CSE 333 25au Exercise 2

out: Monday, September 29, 2025
due: Wednesday, October 1, 2025 by **10:00 am**. **No late exercises accepted.**

Goals: Use pointers to access and print arbitrary memory bytes and addresses, and explore C data structure representation in memory.

Description: Your job is to write a C function that prints, in hex, the values of the bytes allocated to some variable. The function's prototype is

```
void PrintBytes(void* mem_addr, int num_bytes);
```

and its output looks like this:

```
The 4 bytes starting at 0x7fff1081856c are: ff 01 30 4e
```

You should use the following `main` function in the program you submit (you should be able to copy and paste this code from the assignment web page rather than having to re-type it):

```
int main(int argc, char **argv) {
  char     char_val = '0';
  int32_t  int_val = 1;
  float    float_val = 1.0;
  double   double_val  = 1.0;

  typedef struct {
    char     char_val;
    int32_t  int_val;
    float    float_val;
    double   double_val;
  } Ex2Struct;

  Ex2Struct struct_val = { '0', 1, 1.0, 1.0 };

  PrintBytes(&char_val, sizeof(char));
  PrintBytes(&int_val, sizeof(int32_t));
  PrintBytes(&float_val, sizeof(float));
  PrintBytes(&double_val, sizeof(double));
  PrintBytes(&struct_val, sizeof(struct_val));

  return EXIT_SUCCESS;
}
```

Your implementation of `PrintBytes` will need to do a few things:

- convince the compiler to let you access bytes in memory, given the address of the first byte to be printed as a `void*` pointer value.
- use a loop to iterate once for each byte in the variable, using a pointer to extract that byte and print it.
- figure out how to use format specifiers to `printf` in order to print out an `uint8_t` byte in lower-case hexadecimal, using exactly two digits. As a hint, take inspiration from this code:

```
#include <inttypes.h>
...
  uint8_t a_byte = 0xD1;
  printf("The byte is: %02" PRIx8 " -- enjoy!\n", a_byte);
...
```

- figure out how to print a pointer value.

Your code should produce output that is identical to the following, except for values that might change from one execution to the next because of randomness outside of your control. That randomness includes variable addresses and the values of uninitialized data or padding bytes.

```
$bash gcc -Wall -std=c17 -g -o ex2 ex2.c
$bash ls
ex2 ex2.c
$bash ./ex2
The 1 bytes starting at 0x7fff3c84a1ff are: 30
The 4 bytes starting at 0x7fff3c84a1f4 are: 01 00 00 00
The 4 bytes starting at 0x7fff3c84a1f8 are: 00 00 80 3f
The 8 bytes starting at 0x7fff3c84a1e8 are: 00 00 00 00 00 00 f0 3f
The 24 bytes starting at 0x7fff3c84a1d0 are: 30 b0 f0 00 01 00 00 00 00 00 80 3f 0
0 00 00 00 00 00 00 00 00 00 f0 3f
```

(Note that the last output line is actually a single line, but probably wraps in your browser. Also, note that the number of bytes shown on the last line is not the sum of the number of bytes shown on the previous lines because of uninitialized padding bytes in the struct used to align the struct fields.)

You must not alter the given `main` function code. If you notice any style issues with it, ignore them and focus on getting your new code right.

Your code must:

- compile without errors or warnings on current CSE Linux machines (lab workstations, attu, or CSE home VM)
- have no crashes, memory leaks, or memory errors on CSE linux machines (note: if you check your program with valgrind, it may report that your program is reading uninitialized bytes; that is not an error if, in fact, that's what the code is supposed to actually be doing).

- be contained in a single file called `ex2.c` that compiles with the command `gcc -Wall -g -std=c17 -o ex2 ex2.c` -- do not submit a Makefile.
- be pretty: the formatting, modularization, variable and function names, and so on must make us smile rather than cry. (Suggestion: use `cpplint.py --clint` from hw0 to check for potential style problems.)
- be robust: you should think about handling bogus input from the user (if any), and you should handle hard-to-handle cases (if there are any) gracefully.
- have a comment at the top of your `.c` file with your name, and CSE or UW email address.

You should submit your exercise using the Gradescope dropbox linked on the course resources web page.
