72.1.

Za pomocą funkcji length() uzyskujemy długości obu napisów i porównujemy je ze sobą, pamiętając o tym, aby sprawdzić obie możliwości (albo pierwszy, albo drugi z nich może być tym dłuższym). Jeśli warunek podany w zadaniu jest spełniony, zwiększamy licznik przechowywany w zmiennej *wynik*:

```
int wynik = 0;
bool wypisalem = false;
for(int i=0; i<200; i++)
{
     string a,b;
     in >> a >> b;
     if (a.length()>=3*b.length() || a.length()*3 <=
b.length())
     {
          wynik++;
          (...)
     }
}
```

Konieczne jest jeszcze wypisanie pierwszej znalezionej pary. Po wypisaniu ustalamy wartość zmiennej logicznej *wypisalem* na *prawda*, co pozwoli nam uniknąć wypisania kolejnych par:

```
     if (a.length()>=3*b.length() || a.length()*3 <=
b.length())
     {
          wynik++;
          if (!wypisalem)
          {
               wypisalem = true;
               out << "Pierwsza para: " << endl;
               out << a << " " << b << endl;
          }
     }
```

72.2.

Dla dwóch zmiennych — łańcuchów znaków — chcemy sprawdzić, czy pierwszy z nich jest początkowym fragmentem drugiego. Pierwszy ma długość s = *a.length()*, zaś drugi t = *b.length()*. Wystarczy sprawdzić dwa warunki: czy $s < t$ oraz czy pierwsze s znaków jest w obu napisach identyczne:

```
for(int i=0; i<N; i++)
{
     string a,b;
     in >> a >> b;
     if (a.length() >= b.length())
          continue;
     bool dobry = true;
     for(int j=0; j<a.length(); j++)
          if (a[j]!=b[j])
               dobry = false;
     (...)

}
```

Instrukcja:

```
if (a.length() >= b.length())
     continue;
```

powoduje natychmiastowe przejście do następnej iteracji pętli, jeśli drugi ciąg znaków jest za krótki (nie przewyższa pierwszego długością). Kolejne instrukcje porównują ze sobą kolejne znaki *a* i *b*. Pierwsza niezgodność powoduje ustawienie zmiennej *dobry* na *false*. Jeśli do końca algorytmu wartość tej zmiennej pozostanie *true*, musimy wypisać oba napisy, a także różniące je litery:

```
if (dobry)
{
     out << a << " " << b << " ";
     for(int j=a.length(); j<b.length(); j++)
          out << b[j];
     out << endl;
}
```

Litery, które różnią napisy, to te, znajdujące się na końcu dłuższego z nich (czyli *b*), licząc od miejsca, w którym skończył się krótszy (czyli od *a.length()*). Stąd wynika konstrukcja pętli taka jak powyżej.

72.3.

Napiszmy najpierw funkcję *koncowka(A,B)*, która przyjmując dwa napisy, określi, jak długa jest ich wspólna końcówka. Aby to określić, liczymy najpierw długości obu, odpowiednio *dl_A* i *dl_B*. Teraz trzeba sprawdzić, czy ostatnie litery *A* i *B* są równe (czyli czy *A[dl_A-1]* jest równe *B[dl_B-1]*), potem przedostatnie litery (*A[dl_A – 2]* oraz *B[dl_B – 2]*) i tak dalej:

```
int koncowka(string A, string B)
{
     int dl_A = A.length();
     int dl_B = B.length();
     int k = 0;
     while(k<dl_A && k<dl_B && A[dl_A-1-k]==B[dl_B-1-k])
          k++;
     return k;
}
```

Zmienna *k* przechowuje liczbę wcześniej już znalezionych wspólnych liter. Ważne jest, aby przerwać pętlę, kiedy *k* osiągnie jedną z liczb *dl_A*, *dl_B*, co będzie oznaczać, że sprawdziliśmy już jeden z napisów w całości. Gdybyśmy próbowali kontynuować, wyrażenie (na przykład) *dl_A -1-k* przyjęłoby wartości ujemne, a więc sprawdzalibyśmy nieistniejące wartości w zmiennej tablicowej *A*. Szczególnie w języku C++ to bardzo niebezpieczny rodzaj błędu, jego skutki są bowiem trudne do przewidzenia. Czasem taki program zostanie przerwany z błędem wykonania, ale zdarza się, że na przykład kontynuuje on działanie z nagle pojawiającymi się absurdalnymi wartościami zmiennych albo źle wykonuje niektóre instrukcje.

Skoro mamy już prawidłową funkcję *koncowka()*, reszta działań programu sprowadza się do tego, aby najpierw znaleźć najdłuższy wspólny końcowy fragment, a następnie wypisać wszystkie pary napisów, dla których wartość jest maksymalna. W tym celu użyjemy tablic przechowujących łańcuchy znaków: *pierwszy[200]*, *drugi[200]* oraz tablicy liczb *wspolny[200]*:

```
int dlugosc = 0;
string pierwszy[200],drugi[200];
int wspolna[200];
for(int i=0; i<N; i++)
{
     in >> pierwszy[i] >> drugi[i];
     wspolna[i] = koncowka(pierwszy[i],drugi[i]);
     if (wspolna[i] > dlugosc)
          dlugosc = wspolna[i];
}
out << "Maksymalna koncowka: " << dlugosc << endl;
for(int i=0; i<N; i++)
     if (wspolna[i]==dlugosc)
          out << pierwszy[i] << " " << drugi[i] << endl;
```

Pierwsza część programu (pętla *for*) zapisuje wczytane napisy w zmiennych *pierwszy[i]* oraz *drugi[i]*, a obliczoną końcówkę w zmiennej *wspolna[i]*. Najdłuższą dotychczas znalezioną końcówkę pamiętamy w zmiennej *dlugosc*, zmieniając jej wartość, kiedy trafimy na większą.

W drugiej części programu jeszcze raz sprawdzamy wszystkie znalezione końcówki — jeśli któraś z nich ma długość maksymalną, wypisujemy odpowiednie napisy na wyjście.