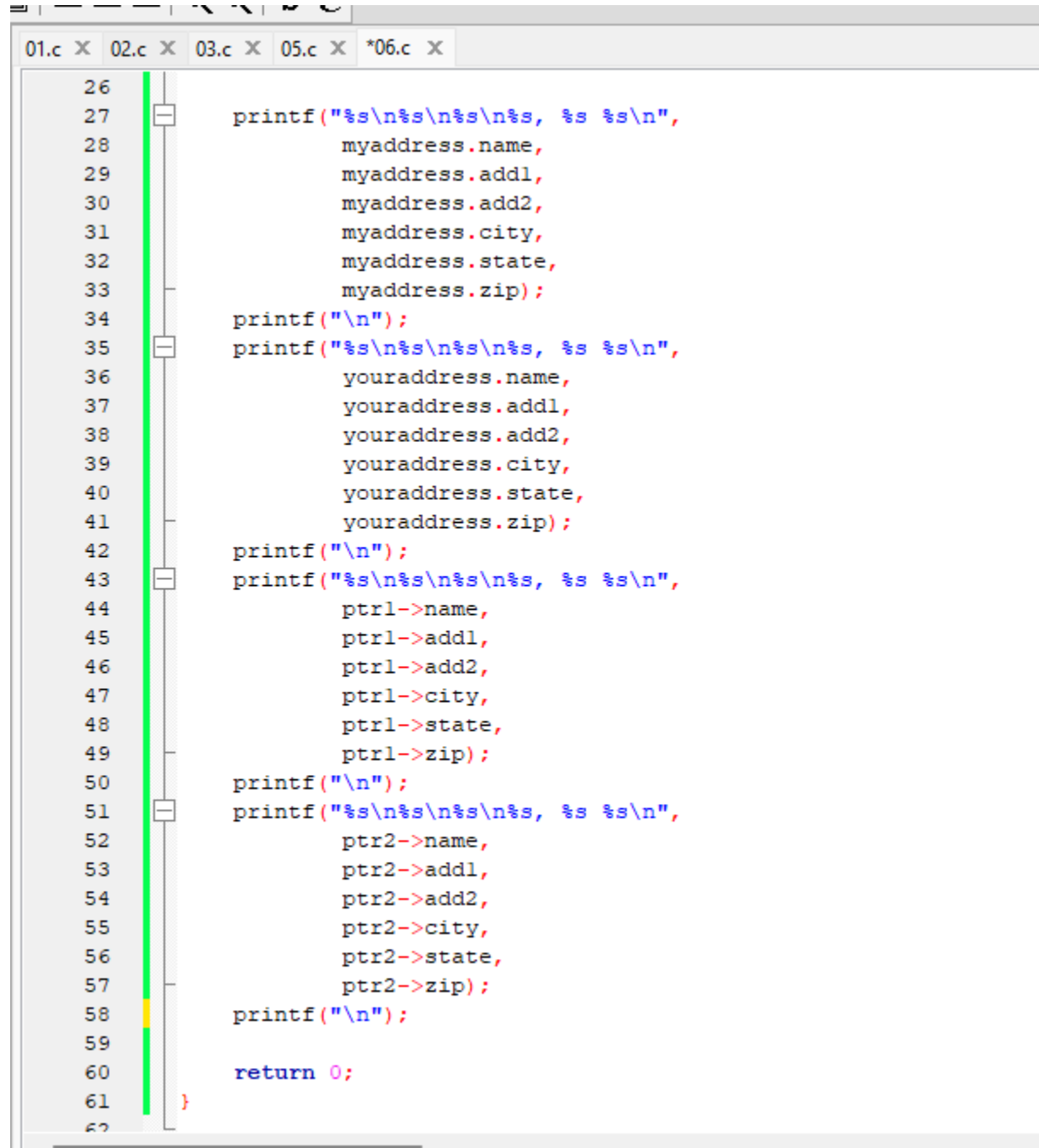
Create two pointers **ptr1**, and **ptr2**. Make **ptr1** point to
myaddress and **ptr2** to **youraddress**.

Print the structures two ways:

- using the variable names
- using the pointers.

```
struct address
{
    char name[31];
    char add1[31];
    char add2[31];
    char city[11];
    char state[3];
    char zip[11];
} myaddress = { "Bradley Jones",
                "RTSoftware",
                "P.O. Box 1213",
                "Carmel", "IN", "46082-1213"};
```

```
01.c X 02.c X 03.c X 05.c X 06.c X
1  #include <stdio.h>
2  struct address
3  {
4      char name[31];
5      char add1[31];
6      char add2[31];
7      char city[11];
8      char state[3];
9      char zip[11];
10 };
11
12 int main()
13 {
14     struct address myaddress = {"Bradley Jones",
15                                 "RTSoftware",
16                                 "P.O. Box 1213",
17                                 "Carmel", "IN", "46082-1213"};
18     struct address youraddress = { "Jeffrey Morris",
19                                     "AuroraBotics",
20                                     "P.O. Box 54",
21                                     "Glen Rose", "TX", "76043-0054"};
22
23     struct address *ptr1, *ptr2;
24     ptr1 = &myaddress;
25     ptr2 = &youraddress;
26
27     printf("%s\n%s\n%s\n%s, %s %s\n",
28            myaddress.name,
29            myaddress.add1,
30            myaddress.add2,
31            myaddress.city,
32            myaddress.state,
33            myaddress.zip);
34     printf("\n");
35     printf("%s\n%s\n%s\n%s, %s %s\n",
36            youraddress.name,
37            youraddress.add1
```



```
01.c X 02.c X 03.c X 05.c X *06.c X
26
27     printf("%s\n%s\n%s\n%s, %s %s\n",
28             myaddress.name,
29             myaddress.add1,
30             myaddress.add2,
31             myaddress.city,
32             myaddress.state,
33             myaddress.zip);
34     printf("\n");
35     printf("%s\n%s\n%s\n%s, %s %s\n",
36             youraddress.name,
37             youraddress.add1,
38             youraddress.add2,
39             youraddress.city,
40             youraddress.state,
41             youraddress.zip);
42     printf("\n");
43     printf("%s\n%s\n%s\n%s, %s %s\n",
44             ptr1->name,
45             ptr1->add1,
46             ptr1->add2,
47             ptr1->city,
48             ptr1->state,
49             ptr1->zip);
50     printf("\n");
51     printf("%s\n%s\n%s\n%s, %s %s\n",
52             ptr2->name,
53             ptr2->add1,
54             ptr2->add2,
55             ptr2->city,
56             ptr2->state,
57             ptr2->zip);
58     printf("\n");
59
60     return 0;
61 }
```



```
"C:\Users\ffejs\OneDrive - tar  X + v  
Bradley Jones  
RTSoftware  
P.O. Box 1213  
Carmel, IN 46082-1213  
  
Jeffrey Morris  
AuroraBotics  
P.O. Box 54  
Glen Rose, TX 76043-0054  
  
Bradley Jones  
RTSoftware  
P.O. Box 1213  
Carmel, IN 46082-1213  
  
Jeffrey Morris  
AuroraBotics  
P.O. Box 54  
Glen Rose, TX 76043-0054  
  
Process returned 0 (0x0)   execution time : 0.030 s  
Press any key to continue.  
|
```

7] Circle all that apply:

If one or more members of a structure are other structures, the structure is known as:

- ☒ A. nested structure B. invalid structure
C. self-referential structure D. unstructured structure

Do all the members of a structure need to have the same size?

A. Yes

☒ B. No