Strings

An example of string use:

```c++
// Strings3.cpp
// Demonstrates the pre-defined STL string type
//--------------------------------------------------------
#include <iostream>
#include <iomanip>
#include <string>
#include <cstring>
#include <cstdio>
#include <cstdlib>
#include <cctype>
using namespace std;
//--------------------------------------------------------
int main()
{
    string str1 = "hello";
    string str2 = "world";

    cout << "str1 is '" << str1 << "' and str2 is '" << str2 << "' \n\n";
    cout << "Enter a new string for string 2: ";
    cin >> str2;
    cout << "str1 is '" << str1 << "' and str2 is '" << str2 << "' \n\n";
    str1 = str2 + " says hello";
    cout << "str1 is '" << str1 << "' and str2 is '" << str2 << "' \n\n";
    return 0;
}
```
A few tricks:

- Always include `<iostream>`, `<iomanip>`, `<string>`, `<cctype>`, `<cstring>`, `<cstdio>` and `<cstdlib>`. It doesn’t matter if they are included and not needed.

- Always put “using namespace std” underneath the includes.

- Any standard C function—such as `strcpy`—can be found in the man pages. Type: `man strcpy`.

- Conversion from lower case to upper case:
  ```cpp
  ch = toupper(ch);
  ```
  and vice-versa:
  ```cpp
  ch = tolower(ch);
  ```

- Conversion of an ascii digit (such as '8') to a number:
  ```cpp
  num = ch - '0';
  ```
  and vice-versa:
  ```cpp
  ch = num + '0';
  ```

- Getting the char* string out of an STL string:
  ```cpp
  str.c_str();
  ```

- Conversion of a char* string to an integer:
  ```cpp
  int num = atoi(str);
  ```
  or
  ```cpp
  sscanf (str, "%d", &num);
  ```

- Conversion of a char* string to a floating point:
  ```cpp
  float num = atof(str);
  ```
  or
  ```cpp
  sscanf (str, "%f", &num);
  ```

- Conversion of numbers to strings:
  ```cpp
  sprintf (str, "%d", &num);
  ```
  and
  ```cpp
  sprintf (str, "%f", &num);
  ```

- Copying a char* string to another char* string (but with no bounds checking!):
  ```cpp
  strcpy(targetStr, sourceStr);
  ```
  and to an STL string:
  ```cpp
  targetSTLStr = sourceCharStr;
  ```

- Comparing two char* strings:
  ```cpp
  if (strcmp(str1, str2) == 0)
  ```
  and to an STL string:
  ```cpp
  if (strcmp(str1, str2.c_str()) == 0))
  ```
  and two STL strings:
  ```cpp
  if (str1 == str2)
  ```

- Reading a whole line, with spaces, into a char*:
  ```cpp
  cin.getline(name, LINE_SIZE+1, '\n');
  ```
• Testing for types of characters, these all return 0 if false, they require the `<cctype>` header file:

  isalpha(ch), isascii(ch), isdigit(ch), isalnum(ch), ispunct(ch), isspace(ch), isupper(ch), isxdigit()

• If you need some kind of manipulation you don’t know about, then you can get information about most C type manipulation by typing “info libc” at the command line.

Output

1. New-fashioned C++ output makes some tasks easier because cout ‘knows’ the type it is trying to output. However I find formatting much more difficult. Make sure you always include `<iomanip>` so that you can use these functions. Examples of formatting are shown below:

   • Justification is set and then stays set until changed:
     To specify left justification:
     ```
     cout.setf(ios::left)
     ```
     or right:
     ```
     cout.setf(ios::right)
     ```
     to unset a justification:
     ```
     cout.unsetf(ios::right)
     ```

   • Column widths have to be set individually for each item output:
     ```
     cout << setw(8) << num1 << setw(8) << num2 << setw(20) << name << "\n";
     ```
   
   • Setting the column width will put the name in the example above right justified in the column. To make it left justified (as per normal), but then ensure all future output uses the default:
     ```
     cout << setw(8) << num1 << setw(8) << num2 << " ";
     cout.setf(ios::left)
     cout << setw(20) << name << "\n";
     cout.unsetf(ios::left)
     ```

   • To add leading zeros to pad a number for the width of the column:
     ```
     cout << setfill('0')
     ```
     and remember to turn it back to the default afterwards:
     ```
     cout << setfill(' ') 
     ```

   • To output in hexadecimal:
     ```
     cout.setf(ios::hex, ios::basefield)
     ```
     and return it to normal afterwards:
     ```
     cout.unsetf(ios::hex);
     ```

   • To output all floats in five decimal places until further notice:
     ```
     cout << fixed << setprecision(5) << "\n";
     ```
     and return it to normal:
     ```
     cout.unsetf(ios::fixed);
     ```
• If you do not know a width then a variable can be used (note also the use of two lines):
  
  cout << setw(maxWidth) << num1 << setw(maxWidth) << num2
  << setw(strWidth) << name << "\n";

2. Note that you can mix and match C++ and C input and output statements within one program with no problems.

3. If you want to use old-fashioned C output statements, that is fine (I do in competition stuff!). However you do need to make sure you tell `printf` what type you are outputting or you will get garbage out, not an error message!

Justification, column widths, leading 0s and type are all specified for each individual piece of output data. You need to include `<stdio>` and `<stdlib>`. Examples:

(a) A floating point number with the default decimal places:
  
  `printf ("%f", num);`

(b) A floating point number on a line of its own:
  
  `printf ("%f\n", num);`

(c) A floating point number with exactly 5 decimal places:
  
  `printf ("%.5f", num);`

(d) A floating point number with exactly 4 decimal places in a column 8 wide:
  
  `printf ("%8.4f", num);`

(e) An integer:
  
  `printf ("%d", num);`

(f) An integer in a column 8 wide:
  
  `printf ("%8d", num);`

(g) An integer in a column 6 wide, with leading 0s:
  
  `printf ("%06d", num);`

(h) A hexadecimal integer:
  
  `printf ("%x", num);`

(i) A pointer (address):
  
  `printf ("%p", num);`

(j) A string:
  
  `printf ("%s", str);`

(k) A string right justified in a column 20 wide:
  
  `printf ("%20s", str);`

(l) A string left justified in a column 20 wide:
  
  `printf ("%-20s", str);`

(m) A character as a character:
  
  `printf ("%c", ch);`

(n) A character as its ASCII code:
  
  `printf ("%d", ch);`

(o) Putting it all together (and note the use of two lines):
  
  `printf ("The character %c has the ASCII code %d, which is %x in hex\n", ch, ch, ch);`
Finally, if you do not know the width, a token can be used which is then replaced from the parameter list:

```
printf("%*d %*.*f\n", width1, num1, width2, decPlaces, num2);
```

**Task**

Write a program that takes as input a file with data such as:

```
Hello
This is a sentence
And another one
just some words.
World
END
```

and outputs the number of alphabetic characters in each string and the number of words in the following format:

```
STRING-----------------ALPHA CHARS---WORDS
"Hello" 5 1
"This is a sentence" 15 4
"And another one" 13 3
"just some words." 13 3
"World" 5 1
```

Note that the heading “ALPHA CHARS” is exactly three character over from the longest sentence, however the heading “WORDS” is always three characters after the previous heading.