- Example

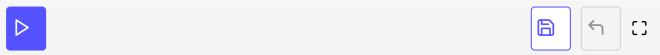
This example demonstrates the usage of std::packaged_task in multithreaded programs.

```
we'll cover the following ^
• Example
• Explanation
```

Example

```
// packagedTask.cpp
#include <utility>
#include <future>
#include <iostream>
#include <thread>
#include <deque>
class SumUp{
  public:
    int operator()(int beg, int end){
      long long int sum{0};
      for (int i = beg; i < end; ++i ) sum += i;</pre>
      return sum;
};
int main(){
  std::cout << std::endl;</pre>
  SumUp sumUp1;
  SumUp sumUp2;
  SumUp sumUp3;
  SumUp sumUp4;
  // wrap the tasks
  std::packaged_task<int(int, int)> sumTask1(sumUp1);
  std::packaged_task<int(int, int)> sumTask2(sumUp2);
  std::packaged_task<int(int, int)> sumTask3(sumUp3);
  std::packaged_task<int(int, int)> sumTask4(sumUp4);
  // create the futures
  std::future<int> sumResult1 = sumTask1.get_future();
  //std:.future<int> sumResult2 = sumTask2 get future():
```

```
auto sumResult2 = sumTask2.get_future();
std::future<int> sumResult3 = sumTask3.get_future();
//std::future<int> sumResult4 = sumTask4.get_future();
auto sumResult4 = sumTask4.get_future();
// push the tasks on the container
std::deque<std::packaged_task<int(int,int)>> allTasks;
allTasks.push_back(std::move(sumTask1));
allTasks.push_back(std::move(sumTask2));
allTasks.push_back(std::move(sumTask3));
allTasks.push_back(std::move(sumTask4));
int begin{1};
int increment{2500};
int end = begin + increment;
// execute each task in a separate thread
while (not allTasks.empty()){
  std::packaged_task<int(int, int)> myTask = std::move(allTasks.front());
  allTasks.pop_front();
  std::thread sumThread(std::move(myTask), begin, end);
  begin = end;
  end += increment;
  sumThread.detach();
// get the results
auto sum = sumResult1.get() + sumResult2.get() +
           sumResult3.get() + sumResult4.get();
std::cout << "sum of 0 .. 10000 = " << sum << std::endl;
std::cout << std::endl;</pre>
```



Explanation

The purpose of the program is to calculate the sum of all numbers from 0 to 10000 with the help of four std::packaged_task, each running in a separate thread. The associated futures are used to sum up the final result. Of course, you can also use the Gaußschen Summenformel.

I. Wrap the tasks: We pack the work packages in std::packaged_task (lines 28 - 31) objects. Work packages are instances of the class SumUp (lines 9 - 16). The work is done in the call operator (lines 11 - 15) which sums up all numbers from beg to end - 1 and returns the sum as a result. std::packaged_task (lines 28 - 31) can handle the callables that need two int s and return an int: int(int, int).

II. **Create the futures**: We must create **future** objects with the help of **std::packaged_task** objects (lines 34 to 39). The **packaged_task** is the promise in the communication channel. The type of the **future** is defined explicitly: **std::future<int> sumResult1= sumTask1.get_future()**. The compiler can complete the task for us: **auto sumResult1= sumTask1.get_future()**.

III. **Perform the calculations**: Now the calculation occurs. The <code>packaged_task</code> are moved onto the <code>std::deque</code> (lines 43 - 46). In the while loop, each <code>packaged_task</code> (lines 54 - 59) is executed. To complete the task, we move the head of the <code>std::deque</code> in a <code>std::packaged_task</code> (line 54). We then move the <code>packaged_task</code> in a new thread (line 56) and let it run in the background (line 59). We used the move semantic in lines 54 and 56 since <code>std::packaged_task</code> objects are not copyable. This restriction holds for all promises, futures, and threads. There is one exception to this rule: <code>std::shared_future</code>.

IV. **Pick up the results**: In the final step, we ask all futures for their value and sum them up (line 63-64).

Test your knowledge on this topic with an exercise in the next lesson.