std::condition_variable

wait:-
* check the condition using supplied callback
* if cond is true - continue, mutex still is locked
* if cond is false -- about to block, but releases mutex

on notify:-
* cond is typically true, locks the mutex and go ahead

consumer:-

```
std::unique_lock<std::mutex> ulck(m1);
cv.wait(ulock, [] () { return flag; } );
//actual code
```

prod:-

```
std::unique_lock<std::mutex> ulck(m1);
//actual code
flag=true
cv.notify
```

Similar pthread APIs:-
* pthread_cond_wait
* pthread_cond_signal

```
std::atomic<int> counter(10);
std::atomic<int> counter=10;                    //error
counter=15;    //noyt thru constructor
-----------------------------------------------
parallelism:-

std::async


int result;
void sumarr(int *arr, int len) {
  int sum=0,i;
  for(i=0;i<len;i++)
        sum+=arr[i];
  result=sum;
}

int arr[10];
//fill with random values
std::thread t1(sumarr, arr, 10);
t1.join();
//print the result
```

Self Study/Additional:-
* get_id from std::thread
* Detachable Threads
* std::launch::async vs std::launch::deferred

std::future
---
std::future & std::promise

Scenario:-1

```
std::promise<int> p1;     //global or common
```

T1:-
```
p1.set_value(10);
```

T2/main:-
```
std::future<int> res = p1.get_future();
res.get();
```

Scenario-2:-

```
std::promise<void> barrier;
```

T1:-

```
barrier.set_value();
```

T2:-

```
std::future<void> barrier_future = barrier.get_future();
barrier_future.wait();
```