



Uniwersytet
Wrocławski

Programowanie współbieżne w C++11

Zbigniew Koza

Wydział Fizyki i Astronomii

Co to jest współbieżność?

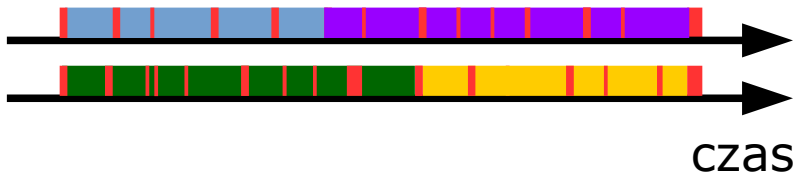
Przetwarzanie:



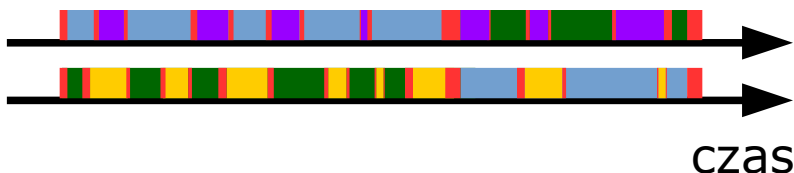
- Szeregowe



- **Współbieżne**



- Równoległe



- **Równoległe i współbieżne**

Problemy

- Synchronizacja
- Bezpieczna wymiana informacji
- Nigdy nie wiemy, kiedy nasz wątek zostanie wywłaszczony
- Potrzebujemy mechanizmów pozwalających stwierdzić, czy/kiedy zasoby współdzielone z innymi wątkami zostały zmodyfikowane

```
#include <thread>
```

Pierwszy program

```
#include <iostream>
#include <thread>

void my_fun_pl()
{
    std::cout << u8"Witaj, świecie!\n";
}

int main()
{
    std::thread th(&my_fun_pl);
    std::cout << "Hello, world!\n";
    th.join();
}
```

Pierwszy program

- **#include <thread>**

```
#include <iostream>
#include <thread>

void my_fun_pl()
{
    std::cout << u8"Witaj, świecie!\n";
}

int main()
{
    std::thread th(&my_fun_pl);
    std::cout << "Hello, world!\n";
    th.join();
}
```

Pierwszy program

- `#include <thread>`
- **Definicja obiektu zarządzającego funkcją wywoływaną współbieżnie**

```
#include <iostream>
#include <thread>

void my_fun_pl()
{
    std::cout << u8"Witaj, świecie!\n";
}

int main()
{
    std::thread th(&my_fun_pl);
    std::cout << "Hello, world!\n";
    th.join();
}
```


Pierwszy program

```
#include <iostream>
#include <thread>

void my_fun_pl()
{
    std::cout << u8"Witaj, świecie!\n";
}

int main()
{
    std::thread th(&my_fun_pl);
    std::cout << "Hello, world!\n";
    th.join();
}
```

- `#include <thread>`
- Definicja obiektu zarządzającego funkcją wywoływaną wspólnie
- **Synchronizacja z wątkiem głównym**

Kompilacja

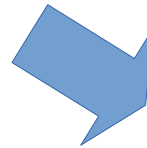
```
> g++ 1.cpp -pthread
```

Wynik

```
#include <iostream>
#include <thread>

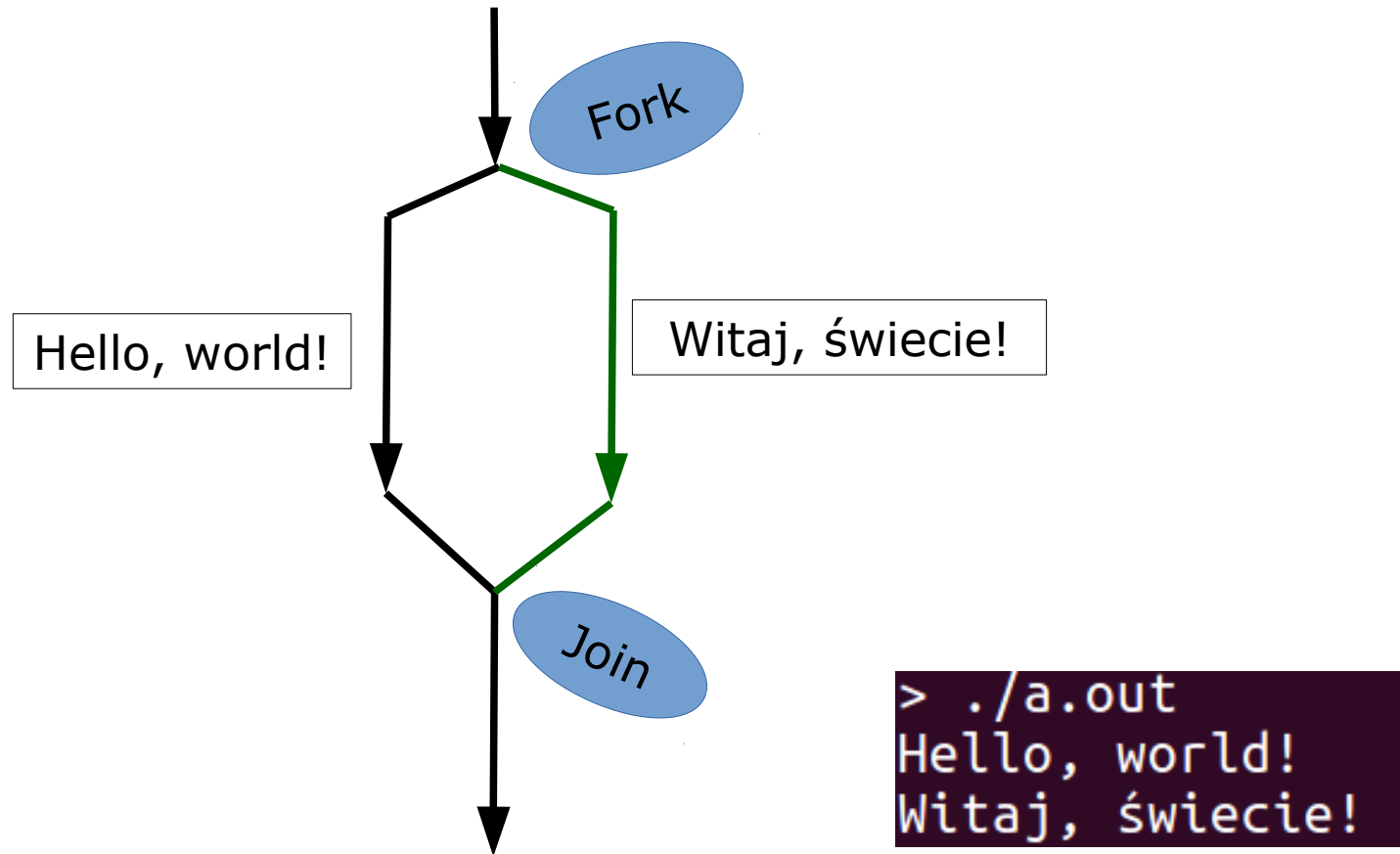
void my_fun_pl()
{
    std::cout << u8"Witaj, świecie!\n";
}

int main()
{
    std::thread th(&my_fun_pl);
    std::cout << "Hello, world!\n";
    th.join();
}
```

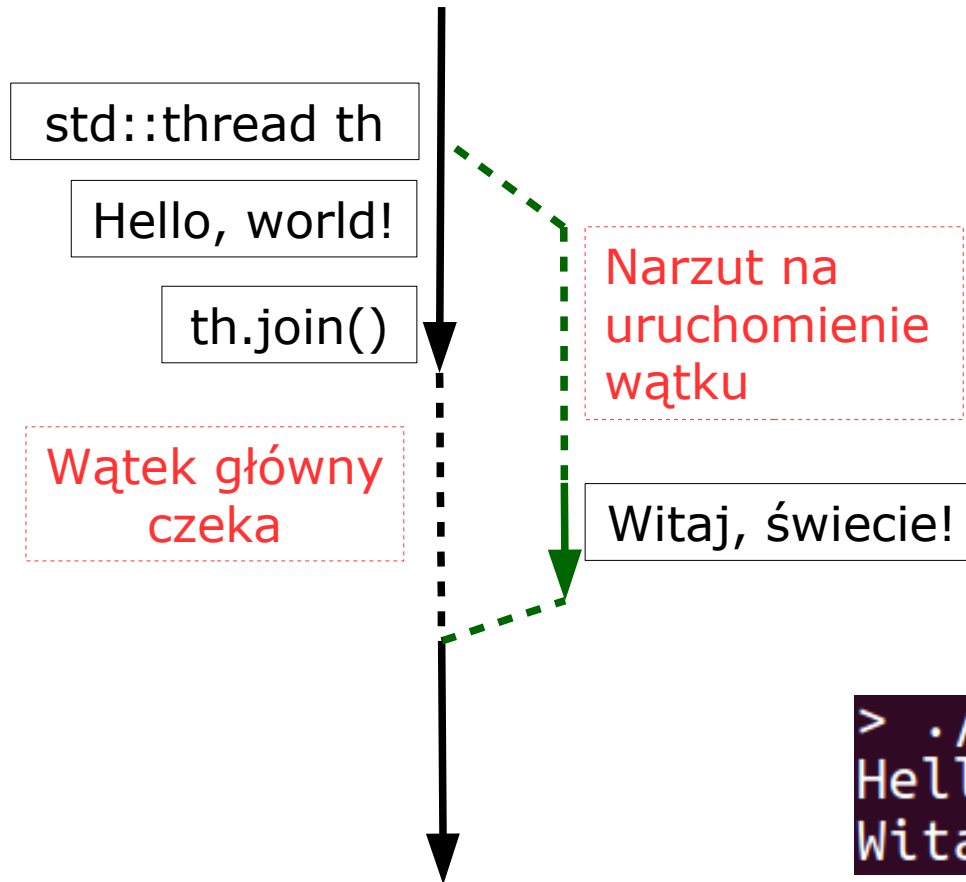


```
> ./a.out
Hello, world!
Witaj, świecie!
```

Fork & join



Fork & join



```
> ./a.out  
Hello, world!  
Witaj, świecie!
```

Uwaga

- Potrafimy uruchomić drugi wątek, ale wciąż nie znamy mechanizmów synchronizacji i wymiany danych...
- Wciąż nie potrafimy w bezpieczny i efektywny sposób korzystać ze współbieżności...

Przykład 2

```
#include <iostream>    // std::cout
#include <thread>      // std::thread, std::this_thread::sleep_for
#include <chrono>       // std::chrono::seconds

void pause_thread(int n)
{
    std::this_thread::sleep_for (std::chrono::seconds(n));
    std::cout << "pause of " << n << " seconds ended\n";
}

int main()
{
    std::cout << "Spawning 3 threads...\n";
    std::thread t1 (pause_thread, 1);
    std::thread t2 (pause_thread, 2);
    std::thread t3 (pause_thread, 3);
    std::cout << "Done spawning threads. Now waiting for them to join:\n";
    t1.join();
    t2.join();
    t3.join();
    std::cout << "All threads joined!\n";
    return 0;
}
```

// źródło: <http://www.cplusplus.com/reference/thread/thread/join/>

Przykład 2

```
#include <iostream>    // std::cout
#include <thread>       // std::thread, std::this_thread::sleep_for
#include <chrono>       // std::chrono::seconds

void pause_thread(int n)
{
    std::this_thread::sleep_for (std::chrono::seconds(n));
    std::cout << "pause of " << n << " seconds ended\n";
}

int main()
{
    std::cout << "Spawning 3 threads...\n";
    std::thread t1 (pause_thread, 1);
    std::thread t2 (pause_thread, 2);
    std::thread t3 (pause_thread, 3);
    std::cout << "Done spawning threads. Now waiting for them to join:\n";
    t1.join();
    t2.join();
    t3.join();
    std::cout << "All threads joined!\n";
    return 0;
}
```

```
Spawning 3 threads...
Done spawning threads. Now waiting for them to join:
pause of 1 seconds ended
pause of 2 seconds ended
pause of 3 seconds ended
All threads joined!
```


Funkcja składowa `join`

- Metoda `join` czeka na zakończenie funkcji, którą zarządza dany obiekt, i dopiero wtedy sama kończy swoje działanie
- Metoda ta tworzy więc **barierę** dla wątku głównego
- Jest to jedna z metod **synchronizacji** wątku głównego z wątkami pobocznymi

Alternatywa dla join

- `thread.detach()`
- Tworzy „wątek działający w tle”
(*deamon thread*)

Nie ma alternatywy dla `join`

- ~~• `thread.detach()`~~
- ~~• Tworzy „wątek działający w tle”
(*deamon thread*)~~
- Nie kombinuj, nie komplikuj sobie życia

Co, jeśli zapomnimy o join?

```
~thread()  
{  
    if (joinable())  
        std::terminate();  
}
```



- Program „pada”

```
Spawning 3 threads...  
Done spawning threads. Now waiting for them to join:  
pause of 1 seconds ended  
pause of 2 seconds ended  
pause of 3 seconds ended  
All threads joined!  
terminate called without an active exception  
Przerwane (zrzut pamięci)
```

Dlaczego `join` nie jest wywoływane w destruktorze?

- Jeśli w wątku głównym zgłoszono by **wyjątek**, to automatyczne uruchomienie `join` w destruktorze `std::thread` mogłoby zablokować wątek główny (bariera!), a więc i obsługę zgłoszonego wyjątku, zwłaszcza jeśli wątek poboczny oczekiwałby na dane z zawieszzonego wątku głównego
- => *deadlock* (zakleszczenie)

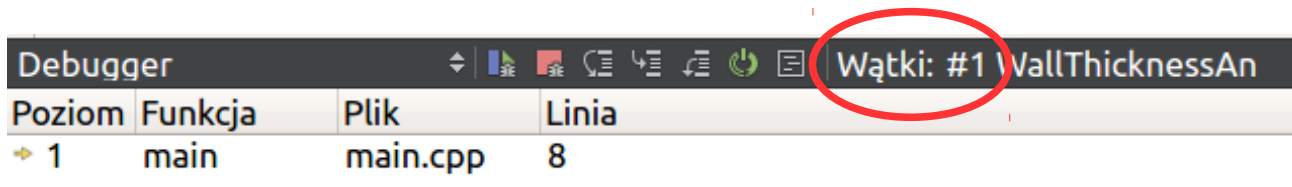
```
~thread()
{
    if (joinable())
        std::terminate();
}
```

Dygresja: debugging

- gdb:

```
(gdb) info threads
Id      Target Id      Frame
* 1      Thread 0x7ffff7fb3740 (LWP 12369) "a.out" main () at p1a_join.cpp:20
 3      Thread 0x7ffff674d700 (LWP 12371) "a.out" 0x00007ffff7632c1d in nanosleep ()
    at ../sysdeps/unix/syscall-template.S:84
 4      Thread 0x7ffff5f4c700 (LWP 12372) "a.out" 0x00007ffff7632c1d in nanosleep ()
    at ../sysdeps/unix/syscall-template.S:84
```

- QtCreator:



- Valgind + helgrind

```
Possible data race during read of size 4 at 0x601038 by thread #1
Locks held: none
  at 0x400606: main (simple_race.c:13)
```

Zamiast funkcji może być lambda

```
#include <iostream>
#include <thread>

int main()
{
    std::thread th([]()
    {
        std::cout << u8"Witaj, świecie!\n";
    });
    std::cout << "Hello, world!\n";
    th.join();
}
```

Grupa wątków

```
#include <iostream>
#include <thread>
#include <vector>

const int N = 10;

int main()
{
    std::vector<std::thread> workers;
    for (int i = 0; i < N; i++)
    {
        workers.push_back(std::thread ([]{
            std::cout << "Witaj, świecie!\n";
        }));
    }
    std::cout << "Hello, world!\n";
    for (auto & w: workers)
        w.join();
}
```

- **std::thread**
kopiowany jest
z użyciem
move semantics
- std::thread
nie może
posiadać
prawdziwej
kopii!

Grupa wątków

```
#include <iostream>
#include <thread>
#include <vector>

const int N = 10;

int main()
{
    std::vector<std::thread> workers;
    for (int i = 0; i < N; i++)
    {
        workers.push_back(std::thread ([]{
            {
                std::cout << "Witaj, świecie!\n";
            }
        }));
    }
    std::cout << "Hello, world!\n";
    for (auto & w: workers)
        w.join();
}
```



```
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Hello, world!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
```

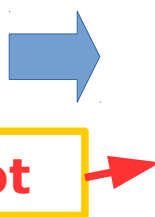
Grupa wątków

```
#include <iostream>
#include <thread>
#include <vector>

const int N = 10;

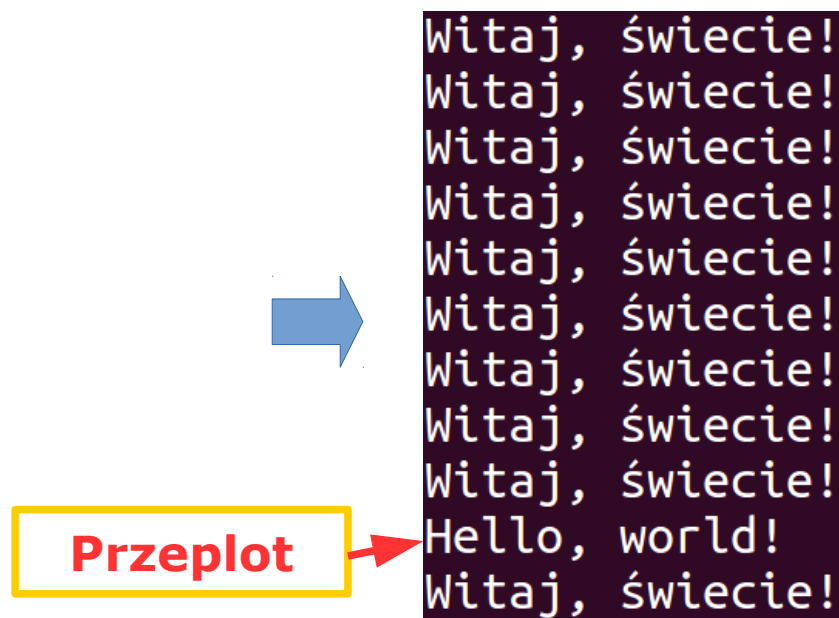
int main()
{
    std::vector<std::thread> workers;
    for (int i = 0; i < N; i++)
    {
        workers.push_back(std::thread ([i]()
        {
            std::cout << "Witaj, świecie!\n";
        })));
    }
    std::cout << "Hello, world!\n";
    for (auto & w: workers)
        w.join();
}
```

Przeplot



Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!
Hello, world!
Witaj, świecie!
Witaj, świecie!
Witaj, świecie!

Kolejne uruchomienie...



Kolejne uruchomienie...

- Niemal za każdym razem inny wynik...

```
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Hello, world!  
Witaj, świecie!
```

```
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Hello, world!  
Witaj, świecie!  
Witaj, świecie!
```

```
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Witaj, świecie!  
Hello, world!  
Witaj, świecie!  
Witaj, świecie!
```

- To jest bardzo niekorzystne zjawisko!

Skomplikujmy funkcję...

```
#include <iostream>
#include <thread>
#include <vector>

const int N = 10;

int main()
{
    std::vector<std::thread> workers;
    for (int i = 0; i < N; i++)
    {
        workers.push_back(std::thread ([i]()
        {
            std::cout << "Witaj, świecie nr " << i << "\n";
        }));
    }
    std::cout << "Hello, world!\n";
    for (auto & w: workers)
        w.join();
}
```

Wynik...

```
Witaj, świecie nr Witaj, świecie nr Witaj, świecie nr 6  
Witaj, świecie nr 7  
Witaj, świecie nr 4  
Witaj, świecie nr 5  
1  
Witaj, świecie nr 3  
Hello, world!  
Witaj, świecie nr 02  
Witaj, świecie nr 9  
Witaj, świecie nr 8
```

- Za każdym razem coś innego
- „Przeplot” jest nieznośny
- Program jest niedeterministyczny :-(

Wyścig (race condition)

- Wyścig to sytuacja, gdy wynik działania kilku procesów lub wątków zależy od względnej kolejności wykonywania poszczególnych operacji w każdym z nich
 - kilka wątków/procesów w tym samym czasie usiłuje modyfikować (i być może odczytywać) ten sam zasób, np. plik lub pamięć współdzieloną
 - Przykład: dwa niesynchronizowane programy/wątki wyświetlające dane na tej samej konsoli

Wyścig - przykład

Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

OK

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

wyścig

Wyścig = błąd

- Program, w którym zachodzi wyścig:
 - jest niedeterministyczny
 - jest niezwykle trudny do testowania
 - jest **błądny**
- Diagnostyka wyścigu jest bardzo trudna, dlatego lepiej zapobiegać niż leczyć
- Zapobieganie polega na stosowaniu określonych reguł/idiomów programowania (język nie daje żadnych gwarancji)

Diagnostyka wyścigu

np. Valgrind

Dołącz kod źródłowy

```
4 void my_fun_pl()
5 {
6   std::cout << u8"Witaj, świecie!\n";
7 }
8
9 int main()
10 {
11   std::thread th(&my_fun_pl);
12   std::cout << "Hello, world!\n";
13   th.join();
14 }
```

```
> g++ -pthread -g p1.cpp
> valgrind --tool=helgrind ./a.out
```

```
Hello, world!
==14608== -----
==14608==
==14608== Possible data race during write of size 8 at 0x604158 by thread #1
==14608== Locks held: none
==14608==   at 0x4F4EE25: std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert<
r> >&, char const*, long) (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21)
==14608==   by 0x4F4F236: std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::
const*) (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21)
==14608==   by 0x400F3F: main (p1.cpp:12)
==14608==
==14608== This conflicts with a previous write of size 8 by thread #2
==14608== Locks held: none
==14608==   at 0x4F4EE25: std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert<
r> >&, char const*, long) (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21)
==14608==   by 0x4F4F236: std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::
const*) (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21)
==14608==   by 0x400EFA: my_fun_pl() (p1.cpp:6)
==14608==   by 0x40249C: void std::Bind_simple<void (*())()>::M_invoke<>(std::Index_tuple<>) (fun
--14608==   by 0x4023E5: std::Bind_simple<void (*())()>::operator()() (functional:1520)
```

Programowanie współbieżne...

- **Jest trudne!** (jeśli program ma być *error-proof*)
- Gdyż trudno je *w pełni* zautomatyzować
- Żeby uniknąć nieoczywistych błędów, trzeba rozumieć (prawie) wszystkie zagadnienia języka na poziomie programów szeregowych
 - W C++: różne rodzaje funkcji, argumenty i wartości funkcji, referencje, semantyka *move*, konstruktor/destruktor, dziedziczenie, szablony, wyjątki, RAII, STL, iteratory, czas życia zmiennych, stos/sterta programu, funktory/wyrażenia lambda...