HW2.6. Pointers

Consider the following code:

```
int main () {
    int x[5];
    x[0] = 254; x[1] = 649; x[2] = 971; x[3] = 1678;
    unsigned char *y = (unsigned char*) x;
    // we'll run some print statements here
}
```

We'll assume the following:

* the address of x's first element is `0x1868`

* sizeof(char) == 1; sizeof(int) == 4;

* this computer is big-endian

Fill in the blank with the printed value.

If the value printed is uncertain, enter "garbage". If this program would cause a compile-time error or the behavior is uncertain, enter "n/a".

Q1.1: `printf("%p\n", x);` 0x1868 ? ✓ 100%

Q1.2: `printf("%d\n", *(x+1));` 649 ? ✓ 100%

Q1.3: `printf("%d\n", x[2]);` 971 ? ✓ 100%

Q1.4: `printf("%d\n", x[4]);` garbage ? ✓ 100%

Q1.5: `printf("%p\n", x+9);` 0x188C ? ✓ 100%

Q1.6: `printf("%p\n", &x);` 0x1868 ? ✓ 100%

Q1.7: `printf("%d\n", x[15]);` n/a ? ✓ 100%

Q1.8: `printf("%d\n", *y);` 0 ? ✓ 100%

Q1.9: `printf("%d\n", *(y+3));` 254 ? ✓ 100%

Try a new variant

Correct answer

Fill in the blank with the printed value.

If the value printed is uncertain, enter "garbage". If this program would cause a compile-time error or the behavior is uncertain, enter "n/a".

Q1.1: `printf("%p\n", x);` `0x1868`

Homework 2

Assessment overview

Total points: 50/100

Score: 50%

Question

Value: 15

History: 15

Awarded points: 15/15

Report an error in this question

Previous question

Next question

Attached files

No attached files

Attach a file

Attach text

Q1.2: `printf("%d\n", *(x+1));` `649`

Q1.3: `printf("%d\n", x[2]);` `971`

Q1.4: `printf("%d\n", x[4]);` `garbage`

Q1.5: `printf("%p\n", x+9);` `0x188c`

Q1.6: `printf("%p\n", &x);` `0x1868`

Q1.7: `printf("%d\n", x[15]);` `n/a`

Q1.8: `printf("%d\n", *y);` `0`

Q1.9: `printf("%d\n", *(y+3));` `254`

Q1.1: An array is a pointer to it's first element. `x` points to the first element, which is stored at `0x1868`.

Q1.2: This is equivalent to `x[1]`. Remember in C, that pointer arithmetic takes into account the size of the pointer type, so this is deferencing the address "x + 1 ints" (address `0x186C`).

Q1.3: 971 is stored at index 2 of array `x`.

Q1.4: C never initializes the contents of local variables for you (for efficiency). Uninitialized values contain garbage. It does not error because we have declared an array of length 5, meaning index 4 exists, is just uninitialized.

Q1.5: Pointer arithmetic takes into account the size of the pointer type. So this is equivalent to the address "`x + 9` ints". If each int is 4 bytes, 9 ints is 36 bytes, which yields a pointer of `0x188c`.

Q1.6: This is one of the trickier cases to remember when dealing with pointers and arrays. In C, while an array name behaves very much like a pointer to the first element of the array, it is not a separate variable storing this pointer. So when placing an `&` in front of the array name to get the "address of" the array, we don't get the address of wherever a pointer to this array might be stored. Instead, we literally get the "address of" the array `x` - i.e. where it begins, which is the location of the first element in the array x.

Q1.7: The valid indexes for `x` are 0-4. It's possible `x[15]` results in a segfault, so the behavior of this program is undefined. C standard keeps this undefined, since the system is only guaranteed to generate space for 5 ints; the memory access could map to inaccessible memory. In practice, though, accessing a array index like this would not segfault. This is because x is stored on the stack, and the stack contains a large space of "accessible" memory. Trying to access `x[15]` would likely return something that's also on the stack, such as other local variables or stack metadata; as a result, bugs involving reading past the end of an array are **very** hard to debug.

Q1.8: Dereferencing a pointer of type `char` returns `sizeof(char)` bytes from that memory location. Since `sizeof(char) == 1`, the first byte from the beginning of the array x would be 0.

Q1.9: Dereferencing a pointer of type `unsigned char` returns `sizeof(unsigned char)` bytes from that memory location. Since `sizeof(char) == 1`, `(y+3)` would be the the fourth byte from the beginning of the array `x`, which is 0xfe.

Submitted answer 6 correct: 100%

Submitted at 2022-09-03 09:20:52 (PDT)

hide

Fill in the blank with the printed value.

If the value printed is uncertain, enter "garbage". If this program would cause a compile-time error or the behavior is uncertain, enter "n/a".

Q1.1: `printf("%p\n", x);` `0x1868` ✓ 100%

Q1.2: `printf("%d\n", *(x+1));` `649` ✓ 100%

Q1.3: `printf("%d\n", x[2]);` `971` ✓ 100%

Q1.4: `printf("%d\n", x[4]);` `garbage` ✓ 100%

Q1.5: `printf("%p\n", x+9);` `0x188C` ✓ 100%

Q1.6: `printf("%p\n", &x);` `0x1868` ✓ 100%

Q1.7: `printf("%d\n", x[15]);` `n/a` ✓ 100%

Q1.8: `printf("%d\n", *y);` `0` ✓ 100%

Q1.9: `printf("%d\n", *(y+3));` `254` ✓ 100%

Submitted answer 5 **partially correct: 88%**

Submitted at 2022-09-03 09:18:54 (PDT)

ℹ show ⌄

Submitted answer 4 **partially correct: 88%**

Submitted at 2022-09-03 09:18:26 (PDT)

ℹ show ⌄

Show/hide older submissions ⌄