CS Basics - Exercises
Pointers in C

C. Grothoff and E. Benoist and P. Mainini

Fall Term 2022-23

1 Pointers

1.1 Swap Variables

Write a function which swaps the contents of two `int32_t` variables. Provide a main function which tests the function you have written.

1.2 Array = pointer

Suppose that we have the following code:

```c
int32_t minimum(int32_t *array, size_t size);

int main(void) {
  int32_t array[] = {34, 54, 2, 43, 78};
  int32_t min = minimum(array, 5);

  printf("%d\n", min); // prints "2"
}
```

Write the code for the function `minimum()` without using any "`[]`"; use only pointers and the dereferencing operator ("`*`").

Write a second function `minmax()`, which takes as input an array and pointers to two variables `min` and `max`. It should write the minimum and maximum value of the elements in the array to `min` and `max`.

```
void minmax(int32_t *array, size_t size, int32_t *min, int32_t *max);

int main(void) {
  int32_t min = 0, max = 0;
  int32_t array[] = {34, 54, 2, 43, 78};

  minmax(array, 5, &min, &max);
  printf("min: %d, max: %d\n", min, max); // prints "min: 2, max: 78"
}
```

Could you also change `minmax()` that it *returns* an array with the minimum and maximum values? What would be the challenge?

1.3 `sizeof()`

Consider the following code:

```
int main(void) {
  int32_t array[] = {34, 54, 2, 43, 78};
  int32_t *a = array;

  printf("%lu %lu %lu %lu\n", sizeof array, sizeof a, sizeof *array,
         sizeof *a);
}
```

Use it as an inspiration to avoid the magic constant “5” in the previous exercises.

2 Arrays vs. Pointers in Assembly

We do want to understand how arrays and pointers are treated by the compiler when generating assembly code. Proceed as follows:

1. Compile `resources/disassemble.c` without optimization, i.e.
   `gcc -std=c17 -Wall -Wextra -Wpedantic -O0 disassemble.c -o disassemble`
2. Disassemble the binary and try to understand the functions `array` and `pointer`.
   - How do they work? Do they differ? If so, what's the most important difference?
   - Can you identify where the variables from the C code are stored in assembly?
   - Why are they stored that way?

   Hint: For disassembling, use "`objdump -M intel -d disassemble`", you can then search for "array", "pointer" or also "main"...
3. Change all the `uint16_t` data types to `uint32_t`. Compile and disassemble again. What has changed?
4. Which parts of the generated assembly code might be inefficient? How could that be improved?
5. Recompile with different optimizations: `-O1`, `-O2` and `-O3`. Can you still understand the code? What changes?