Repeat encodings. In this assignment, you will implement the *repeat encoding*. If a string contains repeated letters, such as "aaaa", then we can replace this with a shorter repeat-encoded string "a4", which says that the letter 'a' should be repeated four times. To *encode* a string into the repeat encoding, all repeated characters should be transformed into their repeat encodings. So, when the string "aaaabbb" is given to the encoder, it should return "a4b3". The *decoder* does the opposite: it takes repeat encoded strings and returns the original. So when the string "a2b2c2" is given to the decoder, it will return "aabbcc".

To make things easier, in parts A and B we will consider a *simplified* repeat encoding with the following properties:

- No character is ever repeated more than nine times.
- Single instances of a character will still be repeat encoded. This means that the string "abc" will be encoded to "a1b1c1", even though the encoding is longer than the original.

These two restrictions mean that *every* encoded string has the format:

<character> <number> <character> <number> ...

where all of the numbers are represented by single characters. Here are some examples of the simplified repeat encoding.

Original	Encoded
"too"	"t1o2"
"hello"	"h1e1l2o1"
"bookkeeper"	"b1o2k2e2p1e1r1"
"aaaaaaaaabbbbccc"	"a9b4c3"

The assignment. In this assignment, you will implement an encoder and decoder for the repeat encoding in Haskell. In Part A you will create a decoder for the simplified encoding, and in Part B you will create an encoder for the simplified encoding.

Restrictions. The purpose of this assignment is to test your skills at creating recursive functions. For this reason, to obtain full marks for your solutions, you should implement your functions using recursion wherever possible, **you should not use any library functions except** head **and** tail, and you should not use list comprehensions. To be clear, all normal Haskell syntax is allowed, including:

- Any operator that is infix by default. For example +, ++, &&, and : are all allowed, but `min` and `max` are not.
- if expressions, let expressions, where syntax.
- List ranges.
- Pattern matching.

The only things that are disallowed are calling are calling named library functions except for head and tail, and list comprehensions. In part C, a few more library functions can be used, as described in that section.

Part A

In part A we will build a decoder for the simple repeat encoding.

Question 1. Write a function char to _int char that takes a character representing a number between zero and nine, and returns the corresponding integer. So char to int '1' should return the integer 1, and char to _ int '2' should return 2, and so on.

Hints:

- You only need to handle single characters as input to this function. So the only inputs will be the characters '0' through '9'. You *do not* need to handle strings with multiple numbers. It does not matter what your function returns if the input is not a number.
- There is no need to use recursion for this function.
- There is an example in Lecture 7 that you can adapt to implement this function. Although we saw some more advanced ways to work with characters in Lecture 10, those examples used library functions that are not allowed in Assignment 1.
- There is no special trick here, it is fine if your function takes quite a few lines.

Question 2. Write a function repeat_char c n that returns a string that contains n copies of the character c. So repeat char 'a' 3 should return "aaa" and repeat char 'b' 9 should return "bbbbbbbbb".
Hints:

- You will need to use recursion to do this.
- First think about the base case: when should we stop the recursion? It doesn't make sense to call repeat char c (-1), so we should probably stop before then. What should we return for the base case?
- Next think about the recursive step: how do we break repeat char c n into something that is closer to the base case? Once we've made a recursive call, how should we transform the output to get what we want?
- Look at the examples in Lecture 8 if you are stuck here.

Question 3. Write a function decode string that decodes a string in the simple repeat encoding. So decode "a1b2" should return "abb" and decode "t1h1e1" should return "the"
Hints:

- Note that, in all of the questions in the assignment, the arguments names are not prescribed. So you can change the name of the argument string to whatever you like, and you can use pattern matching if you wish.
- As always, start with the base case. What is the smallest possible input to decode? What should the output be for that input?
- The recursive rule will be a little complex. You will need to pull the first *two* characters from the front of the string. Pattern matching can do this, and there is an example in Lecture 8 demonstrating this.
- Because we are using the simple repeat encoding, we know that the first character in the string will be a letter, and the second character will be a number.
- So there are two things that we need to do: turn the second character into an integer, and then repeat the first character that many times. We have already written functions to do both of these things.
- Then we just need to put our repeated characters at the front of the decoded string. The Haskell operator for joining two strings can do this.

Part B (worth 35%)

In Part B we will build an encoder for the simple repeat encoding. There are no hints in Part B. You can answer the first three questions in any order, so if you are stuck on one question, try another one. You will probably need to answer Questions 4–6 before you can tackle Question 7.

Question 4. Write a function int to char int that takes a number between 0 and 9 and returns the corresponding character. So int to _char 1 should return '1', and int to char 2 should return '2', and so on.

Question 5. Write a function length char c string that takes a character c and a string string and returns the number of times that c occurs at the start of string. So length char 'a' "aaabaa" should return 3, and length char 'c' "cbcac" should return 1.

Question 6. Write a function drop char c string that takes a character c and a string string and returns a version of string without any leading instances of c. A leading instance of c is any instance that would have been counted by length char. So drop char 'a' "aaabaa" should return "baa", and drop char 'c' "cbcac" should return "bcac".

Question 7. Write a function encode string that encodes string in the simple repeat encoding. So encode "aaabbbccc" should return "a3b3c3" and encode "abcd" should return "a1b1c1d1".