PJC

(ćwiczenia 3)

Zadanie 1

Korzystając z tablic dwuwymiarowych stwórz program który będzie reprezentował i wypisywał trójkąt Pascala dowolnego rozmiaru. Trójkąt Pascala to trójkąt którego kolejne rzędy są zbudowane z sum kolejnych dwóch wartości z poprzedniego rzędu.
Np. dla danych size=4 program ma wypisać

```
1000
1100
1210
1331
```

Zadanie 2

Napisz program definiujący funkcję, która
pobiera wskaźnik do tablicy znaków (czyli napisu);
modyfikuje tę tablicę tak, aby po wyjściu z funkcji:

1. napis nie zawierał początkowych i końcowych spacji (jeśli takie były);
2. każda sekwencja więcej niż jednej spacji była zastąpiona dokładnie jedną Spacją.

UWAGA:
Nie wolno w łączać do programu nag łówka <cstring> (ani, tym bardziej, <string.h>).
Przewidzieć sytuację, gdy napis sk łada się z samych spacji lub jest napisem pustym.
Nie tworzyć żadnych pomocniczych tablic.

Schemat programu mógłby zatem być następujący:

```
#include <iostream>
using namespace std;
void clean(char* n) {
// implementacja funkcji
}
int main() {
char n1[] = "a   bc   def  ghijk ";
cout << "Przed: >" << n1 << "<" << endl;
clean(n1);
cout << " Po: >" << n1 << "<" << endl;
char n2[] = " a  bc  def    ghijk";
cout << "Przed: >" << n2 << "<" << endl;
clean(n2);
cout << " Po: >" << n2 << "<" << endl;
char n3[] = "     ";
cout << "Przed: >" << n3 << "<" << endl;
clean(n3);
cout << " Po: >" << n3 << "<" << endl;
}
```

Powyższy program powinien wydrukować:

```
Przed: >a   bc   def  ghijk <
Po: >a bc def ghijk<
Przed: > a  bc  def    ghijk<
Po: >a bc def ghijk<
Przed: >     <
Po: ><
```

gdzie znaki mniejszości/większości drukowane są po to, aby zobaczyć czy nie ma zbędnych spacji na początku i końcu otrzymanego napisu.

Zadanie 3

Napisz program, w którym definiujemy dowolne dwa C-napisy (tablice znaków), na przykład tak

```
char nap1[] = "Jakis Dlugi Napis";
char nap2[] = "Zupelnie inny napis";
```

i wysyłamy je do funkcji która wypisuje:

- w pierwszej linii wszystkie litery, które występują zarówno w pierwszym jak i drugim napisie (tylko litery, ignorujemy znaki odstępu, cyfry, znaki interpunkcyjne itd.). Każda litera występująca w obu napisach powinna zostać wypisana tylko raz. Odpowiadające sobie litery duże i małe (na przykład 'a' i 'A') są traktowane jako te same;
- w drugiej linii wszystkie litery z pierwszego napisu, których nie ma w drugim. Jak poprzednio, odpowiadające sobie litery duże i małe są traktowane jako te same.

Zadanie powinno zostać wykonane przez funkcję, do której posyłamy utworzone w funkcji main dwie niemodyfikowalne tablice znaków. Żadne dodatkowe tablice nie mogą być tworzone. Nie należy wykorzystywać znajomości kodów ASCII konkretnych liter. Schemat programu mógłby zatem być taki:

```
#include <iostream>
using namespace std;
void wypisz_wyniki(const char* n1, const char* n2);
int main() {
char nap1[] = "Jakiś Długi Napis";
char nap2[] = "Zupełnie inny napis";
wypisz_wyniki(nap1,nap2);
}
void wypisz_wyniki(const char* n1, const char* n2) {
// ...
}
```

gdzie, oczywiście, samą funkcję wypisz_wyniki należy zdefiniować (i, być może, inne funkcje pomocnicze). Wynik programu mógłby więc wyglądać tak:

```
a i l n p s u
d g j k
```

UWAGA:
Nie wolno włączać do programu nagłówków cstring (string.h) i/lub string.
Można założyć, że kody odpowiadające literom są kodami ASCII, to znaczy wartości liczbowe odpowiadające 'A','B',. . . ,'Z' są kolejnymi liczbami całkowitymi, i tak samo dla 'a','b',. . . ,'z'. Pamiętaj, że wartości zmiennych znakowych są w wyrażeniach konwertowane do wartości całkowitych, na przykład wartość wyrażenia 'd'-'a' wynosi 3, bo kod ASCII litery 'a' to 97, litery 'd' to 100 nie musimy tego wiedzieć, wystarczy nam informacja, że kolejne litery mają kolejne kody ASCII. Podobnie wartość wyrażenia 'C'-'A' wynosi 2, bo kod ASCII litery 'A' to 65, litery 'C' to 67.
Nie stosuj liter polskich wyłącznie litery alfabetu łacińskiego.