before async and parallel programing we have
- thread
- async delegares **(**begin invoke, end invoke**)**
- event based pattern **(**background worker**)**
- queue user work item (new)

task – an object denoting (marked-обозначенний) as outgoing operation being perform by machine
task exception 3 phases (saved, stored in task object, rethrown)
lambda expression is invoked as delegate
task.continue (action, another context)
closures – code + supporting data expression
closure variables become shared varaibles
local and global variables
code require thread to execute
resource manager – manage a pool of task and resource

race condition	условия гонки (переменних в параллизме)
when you perform speculative work	когда ви виполняете умозрительную работу
to overlap computation	преривать вичисления

task<>task.factory.startnew(()=>….return) because each access to X.result is an implicit call to wait
task composition – the completion of one task trigger the start to another
doinig t2=t1.continue with grabbing the result
exeption on task is caught, saved to task and re-thrown upon .wait,.result,.waitall
observe exception before garbage collector will running
subscribe taskshedule.un...taskexeption
task cancelation in cooperative model

jumbled output	перемешаний вывод

creator pass a cancelation token
task monitor token, if canceled perform cleanup and throw
async programing – responsiveness – hide latency of long runing I/O operation
parallel programing – performance – reduce time of CPU-bound computation
by dividing workload and execution simultaniously
PLINQ **пи-линк**