- CPSC 275: Introduction to Computer Systems

CPSC 275: Introduction to Computer Systems

Fall 2025

- Syllabus
- Schedule
- Resources
- Upload
- Solution

Solution to Homework 9

1.
   A. `float *fPtr;`
   B. `fPtr = &number1;`
   C. `printf("%f", *fPtr);`
   D. `number2 = *fPtr;`
   E. `printf("%f", number2);`
   F. `printf("%p", &number1);`
   G. `printf("%p", fPtr);` Yes, it is the same as the address of `number1`.

2. `void exchange(float *, float *);`

3.
   A. `number` uninitialized
   B. `integerPtr = (long *) realPtr;`
   C. `x = &y;`
   D. `numPtr` uninitialized
   E. `xPtr` not a pointer
   F. `s` uninitialized

4.
   A. `float numbers[] = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};`
   B. `float *nPtr;`
   C. `for (i = 0; i < SIZE; i++) printf("%f5.1\n", numbers[i]);`
   D. `nPtr = numbers;`

      `nPtr = &numbers[0];`

   E. `for (i = 0; i < SIZE; i++) printf("%f5.1\n", *(nPtr + i));`
   F. `for (i = 0; i < SIZE; i++) printf("%f5.1\n", *(numbers + i));`
   G. `for (i = 0; i < SIZE; i++) printf("%f5.1\n", nPtr[i]);`
   H. `numbers[4]`, `*(numbers + 4)`, `nPtr[4]`, `*(nPtr+4)`
   I. 1002532
   J. 1002504; The number 1.1 is stored at that location.

5.
   A. 14
   B. 34
   C. 4
   D. 1 (true)
   E. 0 (false)

6. `10 9 8 7 6 5 4 3 2 1`

7. Note that `show_bytes` enumerates a series of bytes starting from the one with lowest address and working toward the one with highest address. On a little-endian machine, it will list the bytes from least significant to most. On a big-endian machine, it will list bytes from the most significant byte to the least. Thus, we have:

```
Little endian: 21          Big endian: 87
Little endian: 21 43       Big endian: 87 65
Little endian: 21 43 65    Big endian: 87 65 43
```

8. It prints `61 62 63 64 65 66`. Recall also that the library routine `strlen` does not count the terminating null character, and so `show_bytes` printed only through the character '`f`'.

9. This problem is designed to demonstrate how easily bugs can arise due to the implicit casting from signed to unsigned. It seems quite natural to pass parameter `length` as an unsigned, since one would never want to use a negative length. The stopping criterion `i <= length-1` also seems quite natural. But combining these two yields an unexpected outcome! Since parameter `length` is unsigned, the computation `0 - 1` is performed using unsigned arithmetic. The result is then UMax. The ≤ comparison is also performed using an unsigned comparison, and since any number is less than or equal to UMax, the comparison always holds! Thus, the code attempts to access invalid elements of array `a`. The code can be fixed either by declaring `length` to be an `int`, or by changing the test of the `for` loop to be `i < length`.

10. This example demonstrates a subtle feature of unsigned arithmetic, and also the property that we sometimes perform unsigned arithmetic without realizing it. This can lead to very tricky bugs.
    A. The function will incorrectly return `1` when `s` is shorter than `t`.
    B. Since `strlen` is defined to yield an unsigned result, the difference and the comparison are both computed using unsigned arithmetic. When `s` is shorter than `t`, the difference `strlen(s) - strlen(t)` should be negative, but instead becomes a large, unsigned number, which is greater than 0.
    C. Replace the test with the following:

```
return strlen(s) > strlen(t);
```