Zadanie 1

Zdefiniować pięć funkcji operujących na C-napisach (i jedną, **isLetter**, na pojedynczym znaku):

```
size_t length(const char* cstr);        // 1
bool   isLetter(char c);                // 2
char*  reverse(char* cstr);             // 3
size_t words(const char* cstr);         // 4
size_t words2(const char* cstr);        // 5
char*  concat(char* t, const char* s); // 6
```

gdzie

1. **length** zwraca długość napisu nie licząc znaku '\0' na jego końcu;
2. **isLetter** sprawdza, czy przekazany znak c jest, dużą lub małą literą (nie używać znajomości kodów ASCII);
3. **reverse** odwraca kolejność znaków w przekazanym napisie i zwraca niezmienioną wartość przekazanego wskaźnika cstr;
4. **words** zwraca ilość słów w przekazanym napisie, przy czym za słowo uważamy niepusty ciąg *liter*, dużych i małych, taki, że bezpośrednio przed nim i za nim nie ma litery;
5. **words2** zwraca ilość słów w przekazanym napisie, ale tym razem liczymy tylko „słowa" co najmniej dwuliterowe;
6. **concat** „dokleja" do napisu t (od target) napis s (od source); oczywiście trzeba zadbać o to, aby zarezerwowany pod adresem t obszar pamięci wystarczał na pomieszczenie obu napisów wraz z kończącym znakiem '\0'! Funkcja zwraca niezmodyfikowaną wartość pierwszego argumentu.

UWAGA: wszystkie te funkcje powinny być zaimplementowane samodzielnie bez odwoływania się do funkcji z biblioteki standardowej — w szczególności **strlen**, **isupper**, **isalpha**, **strcpy** itd. Nie należy włączać żadnych plików nagłówkowych z wyjątkiem *iostream*. Nie twórz żadnych pomocniczych tablic.

Na przykład program:

```
#include <iostream>

size_t length(const char* cstr);
bool   isLetter(char c);
char*  reverse(char* cstr);
size_t words(const char* cstr);
size_t words2(const char* cstr);
char*  concat(char* t, const char* s);
```

```
int main() {
    using std::cout; using std::endl;
    char s1[] = "Alice in Wonderland";
    cout << "reverse: " << reverse(s1) << endl;
    cout << "length : " << length(s1)  << endl;

    char s2[] = " ... for (int i = 0; i < n; ++i){...} ...";
    cout << "words  : " << words(s2)  << endl;
    cout << "words2 : " << words2(s2) << endl;

    char s3[100] = "Hello";
    cout << "concat : "
         << concat(concat(s3,", world"),"!") << endl;
}

// definicje funkcji
```

powinien wydrukować

```
reverse: dnalrednoW ni ecilA
length : 19
words  : 6
words2 : 2
concat : Hello, world!
```

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 włączać do programu nagłówka **<cstring>** *(ani, tym bardziej,* **<string.h>***).*
Przewidzieć sytuację, gdy napis składa 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 funkcję

```
bool checkpass(const char* pass);
```

która pobiera hasło (jako C-napis, czyli tablicę znaków) i sprawdza jego poprawność. Przyjmujemy, że poprawne hasło zawiera

1. co najmniej 8 znaków;
2. co najmniej 6 znaków różnych;
3. co najmniej 1 cyfrę;
4. co najmniej 1 dużą literę;
5. co najmniej 1 małą literę;
6. co najmniej 1 znak niealfanumeryczny (nie będący literą ani cyfrą).

Funkcja zwraca **true** jeśli hasło jest poprawne, a jeśli nie, to zwraca **false**, ale przedtem wypisuje komunikat o *wszystkich* przyczynach niepoprawności. Można założyć, że znaki są znakami ASCII o kodach w zakresie $[32, 126]$. [Może być przydatne zdefiniowanie też prostych funkcji pomocniczych.]

Na przykład program następujący

download *CStringPass.cpp*

```
#include <iostream>

// ...

bool checkpass(const char* pass) {
    // ...
}

int main() {
    using std::cout; using std::endl;
    const char* passes[] =
        {
            "AbcDe93",  "A1b:A1b>",  "Ab:Acb<",
            "abc123><", "Zorro@123", nullptr
        };
    for (int i = 0; passes[i] != nullptr; ++i) {
        cout << "checking " << passes[i] << endl;
        if (checkpass(passes[i])) cout << "OK" << endl;
        cout << endl;
    }
}
```

powinien wypisać coś w rodzaju

```
checking AbcDe93
Too short
No non-alphanumeric character

checking A1b:A1b>
Too few different characters

checking Ab:Acb<
Too short
Too few different characters
No digit

checking abc123><
No uppercase letter

checking Zorro@123
OK
```

Nie włączaj żadnych plików nagłówkowych innych niż *iostream*!

Zadanie 4

Sekwencje nukleotydów w cząsteczkach DNA oznaczane są ciągami liter A, C, G i T o dowolnej długości (skróty pochodzą od *adenine*, *cytosine*, *guanine* i *thymine*). Bardzo uproszczona metoda mierzenia podobieństwa między dwoma sekwencjami jest następująca:

- Dla ciągów *o tej samej długości* za każdy identyczny odcinek kodów o długości d doliczamy d^2 punktów; tak obliczona suma jest współczynnikiem podobieństwa obu sekwencji. Na przykład dla sekwencji `ACGTC` i `AGGTG` będzie to 5: jeden punkt za `A` na pierwszej pozycji i cztery za dwuznakowy ciąg `GT` na pozycjach 3-4.
- Dla sekwencji o niejednakowej długości obliczamy powyższym sposobem współczynniki podobieństwa między ciągiem krótszym, o długości, powiedzmy, d, a każdym podciągiem o tejże długości kolejnych elementów ciągu dłuższego. Obliczonym współczynnikiem jest wtedy największy z tak uzyskanych współczynników częściowych. Na przykład dla ciągów `AGG` i `CGGAT` obliczamy podobieństwa między trójznakowym ciągiem `AGG` a, kolejno, `CGG` (4 punkty), `GGA` (1 punkt) i `GAT` (0 punktów). Tak więc współczynnikiem podobieństwa będzie największa z tych wartości, czyli 4.

Napisz funkcję

```
int simil(const char a[], const char b[]);
```

która pobiera dwie sekwencje i zwraca ich podobieństwo (możesz też, jeśli uznasz to za wygodne, zdefiniować jakąś funkcję pomocniczą). Nie twórz żadnych dodatkowych tablic czy kolekcji i nie włączaj żadnych dodatkowych plików nagłówkowych. Na przykład program

download *DNA.cpp*

```
#include <iostream>

int simil(const char a[], const char b[]) {
    // ...
}

int main() {
    char a[] = "AACTACGTC",
         b[] = "ACGTA";
    std::cout << a << " and " << b << " -> "
              << simil(a,b) << std::endl;
    char c[] = "GCGC",
         d[] = "AGGGCA";
    std::cout << c << " and " << d << " -> "
              << simil(c,d) << std::endl;
}
```

powinien wydrukować

```
AACTACGTC and ACGTA -> 16
GCGC and AGGGCA -> 5
```
