Kod programu

```
#include<stdlib.h>
#include<stdio.h>
#include<omp.h>
void zadanie3_4() {
   int a_shared = 1;
   int b_private = 2;
   int c_firstprivate = 3;
   int e_atomic=5;
   printf("przed wejsciem do obszaru rownoleglego -  nr_threads %d, thread ID %d\n", omp_get_num_threads(), omp_get_thread_num());
   printf("\ta_shared \t= %d\n", a_shared);
   printf("\tb_private \t= %d\n", b_private);
   printf("\tc_firstprivate \t= %d\n", c_firstprivate);
   printf("\te_atomic \t= %d\n", e_atomic);
   #pragma omp parallel default(none) shared(a_shared, e_atomic) private(b_private) firstprivate(c_firstprivate )
     {
      int i;
      int d_local_private;
      d_local_private = a_shared + c_firstprivate;
               #pragma omp barrier
      for(i=0;i<10;i++) {
                               #pragma omp atomic
                               a_shared ++;
               }
      for(i=0; i<10; i++) c_firstprivate += omp_get_thread_num();
      for(i=0;i<10;i++) {
         #pragma omp atomic
         e_atomic+=omp_get_thread_num();
      }
               #pragma omp barrier
      #pragma omp critical
      {
        printf("\nw obszarze równoległym: aktualna liczba watkow %d, moj ID %d\n", omp_get_num_threads(), omp_get_thread_num());
        printf("\ta_shared \t= %d\n", a_shared);
        printf("\tb_private \t= %d\n", b_private);
        printf("\tc_firstprivate \t= %d\n", c_firstprivate);
        printf("\td_local_private = %d\n", d_local_private);
        printf("\te_atomic \t= %d\n", e_atomic);
      }
     }
     printf("po zakonczeniu obszaru rownoleglego:\n");
     printf("\ta_shared \t= %d\n", a_shared);
     printf("\tb_private \t= %d\n", b_private);
     printf("\tc_firstprivate \t= %d\n", c_firstprivate);
     printf("\te_atomic \t= %d\n", e_atomic);
}
void zadanie5() {
   // Ustawiam zmienną środowiskową przed uruchomieniem programu.
   // $ set OMP_NUM_THREADS = 33
   // lub
   // $ export OPM_NUM_THREADS = 33
   zadanie3_4();
}
void zadanie8() {
   omp_set_num_threads(33);
   zadanie3_4();
}
void zadanie9() {
   int a_shared = 1;
   int b_private = 2;
   int c_firstprivate = 3;
   int e_atomic=5;
   printf("przed wejsciem do obszaru rownoleglego -  nr_threads %d, thread ID %d\n", omp_get_num_threads(), omp_get_thread_num());
   printf("\ta_shared \t= %d\n", a_shared);
   printf("\tb_private \t= %d\n", b_private);
   printf("\tc_firstprivate \t= %d\n", c_firstprivate);
   printf("\te_atomic \t= %d\n", e_atomic);
   #pragma omp parallel num_threads(7) default(none) shared(a_shared, e_atomic) private(b_private) firstprivate(c_firstprivate )
     {
      int i;
      int d_local_private;

      d_local_private = a_shared + c_firstprivate;
               #pragma omp barrier
      for(i=0;i<10;i++) {
                               #pragma omp atomic
                               a_shared ++;
               }
      for(i=0; i<10; i++) c_firstprivate += omp_get_thread_num();
      for(i=0;i<10;i++) {
         #pragma omp atomic
         e_atomic+=omp_get_thread_num();
      }
               #pragma omp barrier
      #pragma omp critical
      {
        printf("\nw obszarze równoległym: aktualna liczba watkow %d, moj ID %d\n", omp_get_num_threads(), omp_get_thread_num());
        printf("\ta_shared \t= %d\n", a_shared);
```

```
            printf("\tb_private \t= %d\n", b_private);
            printf("\tc_firstprivate \t= %d\n", c_firstprivate);
            printf("\td_local_private = %d\n", d_local_private);
            printf("\te_atomic \t= %d\n", e_atomic);
          }
        }
        printf("po zakonczeniu obszaru rownoleglego:\n");
        printf("\ta_shared \t= %d\n", a_shared);
        printf("\tb_private \t= %d\n", b_private);
        printf("\tc_firstprivate \t= %d\n", c_firstprivate);
        printf("\te_atomic \t= %d\n", e_atomic);
}
void zadanie10() {
    static int i = -1;
    #pragma omp threadprivate(i)
    #pragma omp parallel num_threads(7)
    {
        i = omp_get_thread_num();
    }
    #pragma omp parallel num_threads(7)
    {
        printf("%d\n",i);
    }
}
void zadanie12_13_a() {
    printf("schedule(static,3)\n");
    omp_set_num_threads(4);
    int i;
    #pragma omp parallel for ordered schedule(static, 3)
    for(i = 0; i < 17; i++)
    {
        #pragma omp ordered
        printf("iteracja %d: wątek %d\n",i,omp_get_thread_num());
    }
    printf("\n");
}
void zadanie12_13_b() {
    printf("schedule(static)\n");
    omp_set_num_threads(4);
    int i;
    #pragma omp parallel for ordered schedule(static)
    for(i = 0; i < 17; i++)
    {
        #pragma omp ordered
        printf("iteracja %d: wątek %d\n",i,omp_get_thread_num());
    }
    printf("\n");
}
void zadanie12_13_c() {
    printf("schedule(dynamic,3)\n");
    omp_set_num_threads(4);
    int i;
    #pragma omp parallel for ordered schedule(dynamic, 3)
    for(i = 0; i < 17; i++)
    {
        #pragma omp ordered
        printf("iteracja %d: wątek %d\n",i,omp_get_thread_num());
    }
    printf("\n");
}
void zadanie12_13_d() {
    printf("schedule(dynamic)\n");
    omp_set_num_threads(4);
    int i;
    #pragma omp parallel for ordered schedule(dynamic)
    for(i = 0; i < 17; i++)
    {
        #pragma omp ordered
        printf("iteracja %d: wątek %d\n",i,omp_get_thread_num());
    }
    printf("\n");
}
void histogram() {
                    int n = 100;
                    int m = 150;
                    int i, j;
                    int** obraz = malloc(sizeof(int*) * n);
                    for(i = 0; i < m; i++) {
                        obraz[i] = malloc(sizeof(int) * m);
                    }
                    int *histogram = malloc(sizeof(int )*94);
                    for(i = 0; i < 94; i++) {
                        histogram[i] = 0;
                    }
                    for(i = 0; i < n; i++) {
                        for(j = 0; j < m; j++) {
                                            obraz[i][j] = rand() % 94 + 33;
                                            printf("%c",obraz[i][j]);
                                        }
                    printf("\n");
                }
                omp_set_num_threads(8);
                #pragma omp parallel for ordered schedule(guided)
                for(i = 0; i < 94; i++) {
                                int p,k;
                                for(p = 0; p < n; p++) {
                                                for(k = 0; k < m; k++) {
                                                                if(obraz[p][k] == i+33) {
                                                                                histogram[i]++;
                                                                }
                                                }
                                }
                                #pragma omp ordered
                                printf("%c == %d - liczyl watek %d\n",i+33,histogram[i],omp_get_thread_num());
                }
}
int main(){
#ifdef   _OPENMP
  printf("\nKompilator rozpoznaje dyrektywy OpenMP\n");
#endif
//zadanie3_4();
//zadanie5();
//zadanie8();
//zadanie9();
//zadanie10();
zadanie12_13_a();
zadanie12_13_b();
zadanie12_13_c();
zadanie12_13_d();
//histogram();
}
```

schedule(static,3)	**schedule(static)**	**schedule(dynamic,3)**	**schedule(dynamic)**
it 0: wątek 0 it 1: wątek 0 it 2: wątek 0 it 3: wątek 1 it 4: wątek 1 it 5: wątek 1 it 6: wątek 2 it 7: wątek 2 it 8: wątek 2 it 9: wątek 3 it 10: wątek 3 it 11: wątek 3 it 12: wątek 0 it 13: wątek 0 it 14: wątek 0 it 15: wątek 1 it 16: wątek 1	it 0: wątek 0 it 1: wątek 0 it 2: wątek 0 it 3: wątek 0 it 4: wątek 0 it 5: wątek 1 it 6: wątek 1 it 7: wątek 1 it 8: wątek 1 it 9: wątek 2 it 10: wątek 2 it 11: wątek 2 it 12: wątek 2 it 13: wątek 3 it 14: wątek 3 it 15: wątek 3 it 16: wątek 3	it 0: wątek 0 it 1: wątek 0 it 2: wątek 0 it 3: wątek 2 it 4: wątek 2 it 5: wątek 2 it 6: wątek 3 it 7: wątek 3 it 8: wątek 3 it 9: wątek 1 it 10: wątek 1 it 11: wątek 1 it 12: wątek 0 it 13: wątek 0 it 14: wątek 0 it 15: wątek 2 it 16: wątek 2	it 0: wątek 0 it 1: wątek 2 it 2: wątek 1 it 3: wątek 3 it 4: wątek 0 it 5: wątek 2 it 6: wątek 1 it 7: wątek 3 it 8: wątek 0 it 9: wątek 2 it 10: wątek 1 it 11: wątek 3 it 12: wątek 0 it 13: wątek 2 it 14: wątek 1 it 15: wątek 3 it 16: wątek 0