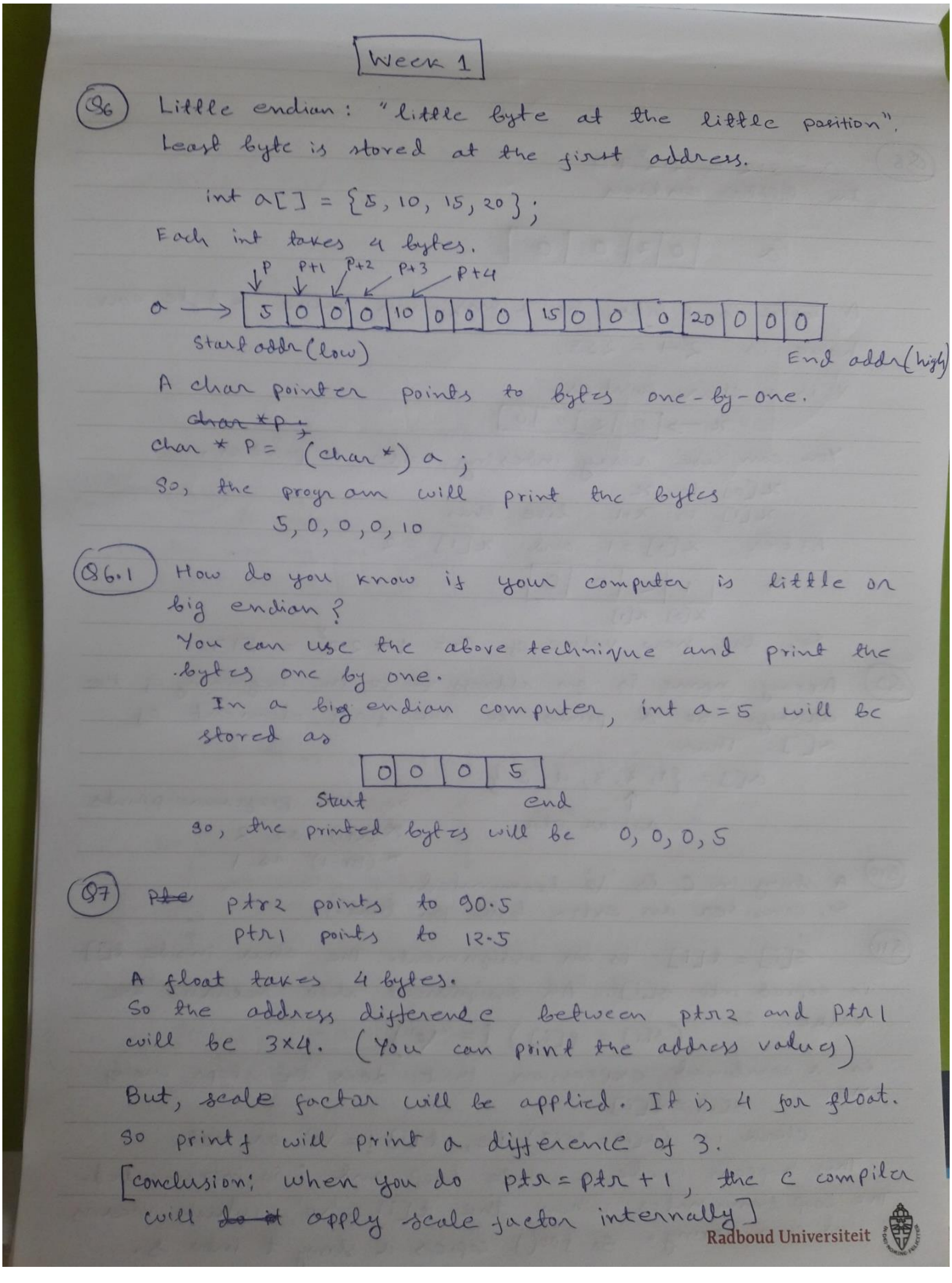
Solutions of Some of the Practice Problems of Week 1



Week 1

Q6 Little endian: "little byte at the little position".
Least byte is stored at the first address.

int a[] = {5, 10, 15, 20};

Each int takes 4 bytes.

p p+1 p+2 p+3 p+4

a →

5	0	0	0	10	0	0	0	15	0	0	0	20	0	0	0

Start addr (low) End addr (high)

A char pointer points to bytes one-by-one.

~~char *p;~~

char * p = (char *) a ;

So, the program will print the bytes

5, 0, 0, 0, 10

Q6.1 How do you know if your computer is little or big endian?

You can use the above technique and print the bytes one by one.

In a big endian computer, int a=5 will be stored as

0	0	0	5

Start end

So, the printed bytes will be 0, 0, 0, 5

Q7 ~~Ptr~~ ptr2 points to 90.5
ptr1 points to 12.5

A float takes 4 bytes.
So the address difference between ptr2 and ptr1 will be 3x4. (You can print the address values)

But, scale factor will be applied. It is 4 for float.
So printf will print a difference of 3.
[Conclusion: when you do ptr = ptr + 1, the c compiler will ~~do it~~ apply scale factor internally]

Radboud Universiteit

(Q8) $a = 512 = 2 \cdot 2^8 + 0$

In little endian

a ·				
	0	2	0	0

A byte is 8 bits. So, a maximum value a byte can have is $2^8 - 1 = 255$.

x is a char pointer

x →				
	0	2	0	0

You can use array indexing with a pointer.

x[0] is x

x[1] is x+1 live this

After x[0] = 1 and x[1] = 2,

a				
	1	2	0	0
	x[0]	x[1]		

So, the new value of $a = 1 + 2 \cdot 2^8 = 513$

(Q9) Array name is an address ~~#~~ to the beginning of the array. So 'a' points to the first element of a[]. Thus

$a[] = \{1, \underset{\uparrow}{2}, 3, 4, 5\}$

a+1 or ptr

So, the program prints

*(a+1) as 2

* (ptr-1) as 1

(Q10) A string in C is '\0' terminated.
So, consider an extra byte for that.

(Q11) s[i] = t[i] is an assignment. The char inside t[i] is copied into s[i]. An assignment also returns the value. So

(s[i] = t[i]) != '\0')

is a combined expression that does the steps one by one:

s[i] = t[i]

check is (s[i] = t[i]) i.e, t[i] != '\0' or not.

This happens inside a while loop and i is incremented. The loop terminates when the t[i] = '\0' which means the end of a string. So foo() copies a string t into s.

(Q12) It does the same thing as Q11.

s[i] with i++ means the next element.

s++ also means pointing to the next element.

So the while loop copies char elements of string t into s one by one.

(Q13)

*s++ = *t ++ is done in steps

*s = *t then

t++ and s++

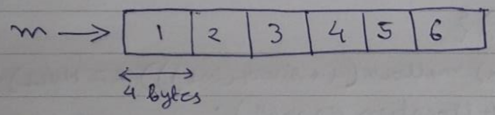
The (*s++ = *t++) is the assignment operation returned value. So it is ~~basicall~~ the value *t.

Thus, again the while loop copies contents of t into c.

(Q14) and (Q15)

A matrix such as int m[2][3] in C is stored in the row major order.

Let $m[2][3] = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ Logical view



Let p be a pointer: P = &m[0][0].

Column major print: 1, 4, 2, 5, 3, 6

```
for (i=0; i<3; i++)
   for (j=0; j<2; j++){
         printf("%d.", *(P+jx3+i))
   }
```

Row major print: 1, 2, 3, 4, 5, 6

```
for (i=0; i<3x2; i++)
     printf("%d", *(P+i));
```