

Lab 5. Stałe i zmienne znakowe. Tablice. Wskaźniki do tablic. Operacje na wskaźnikach. Instrukcja switch, case. Wyrażenie przecinkowe. Funkcje tekstowe.

1. **char** jest to typ **znakowy**, umożliwiający zapis znaków ASCII. Może też być traktowany jako liczba z zakresu 0..255 (Oznacza to, że każdy znak jest reprezentowany w pamięci przez swój kod. Kody znaków alfanumerycznych to liczby z przedziału *32...127+ pozostałe liczby są również kodami znaków, ale mogą być one różnie definiowane – nie odpowiadają standardowi ASCII). Znaki zapisujemy w pojedynczych cudzysłowach (czasami nazywanymi apostrofami), by odróżnić je od tańcuchów tekstowych (pisanych w podwójnych cudzysłowach). Np. 'a'; '7'; '!' ; '\$'.

- i. Stała znakowa ma postać: 'znak'
- ii. Deklaracja zmiennej znakowej poprzez podanie kodu znaku spacji
`char i = 32; // i - znak spacji`
- iii. b. Deklaracja zmiennej znakowej poprzez podanie znaku
`char i = ' '; // i - znak spacji`

```
//Zmienne znakowe wyprowadzanie znaków, ++i
#include <stdio.h>
#include <stdlib.h>
int main()
{
    /* Kody ASCII */
    int i = 32;
    /* int i=' '; jest poprawne */
    while (++i < 128)
    {
        printf("%3d %c ", i, i);
        if (!((i - 32) % 4)) printf("\n");
    }
    printf("\n");
    system("pause");
}
```

```
//Zmienne znakowe wyprowadzanie znaków, i++
#include <stdio.h>
#include <stdlib.h>
int main()
{
    /* Kody ASCII */
    char i = ' ';
    /* char i=32; jest poprawne */
    while (i++ < 127)
    {
        printf("%3d %c ", i, i);
        if (!((i - 32) % 4)) printf("\n");
    }
    printf("\n");
    system("pause");
}
```

2. Typ **char** to zwykły typ liczbowy i można go używać tak samo jak typu **int** (zazwyczaj ma jednak mniejszy zakres). Co więcej literały znakowe (np. 'a') są traktowane jak liczby i w języku C są typu **int** (w języku C++ są typu **char**).
3. **Stała tekstowa** – dowolny ciąg znaków, zapisany w postaci: "to jest stała tekstowa". Jeśli w stałej chcemy umieścić cudzysłów należy wtedy poprzedzić znak " znakiem \ tzn. umieścić w tekście sekwencję \". Kompilator umieszcza tekst w odpowiedniej grupie bajtów (1B na symbol). **Na końcu stałej tekstowej dopisywany jest znak o kodzie zero, czyli symbol '\0'.**

"to jest stała tekstowa" = 22 znaki łącznie ze spacjami. W pamięci komputera stała ta zajmie 23B razem ze znakiem \0.

4. **Tablice znakowe** – używane są do zapisu ciągu znaków w pamięci komputera. Ciąg znaków (wiersz tekstowy) jest umieszczany w kolejnych bajtach pamięci. Każdy znak tego ciągu to odpowiedni element tablicy znakowej.

Definicja tablicy znakowej: `char tekst[num];`

gdzie: tekst – to nazwa (identyfikator) tablicy, num – maksymalna liczba znaków minus jeden – tzw. rozmiar tablicy; musi być zdefiniowany jako stała.

Elementy tablicy numerowane są od 0 do n-1 (numery elementów to tzw. indeksy tablicy)

tekst[0], tekst[1], ..., tekst[n-2], tekst[n-1]

przy czym tekst[n-1] -> '\0', pamięć zarezerwowana na num elementów: num >= n.

Odwołanie do elementu tablicy o indeksie i następuje poprzez użycie operatora [] np.: tekst[i], a w każdym elemencie można zapisać tylko jeden znak np. : tekst[1]= 'a'.

```
//Liczba powtórzeń litery a (1)-tablice
tekstowe
#include <stdio.h>
#include <stdlib.h>
#define MAX_LINE 256
int main()
{
/*Ile razy litera a powtarza sie w tekście*/
    char d[MAX_LINE], b;
    int k, l;
    k = 0; l = 0;
    gets(d);
    while ((b = d[k]) != '\0')
    {
        if (b == 'a') l++;
        k++;
    }
    printf("%s\n%d\n", d, l);
    system("pause");
}
```

```
//Liczba powtórzeń litery a (2) - tablice
tekstowe
#include <stdio.h>
#include <stdlib.h>
#define MAX_LINE 256
int main()
{
/*Ile razy litera a powtarza sie w tekście*/
    char d[MAX_LINE], b;
    int k = 0, l = 0;
    gets(d);
    while ((b = d[k++]) != '\0')
        if (b == 'a') l++;
    printf("%s\n%d\n", d, l);
    system("pause");
}
```

5. Funkcja **gets()** - czyta linię ze standardowego wejścia (usuwa ją stamtąd) i umieszcza ją w tablicy znakowej wskazywanej przez str. Ostatni znak linii (znak nowego wiersza - '\n') zastępuje zerem (znakiem '\0'). Wartością zwracaną funkcji jest str (wskaźnik do tekstu) w przypadku sukcesu lub NULL w przypadku błędu lub natrafienia na koniec pliku. Funkcja nie sprawdza, czy jest miejsce do zapisu w tablicy str.

Deklaracja: `char *gets(char *str);`

6. **Wskaźnik do tablicy** – definicja tablicy rezerwuje odpowiednią grupę bajtów w pamięci i przypisuje nazwie (identyfikatorowi) tekst adres zerowego elementu. Nie wolno zmieniać wartości tekst – jest to stała, wskaźnik początku tablicy. Dodatkowo identyfikator tekst nie jest L-value, tzn. że nie można go używać po lewej stronie instrukcji przypisania.

Przykład 1:

```
char str1[] = "abcde"; //debuger information: str1 = 0x0012fe44 "abcde"
//str1[0] = 97 'a', &str1[0] 0x0012fe44 "abcde"
//str1[1] = 98 'b' &str1[1] 0x0012fe45 "bcde"
```

Przykład 2:

```
char str[256];
char *ptr = NULL;
ptr = gets(str); //Przy wyjściu z gets str zawiera ciąg znaków,
//wprowadzonych z monitora, ptr – to jest wskaźnik do str:
    str 0x0012fe60 "abcdef" char[256]
    ptr 0x0012fe60 "abcdef" char *
```

Przykład 3:

```
char *ptr = NULL; //ptr 0x00000000
```

```

ptr = "abcdefg"; //ptr 0x0041563c "abcdefg" char *
//Uwaga! Pamięć pozostała wydzielona w obszarze roboczym. Ta
//informacja nie będzie chroniona.
//Kod poprawny:
ptr = (char *)malloc(256 * sizeof(char)); //dynamiczne wydzielenie
//pamięci w stercie (heap)
if (!ptr)
{
    printf("błąd \n");
    exit(1);
}
sprintf(ptr, "abcdefg");
//.....
//jeżeli ptr jest już nie potrzebne - zwolnij pamięć!
if (ptr)
    free(ptr);
ptr = NULL;

//Liczba powtórzeń litery a (3) - tablice a
//wskaźniki
#include <stdio.h>
#include <stdlib.h>
#define MAX_LINE 256
int main()
{
    /*Ile razy litera a powtarza sie w tekście*/
    char d[MAX_LINE], *dd;
    int l;
    l = 0;
    dd = gets(d);
    while (*dd != '\0')
    {
        if (*dd == 'a') l++;
        dd++;
    }
    printf("%s\n%d\n", d, l);
    system("pause");
}

//Liczba powtórzeń litery a (4) - krótko
#include <stdio.h>
#include <stdlib.h>
#define MAX_LINE 256
int main()
{
    /*Ile razy litera a powtarza sie w tekście*/
    char d[MAX_LINE], *dd;
    int l;
    l = 0;
    dd = gets(d);
    while (*dd)
        if (*dd++ == 'a') l++;
    printf("%s\n%d\n", d, l);
    system("pause");
}

```

7. Instrukcja **switch, case** - tzw. warunek wielokrotnego wyboru lub przełącznik. Dzięki tej instrukcji możemy wykonywać decyzje tylko i wyłącznie na podstawie wartości jednej zmiennej. Możliwości instrukcji **switch** są nieporównywalnie mniejsze od możliwości funkcji **if**, jednak używanie jej w niektórych przypadkach jest znacznie korzystniejsze dla szybkości działania programu i estetyki kodu niż użycie funkcji **if**.

Składnia:

```

switch (zmienna)
{
case wartosc_1:
    //jakiś kod
    break;
case wartosc_2:
    //jakiś kod
    break;
//...
case wartosc_n:
    //jakiś kod
    break;
default:
    //jakiś kod
    break;
}

```

Algorytm działania:

1. Obliczenie wartości *zmienna*
2. Jeśli *zmienna == wartosc_i* - wykonanie ciągu instrukcji (i + 1)....*ciąg instrukcji n*, goto 3;
jeśli *zmienna != wartosc_i* - ciąg instrukcji *default*, goto 3 ;
3. Pierwszy operator poza blokiem *switch*

W przypadku gdy operator *default* jest pominięty w deklaracji instrukcji *switch*:

1. Obliczenie wartości *zmienna*
2. Jeśli *zmienna == wartosc_i* - wykonanie ciągu instrukcji (i + 1)....*ciąg instrukcji n*; pierwszy operator poza blokiem *switch*

Ostatnim operatorem ciągu instrukcji i może być **break**; - przerywa on działanie *switch* i przekazuje sterowanie do pierwszego operatora poza blokiem *switch*.

```
switch (i)
{
case -1:
    n++;
    break;
case 0:
    z++;
    break;
case 1:
    p++;
    break;
}
```

8. Przykład: liczba wystąpień liter a, b oraz liter od b do y.

```
//Instrukcje switch, case, break, continue
#include <stdio.h>
#include <stdlib.h>
#define MAX_LINE 256

int main()
{
    /* Liczba wystąpień liter a, b oraz liter od b do y */
    char d[MAX_LINE], *dd, b;
    int l, k, m;
    l = k = m = 0;
    dd = gets(d);
    while (b = *dd++, b != '\0')
    {
        if (!(b >= 'a' && b <= 'y')) continue;
        switch (b)
        {
            case 'a':
                l++;
                break;
            case 'b':
                k++;
            default:
                m++;
        }
    }
    printf("%d %d %d\n", l, k, m);
    system("pause");
}
```

9. Operator przecinkowy, wyrażenie przecinkowe.

Operator przecinkowy jest operatorem o najniższym priorytecie i jest lewostronnie łączny.

Wyrażenie przecinkowe to wyrażenie postaci:

wyrażenie_1, wyrażenie_2

Wartością wyrażenia przecinkowego jest wartość wyrażenie_2.

10. Przykład: liczba słów w jednej linii tekstu.

```
//Liczba słów w linii tekstu (1)
#include <stdio.h>
#include <stdlib.h>
#define MAX_LINE 256

int main()
{
    char d[MAX_LINE], *dd, p = ' ', b;
    int l = 0;
    dd = gets(d);
    printf("%s\n", dd);
    /* Tu zostanie wyprowadzony cały wczytany tekst */
    while ((b = *dd) != '\0')
    {
        if (b != ' ')
            if (p == ' ') l++;
        p = b;
        dd++;
    }
    printf("%d\n", l);
    /* A tu zostanie wyprowadzony pusty tekst */
    printf("%s\n", dd);
    system("pause");
}
```

11. Przykład: liczba słów w jednej linii tekstu – tablice w argumentach funkcji.

```
//Liczba słów w linii tekstu (2) - tablice w argumentach funkcji
#include <stdio.h>
#include <stdlib.h>
#define MAX_LINE 256
int ile_slow_1(char *), ile_slow_2(char *), ile_slow_3(char *), ile_slow(char *);

int main()
{
    char d[MAX_LINE], *dd;
    int l;
    dd = gets(d);
    /* Tu zostanie wyprowadzony cały wczytany tekst */
    printf("%s\n", dd);
    l = ile_slow_1(d);
    printf("%d\n", l);

    l = ile_slow_2(dd);
    printf("%d\n", l);
    /* I tu zostanie wyprowadzony cały wczytany tekst */
    printf("%s\n", dd);

    l = ile_slow_3(d);
    printf("%d\n", l);

    l = ile_slow(dd);
}
```

```

        printf("%d\n", l);
        system("pause");
    }

    int ile_slow_1(char te[])
    {
        /* Ile jest slow w linii tekstu. Tablice w argumentach funkcji*/
        char p = ' ', b;
        int l = 0, i = 0;
        while (b = te[i++])
        {
            if (b != ' ')
                if (p == ' ') l++;
            p = b;
        }
        return l;
    }

    int ile_slow_2(char *te)
    {
        /* Ile jest slow w linii tekstu. Wskaźniki w argumentach funkcji*/
        char p = ' ', b;
        int l = 0;
        while (b = *te)
        {
            if (b != ' ')
                if (p == ' ') l++;
            p = b;
            te++;
        }
        return l;
    }

    int ile_slow_3(char *te)
    {
        /* Ile jest slow w linii tekstu. Elementy tablic == wskazanie pośrednie*/
        char p = ' ', b;
        int l = 0, i = 0;
        while (b = te[i])
        {
            if (b != ' ')
                if (p == ' ') l++;
            p = b;
            i++;
        }
        return l;
    }

    int ile_slow(char *te)
    {
        char p, b = ' ';
        int l = 0;
        while (p = b, b = *te++)
            if (b != ' ' && p == ' ') l++;
        return(l);
    }

```

12. Operacje na tekstach – wybrane funkcje.

Funkcja **strcpy()** – Funkcja kopiuje łańcuch znaków (src) do tablicy znaków (dest). Funkcja nie sprawdza czy łańcuch kopiowany zmieści się w tablicy docelowej. Nie istnieje wartość, która wskazywałaby na błędne wykonanie.

Deklaracja: `char * strcpy(char * dest, const char * src);`

Funkcja **strcat()** – *Dopisuje ciąg znaków (string) strFrom na końcu bufora (string) strTo i zamyka symbolem końca \0. Zachowanie funkcji jest nieokreślone, jeśli bufory strTo oraz strFrom wzajemnie się pokrywają. Funkcja zwraca wskaźnik do strTo. Nie istnieje wartość, która wskazywałaby na błędne wykonanie.*

Deklaracja: `char * strcat(char * strTo, const char * strFrom);`

Funkcja **strcmp()** – (znajduje się w bibliotece <string.h>) – *Porównuje ciągi znaków string1 i string2. Zwraca wartość (RetVal) typu int.*

RetVal: < 0 – $string1 < string2$

$= 0$ – $string1 == string2$ – *jeśli każdy znak ze string1 jest równy odpowiedniemu znakowi ze string2*

> 0 – $string1 > string2$

Deklaracja: `int strcmp(const char * string1, const char * string2);`

Relacja nierówności: w języku C przyjęto, że $string1 > string2$, jeśli kod pierwszego symbolu ze string1 jest większy od kodu pierwszego symbolu ze string2. Jeśli pierwsze znaki są równe, porównujemy drugie itd.

Funkcja **strlen()** – (<string.h>) - *oblicza długość łańcucha str. Jej działanie polega na zliczaniu znaków aż do napotkania 0 (znaku '\0'). 0 nie jest wliczane do długości. W przypadku łańcuchów nie zakończonych 0 jej działanie jest nieokreślone. Zwraca długość łańcucha str.*

Deklaracja: `int strlen(char *str);`

Funkcja **getchar()** - *czyta znak ze standardowego wejścia i go stamtąd usuwa. Wywołanie getchar() jest równoważne wywołaniu getc(stdin).*

Deklaracja: `int getchar(void);`

Funkcja **putchar ()** - *wysyła znak na standardowe wyjście. Wywołanie putchar (c) jest równoważne wywołaniu putc(c, stdout).*

Deklaracja: `int putchar(int c);`

13. Znajdź pozycję wybranego znaku w tekście.
14. Usuń z tekstu pierwsze wystąpienie wybranego znaku.
15. Zamień każde wystąpienie wybranego znaku na inny znak.
16. Dołącz tekst do innego tekstu.
17. Dołącz tekst od znaku o podanej pozycji na koniec innego tekstu.
18. Zaproponuj i zaimplementuj algorytm, który przesunie zawartość N-elementowej tablicy cyklicznie o jedną pozycję „w górę”. (tzn. pierwszy element na miejsce drugiego, drugi na trzeci, ..., ostatni na pierwszy).
19. Napisz funkcje:

```
int moje_strcmp(const char*, const char*);
char* moje_strcat(char*, const char*);
char* moje_strcpy(char*, const char*);
```

realizujące te same operacje co standardowe funkcje strcmp, strcmpat, strcmpi z biblioteki <string.h>.

20. Przeanalizuj w trybie pracy krokowej następujące przykłady do wykładu 5:

http://torus.uck.pk.edu.pl/~fialko/text/CC/przykl/WW5_1.pdf

http://torus.uck.pk.edu.pl/~fialko/text/CC/przykl/WW5_2.pdf

http://torus.uck.pk.edu.pl/~fialko/text/CC/przykl/WW5_3.zip

Wszystkie przykłady dostępne pod adresem:

<http://torus.uck.pk.edu.pl/~fialko/text/CC/przykl/>

**Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.*