# Contemporary C++: Learning Modern C++ in a Modern Way

**الماس فناوری ابری پاسارگاد- آلفا**

مدرس: سعید امراللهی بیوکی

# Agenda 7/24

## Session 7. Writing bread-and-butter programs (part I) and some technicalities (part I)

- Duff’s device
- Binary search: Recursion and recursive functions
- Function name overloading
- Some namespace technicalities: using declaration and using directive
- Constant references
- Time and chrono utility
- Fibonacci sequence: Two approaches- recursive and iterative
- long long types
- Counting function call: the static local variable
- Jagged arrays
- Removing right angle bracket problem
- Q&A

150 min (incl. Q & A)

# Duff's device

- Tom Duff
- Real-time animation program, 1983
- Loop unrolling
- The basic idea of loop unrolling is that the number of instructions executed in a loop can be reduced by reducing the number of loop tests, sometimes reducing the amount of time spent in the loop.
- Do-while + switch statement
- C's case label fallthrough

```c
void send(int* to, int* from, int count)
{
        int n = (count + 7) / 8;
        switch (count % 8) {
        case 0:  do { *to++ = *from++;
        case 7:       *to++ = *from++;
        case 6:       *to++ = *from++;
        case 5:       *to++ = *from++;
        case 4:       *to++ = *from++;
        case 3:       *to++ = *from++;
        case 2:       *to++ = *from++;
        case 1:       *to++ = *from++;
                } while (--n > 0);
        }
}
```

# Duff's device cont.

- Loop unrolling

1

```
void execute_loop(int & data, int loop_size)
{
    for (int i = 0 ; i < loop_size ; ++i)
    {
        computation(data);
    }
}
```

2

```
void execute_loop(int & data, const int loop_size)
{
    for (int i = 0 ; i < loop_size/4 ; ++i)
    {
        computation(data);
        computation(data);
        computation(data);
        computation(data);
    }
}
```

3

```
void execute_loop(int & data, int loop_size)
{
    int i = 0;
    switch(loop_size%4)
    {
        do{
            case 0: computation(data);
            case 3: computation(data);
            case 2: computation(data);
            case 1: computation(data);
            ++i;
        } while (i < (loop_size+3)/4);
    }
}
```

- Program speed vs. binary size

## Binary search

# Binary search

- Binary search algorithm:

… if a sequence is sorted, we can check the midpoint of the sequence against and eliminate half of the sequence from further consideration …

- Thoms Cormen et al. Introduction to Algorithms, Addison-Wesley, 2009, 2nd edition.

## Binary search

- Binary search algorithm:

… if a sequence is sorted, we can check the midpoint of the sequence against and eliminate half of the sequence from further consideration …

- Thoms Cormen et al. Introduction to Algorithms, Addison-Wesley, 2009, 2nd edition.

- Sorted elements

## Binary search

- Binary search algorithm:

… if a sequence is sorted, we can check the midpoint of the sequence against and eliminate half of the sequence from further consideration …

- Thoms Cormen et al. Introduction to Algorithms, Addison-Wesley, 2009, 2nd edition.

- Sorted elements
- Halving method

## Binary search

- Binary search algorithm:

… if a sequence is sorted, we can check the midpoint of the sequence against and eliminate half of the sequence from further consideration …

- Thoms Cormen et al. Introduction to Algorithms, Addison-Wesley, 2009, 2nd edition.

- Sorted elements
- Halving method
- Worst case: O(log n)

## Binary search

- Binary search algorithm:

… if a sequence is sorted, we can check the midpoint of the sequence against and eliminate half of the sequence from further consideration …

- Thoms Cormen et al. Introduction to Algorithms, Addison-Wesley, 2009, 2nd edition.

- Sorted elements
- Halving method
- Worst case: O(log n)
- $\text{Log}_2$ 1’048’576 = 20

# Binary search

- Binary search algorithm:

… if a sequence is sorted, we can check the midpoint of the sequence against and eliminate half of the sequence from further consideration …

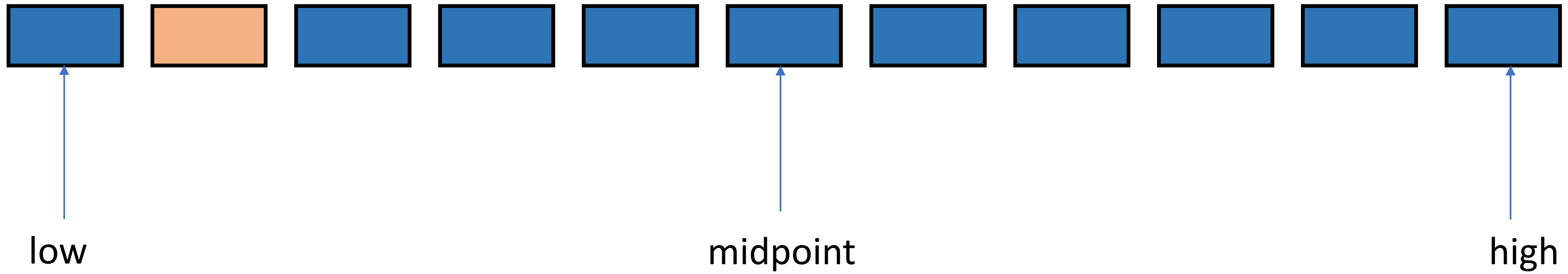
- Thoms Cormen et al. Introduction to Algorithms, Addison-Wesley, 2009, 2nd edition.

- Sorted elements
- Halving method
- Worst case: O(log n)
- $\mathrm{Log}_2$ 1’048’576 = 20
- Jon Bentley: Programming Pearls

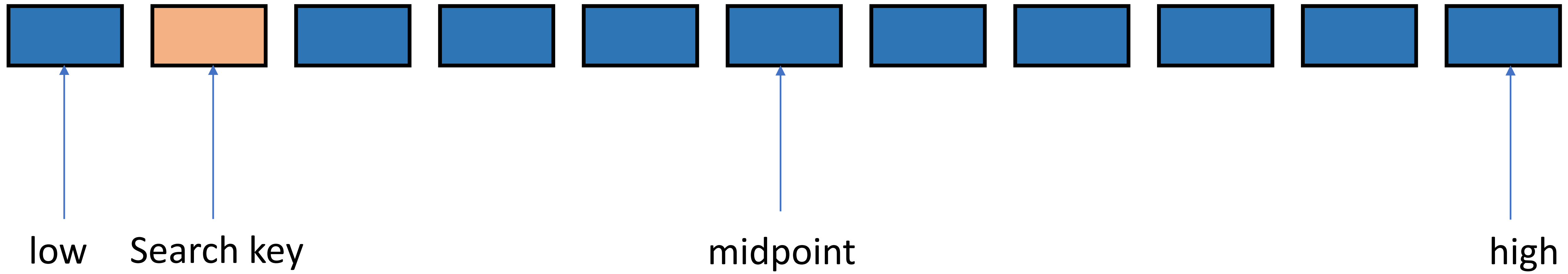
… Only 10% of programmers can write a binary search …

## Binary search

# Binary search

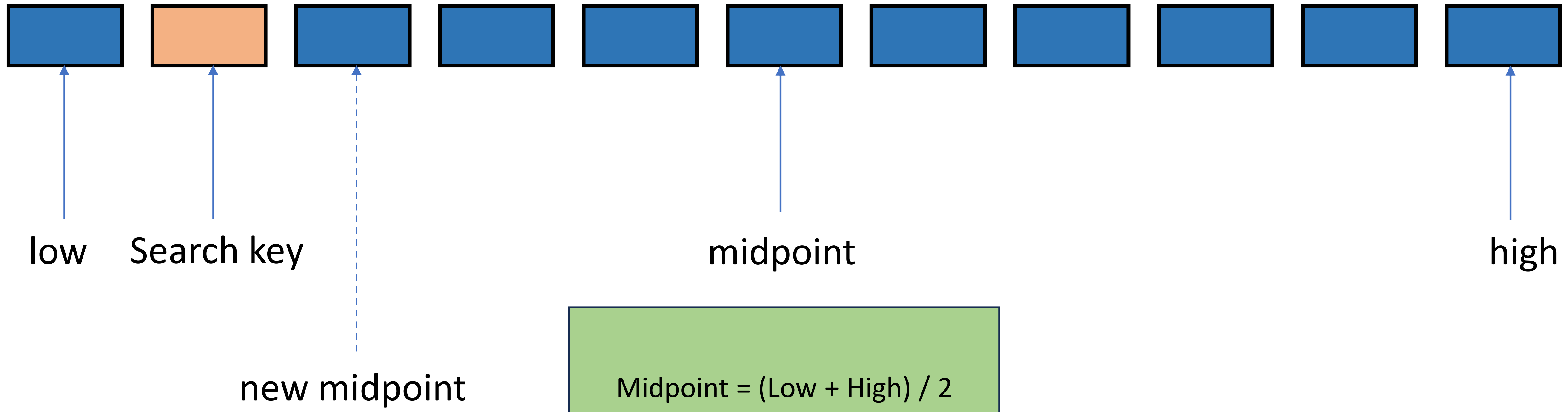


## Binary search



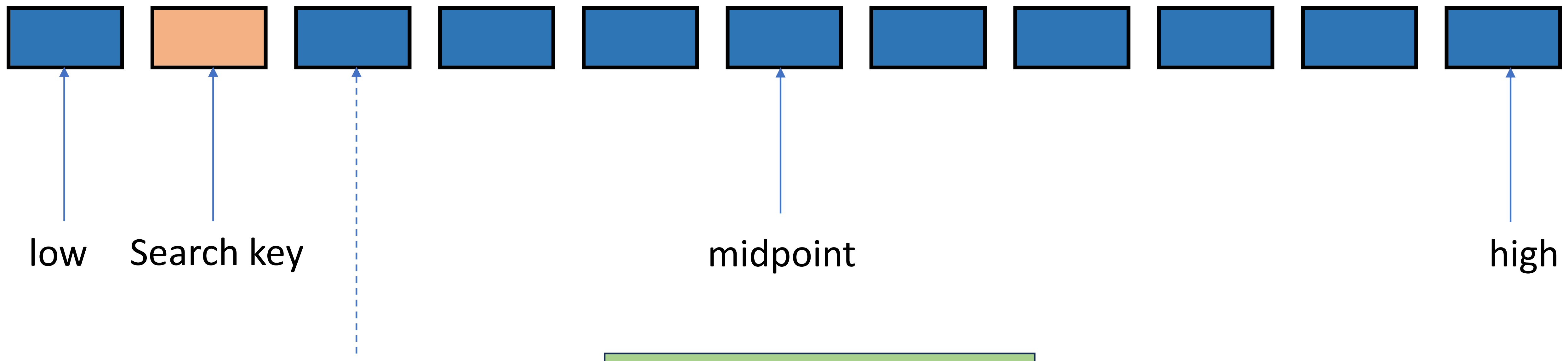
Midpoint = (Low + High) / 2

## Binary search



Midpoint = (Low + High) / 2

# Binary search



Midpoint = (Low + High) / 2

## Binary search- some technicalities

- Function name overloading
- Constant references
- Namespaces: using declaration and using directive
- Time and benchmark the binary search: The chrono library
- iota and is_sorted generic functions
- ?: conditional operator

## Function name overloading

- different functions → different names

```
// ugly
void print_int(int);
void print_char(char);
void print_string(const char*); // C-style string
```

```
// good
void print (int);
void print (char);
void print (const char*); // C-style string
```

```
// conceptually perform the same task
int max(int, int); // return maximum of two integers
int max(const int*, int); // return maximum of array of integers
int max(const List&); // return maximum of list of something
double max(double, double); // return maximum of two doubles
```

```
// overloaded +
int i = 2 + 2; // int + int
double d = 2.1 + 3.4; // double + double
bool b = true + false; // bool + bool
int a[10];
int* p = &a;
p = p + 5;
```

- Using the same name for operations on different types is called overloading.

## Resolving an overloaded function call

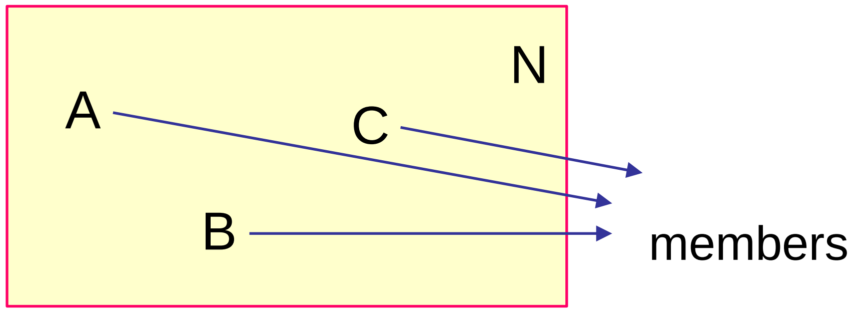
- Exact match
- Match using promotion
- Match using standard conversion
- Match using user-defined conversions
- Match using the ellipsis ... in a function declaration

Higher priority

Lower priority

# Namespace: Some technicalities

- A namespace is a scope.



- Qualified names
- using declaration
- using directive

*namespace name*::*member name*

using *namespace name*::*member name*

using namespace *namespace name*

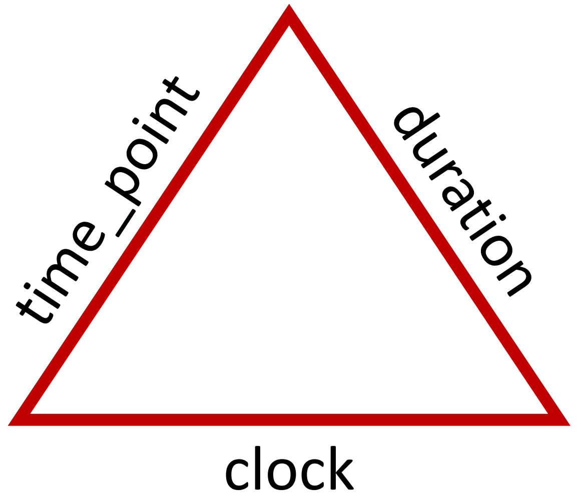
```
// Hello world: using declaration
#include <iostream>
using std::cout;
int main()
{
  cout << "Hello, world!" << std::endl;

  return 0;
}
```

```
N::A; // qualified name
using N::A; // using declaration
using namespace N; // using directive
```

```
// Hello world: using directive
#include <iostream>
using namespace std;
int main()
{
  cout << "Hello, world!" << endl;

  return 0;
}
```

## the Chrono namespace

- using namespace std::chrono
- All chrono facilities are in the std::chrono (sub)namespace



- Why does the chrono header has its own namespace? Q

A To avoid potential name collision.

# Duration

- The type duration to represent the time between two points in time (*time_points*).

  - A duration of time is defined as a specific number of ticks over a time unit.

type of ticks

```
template <class Rep, /* representation */ class Period = std::ratio<1>>
class duration {
public:
     using rep = Rep; // the type of tick
     using period = Period; // unit type in seconds
};
```

Unit types

- Examples:

```
duration<long long, milli> d1{7}; // 7 milliseconds
duration<int> twentySeconds(20);
duration<double, std::ratio<60>> halfAMinute{0.5}; // 0.5 * (60/1 seconds) = 30 sec
duration<long, std::ratio<1,1000>> oneMillisecond{1}; // 1 * (1/1000 seconds) = 1/1000 sec
duration<double, pico> p{3.33};
cout << d1.count() << '\n'; // 7
cout << twentySeconds.count() << '\n'; // 20
cout <<  p.count() << '\n'; // 3.33
```

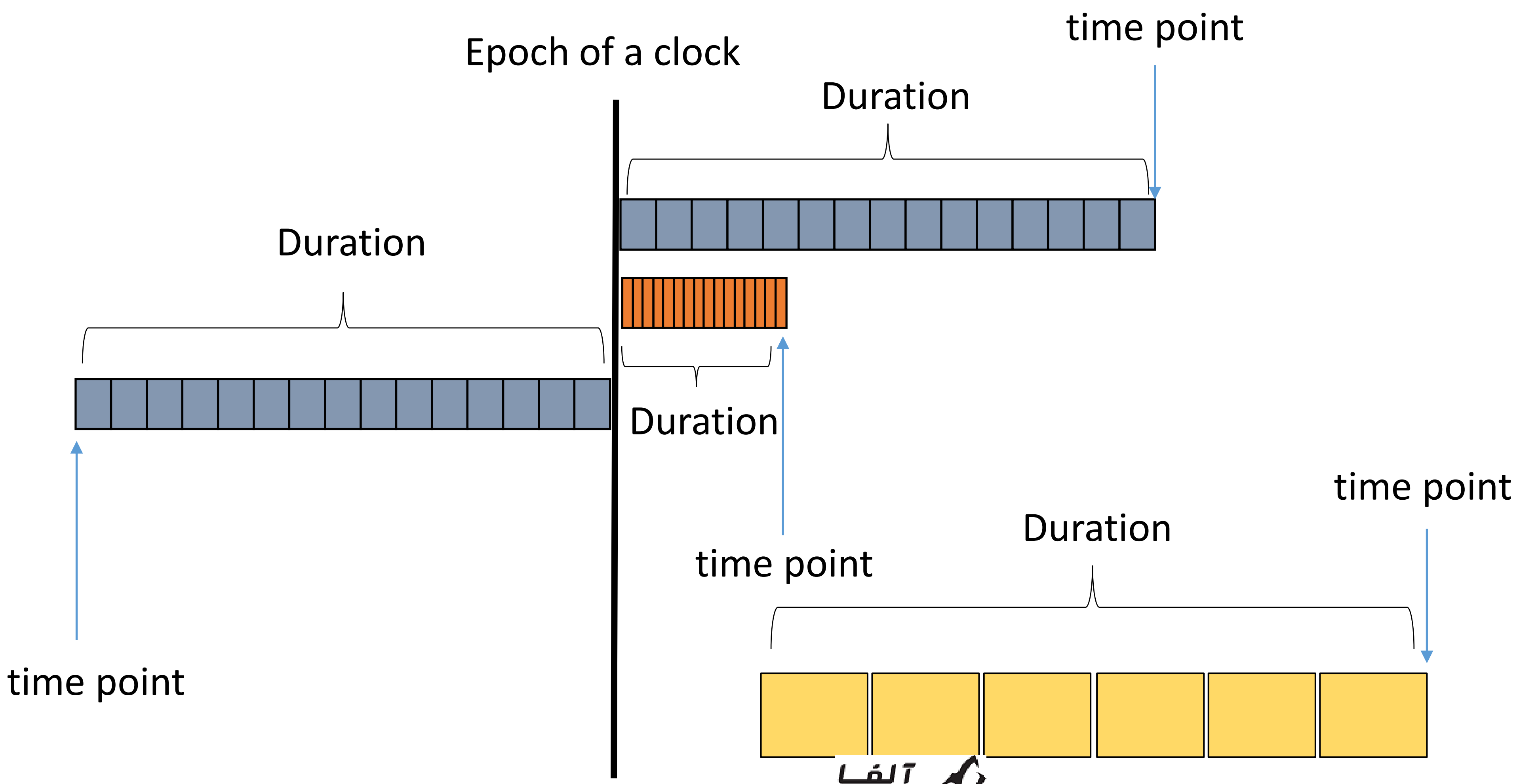
## System clock

- Objects of class system_clock represent wall clock time from the system-wide real-time clock.

```
class system_clock {
public:
     using rep = /* implementation-defined signed type */
     using period = /* implementation-defined ration<> */
     using duration = chrono::duration<rep, period>;
     using time_point = chrono::time_point<system_clock>;
     static time_point now() noexcept;
     // …
};
```

## Epoch, duration and time point

- A timepoint is defined as combination of a duration and a beginning of time (the so-called epoch).



ALPHA

## High-resolution clock and duration_cast: an example

```
#include <chrono>
#include <iostream>

using std::cout;                    using std::chrono::duration_cast;
using std::chrono::nanoseconds; using std::chrono::high_resolution_clock;

void do_work() { /* … */ }

int main()
{
     using Clock = high_resolution_clock;

     auto t0 = Clock::now();
     do_work();
     auto t1 = Clock::now();
     auto nano =  duration_cast<nanoseconds>(t1 - t0).count();
     cout << "do_work() takes" << nano << " nanoseconds!\n";
}
```

## Fibonacci sequence

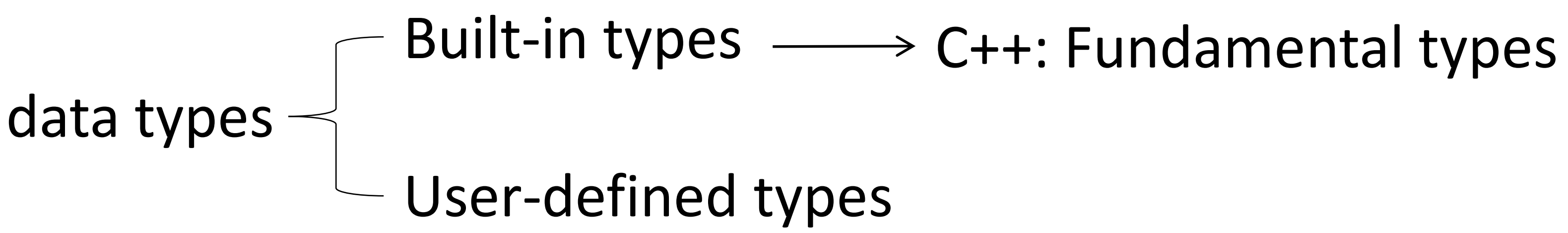
In mathematics, the **Fibonacci sequence** is a sequence in which each number is the sum of the two preceding ones. Numbers that are part of the Fibonacci sequence are known as **Fibonacci numbers**, commonly denoted $F_n$.

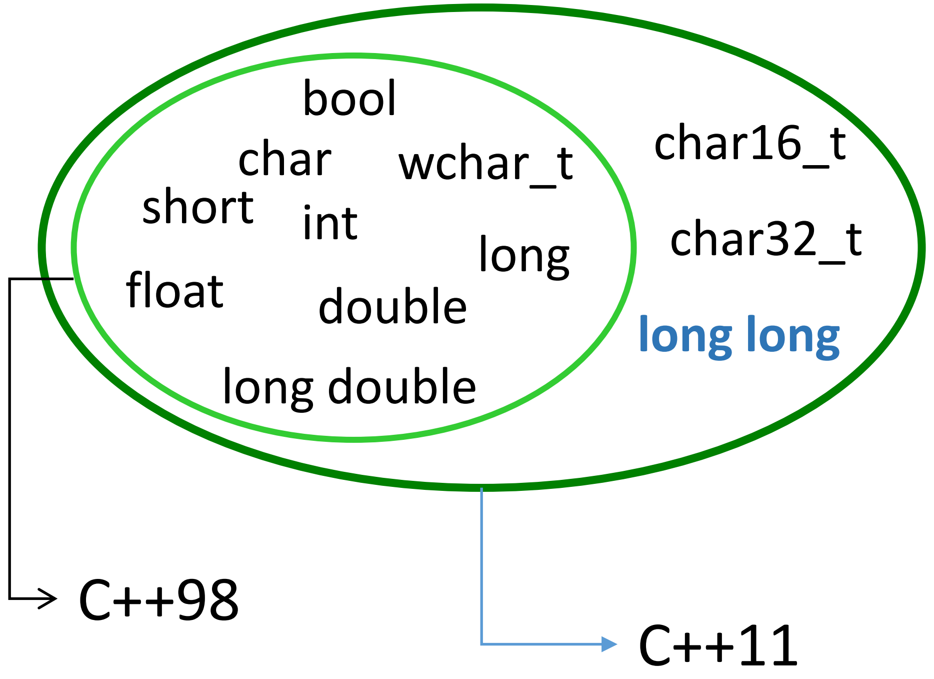
$F_0 = 0, F_1 = 1$

$F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 1$

- Iterative vs. Recursive Fibonacci sequence generation

# Longer integers: long long





- A longer integer that's <u>at least</u> 64-bits.
- In 2's complement representation:
  $[-2^{n-1}, 2^{n-1} - 1]$, n = number of bits
- signed long long, unsigned long long
- There is no *long long long* or *short long long*.

```
long long int too_large = 3000000000000; // No comment ;)
long long RAM = 4E9; // 4GB RAM
long long wp = 7000000000; // World population
int shares = 500000000; // # shares at TSE
short price = 4000; // the closing price of each share in IRR
long long capital = shares * price; // the corp. capital
long long dataset_size = 16E9; // typical facebook dataset size
long long chunk_handle; // Google File System (GFS) chunk handle
```

- C99: First added to C

## Long long- cont.

• Many important things about C++ fundamental data types like the exact size of types are *implementation-defined* by the standard.

sizeof(char) == 1
sizeof(long) <= sizeof(long long)
sizeof(long long) >= 8
…

• <limits> header file

```
template <class T> class numeric_limits {
public:
   // uninteresting defaults
};
```

← class template

```
template <> class numeric_limits<long long> {
public:
     inline static long long max()
     {
          return 9223372036854775807; // largest value
     }
     inline static long long min()
     {
          return -9223372036854775808; // smallest value
     }
     // ...
};
```

←template specialization

```
const unsigned long long int a = 1LLU;
const long long int b = -2LL;
```

• long long int suffix to make literals : LL, ll for long long and both LL/ll and u/U for unsigned long long

## static (local) variables

- A local variable is initialized when the thread of execution reaches its definition.

thread of execution

```
void f() {
  int i = 1; // automatic variable
}

int main() {
  for (int i = 0; i < 1000; ++i) {
    f(); // i initialized 1000 times
  }
}
```

thread of execution

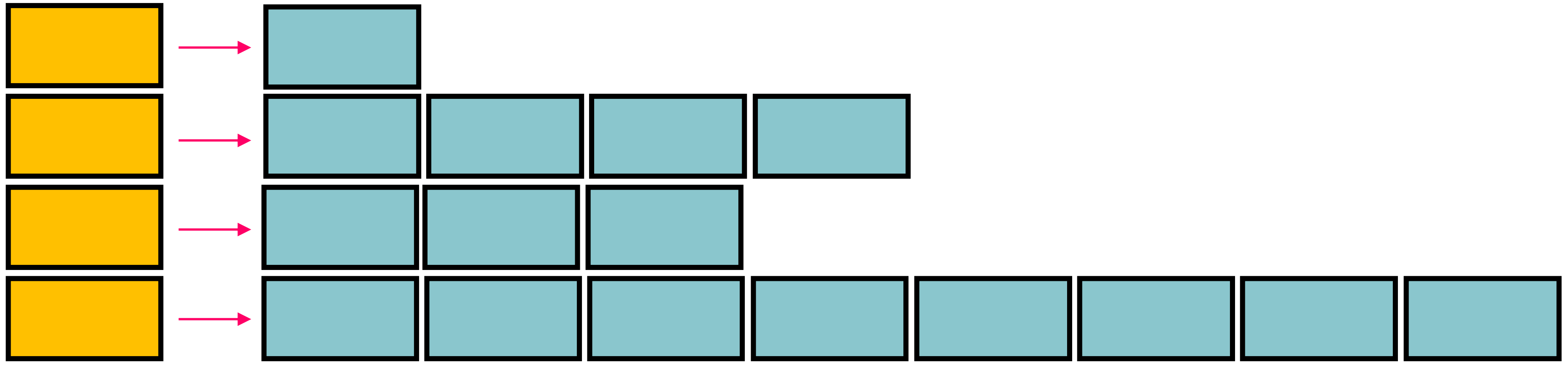
```
void f() {
  static int i = 1; // static variable
}

int main() {
  for (int i = 0; i < 1000; ++i) {
    f(); // i initialized once
  }
}
```

```
void f(int a)
{
  while (a--) {
    static int n = 0; // initialized once
    int x = 0; // initialized n times
    cout << "n == " << n++ << ", x == " << x++ << ´\n ´;
  }
}
int main()
{
  f(3);
}
```

# Jagged arrays

- https://en.wikipedia.org/wiki/Jagged_array

In computer science, a **jagged array**, also known as a **ragged array** [1] or **irregular array** is an array of arrays of which the member arrays can be of different lengths, producing rows of jagged edges when visualized as output.



# Removing right-angle brackets problem

```
template
<
  class T
>
class vector {
  // …
};
```

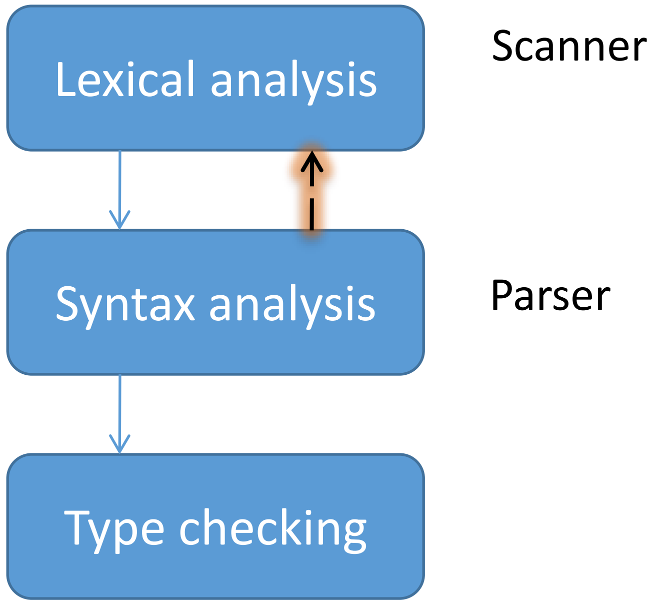
Angle brackets

```
int i = 32;
i >> 2;   // i == 8
cin >> i; // read from standard input and put to i
```

```
vector<list<int> > vl; // additional space
stack<complex<int>> sc; // error in C++98
vector<list<map<int, string>>> vlm; // ok in C++11
```

- The *Maximum Munch* principle
  - "unsigned long int" is one token.
  - "++" is one token.
  - ">>=" is one token.

- Compilation phases
  - Lexical analysis: make tokens
  - Syntax analysis: check the grammar
  - Type checking: find the type of names



- The > token following the template-parameter-list of a template-declaration may be the product of replacing a >> token by two consecutive > tokens.

# Programming projects

1. Design and Implementation of a simple Desk Calculator
2. Design and Implementation of a typical Phonebook
3. Design and Implementation of doubly linked-list
4. Design and Implementation of simple Producer/Consumer

# *Thanks for your patience …*

A man who asks a question is a fool for minute,
The man who does not ask, is a fool for a life.
- Confucius

Learning to ask the right (often hard) questions is an essential part of learning to think as a programmer.
- Bjarne Stroustrup *programming Principles and Practice Using C++, page 4.*

There is no stupid question, but there is stupid answer.
- Howard Hinnant

Q & A