# MORE ON STRINGS

## Today

- Last time we looked at some basic operations one can carry out on strings
- This time we will look at more complex operations.
- We will also look at some operations which are not strictly to do with strings, but which can be useful in dealing with strings.

## More member functions

- Last time we saw how to find the length of a string.

  ```
  int    len;
  string dna;

  len = dna.length();
  ```

  will do this for the string `dna`.

- It turns out that:

  ```
  len = dna.size();
  ```

  does much the same.

- So far as I can tell, `length` and `size` give exactly the same thing.

- Now, stroctly speaking, `len` shouldn't be an `int`.
- We should really use:

```
string::size_type len;
```

- In other words, what gets returned by `size` and `length` is a value of type `string::size_type`.
- If you try to use `int` you may find some strange compiler errors crop up.

## Finding things in strings

- Often we want to look for things in a string.
- C++ allows us to do this:

  ```
  string::size_type pos;
  pos = dna.find("tata", 0);
  ```

  `pos` gives the location of the start of the first occurence of the string `tata` in the string `dna`.

  The `0` says to start looking from the first character in `dna`. (Since the string is an array, the first character is numbered 0).
- We can also look for a single character:

  ```
  pos = dna.find('c', 0);
  ```

- If `dna.find` doesn’t find the thing we are looking for, it returns the value `dna.npos`.
- This gives us a neat way to search for things in `dna`.
- We keep looking until we get `dna.npos`.
- So, to count how many times we have `g` in `dna`, we would do this:

```
int countG = 0;

pos = dna.find(’g’,0);
while (pos != dna.npos)
{
    countG++;
    pos = dna.find(’g’, pos + 1);
}
```

- This code works as follows:

  1. We look for `g` starting at the beginning of the string.
  2. If we don't get `npos` we have found a `g`, so increase the counter.
  3. Look again, starting with the character just after the one you just found.
  4. Go to 2.

- This is a common way of using a `while` loop.

## Erase and insert

- Last time we saw how to use `replace` to exchange one bit of a string for another.
- To swap two bits of a string that aren’t the same length, we have to first `erase` one and then `insert` another.
- For example:

```
dna.erase(7, 4);
dna.insert(7, "ctctc");
```

  will remove the four characters of `dna` that start with the character in place number seven, and then insert the string `ctctc` at the same place.

- A slightly more sophisticated use of `insert` and `erase` is:

```
pos = dna.find("ggaa", 0);
dna.erase(pos, 4);
dna.insert(pos, "tatatt");
```

- This finds the location of the first string `ggaa`, `erase`s four characters at that position, and then `insert`s `tatatt` in the same place.
- The overall effect is to replace `ggaa` with `tatatt`

## One other thing

- Just as we can concatenate two strings using +

  ```
  dna = dna + dna2;
  ```

  we can combine concantenation and assignment using +=

  ```
  dna += dna2;
  ```

## cctype

- When we are processing strings, it is often useful to be able to identify what individual characters are.
- Clearly we can do this like so:

```
string s1

if(s1[2] == ’c’){
...
}
```

testing individual characters from a string against specific character constants.

- This is fine if we want to test against individual values, but is less helpful if we want, for example, to know if a specific character from a string is a lower case letter.
- Luckily there are some library functions that can help us out.
- The header file to use is

  ```
  #include <cctype>
  ```

  for the C-library `cctype`
- This includes the following functions.
- Note that they take an integer as an argument — you have to cast a character as an integer in order to use them, and return an integer.
- For most of the functions we want a true/false answer and if the integer that is returned is 0, that means `false`. If the integer is non-zero, that means `true`.

- These are some ofthe more useful functions:
- `int isalnum( int c )` checks if character argument is alphanumeric
- `int isalpha( int c )` checks if character argument is alphabetic
- `int isdigit( int c )` checks if character argument is a decimal digit
- `int islower( int c )` checks if character argument is a lowercase letter

- `int ispunct( int c )` checks if character argument is a punctuation character
- `int isupper( int c )` checks if character argument is an uppercase letter
- `int tolower( int c )` converts uppercase letter argument to lowercase
- `int toupper( int c )` converts lowercase letter argument to uppercase
- For these last two, the integer that is returned is the ASCII value of the corresponding letter — you'll have to cast it to a character.
- The on-line reference for **cctype** is: `http://www.cplusplus.com/reference/clibrary/cctype/`

```
#include <iostream>
#include <cctype>
using namespace std;

int main() {
  bool q = false;
  char c;
  while ( ! q ) {
    cout << "enter a character (q to quit): ";
    cin >> c;
    cout << "you entered: " << c << endl;
    if ( islower( (int)c )) {
      c = (char)toupper( c );
    }
    cout << "upper case = " << c << endl;
    q = ( c == ’Q’ );
  } // end while
} // end of main()
```

## Extracting numbers from strings

- Another thing we often want to do with strings is to extract numbers from them.
- `string s1 = "12";`

  is very different from

  `int i = 12;`
- To turn a set of numeric characters in a string into a number, the C standard library (`cstdlib`) provides the function `atoi`.
- Because it is a C-library function, it won't work directly on strings as we know them.
- Rather we have to use `atoi` like this:

  ```
  i = atoi(s1.c_str());
  ```

- There is a similar function `atof` which will convert a string representing a decimal number into a `double`.
- BTW, the member function

  `c_str()`

  generates what is known as a C-string, a string as it was represented in C.
- This is not a class, and has no member functions, but there are many functions that do for C-strings what the string member functions do for strings.
- As ever, these functions are documented in:
  `http://www.cplusplus.com/reference/clibrary/`

## Summary

- This lecture looked in some more detail at strings.
- We looked at some additional member functions, especially those that allow us to search in strings.
- We also looked at some functions from the C library that allow us to process string content.
  - Functions that tell us what kind of character we are dealing with.
  - Functions that convert numeric characters into numbers.
- We will talk more of strings in the next lecture.