# Modern C++ - Static typing

# 2. Static typing

The **static typing** means that the compiler decides about the (static) type of all expressions and subexpressions in compilation time. Under run-time these types do not change.

## Second C++ program: Fahrenheit to Celsius conversion

(aka. The Bad, the Ugly and the Good)

The task is to convert Fahrenheit values to Celsius between -100 and +400 with stepping by 25.

We start with a C-like solution.

```
 1 /*
 2  *  BAD VERSION !!!
 3  *  Convert Fahrenheit to Celsius
 4  *  between -100F and +400F  by 25F
 5  */
 6 #include <stdio.h>
 7 int main()
 8 {
 9   int fahr;
10
11   for ( fahr = -100; fahr <= 400; fahr += 25 )
12   {
13     printf( "Fahr = %d\tCels = %d\n", fahr, 5/9*(fahr-32) );
14   }
15   return 0;
16 }
```

```
$ g++ -ansi -pedantic -W -Wall fahr2cels.cpp -o fahr2cels

$ ./fahrenheit
Fahr = -100     Cels = 0
Fahr = -75      Cels = 0
Fahr = -50      Cels = 0
```

```
Fahr = -25      Cels = 0
Fahr = 0        Cels = 0
Fahr = 25       Cels = 0
Fahr = 50       Cels = 0
Fahr = 75       Cels = 0
Fahr = 100      Cels = 0
Fahr = 125      Cels = 0
Fahr = 150      Cels = 0
Fahr = 175      Cels = 0
Fahr = 200      Cels = 0
Fahr = 225      Cels = 0
Fahr = 250      Cels = 0
Fahr = 275      Cels = 0
Fahr = 300      Cels = 0
Fahr = 325      Cels = 0
Fahr = 350      Cels = 0
Fahr = 375      Cels = 0
Fahr = 400      Cels = 0
```

The reason is that in *strongly typed* programming languages the compiler decides the type of all (sub)expressions in compilation time, and that depends only *from the type* of the operands (and from the operator), *not from the value* of the operands. Thus 5/9 is always integer (with value 0).

Let try to fix it using 5./9. istead of 5/9

```
 1 /*
 2  *  BAD VERSION !!!
 3  *  Convert Fahrenheit to Celsius
 4  *  between -100F and +400F  by 25F
 5  */
 6 #include <stdio.h>
 7 int main()
 8 {
 9   int fahr;
10
11   for ( fahr = -100; fahr <= 400; fahr += 25 )
12   {
13     printf( "Fahr = %d\tCels = %d\n", fahr, 5./9.*(fahr-32) );
14   }
15   return 0;
```

```
16 }
```

```
$ g++ -ansi -pedantic -W -Wall fahr2cels.cpp -o fahr2cels
fahr2cels.cpp: In function ‘main’:
fahr2cels.cpp:17:5: warning: format ‘%d’ expects argument of type ‘int’, but
argument 3 has type ‘double’ [-Wformat=]
     printf( "Fahr = %d,\tCels = %d\n", fahr, 5./9.*(fahr-32) );
     ^

$ ./fahrenheit
Fahr = -100     Cels = 913552376
Fahr = -75      Cels = -722576928
Fahr = -50      Cels = -722576928
Fahr = -25      Cels = -722576928
Fahr = 0        Cels = -722576928
Fahr = 25       Cels = -722576928
Fahr = 50       Cels = -722576928
Fahr = 75       Cels = -722576928
Fahr = 100      Cels = -722576928
Fahr = 125      Cels = -722576928
Fahr = 150      Cels = -722576928
Fahr = 175      Cels = -722576928
Fahr = 200      Cels = -722576928
Fahr = 225      Cels = -722576928
Fahr = 250      Cels = -722576928
Fahr = 275      Cels = -722576928
Fahr = 300      Cels = -722576928
Fahr = 325      Cels = -722576928
Fahr = 350      Cels = -722576928
Fahr = 375      Cels = -722576928
Fahr = 400      Cels = -722576928
```

Still bad. Now the value of the celsius is computed correctly, but we try to print it as an integer: using *%d format*. What happened: we placed (a likely 8 byte) *double* to the stack, but used only 4 bytes (interpreted as *integer*) to print. Obviously, this is wrong.

```
1 /*
2  *  UGLY VERSION
3  *  Convert Fahrenheit to Celsius
4  *  between -100F and +400F  by 25F
5  */
```

```cpp
#include <stdio.h>
int main()
{
  int fahr;

  for ( fahr = -100; fahr <= 400; fahr += 25 )
  {
    printf( "Fahr = %d\tCels = %f\n", fahr, 5./9.*(fahr-32) );
  }
  return 0;
}
```

```
$ g++ -ansi -pedantic -W -Wall fahr2cels.cpp -o fahr2cels

$ ./fahr2cels
Fahr = -100     Cels = -73.333333
Fahr = -75      Cels = -59.444444
Fahr = -50      Cels = -45.555556
Fahr = -25      Cels = -31.666667
Fahr = 0        Cels = -17.777778
Fahr = 25       Cels = -3.888889
Fahr = 50       Cels = 10.000000
Fahr = 75       Cels = 23.888889
Fahr = 100      Cels = 37.777778
Fahr = 125      Cels = 51.666667
Fahr = 150      Cels = 65.555556
Fahr = 175      Cels = 79.444444
Fahr = 200      Cels = 93.333333
Fahr = 225      Cels = 107.222222
Fahr = 250      Cels = 121.111111
Fahr = 275      Cels = 135.000000
Fahr = 300      Cels = 148.888889
Fahr = 325      Cels = 162.777778
Fahr = 350      Cels = 176.666667
Fahr = 375      Cels = 190.555556
Fahr = 400      Cels = 204.444444
```

In C++ we use *operators* to read from an input stream and write to an output stream. These operators are *overloaded* on parameter types. Therefore we cannot make errors like the previous ones, because the either the correct operator is selected based on the

parameter(s) or there is syntax error if no available operator.

```
 1 //
 2 // fahr2cels: convert fahrenheit to celsius in [-100 ... 400] step 20
 3 // the ugly
 4 //
 5 #include <iostream>
 6
 7 int main()
 8 {
 9   for( int fahr = -100; fahr <= 400; fahr += 20 )
10   {
11     std::cout << "fahr = " << fahr
12               << ", cels = " << 5./9. * (fahr-32)
13               << std::endl;
14   }
15   return 0;
16 }
```

```
$ ./a.out
fahr = -100, cels = -73.3333
fahr = -80, cels = -62.2222
fahr = -60, cels = -51.1111
fahr = -40, cels = -40
fahr = -20, cels = -28.8889
fahr = 0, cels = -17.7778
fahr = 20, cels = -6.66667
fahr = 40, cels = 4.44444
fahr = 60, cels = 15.5556
fahr = 80, cels = 26.6667
fahr = 100, cels = 37.7778
fahr = 120, cels = 48.8889
fahr = 140, cels = 60
fahr = 160, cels = 71.1111
fahr = 180, cels = 82.2222
fahr = 200, cels = 93.3333
fahr = 220, cels = 104.444
fahr = 240, cels = 115.556
fahr = 260, cels = 126.667
fahr = 280, cels = 137.778
```

```
fahr = 300, cels = 148.889
fahr = 320, cels = 160
fahr = 340, cels = 171.111
fahr = 360, cels = 182.222
fahr = 380, cels = 193.333
fahr = 400, cels = 204.444
```

This works, but the input is not nicely formatted. We can use input and output formatters, called *manipulators* defined in header file to formatting input or output.

```
 1 //
 2 // fahr2cels: convert fahrenheit to celsius in [-100 ... 400] step 20
 3 // the good
 4 //
 5 #include <iostream>
 6 #include <iomanip>
 7
 8 int main()
 9 {
10   for( int fahr = -100; fahr <= 400; fahr += 20 )
11   {
12     std::cout << "fahr = " << std::setw(5) << fahr
13               << ", cels = " << std::setw(8) << 5./9. * (fahr-32)
14               << std::endl;
15   }
16   return 0;
17 }
```

```
$ g++ -ansi -pedantic -W -Wall fahr2cels.cpp -o fahr2cels
$ ./fahr2cels
Fahr = -100,    Cels =  -73.33
Fahr =  -75,    Cels =  -59.44
Fahr =  -50,    Cels =  -45.56
Fahr =  -25,    Cels =  -31.67
Fahr =    0,    Cels =  -17.78
Fahr =   25,    Cels =   -3.89
Fahr =   50,    Cels =   10.00
Fahr =   75,    Cels =   23.89
Fahr =  100,    Cels =   37.78
Fahr =  125,    Cels =   51.67
```

```
Fahr =  150,    Cels =   65.56
Fahr =  175,    Cels =   79.44
Fahr =  200,    Cels =   93.33
Fahr =  225,    Cels =  107.22
Fahr =  250,    Cels =  121.11
Fahr =  275,    Cels =  135.00
Fahr =  300,    Cels =  148.89
Fahr =  325,    Cels =  162.78
Fahr =  350,    Cels =  176.67
Fahr =  375,    Cels =  190.56
Fahr =  400,    Cels =  204.44
```

There are still room to improve. To help maintenance, we can lift out the *magic constants* from the source to the head of the program.

We can use *named constants* to remove magic numbers from the source and define program parameters in one well-defined place.

```
 1 //
 2 // fahr2cels: using const
 3 //
 4 #include <iostream>
 5 #include <iomanip>
 6
 7 const int lower = -100;
 8 const int upper =  400;
 9 const int step  =   40;
10
11 int main()
12 {
13   for( int fahr = lower; fahr <= upper; fahr += step )
14   {
15     std::cout << "fahr = " << std::setw(4) << fahr
16               << ", cels = " << std::fixed << std::setw(7)
17               << std::setprecision(2) << 5./9. * (fahr-32)
18               << std::endl;
19   }
20   return 0;
21 }
```

Also, the complex expression inside the printf expression is not easy to understand or

maintainde. We refactor the Fahrenheit to Celsius conversion to a separate function.

Recognize, that the *signature* of the function is **double fahr2cels(double)** and we pass an integer parameter. This is ok, since we declared the full prototype of the function, and the integer will be converted to double on parameter passing. This happens only when we declare the parameterlist.

```
 1 //
 2 // fahr2cels: with function call
 3 //
 4
 5 #include <iostream>
 6 #include <iomanip>
 7
 8 const int lower = -100;
 9 const int upper =  400;
10 const int step  =   20;
11
12 // declaration of the fahr2cels function
13 double fahr2cels(double);
14
15 int main()
16 {
17   for( int fahr = lower; fahr <= upper; fahr += step )
18   {
19     std::cout << "fahr = " << std::setw(4) << fahr
20               << ", cels = " << std::fixed << std::setw(7)
21               << std::setprecision(2)
22               << fahr2cels(fahr)  // call of fahr2cels(double)
23               << std::endl;
24   }
25   return 0;
26 }
27
28 // definition of the fahr2cels function
29 double fahr2cels(double f)
30 {
31   return 5./9. * (f-32);
32 }
```

## Homework:

1. Create a program printing your name. Compile, link, run.
2. Separate the program to 2 sources: one return the name, other prints Hints: - use char *my_name() as a function prototype. - use “%s” as a format string for printf