3. Основы технологии OpenMP.

Модель «fork-join».

Классификация переменных.

Основные директивы и их опции.

Распараллеливание по данным и по операциям.

Решение проблемы синхронизации.

Технология разработки параллельных программ для МВС с общей памятью (OpenMP)

- **OpenMP** (Open Multi-Processing) —
  - открытый развивающийся стандарт для распараллеливания программ на языках C, C++, Fortran,
  - включает описание
    - директив компилятора,
    - библиотечных процедур,
    - переменных ОС,

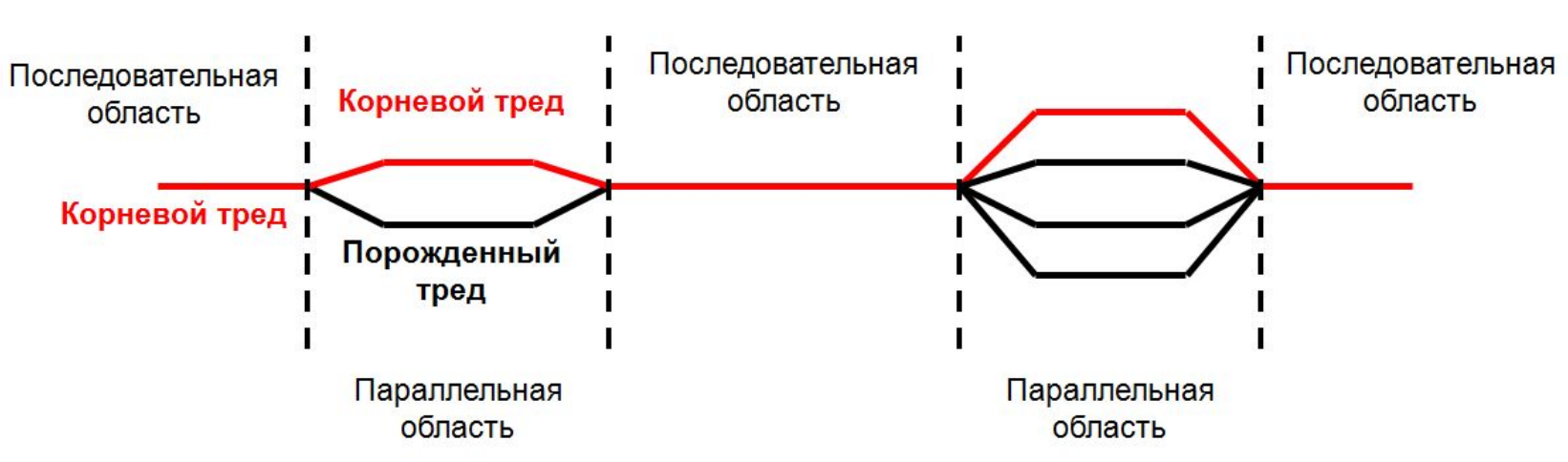
  для программирования многопоточных приложений для МВС с общей памятью (имеется версия и для кластеров).
- Наиболее популярная задача OpenMP — написание программ, ориентированных на циклы

Модель программирования OpenMP

Разветвление-объединение (***fork-join***)

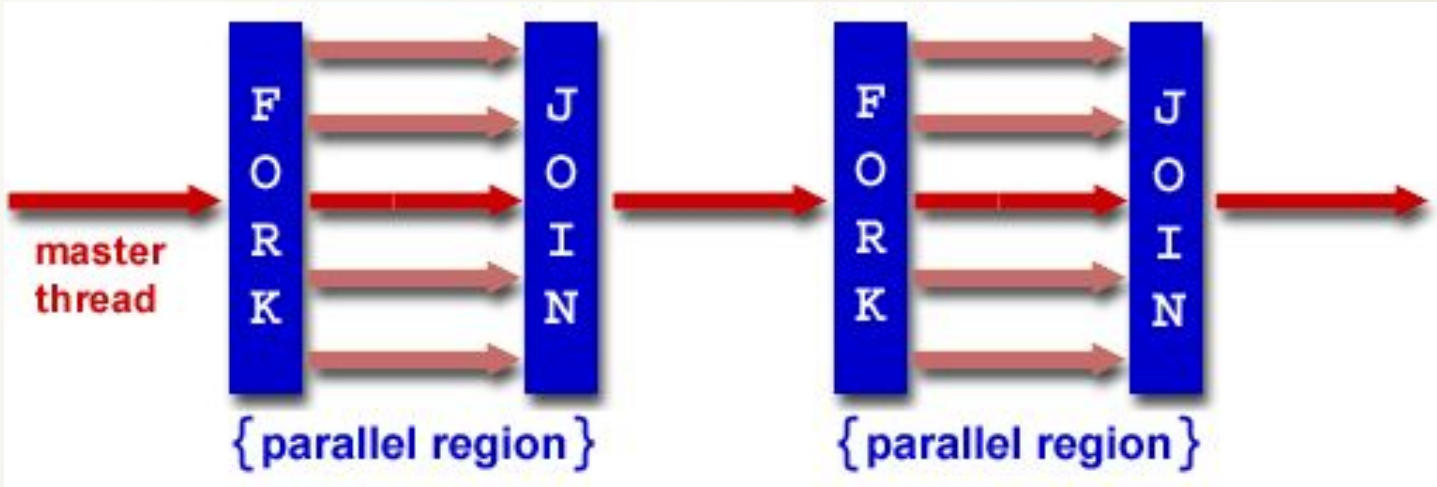
- Работа программы начинается с **одного** (корневого) **потока**, или **нити, треда** (**thread** – нить).
- Для добавления в программу параллелизма выполняется **разветвление** (***fork***) на несколько тредов, создается **группа**.
- Треды группы выполняются параллельно в рамках фрагмента кода, т.наз. **параллельной области**.
- В конце параллельной области все треды **синхронизируются** и заканчивают свою работу (***join***), оставив корневой .
- После этого корневой тред продолжает выполняться до тех пор, пока не начнется следующая **параллельная** область (или не наступит **конец** программы).

Модель программирования OpenMP



- Программа OpenMP начинается как единственный начальный **корневой тред** выполнения (номер =0).
- Встретив в коде параллельную конструкцию, он создает новую **группу тредов (порожденных)**.
- После параллельной конструкции выполнение кода продолжает только **корневой тред**.

Из OpenMp Tutorial
computing.llnl.gov/tutorials/openMP/



Терминология

- **Поток** (выполнения),
- или **нить**,
- или **тред** (*thread of the execution*) –

  последовательность выполняющихся команд.

- **Программа**,
- или **приложение**,
- или **процесс** –
  может состоять из **нескольких потоков**

Не путать с **потоками данных** (*stream*).

Синтаксис и семантика OpenMP

- **Директива компилятора** – указание компилятору на особенности обработки исходного кода при компиляции.
- Языковые конструкции в OpenMP определены как **директивы компилятора**, сообщающие компилятору, **что сделать для реализации параллелизма**.
- В C и C++ такие директивы называются **прагмы** (***pragmatic information* - полезная информация, ЯП Ada**).

- **Синтаксис** прагмы

#pragma omp *имя* [*опции*]

- **Пример** – указание на распараллеливание операторов блока:

```
#pragma omp parallel
{…   }
```

Синтаксис и семантика OpenMP

- **Библиотечные функции** – позволяют:
  - получать сведения (**get**) о параметрах тредов в программе,
  - изменять их значения (**set**),
  - выполнять блокировку выполнения для синхронизации тредов.
- **Синтаксис**:
  omp_имя()
- **Пример** – определение номера треда (по месту вызова функции):

```
i = omp_get_thread_num();
```

Синтаксис и семантика OpenMP

- **Переменные окружения** – переменные операционной системы
  - хранят данные ее настройках,
  - позволяют **управлять** поведением программы (в нашем случае - **параллелизмом**) во время ее выполнения (*run-time*).
- **Синтаксис**:
  OMP_имя
- **Пример** – переменная, задающая максимально возможное количество тредов:

  OMP_NUM_THREADS

Значение **OMP_NUM_THREADS** изменяет функция

```
void omp_set_num_threads (int n);
```

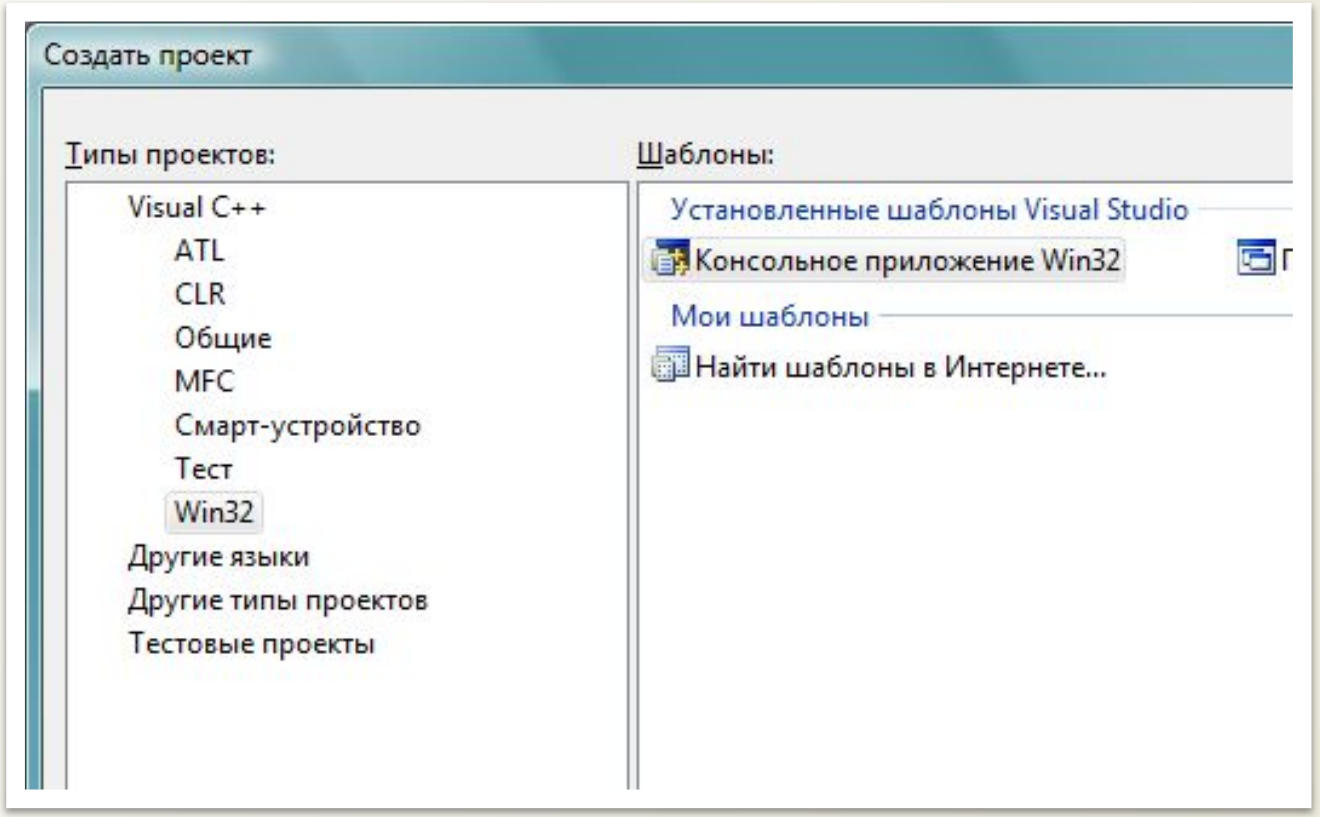
- !!!**Функции** вида
  set (назначения параметров)
  имеют приоритет над соответствующими переменными окружения

Структура программы OpenMP

```
int main ()
{
   //последовательная область, выполняется корневой тред
   . . .
   //Начало параллельной области
   #pragma omp parallel …
   {
   //операторы выполняются всеми тредами
   . . .
   //все треды завершают работу, остается только корневой тред
   }
   //последовательная область, выполняется корневой тред
   . . .
}
```

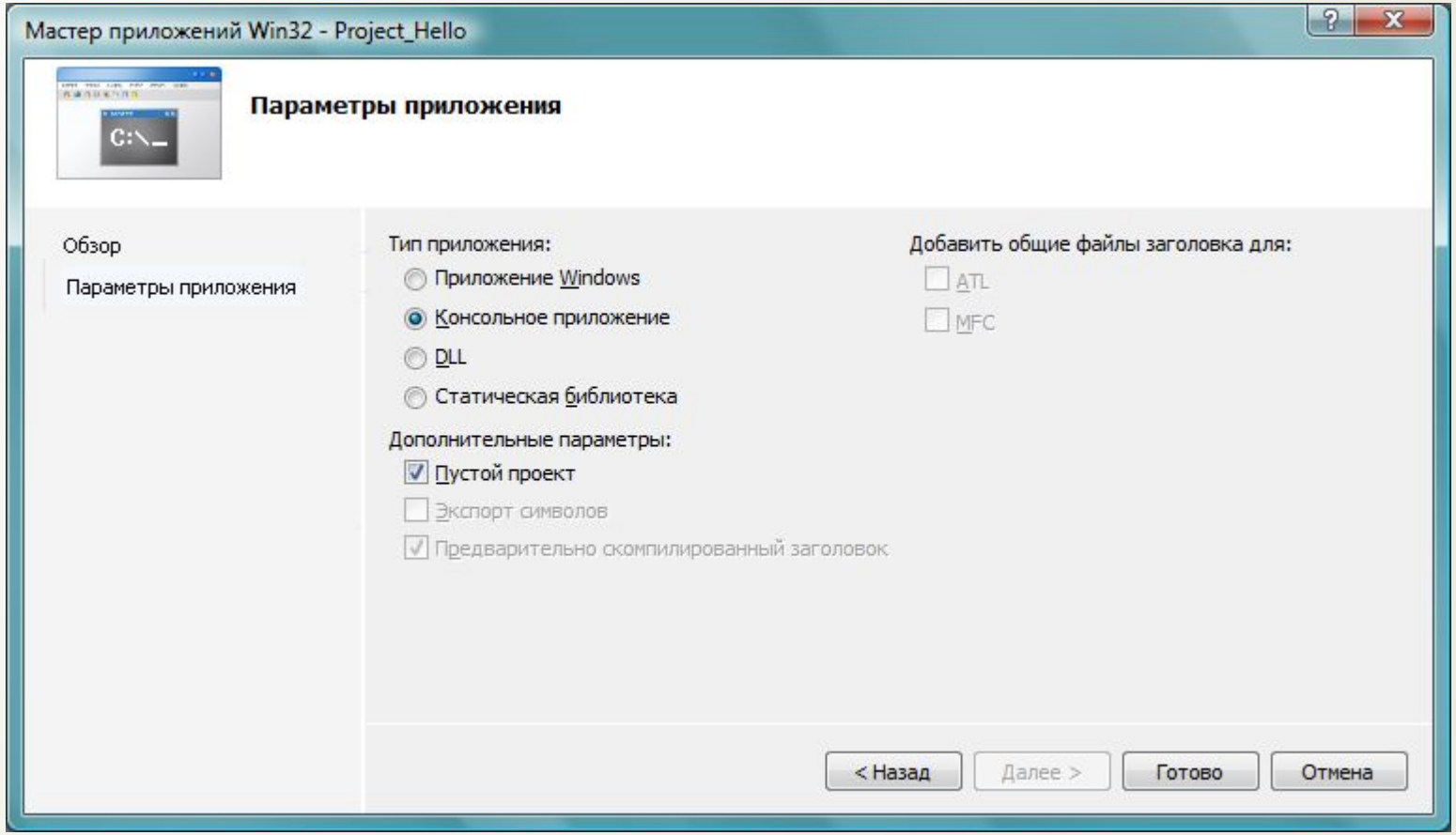
Простейшая программа. Настройки для работы с OpenMP.

Создать новый проект по шаблону «Консольное приложение Win32»



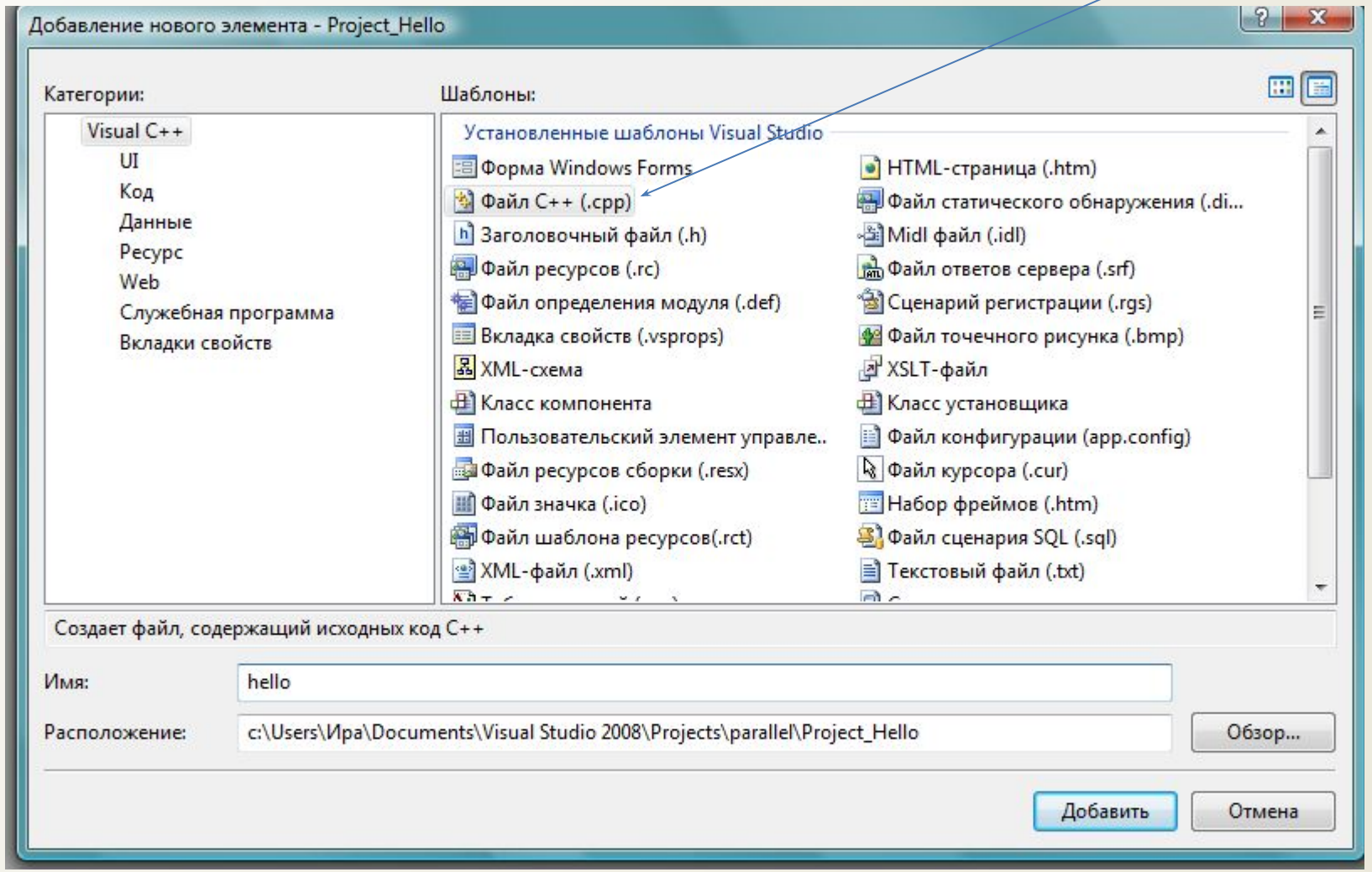
Простейшая программа. Настройки для работы с OpenMP.

В Мастере выбрать «Пустой проект»

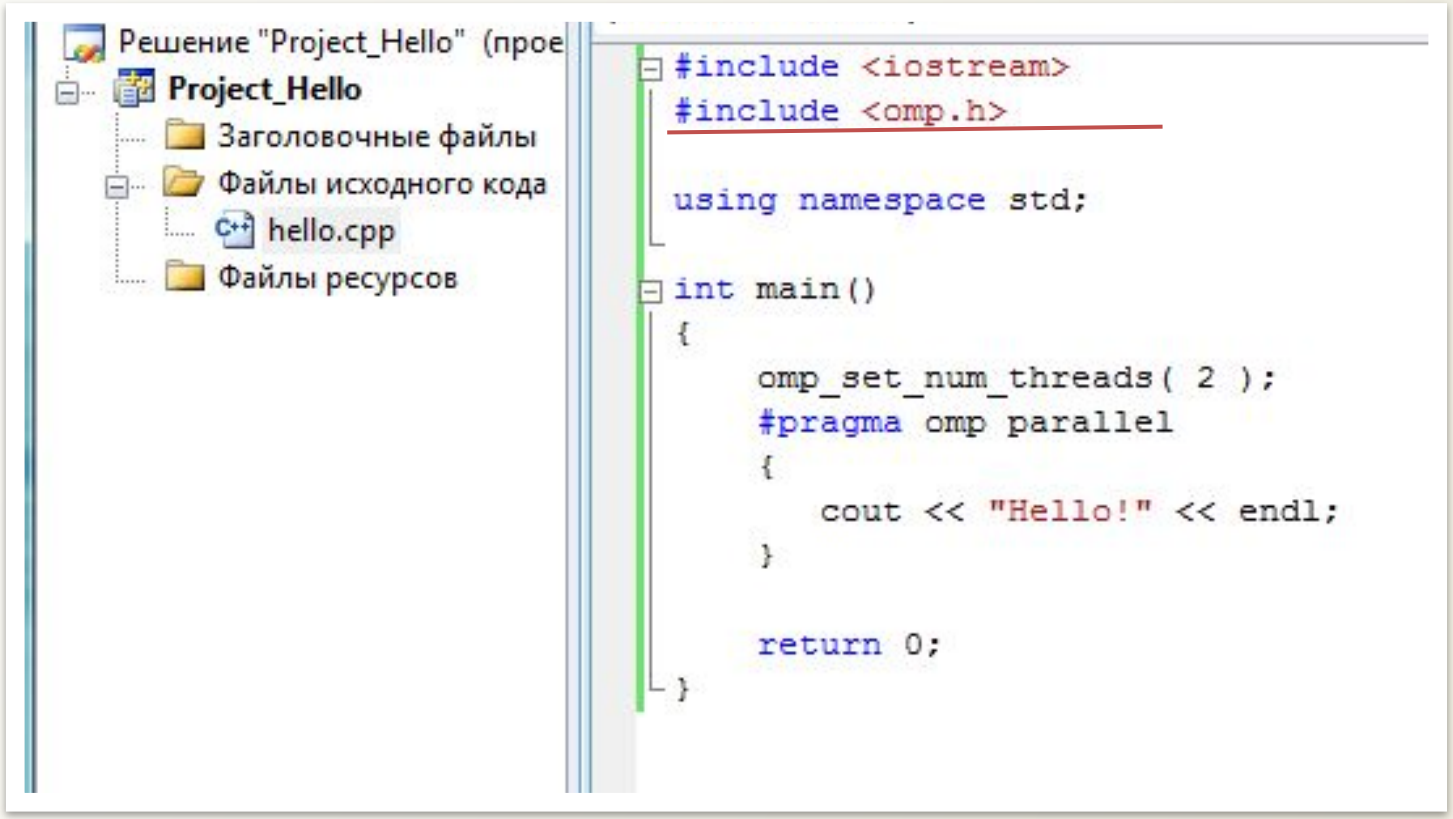


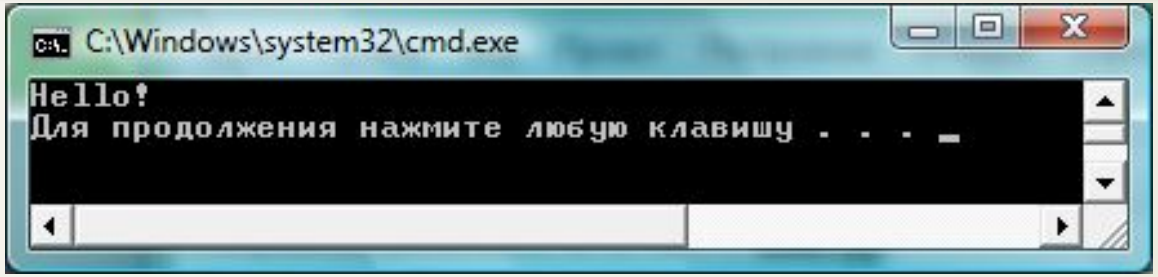
Простейшая программа. Настройки для работы с OpenMP.

Выбрать Проект – Добавить новый элемент

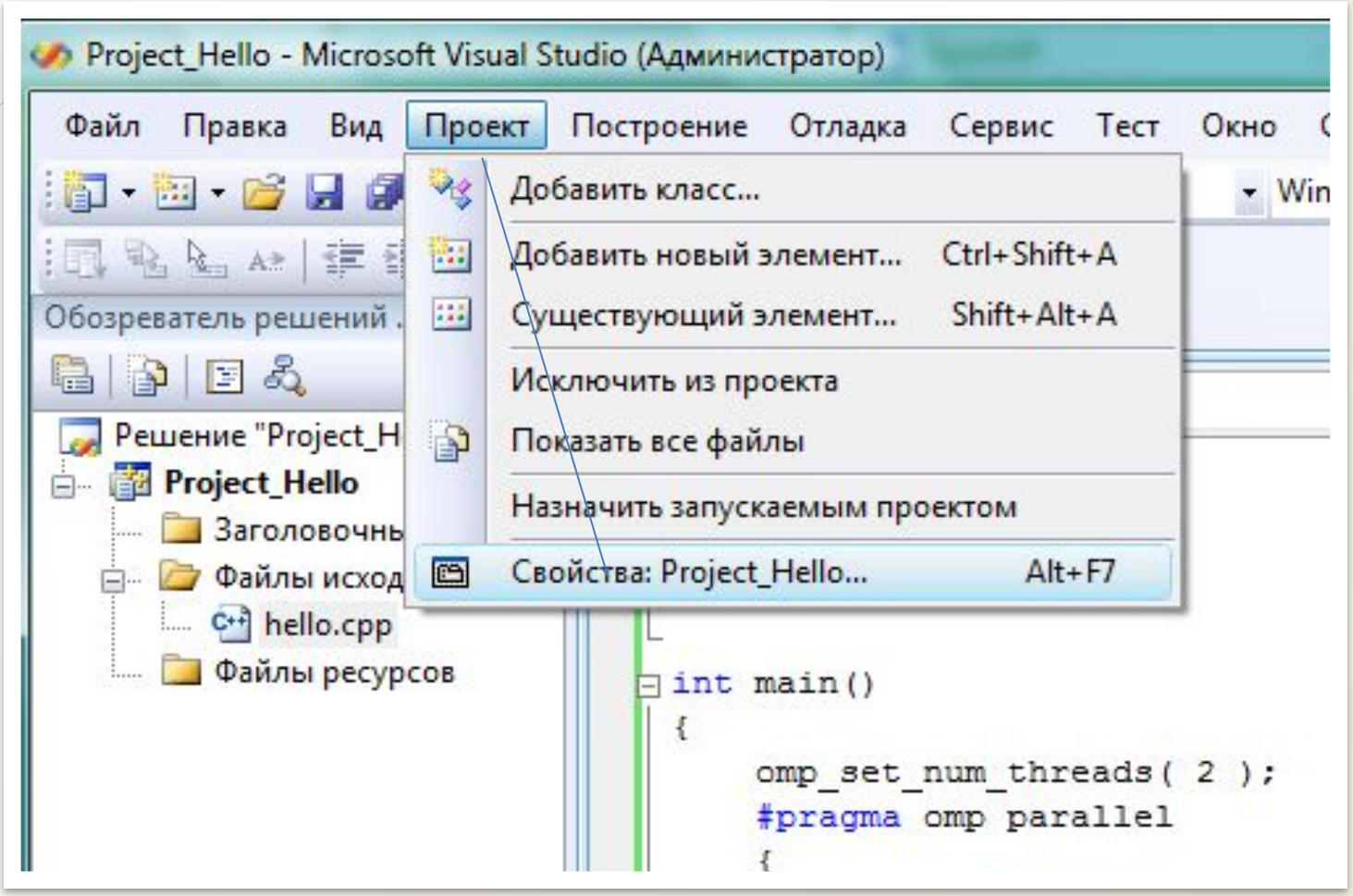


Текст программы:

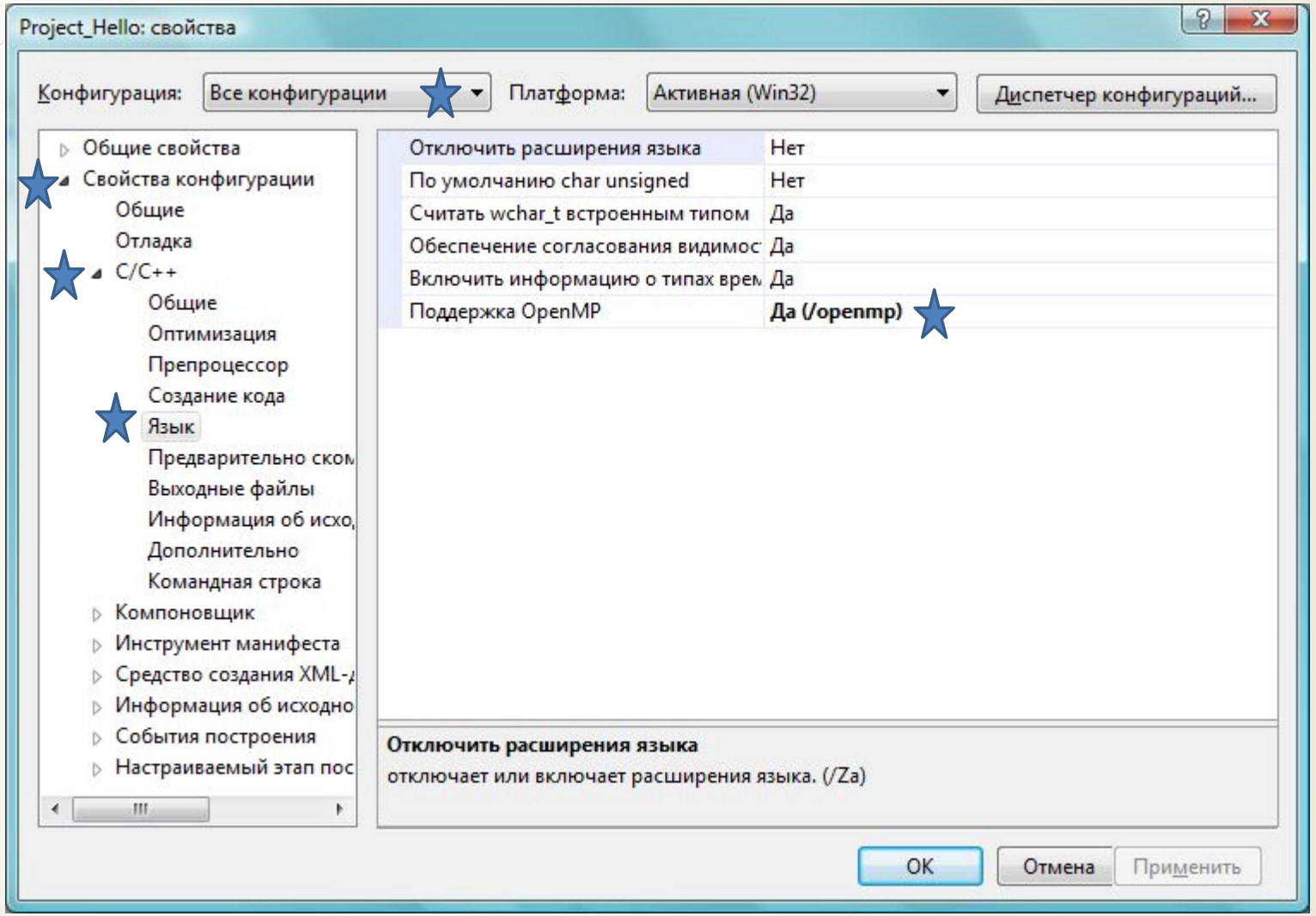




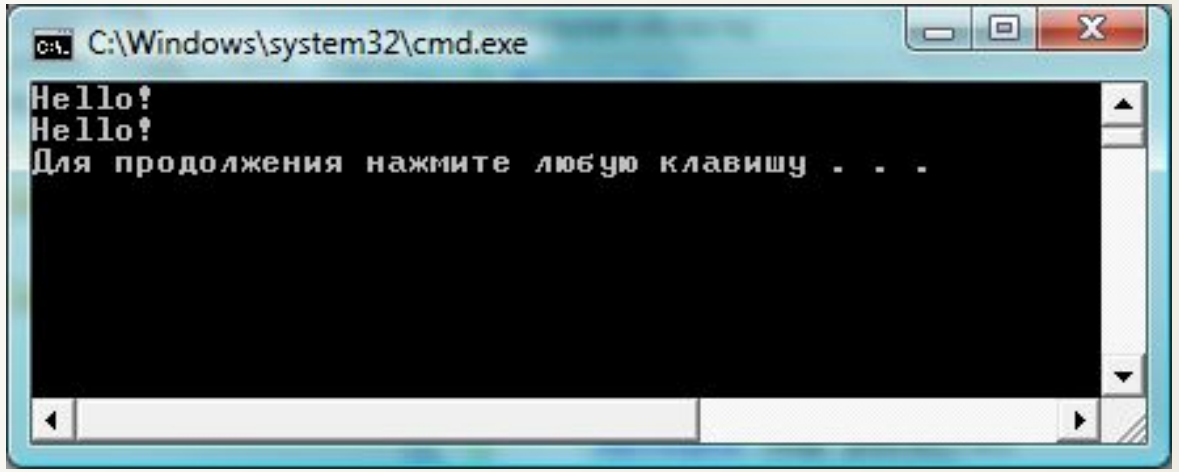
Дальнейшая настройка



Дальнейшая настройка



Новый результат:



Задание:

1) измените параметр, например

```
omp_set_num_threads( 10 );
```

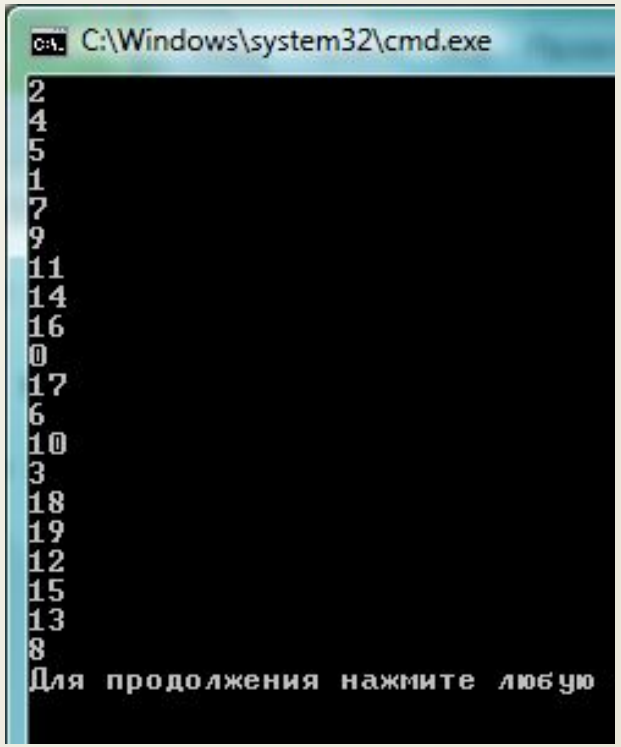
2) Закомментируйте эту строку.

Какие будут результаты?

Какого числа тредов удалось добиться?

Пример: вывод номера треда

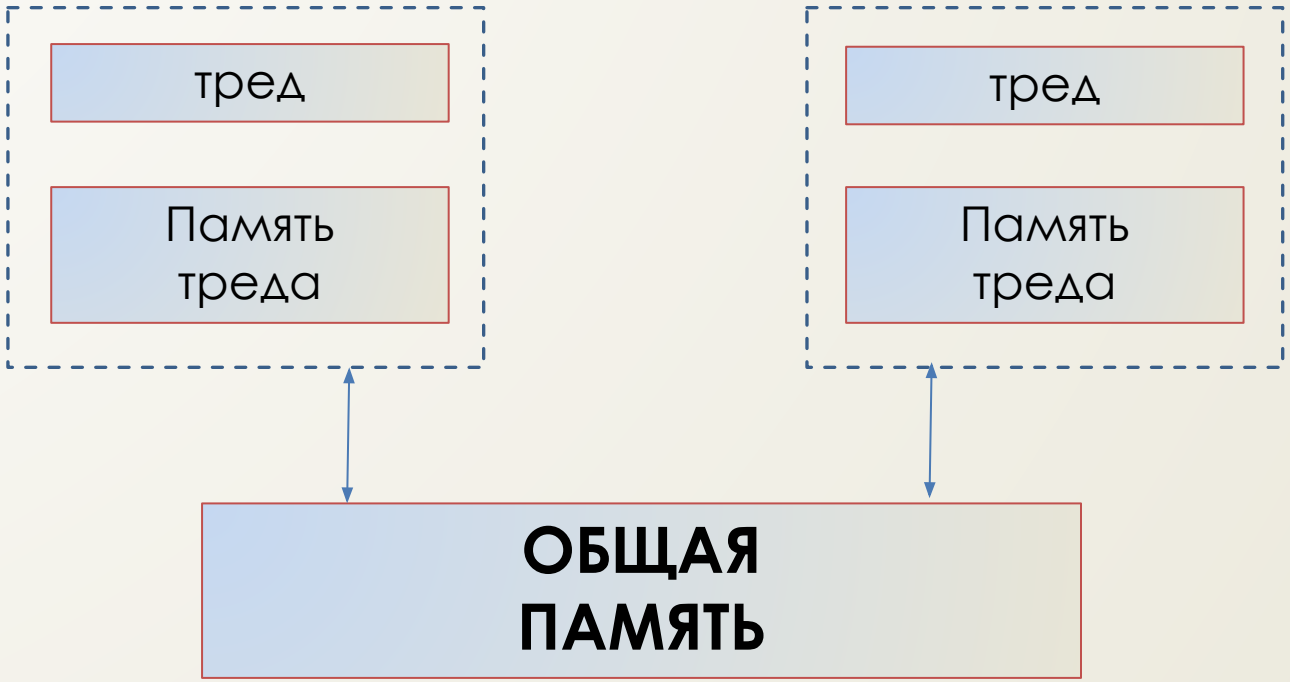

```
#include <iostream>
#include <omp.h>

using namespace std;

int main()
{
    omp_set_num_threads( 20 );
    #pragma omp parallel
    {
        cout << omp_get_thread_num() << endl;
    }

    return 0;
}
```



Переменные. Локализация (область видимости)

Модель данных в OpenMP:

- **общая** область памяти - для **всех** тредов,
- **локальные** области памяти - для **каждого** треда.



Переменные. Локализация (область видимости)

- Переменные, использующиеся в параллельных областях программы, разделяются на **два основных класса**:
- **shared – общие**
  - все треды видят одну и ту же переменную;
- **private - локальные**
  - каждый тред видит свой экземпляр данной переменной,
  - создаются для каждого треда только на время его выполнения.

Переменные. Локализация (область видимости)

Правила видимости переменных:

- все переменные, определенные **вне** параллельной области – **общие**;
- все переменные, определенные **внутри** параллельной области – **локальные**

Пример: локализация

localization.cpp

(Глобальная область) main()

```
#include <iostream>
#include <omp.h>

using namespace std;

int main()
{
    int n = 1;
    #pragma omp parallel
    {
        int k = omp_get_thread_num();
        cout << "in thread #" << k;
        cout << " n = " << n  << endl;
    }

    return 0;
}
```

глобальная

локальная

C:\Windows\system32\cmd.exe

```
in thread #0 n = 1
in thread #1 n = 1
Для продолжения нажмите любую клавишу . . .
```

Директивы OpenMP - parallel

Основная директива для создания параллельной области

```
int main ()
{
   //последовательная область, выполняется корневой тред
   . . .
   //Начало параллельной области
   #pragma omp parallel [опции]
   {
   //операторы выполняются всеми тредами
   . . .
   //все треды завершают работу, остается только корневой тред
   }
   //последовательная область, выполняется корневой тред
   . . .
}
```

Синтаксис директивы **parallel**

```
#pragma omp parallel [опции ...] newline
{


}
if (scalar_expression)
num_threads (integer_expression)
private (list)
firstprivate (list)
shared (list)
default (shared | none)
reduction (operator: list)
copyin (list)
```

Опция if

`if (scalar_expression)`

– распараллеливание по **условию**.

Если значение выражения ≠ 0,

то осуществляется **распараллеливание**.

Иначе операторы

параллельной области выполняются

единственным корневым тредом.

Пример


```
#include <iostream>
#include <omp.h>
using namespace std;
int main()
{
    int n;
    cout << "one thread" << endl;
    cout << "input number of threads: ";
    cin >> n;
    omp_set_num_threads(n);
    #pragma omp parallel if( n>1 )
     {
     int k = omp_get_thread_num();
     cout << "in thread #" << k << endl;
     }
    cout << "one thread" << endl;
    return 0;
}
```