| #include <stdio.h>  #include <omp.h>    int main() {  #pragma omp parallel  {  int id = omp\_get\_thread\_num();  printf("Hello from thread %d\n", id);  }  return 0;  } |
| --- |

| #include <stdio.h>  #include <omp.h>    int main() {  int n = 10;  int a[n];    #pragma omp parallel for  for (int i = 0; i < n; i++) {  a[i] = i \* i; // Calculate square of i  printf("Thread %d calculated a[%d] = %d\n", omp\_get\_thread\_num(), i, a[i]);  }  return 0;  } |
| --- |

| #include <stdio.h>  #include <omp.h>    int main() {  #pragma omp parallel sections  {  #pragma omp section  printf("Thread %d: Task 1\n", omp\_get\_thread\_num());    #pragma omp section  printf("Thread %d: Task 2\n", omp\_get\_thread\_num());  }  return 0;  } |
| --- |

| #include <stdio.h>  #include <omp.h>    int main() {  int sum = 0;    #pragma omp parallel for reduction(+:sum)  for (int i = 1; i <= 10; i++) {  sum += i;  }  printf("Sum = %d\n", sum); // Expected output: 55  return 0;  } |
| --- |

| #include <pthread.h>  #include <stdio.h>    void\* print\_message(void\* arg) {  printf("Hello from thread!\n");  return NULL;  }    int main() {  pthread\_t thread;  pthread\_create(&thread, NULL, print\_message, NULL);  pthread\_join(thread, NULL);  return 0;  } |
| --- |

| #include <iostream>  #include <thread>    void print\_message() {  std::cout << "Hello from thread!" << std::endl;  }    int main() {  std::thread t(print\_message);  t.join();  return 0;  } |
| --- |

| class MyThread extends Thread {  public void run() {  System.out.println("Hello from thread!");  }  }    public class Main {  public static void main(String[] args) {  MyThread t = new MyThread();  t.start();  }  } |
| --- |

| for (int i = 0; i < N; i += 4) {  output[i] = input[i] \* factor;  output[i + 1] = input[i + 1] \* factor;  output[i + 2] = input[i + 2] \* factor;  output[i + 3] = input[i + 3] \* factor;  } |
| --- |