Pomoc do zadania 1:

Przed przystąpieniem do rozwiązania zadania zadeklarujemy odpowiednie zmienne i obiekty, m.in. takie, które będą odpowiadały za zawartość pliku z danymi. Można to zrobić w następujący sposób:

```
#include<fstream>

ifstream fin;
fin.open("liczby.txt");
```

Ponieważ wiemy, ile liczb znajduje się w pliku, do przetwarzania danych (czytania z pliku kolejnych elementów) wystarczy użyć pętli for, na przykład:

```
for(i=0; i<1000; i++)
   {
      fin>>liczba[i];
      ....
   }
```

Warto również utworzyć plik, który będzie przechowywał rozwiązania zadania, np.:

```
ofstream fout;
fout.open("wyniki.txt");
```

Po zakończeniu zadania należy pamiętać o instrukcjach:

```
fin.close();
fout.close();
```

1. W rozwiązaniu zadania 1. wykorzystamy algorytm rozkładu liczby na czynniki pierwsze i zmodyfikujemy go na potrzeby zadania — szukamy tych liczb, które mają tylko nieparzyste czynniki pierwsze oraz mają dokładnie trzy różne czynniki pierwsze.

   Jeżeli liczba jest parzysta, możemy ją wykluczyć z naszych rozważań. W pozostałych przypadkach stosujemy znany algorytm rozkładu na czynniki pierwsze z drobną modyfikacją. Zaczynamy od czynnika równego 3 i liczymy, ile różnych czynników pierwszych wystąpiło w liczbie. Za to odpowiedzialna jest zmienna `ile`. Jeżeli wartość tej zmiennej przekroczy 3, wiemy, że liczba ma więcej niż 3 różne czynniki pierwsze nieparzyste i nie wymaga dalszego sprawdzania. Funkcja zwraca wartość `true` jedynie wtedy, gdy liczba różnych nieparzystych czynników pierwszych jest równa dokładnie 3.

   Poniżej przedstawiono przykładową funkcję, która realizuje to zadanie.

```
bool czynniki(int liczba)
{
  int ile=0;
  int czynnik=3;
  if(liczba %2 == 0) return false;
  while (liczba>1)
  {
    if (liczba % czynnik == 0) ile++;
    while (liczba%czynnik==0){
      liczba = liczba/czynnik;
    }
    czynnik=czynnik+2;
    if (ile>3) return false;
  }
  if (ile == 3) return true;
  if (ile <3) return false;
}
```

2. Zadanie można podzielić na dwa etapy: znajdowanie odwrotności danej liczby oraz sprawdzanie, czy liczba jest palindromem. Do rozwiązywania tego zadania można podejść na

dwa sposoby. Jeśli wczytywaną liczbę potraktujemy jako napis, to jego odwrócenie będzie bardzo proste, podobnie jak sprawdzenie, czy napis jest palindromem. Problemem dla uczniów może okazać się sumowanie dwóch liczb, które są pamiętane jako napisy.
Drugi sposób polega na przetwarzaniu liczby przechowywanej w zmiennej typu int. Aby otrzymać liczbę odwróconą, należy uzyskać z niej kolejne cyfry, które będą tworzyć nową, odwróconą liczbę. Dodatkowo chcemy zapewnić dostęp do kolejnych cyfr liczby poprzez „odcięcie" cyfr liczby, począwszy od cyfry najmniej znaczącej. Można otrzymać taki rezultat dzięki dwóm operacjom: obliczaniu reszty z dzielenia przez 10 (operacja modulo 10) oraz obliczaniu dzielenia całkowitego liczby przez 10. Prosty algorytm odwracania może wyglądać następująco: dopóki nie odcięliśmy wszystkich cyfr liczby (liczba nie jest zerem), pobierz ostatnią cyfrę z liczby, zmodyfikuj liczbę odwróconą, wykorzystując pobraną cyfrę, odetnij ostatnią cyfrę z liczby. Realizuje to poniższa funkcja:

```
int odwroc (int liczba)
{
   int nowa=0;
   while(liczba>0)
   {
      nowa=10*nowa+liczba%10;
      liczba=liczba/10;
   }
   return nowa;
}
```

Teraz wystarczy sprawdzić, czy suma liczby i jej odwróconej postaci tworzą palindrom.
Można napisać funkcję, która będzie sprawdzała, czy uzyskana suma jest palindromem, ale możemy też wykorzystać funkcję odwroc — palindrom to takie słowo (w naszym przypadku liczba), która czytana od lewej do prawej da ten sam wynik co czytana od prawej do lewej. Wystarczy zatem sprawdzić, czy obliczona suma jest równa sumie odwróconej. Powyższe rozważania obrazuje następujący fragment kodu:

```
odwrocona = odwroc(liczba);
suma=odwrocona+liczba;
if (odwroc(suma)==suma)
   ile++;
```

3.
Zadanie wprowadza pojęcie mocy liczby. Dla danych liczb z pliku należy policzyć, ile jest liczb mocy od 1 do 8, oraz podać minimalną i maksymalną liczbę o mocy 1. To zadanie również można podzielić na etapy.
Aby obliczyć moc liczby, trzeba obliczyć iloczyn kolejnych cyfr danej liczby. Należy zauważyć, że obliczony iloczyn staje się nową liczbą, dla której, o ile nie jest to liczba jednocyfrowa, obliczamy ponownie iloczyn cyfr. Wbrew definicji, która mogłaby sugerować wykorzystanie tablicy do pamiętania kolejnych iloczynów, można obliczyć moc liczby „na bieżąco". Na pewno przyda się następująca funkcja iloczyn_cyfr:

```
int iloczyn_cyfr(int x)
{
int wynik=1;
    while (x>0)
    {
       wynik=wynik*(x%10);
       x=x/10;
    }
    return wynik;
    }
```

Funkcji iloczyn_cyfr będziemy używać do obliczania mocy liczby. Obliczamy iloczyn początkowy, który staje się nową liczbą. Tak długo, jak liczba nie jest jednocyfrowa (jest większa niż 9), wyliczamy iloczyn cyfr liczby, który to iloczyn za każdym razem ponownie staje się liczbą, której iloczyn dalej obliczamy. Każde obliczenie iloczynu cyfr powoduje zwiększenie licznika obliczającego moc liczby o 1. Poniżej przedstawiono funkcję realizującą obliczanie mocy liczby.

```
int moc(int liczba)
{

    int ile=1;

    liczba=iloczyn_cyfr(liczba); //iloczyn poczatkowy
    while (liczba>9)
    {
       liczba=iloczyn_cyfr(liczba);
       ile++;
     }
    return ile;
```

Teraz wystarczy dla danych liczb zliczać, ile jest liczb o mocy od 1 do 8. Do zliczania można użyć tablicy w taki sposób, by indeksy jej komórek odpowiadały liczbom od 1 do 8, zaś wartości tablicy odpowiadały za liczbę wystąpień liczb o danej mocy. Dodatkowo w przypadku liczby o mocy 1 należy wyznaczyć minimalną i maksymalną liczbę. Ponieważ wiemy, jaki jest zakres danych, możemy sobie uprościć wyszukiwanie, przyjmując jako początkowe minimum maksymalną liczbę, jaka może się pojawić w pliku, zaś jako maksimum — minimalną liczbę, jaka może wystąpić w pliku. Dalej wystarczy zastosować klasyczny algorytm znajdowania minimum (lub maksimum) w nieuporządkowanym ciągu liczb — w sytuacji, gdy liczba będzie miała moc równą 1 sprawdzamy, czy jest ona większa od aktualnego maksimum/ mniejsza od aktualnego minimum, i w razie potrzeby ustalamy nowego kandydata na minimum/maksimum. Oto fragment kodu:

```
    int i, liczba, tmp;
    int min=999999999, max=10;

    int ile=0;
    int t[9]={0};

    for(i=1; i<=1000; i++)
       {
          fin>>liczba;
          tmp=moc(liczba);
          t[tmp]++;
          if (tmp==1)
          {
             if (liczba<min) min=liczba;
             if (liczba>max) max=liczba;
          }

       }
```