Programming basics
(GKNB_INTA023)

Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

https://github.com/sze-info/ProgrammingBasics

November 5, 2020

Function to swap the values of variables in the caller

Goal:

> Write a function to swap the values of two variables! The effect must be visible in the caller!

Problem:

- Actual parameters are passed by value (the formal parameters are copies of the actual parameters and our goal is to swap the values of the *original* variables and not their *copies*).
- A function may have at most one return value.

swap1.c

```
1 #include <stdio.h>
2
3 void display(int a, int b) {
4   printf("a = %d, b = %d\n", a, b);
5 }
```

Function to swap the values of variables in the caller

swap1.c – First attempt, swap1

```
7  void swap1(int a, int b) {
8    int temp = a;
9    a = b;
10   b = temp;
11 }
```

swap1.c – First lines of main

```
26 int main(void) {
27   int a = 1, b = 2;
28   printf("Original values:\t"); display(a, b);
29   swap1(a, b); printf("After swap1:\t\t"); display(a, b);
```

First lines of the output

```
Original values:  a = 1, b = 2
After swap1:      a = 1, b = 2
```

Function to swap the values of variables in the caller

swap1.c – Second attempt, swap2

```
13 struct twoNumbers { int a, b; };
14
15 struct twoNumbers swap2(int a, int b) {
16   struct twoNumbers temp = {b, a};
17   return temp;
18 }
```

swap1.c – Some lines of main

```
30   struct twoNumbers tn = swap2(a, b); a = tn.a; b = tn.b;
31   printf("After swap2:\t\t"); display(a, b);
```

Corresponding line of output

```
After swap2:        a = 2, b = 1
```

Function to swap the values of variables in the caller

swap1.c – Third attempt, swap3

```
20 void swap3(int* a, int* b) {
21   int temp = *a;
22   *a = *b;
23   *b = temp;
24 }
```

swap1.c – Some lines of main

```
32   swap3(&a, &b); printf("After swap3:\t\t"); display(a, b);
33   return 0;
34 }
```

A snippet of output

```
After swap2:    a = 2, b = 1
After swap3:    a = 1, b = 2
```

Function to swap the values of variables in the caller

swap2.c – Swapping functions

```
3  void swap1(int a, int b) {
4    printf("swap1: address of 'a': %p, address of 'b': %p\n", &a, &b);
5    printf("swap1: value of 'a': %d, value of 'b': %d\n", a, b);
6    int temp = a;
7    a = b;
8    b = temp;
9  }
10
11 void swap3(int* a, int* b) {
12   printf("swap3: address of 'a': %p, address of 'b': %p\n", &a, &b);
13   printf("swap3: value of 'a': %p, value of 'b': %p\n", a, b);
14   printf("swap3: value@address 'a': %d, "
15          "value@address 'b': %d\n", *a, *b);
16   int temp = *a;
17   *a = *b;
18   *b = temp;
19 }
```

Function to swap the values of variables in the caller

swap2.c – The main function

```
21 int main(void) {
22   int a = 1, b = 2;
23   printf("main: address of 'a': %p, address of 'b': %p\n", &a, &b);
24   printf("main: value of 'a': %d, value of 'b': %d\n", a, b);
25   swap1(a, b);
26   printf("main, after calling swap1: "
27          "value of 'a': %d, value of 'b': %d\n", a, b);
28   swap3(&a, &b);
29   printf("main, after calling swap3: "
30          "value of 'a': %d, value of 'b': %d\n", a, b);
31   return 0;
32 }
```

Function to swap the values of variables in the caller

Output

```
main: address of 'a': 0x7ffd85320ef0, address of 'b': 0x7ffd85320ef4
main: value of 'a': 1, value of 'b': 2
swap1: address of 'a': 0x7ffd85320ecc, address of 'b': 0x7ffd85320ec8
swap1: value of 'a': 1, value of 'b': 2
main, after calling swap1: value of 'a': 1, value of 'b': 2
swap3: address of 'a': 0x7ffd85320ec8, address of 'b': 0x7ffd85320ec0
swap3: value of 'a': 0x7ffd85320ef0, value of 'b': 0x7ffd85320ef4
swap3: value@address 'a': 1, value@address 'b': 2
main, after calling swap3: value of 'a': 2, value of 'b': 1
```

Drawing rectangles

rectangle2.c `readTLX`: Do you want to enter further rectangles? If yes, what is the X coord. of the TL corner?

```
39 bool readTLX(int count, int min, int max, int* k) {
40   bool goon;
41   do {
42     printf("X coordinate of the top left corner of rectangle #%d [%d, %d] "
43       "(exits to a negative value) ", count, min, max);
44     scanf("%d", k);
45     goon = *k>=0;
46   } while(goon && (*k<min or *k>max));
47   return goon;
48 }
49
50 int read(int count, char s[], int min, int max) {
51   int k;
52   do {
53     printf("%s rectangle #%d [%d, %d] ",
54       s, count, min, max);
55     scanf("%d", &k);
56   } while(k<min or k>max);
57   return k;
58 }
```

Drawing rectangles

rectangle2.c

```
60 int main(void) {
61   struct rectangle ar[MAXSHAPE]; int count; bool goon = true;
62   printf("Please enter the data of rectangles!\n");
63   for(count=0; count<MAXSHAPE and goon; count++) {
64     goon = readTLX(count+1, MINX, MAXX-1, &ar[count].tl.x);
65     if(goon) {
66       ar[count].tl.y = read(count+1, "Y coordinate of the top left corner", MINY, MAXY-1);
67       ar[count].br.x = read(count+1, "X coordinate of the bottom right corner",
68         ar[count].tl.x+1, MAXX);
69       ar[count].br.y = read(count+1, "Y coordinate of the bottom right corner",
70         ar[count].tl.y+1, MAXY);
71       printf("Drawing character of rectangle #%d: ", count+1);
72       scanf(" %c", &ar[count].c);
73       count++;
74     }
75   }
76   draw(ar, count-1);
77   return 0;
78 }
```

Further information regarding pointers

- Example: pointers.c
- Watch out! `i` is an `int`, but `pi1` and `pi2` are pointers to `int`s!
- Any number of white spaces can be placed on both sides of *
- A pointer can be initialized, too

```
4  int i=3, *pi1, *pi2;
5  pi1 = pi2 = &i; // OK
6  double d=1.5;
7  double* pd = &d; // initialization
```

Pointers

- If an object does not have a memory location/address, even operator & cannot determine it

```
8   // pd = &12.34;
9   // error: lvalue required as unary '&' operand
10  // A literal does not have a memory address, meaningless
```

- In general, assignments can be carried out with pointers of the same type

```
11  // pd = pi1; warning: assignment from
12  //                    incompatible pointer type
```

Pointers

- Exception: any pointers can be assigned to a `void*` pointer
  (≈ dropping type information)

```
13  void *pv;
14  pv = pi1; // OK
```

- It can be done in the opposite direction, but the type of data at the specified address may be incompatible

```
15  pi1 = pv; // OK, but did pv really address an int?
```

- `NULL` is a special address: no data is stored there
- It indicates an error or lack of something
- Can be assigned to any type of pointers

```
16  pv = NULL; // OK
```

Date management

Practical problem:

the structures are usually big, parameter passing needs too much time

Solution:

- pass the *address* of the structure!
- Danger! If the *called* fn. modifies the parameter, that affects the variable in the *caller*, too!
- To avoid the unintended modifications of variables in the *called* fn.: modifier `const` makes the parameter read-only (it can be used with other types as well)
- Indirection + member access: with operator `->`, eg. `(*d).day` ≡ `d->day`

Date management

calendar2.c

```
23 bool check(const struct date* d) { // content validation
24   if(d->month<1 or d->month>12) return false;
25   int days = daysOfMonth(d->year, d->month);
26   if(d->day<1 or d->day>days) return false;
27   return true;
28 }
29
30 int dayOfYear(const struct date* d) { // determining the day of the year
31   int days = d->day;                   // based on year, month and day
32   for(int month=1; month<d->month; month++) {
33     days += daysOfMonth(d->year, month);
34   }
35   return days;
36 }
```

Date management

calendar2.c

```
64 int main(void) {
65   struct date d = {23, 10, 2020};
66   printf("The given date is %s.\n"
67          "%d.%d.%d is the %dth day of the year.\n",
68          (check(&d)?"valid":"invalid"), d.day, d.month, d.year,
69          dayOfYear(&d));
70   struct date xmas = {24, 12, 2020};
71   printf("How many days are left to christmas? %d\n",
72     difference(&d, &xmas));
73   int dy = 300;
74   d = monthAndDay(d.year, dy);
75   printf("The %dth day of %d is: %d.%d\n",
76     dy, d.year, d.day, d.month);
77   return 0;
78 }
```