EDAF50 – C++ Programming

10. <chrono>. *Concurrency. Integer types.*

Sven Gestegård Robertz
Computer Science, LTH

2018

Outline

What is a value

The semantics of a value often include

- a quantity
- a number
- a unit

E.g `int length = 2;`

- two meters?
- two millimeters?

Including quantity and unit in the ***type*** *helps avoid mistakes.*

Time representation

- A "time value" can be either
  - A duration – a time interval
  - A point in time
    - relative to a particular *clock*
- Different units
  - seconds
  - milliseconds
  - nanoseconds
  - *manual conversion error prone*
- Different semantics
  - `duration + duration = duration`
  - `duration - duration = duration`
  - `time_point + duration = time_point`
  - `time_point - duration = time_point`
  - `time_point - time_point = duration`
  - `time_point + time_point =` *error*

Time representation
<chrono>

- Uses the type system to denote
  - if a value is a duration or a point in time
  - the unit used (seconds, milliseconds, etc.)
  - which clock a point in time is relative to
    - `system_clock` – wall clock time
    - `steady_clock` – stopwatch
- Uses compile-time computations for
  - conversions between units
    - implicit conversions when safe
    - explicit conversions when loosing information
    - E.g. `duration_cast<seconds>(milliseconds)`

Time representation
<chrono>

A duration is

- an *integer value* and
- *a ratio* (the number of seconds between two values).

```
std::chrono::nanoseconds duration</*signed int, at least 64 bits*/,
                                  std::nano>
std::chrono::microseconds duration</*signed int, at least 55 bits*/,
                                  std::micro>
std::chrono::milliseconds duration</*signed int, at least 45 bits*/,
                                   std::milli>
std::chrono::seconds duration</*signed integer, at least 35 bits*/>
std::chrono::minutes duration</*signed integer, at least 29 bits*/,
                              std::ratio<60>>
std::chrono::hours   duration</*signed integer, at least 23 bits*/,
                              std::ratio<3600>>
```

`std::ratio` provides compile-time rational arithmetic

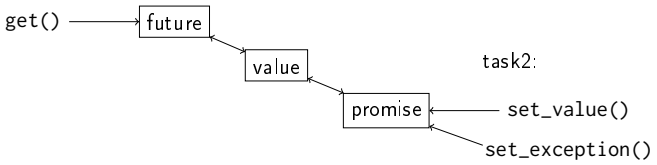
Concurrency

- Tasks and threads
- Passing arguments
- Returning results
- Sharing data
- Waiting for events
- Communicating tasks

Concurrency
futures and promises

- Transfer a value between tasks without an explicit lock
- A future represents a (possibly not yet existing) result of a computation
- A promise is used to deliver a value to a future

task1:



Concurrency
packaged_task

A future is connected to a promise

- create a promise
- get a future by calling promise::get_future()

More conventient to use a packaged_task

- a function (object) and the associated future and promise

Concurrency

Demo

Integer types

- Signed integers

Type	Size	Range (at least)
signed char	8 bits	$[-127, 127]^*$
short	at least 16 bits	$[-2^{15}+1,\ 2^{15}-1]$
int	at least 16 bits, usually 32	$[-2^{15}+1,\ 2^{15}-1]$
long	at least 32 bits	$[-2^{31}+1,\ 2^{31}-1]$
long long	at least 64 bits	$[-2^{63}+1,\ 2^{63}-1]$

*typically $[-128, 127]$, etc.

- Unsigned integers
  - same size as corresponding **signed** type
  - **unsigned char**: $[0, 255]$, **unsigned short**: $[0,\ 2^{16}-1]$. etc.
- special case
  - **char** (can be *represented* as **signed char** *or* **unsigned char**)
  - Use **char** only for characters
  - Use **signed char** or **unsigned char** for integer values
- Sizes according to the standard:
  char $\leq$ **short** $\leq$ **int** $\leq$ **long** $\leq$ **long long**

Integer types
Overflow

- overflow of an **unsigned** n-bit integer is defined as *the value modulo* 2^n
- overflow of a **signed** integer is *undefined*

Integer types

Example with sizeof

```
#include <iostream>
using namespace std;
int main () {
   cout << "sizeof(char)= \t" << sizeof(char)<<endl;
   cout << "sizeof(short)= \t" << sizeof(short) <<endl;
   cout << "sizeof(int) = \t" << sizeof(int) <<endl;
   cout << "sizeof(long)= \t" << sizeof(long)<<endl;
}
```

```
sizeof(char)=   1
sizeof(short)=  2
sizeof(int) =   4
sizeof(long)=   8
```


Integer types – Example of value range by casting
or: *be careful with casts* from **signed** to **unsigned** types

```
int main () {
 cout << "(signed char) -1 = " << (int)(signed char) -1 << endl;
 cout << "(unsigned char) -1 = " << (int)(unsigned char) -1 << endl;
 cout << "(short int) -1 = " << (short int) -1 << endl;
 cout << "(unsigned short int) -1 = "<<(unsigned short int)-1<<endl;
 cout << "(int) -1 = " << (int) -1 << endl;
 cout << "(unsigned int) -1 = " << (unsigned int) -1 << endl;
 cout << "(long) -1 = " << (long) -1 << endl;
 cout << "(unsigned long) -1 = " << (unsigned long) -1 << endl;

}
```

```
 (char) -1 = -1
 (unsigned char) -1 = 255
 (short int) -1 = -1
 (unsigned short int) -1 = 65535
 (int) -1 = -1
 (unsigned int) -1 = 4294967295
 (long) -1 = -1
 (unsigned long) -1 = 18446744073709551615
```


Integer types
Sizes are specified in <climits>

```
CHAR_BIT    Number of bits in a char object (byte)  (>=8)
SCHAR_MIN   Minimum value for an object of type signed char
SCHAR_MAX   Maximum value for an object of type signed char
UCHAR_MAX   Maximum value for an object of type unsigned char
CHAR_MIN    Minimum value for an object of type char
            (either SCHAR_MIN or 0)
CHAR_MAX    Maximum value for an object of type char
            (either SCHAR_MAX or UCHAR_MAX)
SHRT_MIN    Minimum value for an object of type short int
SHRT_MAX    Maximum value for an object of type short int
USHRT_MAX   Maximum value for an object of type unsigned short int
INT_MIN     Minimum value for an object of type int
INT_MAX     Maximum value for an object of type int
UINT_MAX    Maximum value for an object of type unsigned int
LONG_MIN    Minimum value for an object of type long int
LONG_MAX    Maximum value for an object of type long int
ULONG_MAX   Maximum value for an object of type unsigned long int
LLONG_MIN   Minimum value for an object of type long long int
LLONG_MAX   Maximum value for an object of type long long int
ULLONG_MAX  Maximum value for an object of type unsigned long long
```


Integer types
Sizes are specified in <climits>

```
#include <iostream>
#include <climits>
int main()
{
    std::cout << CHAR_MIN << ", " << CHAR_MAX << ", ";
    std::cout << UCHAR_MAX << std::endl;
    std::cout << SHRT_MIN << ", " << SHRT_MAX << ", ";
    std::cout << USHRT_MAX << std::endl;
    std::cout << INT_MIN << ", " << INT_MAX << ", ";
    std::cout << UINT_MAX << std::endl;
    std::cout << LONG_MIN << ", " << LONG_MAX << ", ";
    std::cout << ULONG_MAX << std::endl;
    std::cout << LLONG_MIN << ", " << LLONG_MAX << ", ";
    std::cout << ULLONG_MAX << std::endl;

}
```

```
128, 127, 255
-32768, 32767, 65535
-2147483648, 2147483647, 4294967295
-9223372036854775808, 9223372036854775807, 18446744073709551615
-9223372036854775808, 9223372036854775807, 18446744073709551615
```


Integer types
Sizes are implementation defined

Typedefs for specific sizes are in <cstdint> (<stdint.h>)

- integer types with exact with:
  `int8_t int16_t int32_t int64_t`
- fastest signed integer type with at least the width
  `int_fast8_t int_fast16_t int_fast32_t int_fast64_t`
- smallest signed integer type with at least the width
  `int_least8_t int_least16_t int_least32_t int_least64_t`
- signed integer type capable of holding a pointer:
  `intptr_t`
- unsigned integer type capable of holding a pointer:
  `uintptr_t`

The corresponding unsigned typedefs are named uint_..._t

Next lecture
Low-level details and loose ends

References to sections in Lippman

C-style strings 3.5.4
Multi-dimensional arrays 3.6
Bitwise operations 4.8
The comma operator 4.10
Union 19.6
Bit-fields 19.8.1

Suggested reading

References to sections in Lippman

Built-in types 2.1