Feladat 1

Írjon egy **megfordit(char* input, int len)** nevű függvényt, amely az input vektorban megadott len-karakter hosszúságú szöveget megfordítja.

```
void megfordit(char* input, int len)
{
    for(int i = 0; i < len/2; i++)
    {
        char temp = input[i];
        input[i] = input[len-1-i];
        input[len-1-i] = temp;
    }
}
```

Feladat 2

Írjon egy kiszed(char* input, char *output) nevű függvényt, amely az input vektorban megadott szövegből kiveszi az 'a' és 'e' betűket és az output verktorba kerül az eredmény.

```
kiszed(char* input, char *output)
{
    while(*input != '\0')
    {
        if(*input == 'a' || *input == 'e')
        {
            *output = *input;
            output++;
        }
        input++;
    }
}
```

Feladat 3

Írjon egy **char* kiegeszit(char* input, int len)** nevű függvényt, ami az input vektorban megadott **len**-karakter hosszúságú szöveg minden karaktere után egy _szóközt_ szúr be.

Ha a **bemenet**: HELLO, akkor a **függvény visszatérési értéke**: H E L L O legyen. * hozzon létre egy megfelelő méretű új karaktervektort a függvényben dinamikusan * másolja bele az input karaktertömb értékeit és a szóközöket * az új karaktervektor utolsó karakterét nullázza le.

```
char* kiegeszit(char* input, int len)
{
    // egyel kisebb mint a kétszerese lesz az új char*
    char *res = (char*)malloc(len + len - 1);
```

```
    for(int i = 0; i < len*2-1; i++)
    {
        if(i % 2 == 0)
        {
            res[i] = input[i/2];
        }
        else
        {
            res[i] = ' ';
        }
    }
    // a legutolsó karakter 0 kell legyen, a lezárás miatt
    res[len*2-1] = '\0';
    return res;
}
```

Feladat 4

Írjon egy **int wordCounter(char* input, int len)** nevű függvényt, ami az input vektorban megadott **len**-karakter hosszúságú szöveget megvizsgálja és visszaadja, hogy hány szó van benne.

Ha a **bemenet**: “Hello world here”, akkor a **függvény visszatérési értéke**: 3.

(megjegyzés: a szavak között csak 1 szóköz lehet a bemenetekben)

```
int wordCounter(char* input, int len)
{
    // számoljuk meg a szóközöket
    int space = 0;
    for(int i = 0; i < len; i++)
    {
        if(input[i] == ' ')
        {
            space++;
        }
    }
    return space + 1;
}
```

Feladat 5

Írjon egy **char leggyakoribb(char* input, int len)** nevű függvényt, ami az input vektorban megadott **len**-karakter hosszúságú szöveget megvizsgálja és visszaadja, hogy melyik betű fordul elő benne a legtöbbször.

Ha a **bemenet**: “Hello here”, akkor a **függvény visszatérési értéke**: e.

(megjegyzés: a legtöbbször előfordulóból mindig 1 lesz)

* készítsünk egy tömböt, ami az egyes előfordulásokat fogja tartalmazni, 0-val inicializáljuk az elemeit * karakterenként menjünk végig az input vektoron és a számlálótömb megfelelő elemét növeljük * válasszuk ki melyik elem fordul elő a tömbben legtöbbször és adjuk vissza az indexét

```
char leggyakoribb(char* input, int len)
{
    char gyakorisag[255];
    for(int i=0; i<255; i++)
    {
        gyakorisag[i] = 0;
    }

    for(int i=0; i<len; i++)
    {
        gyakorisag[(int)input[i]]+=1;
    }

    int max = gyakorisag[0];
    char gyakori;
    for(int i=0; i<255; i++)
    {
        if(max < gyakorisag[i])
        {
            max = gyakorisag[i];
            gyakori = i;
        }
    }
    return gyakori;
}
```

Feladat 6

Írjon egy C függvényt, folytatva a megkezdettet, amely kap egy sztringet, majd megvizsgálja, hogy a sztring palindróm-e (azaz ugyanaz, ha visszafelé olvassuk)

```
int palindrom(const char* input)
{
    int palindrom = 1;
    for (int i = 0, j = strlen(input) - 1; i < j; i++, j--) {
        if (input[i] != input[j]) {
            palindrom = 0;
            break;
        }
    }
    return palindrom;
}
```

Feladat 7

Írjon egy C szamol() függvényt, amely megkap egy sztringet paraméterként, majd kiírja a sztringben található számjegyek összegét.

```
int szamjegyekOsszege(const char *sztring) {
    int sum = 0;
    for (int i = 0; sztring[i] != '\0'; ++i) {
        if (sztring[i] >= '0' && sztring[i] <= '9') {
            sum += (sztring[i] - '0');
        }
    }
    return sum;
}
```

Feladat 8

Írjon egy “int maganhangzo_szamlal(const char*)” nevű függvényt, amely visszaadja, hány magánhangzó található a kapott szövegben (a, e, i, o, u betűk kis- és nagybetűs változata).

```
int maganhangzo_szamlal(const char *szoveg)
{
    int darab = 0;
    for (int i = 0; szoveg[i] != '\0'; ++i) {
        char c = szoveg[i];
        if (c == 'a' || c == 'A' || c == 'e' || c == 'E' ||
            c == 'i' || c == 'I' || c == 'o' || c == 'O' ||
            c == 'u' || c == 'U') {
            ++darab;
        }
    }
    return darab;
}
```

Feladat 9

Írjon egy “void kisbetusit(char*)” nevű függvényt, amely minden karaktert kisbetűssé alakít a kapott szövegben.

```
#include <ctype.h>
void kisbetusit(char *szoveg)
{
    for (int i = 0; szoveg[i] != '\0'; ++i) {
        szoveg[i] = (char)tolower((unsigned char)szoveg[i]);
```

```
    }
}
```

Feladat 10

Írjon egy “void eltavolit_karakter(char*, char)” nevű függvényt, amely eltávolítja a megadott karakter minden előfordulását a szövegből.

```
void eltavolit_karakter(char *szoveg, char c)
{
    int j = 0;
    for (int i = 0; szoveg[i] != '\0'; ++i) {
        if (szoveg[i] != c) {
            szoveg[j++] = szoveg[i];
        }
    }
    szoveg[j] = '\0';
}
```

Feladat 11

Írjon egy “int suffixel(const char*, const char*)” nevű függvényt, amely 1-et ad vissza, ha az első szöveg utolsó karakterei megegyeznek a másodikkal, különben 0-t.

```
int suffixel(const char *szoveg, const char *suf)
{
    int n = 0;
    while (szoveg[n] != '\0') {
        ++n;
    }
    int m = 0;
    while (suf[m] != '\0') {
        ++m;
    }
    if (m > n) {
        return 0;
    }
    for (int i = 0; i < m; ++i) {
        if (szoveg[n - m + i] != suf[i]) {
            return 0;
        }
    }
    return 1;
}
```

Feladat 12

Írjon egy “void rovidit(char*, int)” nevű függvényt, amely a megadott hosszra csonkolja a szöveget (és lezárja nullával).

```
void rovidit(char *szoveg, int hossz)
{
    if (hossz < 0) {
        hossz = 0;
    }
    szoveg[hossz] = '\0';
}
```

Feladat 13

Írjon egy “int csak_alphanumerikus(const char*)” nevű függvényt, amely 1-et ad vissza, ha a szöveg minden karaktere betű vagy szám, különben 0-t.

```
int csak_alphanumerikus(const char *szoveg)
{
    for (int i = 0; szoveg[i] != '\0'; ++i) {
        if (!isalnum((unsigned char)szoveg[i])) {
            return 0;
        }
    }
    return 1;
}
```

Feladat 14

Írjon egy “void szokozok_torol(char*)” nevű függvényt, amely eltávolítja a szövegből az összes szóközt.

```
void szokozok_torol(char *szoveg)
{
    int j = 0;
    for (int i = 0; szoveg[i] != '\0'; ++i) {
        if (szoveg[i] != ' ') {
            szoveg[j++] = szoveg[i];
        }
    }
    szoveg[j] = '\0';
}
```

Feladat 15

Írjon egy “int leghosszabb_szo(const char*)” nevű függvényt, amely visszaadja a leghosszabb szó hosszát a szóközökkel elválasztott szövegben.

```
int leghosszabb_szo(const char *szoveg)
{
    int max = 0;
    int akt = 0;
    for (int i = 0;; ++i) {
        char c = szoveg[i];
        if (c != '\0' && c != ' ') {
            ++akt;
        } else {
            if (akt > max) {
                max = akt;
            }
            if (c == '\0') {
                break;
            }
            akt = 0;
        }
    }
    return max;
}
```

Feladat 16

Írjon egy “void karakter_csere(char*, char, char)” nevű függvényt, amely a régi karaktert az újra cseréli a szövegben.

```
void karakter_csere(char *szoveg, char regi, char uj)
{
    for (int i = 0; szoveg[i] != '\0'; ++i) {
        if (szoveg[i] == regi) {
            szoveg[i] = uj;
        }
    }
}
```

Feladat 17

Írjon egy “void fordit_masol(const char*, char*)” nevű függvényt, amely a forrás szöveget megfordítva másolja a cél tömbbe.

```
#include <string.h>
```

```
void fordit_masol(const char *forras, char *cel)
{
    int hosszn = (int)strlen(forras);
    for (int i = 0; i < hosszn; ++i) {
        cel[i] = forras[hosszn - 1 - i];
    }
    cel[hosszn] = '\0';
}
```