Strings

Charlie Dey, Virginia Trueheart

Spring 2020

Characters

Characters and ints

- Type *char*;
- represents '7-bit ASCII': printable and (some) unprintable characters.
- Single quotes: `char c = 'a'`

Char / int equivalence

Equivalent to (short) integer:

Code:

```
char ex = 'x';
int x_num = ex, y_num = ex+1;
char why = y_num;
cout << "x is at position " << x_num
     << endl;
cout << "one further lies " << why
     << endl;
```

Output
[string] intchar:

```
x is at position 120
one further lies y
```

Also: `'x'-'a'` is distance `a--x`

Exercise 1

Write a program that accepts an integer $1 \cdots 26$ and prints the so-manieth letter of the alphabet.

Extend your program so that if the input is negative, it prints the minus-so-manieth uppercase letter of the alphabet.

Strings

String declaration

```
#include <string>
using std::string;

// .. and now you can use 'string'
```

(Do not use the C legacy mechanisms.)

String creation

A string variable contains a string of characters.

```
string txt;
```

You can initialize the string variable or assign it dynamically:

```
string txt{"this is text"};
string moretxt("this is also text");
txt = "and now it is another text";
```

Quotes in strings

You can escape a quote, or indicate that the whole string is to be taken literally:

Code:

```
string
  one("a b c"),
  two("a \"b\" c"),
  three( R"("a ""b """c)" );
cout << one << endl;
cout << two << endl;
cout << three << endl;
```

Output
[string] quote:

```
a b c
a "b" c
"a ""b """c
```

Concatenation

Strings can be *concatenated*:

```
txt = txt1+txt2;
txt += txt3;
```

String indexing

You can query the *size*:

```
int txtlen = txt.size();
```

or use subscripts:

```
cout << "The second character is <<"
     << txt[1] << ">>" << endl;
```

Ranging over a string

Same as ranging over vectors.

Range-based for:

Code:

```
cout << "By character: ";
for ( char c : abc )
  cout << c << " ";
cout << endl;
```

Output
[string] stringrange:

```
By character: a b c
```

Ranging by index:

Code:

```
string abc = "abc";
cout << "By character: ";
for (int ic=0; ic<abc.size(); ic++)
  cout << abc[ic] << " ";
cout << endl;
```

Output
[string] stringindex:

```
By character: a b c
```

Range with reference

Range-based for makes a copy of the element
You can also get a reference:

Code:

```
for ( char &c : abc )
  c += 1;
cout << "Shifted: " << abc << endl;
```

Output
[string]
stringrangeset:

```
Shifted: bcd
```

Review quiz 1

True or false?

- `'0'` is a valid value for a `char` variable
- `"0"` is a valid value for a `char` variable
- `"0"` is a valid value for a `string` variable
- `'a'+'b'` is a valid value for a `char` variable

Exercise 2

The oldest method of writing secret messages is the Caesar cypher. You would take an integer s and rotate every character of the text over that many positions:

$$s \equiv 3 \colon \text{"acdz"} \Rightarrow \text{"dfgc"}.$$

Write a program that accepts an integer and a string, and display the original string rotated over that many positions.

More vector methods

Other methods for the vector class apply: `insert`, `empty`, `erase`, `push_back`, et cetera.

Methods only for `string`: `find` and such.

`http://en.cppreference.com/w/cpp/string/basic_string`

Exercise 3

Write a function to print out the digits of a number: `156` should print `one five six`. You need to convert a digit to a string first; can you think of more than one way to do that?

Start by writing a program that reads a single digit and prints its name.

For the full program it is easiest to generate the digits last-to-first. Then figure out how to print them reversed.

Optional exercise 4

Write a function to convert an integer to a string: the input 215 should give `two hundred fifteen`, et cetera.

String stream

Like cout (including conversion from quantity to string), but to object, not to screen.

- Use the << operator to build it up; then
- use the *str* method to extract the string.

```
#include <sstream>
stringstream s;
s << "text" << 1.5;
cout << s.str() << endl;
```

Exercise 5

Use integer output to print real numbers aligned on the decimal:

Code:

```
string quasifix(double);
int main() {
  for ( auto x : { 1.5, 12.32, 123.456,
      1234.5678 } )
    cout << quasifix(x) << endl;
```

Output
[string] quasifix:

```
make[4]: *** No rule to make ta
```

Use four spaces for both the integer and fractional part; test only with numbers that fit this format.