

1/6

Week 5

Exercise 40

an array is NOT a pointer

../40/40.txt

1. Pointer variables and arrays

They are very similar, in that declaring a `size_t` array of size 10, i.e. `size_t array[10]` is actually just a pointer to the first element of that array (i.e. `array[0] = *array`). The difference lies in the fact that the location that an array points to is immutable, whereas a pointer variable can be changed.

Q

2. Pointer variables and reference variables

See the drawing below (Figure 1).

3. Pointer arithmetic

An example of this can be found in Figure 1, part b. It refers to the fact that pointers of a certain type can be incremented or decremented to reach the next element from its starting position. For example, given an integer array named 'intArray', defining an integer pointer `*intArray` will point it to the start of said array. Thus, `*intArray + 1` will point towards the second element in that array, as the pointer now points one integer-sized storage block further than the start of said array. Or rather, it points towards the addresses associated therewith.

incomplete

4. Accessing an element in an array using only a pointer vs. index expression

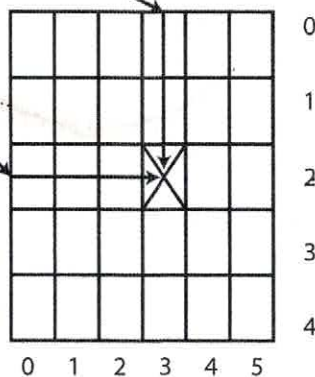
Using a pointer will skip a step when accessing an element, which is to determine the size of the array element and add that to the address of the first element.

Instead, it can simply move over the size of a single element over and over again

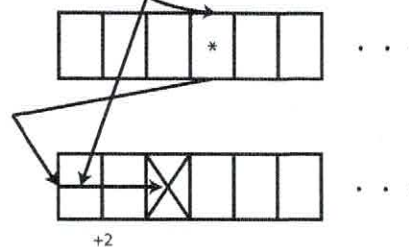
In the exercise, `size_t` is mentioned, of which the size is known and constant and so elements can be accessed directly. This can be especially true when elements are repeatedly accessed, such as in loops, resulting in cumulative benefits. However, personally, I feel that the index expressions establish a closer link to mathematical equivalents, such as matrices, which makes their use more accessible, and I wonder how much of the advantages are still present given the state of compiler optimisation.

? un clear

Element [3][2]
a. for `int array[30][20]`
An array of 30 elements, each element itself an array of 20 integers.
Hence: `array[3]` points at the memory location of the start of the fourth array of integer array-elements.
Then, `array[3][2]` points to the third element within that array
In other words, it is equivalent to `array[3] + 2`



Element [3][2]
b. for `int *pointer[20]`
An array length 20 of pointers of. Say that each pointer itself points towards integer arrays. Then, the element [3][2] is reached as follows:
value = `(*pointer[3] + 2)`



That's true, but you shouldn't think for the compiler

N? 40

Figure 1: Illustration

Exercise 41

../41/main.cc

```
1 // Iterating over environ and argv: main file
2
3 #include "main.ih"
4
5 int main(int argc, char const *argv[])
6 {
7     extern char **environ;
8
9     for (size_t index = 0; index != argc; ++index) // For elements of argv (0-argc)
10         cout << environ[index] << '\n';           // print associated env var
11
12     for (size_t index = 0; environ[index] != nullptr; ++index) // For all elements
13         cout << argv[index] << '\n';                     // of environ[] print
14 }                                                     // associated argv
```

declarations in the
.ih file, please

here here environ's contents are printed,
not argv's

(64)

and here
argv's contents.
Also, most likely a
segfault. At least
an array bound error

Exercise 43

743

../43/43.txt

definition:	rewrite:
int x[8];	x[3] = x[2];
pointer notation:	*(x + 3) = *(x + 2)
semantics:	x + 3 points to the location of the 3th int beyond x. Which is set to be equal to x + 2 which is the location of the 2nd int beyond x.
char *argv[8];	cout << argv[2];
pointer notation:	cout << *(argv + 2);
semantics:	argv + 2 points to the location of the 2nd argument beyond the first argument which is the programs name. Which is then passed to cout.
int x[8];	&x[10] - &x[3];
pointer notation:	*(x+10) - &*(x+3) = ?
semantics:	*(x+10) points to the the 10th int beyond x. Then the reference & makes it instead return its location. The same happens for &*(x+3). Since one points to the 3rd int beyond x and the other points to the 10th int beyond x the result is the difference 10 - 3 = 7. Ah :-)
char *argv[8];	argv[0]++;
pointer notation:	*argv = *argv + 1;
semantics:	*argv then points to the start of the programs name. Then by adding 1 to it, it still points to the programs name. But now it points to 1 byte beyond the start of the programs name. Such that the programs name would be /45 instead of the initial ./45 if it were passed to cout. Nope: This is an assignment, not an increment
char *argv[8];	argv++[0];
pointer notation:	*(argv + 1) → no post-fix increment
semantics:	*(argv + 1) points to the first argument beyond the programs name.
char *argv[8];	++argv[0];
pointer notation:	1 + *(argv) → no prefix increment
semantics:	1 + *(argv) then points to the start of the programs name. Then by adding 1 to it, it still points to the programs name. But now it points to 1 byte beyond the start of the programs name. Such that the programs name would be /45 instead of the initial ./45 if it were passed to cout.
char **argv;	++argv[0][2]; SF
pointer notation:	++*(*(argv) + 2)
semantics:	First the outer pointer (the 2) points to the 2nd column. Then the inner pointer points the the 0th element in the 2nd row. Then 1 is added to its contents. Which is of type char.

THE HISTORY OF THE
CITY OF BOSTON

FROM THE FIRST SETTLEMENT TO THE PRESENT TIME

CHAPTER I

THE CITY OF BOSTON WAS FIRST SETTLED BY A COMPANY OF
PURITANS WHO ARRIVED IN 1630. THEY WERE LEADED BY
JOHN WINSTON, A MINISTER OF THE CHURCH OF
ENGLAND.

THEY FOUND THE CITY ALREADY SETTLED BY
INDIANS WHO CALLED IT QUINCY. THE
PURITANS TOOK POSSESSION OF THE CITY
AND BUILT A FORT ON THE TIP OF THE
NECK.

THE CITY GROWED RAPIDLY AND BECAME
ONE OF THE MOST IMPORTANT PORTS IN
THE COLONIES. IT WAS THE CENTER OF
COMMERCE AND INDUSTRY.

THE CITY WAS THE SCENE OF MANY
IMPORTANT EVENTS. IT WAS THE
PLACE WHERE THE DECLARATION OF
INDEPENDENCE WAS SIGNED. IT WAS
THE PLACE WHERE THE BOMBING OF
BOSTON OCCURRED.

THE CITY WAS THE HOME OF MANY
FAMOUS MEN. IT WAS THE HOME OF
JOHN ADAMS, THE SECOND PRESIDENT
OF THE UNITED STATES. IT WAS THE
HOME OF SAMUEL JOHNSON, THE
AUTHOR OF THE FIRST DICTIONARY OF
THE ENGLISH LANGUAGE.

THE CITY WAS THE CENTER OF
LITERATURE AND ART. IT WAS THE
PLACE WHERE MANY FAMOUS WRITERS
LIVED AND WORKED. IT WAS THE
PLACE WHERE MANY FAMOUS ARTISTS
LIVED AND WORKED.

THE CITY WAS THE CENTER OF
SCIENCE AND EDUCATION. IT WAS THE
PLACE WHERE MANY FAMOUS SCIENTISTS
LIVED AND WORKED. IT WAS THE
PLACE WHERE MANY FAMOUS
EDUCATORS LIVED AND WORKED.

THE CITY WAS THE CENTER OF
POLITICS AND GOVERNMENT. IT WAS
THE PLACE WHERE MANY FAMOUS
POLITICIANS LIVED AND WORKED. IT
WAS THE PLACE WHERE MANY
FAMOUS GOVERNMENT OFFICIALS
LIVED AND WORKED.