

Lab 6. Tablice znakowe o dwóch indeksach, przekazywanie tablic do funkcji cd., dynamiczna alokacja pamięci, funkcje przetwarzające ciągi znakowe. Projekt nr 2.

1. Zapoznaj się z pojęciami stos, sarta, alokacja pamięci (statyczna i dynamiczna), przepełnienie stosu, przepełnienie sterty (Wikipedia).
2. Tablice możemy deklarować na kilka sposobów. W przypadku alokacji statycznej możemy je zadeklarować m.in. następująco:

```
int tab1[5];
int tab2[5] = { 2, 5, 3, 2, 5 };
int tab3[5] = { 2, 2, 1 };
int tab4[5] = {0};
int tab5[] = { 1, 2, 3, 4, 5 };
```

Czym różnią (jeśli się różnią) powyższe sposoby? Zadeklaruj tablicę statyczną (bez inicjalizacji jej wartości - pierwszy sposób), podobnie tablicę globalną - jakie wartości przyjmują poszczególne elementy tablicy?

Język C pozwala w sposób analogiczny na deklarację tablic wielowymiarowych. Przykłady tablic dwuwymiarowych:

```
int matrix1[2][2];
int matrix2[2][2] = { { 1, 2 }, { 3, 4 } };
int matrix3[2][2] = {0};
```

Czym jest nazwa tablicy - jaką przyjmuje wartość? Udowodnij swoją odpowiedź krótkim przykładem.

3. Tablice mogą być przekazywane do funkcji na dwa sposoby: poprzez referencję lub poprzez wskaźnik. Oznacza to, że przekazując tablicę nie jest tworzona jej kopia, a działania są wykonywane bezpośrednio na oryginalnej tablicy. Przekazywanie przez wartość jest niemożliwe.

```
void myfun(int *tab)
void myfun(int tab[])
```

4. Tablica dwuwymiarowa to jednowymiarowa tablica wskaźników do jednowymiarowych tablic danego typu.

```
char d[200][256];
```

- rezerwuje miejsce w pamięci dla 200*256 zmiennych typu `char`;
- identyfikator `d` jest stałą o wartości równej adresowi elementu `d[0][0]`;
- wartość `d[0]` wskazuje na element `d[0][0]`, wartość `d[1]` wskazuje na element `d[1][0]`, ... itd.;
- wartości `d`, `d[0]`, `&d[0][0]` są takie same. `d`, `d[0]`, `d[1]`, `d[2]`,... są stałymi – nie wolno zmieniać ich wartości.

Przykład:

```
//Liczba słów w wielu liniach tekstu (1) - tablice o dwóch indeksach
#pragma warning (disable: 4996)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LINE 256
#define MAX_LINES 200
FILE *fd = NULL;
int ile_slow(char *);

int main()
```

```

{
    char d[MAX_LINES][MAX_LINE];
    int i, l;
    if (!(fd = fopen("DANE.txt", "r")))
    {
        printf("Bład otwarcia zbioru\n");
        exit(2);
    }
    i = 0;
    l = 0;
    while (i < MAX_LINES && fgets(d[i], MAX_LINE, fd) != (char*)NULL)
    {
        l += ile_slow(d[i]);
        i++;
    }
    fclose(fd);
    fd = NULL;
    printf("%d\n", l);
    system("pause");
}

/* TU funkcja ile_slow */

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

```

5. Funkcja **fgets()** czyta kolejne znaki ze strumienia stream i umieszcza je w tablicy znakowej wskazywanej przez str. Czytanie przerywa, gdy przeczyta size - 1 znaków, natrafi na koniec pliku lub znak końca linii (znak ten jest zapisywany do str). Na końcu fgets() dopisuje znak '\0'. W przypadku błędu lub natrafienia na koniec pliku wartością funkcji jest NULL.

Deklaracja: `char *fgets(char *str, int size, FILE *stream);`

6. **Dynamiczna alokacja pamięci** - tworząc aplikacje zazwyczaj nie znamy z góry rozmiarów tablic jakie będą nam potrzebne. Często pamięć zarezerwowana przez stos jest niewystarczająca. Możemy wtedy skorzystać z dynamicznej alokacji pamięci. Język C dostarcza w tym celu funkcję **malloc**, która jako argument przyjmuje rozmiar bloku pamięci, który ma zaalokować, natomiast zwraca wskaźnik do zaalokowanej pamięci lub NULL w przypadku błędu. Przykładowo alokację 10 elementowej tablicy typu double możemy zapisać następująco:

```

double *tab;
tab = (double *)malloc(sizeof(double) * 10);
if (!tab)
{
    //bład alokacji
}

```

Język C nie posiada mechanizmów automatycznego "odśmiecania pamięci" (tzw. garbage collecting). Procedura zarówno alokacji jak i dealokacji spoczywa na programiście. W celu zwolnienia pamięci korzystamy w funkcji **free**. Zwolnienie może być wykonane tylko raz. Pamiętajmy, aby po każdym użyciu funkcji **free** "zniszczyć" wskaźnik poprzez przypisanie wartości NULL.

```

if (tab)

```

```

{
    free(tab);
    tab = NULL;
}

```

7. W przypadku tablic wielowymiarowych ich przekazywanie do funkcji znacznie się komplikuje. Aby przekazać tablicę wielowymiarową w sposób podobny do tablic jednowymiarowych musimy z góry znać stały rozmiar tablicy:

```

void myfun(int tab[2][2])
void myfun(int (*tab)[2])

```

Rozwiązaniem powyższego problemu może być zapis tablicy dwuwymiarowej w jednym wymiarze - jednak nie zawsze jest to pożądane rozwiązanie.

8. **Dynamiczna alokacja pamięci dla tablic wielowymiarowych** - cały proces wygląda analogicznie do alokacji tablic jednowymiarowych - należy jedynie pamiętać, że musimy zaalokować każdy wiersz z osobna (podobnie w przypadku zwalniania pamięci). Przykładowa tablica 10x5 o elementach typu double:

```

double **tab;
tab = (double **)malloc(sizeof(double) * 10);
if (!tab)
{
    //błąd alokacji
}
for (int i = 0; i < 10; i++)
{
    tab[i] = (double *)malloc(sizeof(double) * 5);
    if (!tab[i])
    {
        //błąd alokacji
    }
}
if (tab)
{
    for (int i = 0; i < 10; i++)
    {
        if (tab[i])
        {
            free(tab[i]);
        }
    }
    free(tab);
    tab = NULL;
}

```

9. Przykład dla tablic tekstowych:

```

//Liczba słów w wielu liniach tekstu (2) - alokacja pamięci
#pragma warning (disable: 4996)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LINE 256
#define MAX_LINES 200
FILE *fd = NULL;
int ile_slow(char *);

int main()
{

```

```

/* Ile slow w wielu liniach tekstu, alokacja pamieci */
char *d[MAX_LINES], bufor[MAX_LINE];
int len, i, l;
if (!(fd = fopen("DANE.txt", "r")))
{
    printf("Blad otwarcia zbioru\n");
    exit(2);
}
i = 0;
l = 0;
while (i < MAX_LINES && fgets(bufor, MAX_LINE, fd))
{
    len = strlen(bufor);
    bufor[len - 1] = '\0';
    if ((d[i] = (char *)malloc((unsigned)len)) == (char*)NULL)
    {
        printf("Brak pamieci\n");
        exit(3);
    }
    strcpy(d[i], bufor);
    l += ile_slow(d[i]);
    i++;
}
fclose(fd);
fd = NULL;
printf("%d\n", l);
system("pause");
/* Tekst w pamieci, tablica d przechowuje wskaźniki do linii tekstu */
}

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

```

10. Czym różnią się funkcje: **malloc**, **calloc**, **realloc**?
11. Korzystając z funkcji **fgets** napisz program, który będzie zaczytywał linia po linii tekst zapisany w pliku tekstowym, a następnie wyświetlał numer linii, liczbę dużych liter, liczbę małych liter i liczbę pozostałych znaków. Dla uproszczenia założmy, że tekst nie zawiera polskich znaków oraz maksymalna liczba znaków w pojedynczej linii wynosi 256.
12. Biblioteka **stdlib.h** dostarcza funkcji konwertujących ciąg znaków na liczbę (**atoi**, **atof**, **atol**, **atoll**) oraz liczby na ciąg znaków (**itoa**, **ecvt**, **fcvt**) - więcej informacji w dokumentacji MSDN oraz na wykładzie.
13. Tablice dwuwymiarowe w argumentach funkcji. Zmodyfikuj program z przykładu nr 9 tak, by algorytm liczenia liczby słów nie był zapisany w funkcji **main()** ale w funkcji o nazwie **licz_slowa_1**. W której zostanie wykorzystana funkcja **ile_slow**. Do funkcji **licz_slowa_1** winien być przekazany cały przeczytany tekst – prześlemy jedynie tablice wskaźników do poszczególnych linii tekstu.

```

//Liczba słów w wielu liniach tekstu (3) - tablice 2-indeksowe w argumentach -
argumenty funkcji main()
#pragma warning (disable: 4996)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

#define MAX_LINE 256
#define MAX_LINES 200
FILE *fd;
int ile_slow(char *), licz_slowa(char **), licz_slowa_1(char **);
int main(int argc, char **argv)
{
    /* Ile slow w ielu liniach tekstu */
    char *d[MAX_LINES], bufor[MAX_LINE];
    int len, i, l;
    if (argc != 2)
    {
        printf("Zła liczba argumentów\n");
        exit(1);
    }

    if (!(fd = fopen(argv[1], "r")))
    {
        printf("Błąd otwarcia zbioru\n");
        exit(2);
    }
    i = 0;
    l = 0;
    while (i < MAX_LINES && fgets(bufor, MAX_LINE, fd))
    {
        len = strlen(bufor);
        bufor[len - 1] = '\0';
        if (!(d[i] = (char*)malloc((unsigned)len)))
        {
            printf("Brak pamieci\n");
            exit(3);
        }
        strcpy(d[i], bufor);
        i++;
    }
    d[i] = (char *)0; //znacznik końca tablicy wskaźników
    l = licz_slowa_1(d);
    printf("%d\n", l);
    system("pause");

    /* Tekst w pamieci, tablica d - wskazniki do linii tekstu */
}

/* Tu funkcja ile_slow */

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

int licz_slowa_1(char *te[])
{
    int i, l = 0;
    i = 0;
    while (te[i] != (char *)0)
    {
        l += ile_slow(te[i]);
        i++;
    }
}

```

14. Przekazywanie tablic dwuindeksowych sprowadza się do przekazywania jednoindeksowej tablicy wskaźników do jej wierszy , czyli adresu początkowego elementu tablicy wskaźników. Poniższa funkcja `licz_slowa` jest modyfikacją funkcji `licz_slowa_1` . Deklaracje identyfikatora `te` w obu wersjach funkcji są w pełni równoważne. Bez względu na sposób deklarowania można odwoływać się do tekstów przez elementy tablic lub przez wskazanie pośrednie.

```
int licz_slowa(char **te)
{
    int l = 0;
    while (*te)
    {
        l += ile_slow(*te++);
    }
    return l;
}
```

15. Zamień miejscami tekst z dwóch wybranych linii.
16. Wybierz z tekstu składającego się z wielu linii te linie, w których na początku znajduje się wybrany tekst.
17. Wstaw jedną linię tekstu po linii o numerze n.
18. Wybierz z tekstu słowo o podanej pozycji. Załóż, że ogranicznikiem słowa jest dwukropek.
19. Napisz program kodujący i dekodujący dowolne teksty posługując się alfabetem Morse'a. W rozwiązaniu możesz wykorzystać poniższe tablice.

[illegible]

20. Przeanalizuj w trybie pracy krokowej następujące przykłady do wykładu 6:

http://torus.uck.pk.edu.pl/~fialko/text/CC/przykl/WW6_1.pdf

http://torus.uck.pk.edu.pl/~fialko/text/CC/przykl/WW6_2.pdf

http://torus.uck.pk.edu.pl/~fialko/text/CC/przykl/WW6_3.pdf

Wszystkie przykłady dostępne pod adresem:

<http://torus.ucl.ac.uk/~fialko/text/CC/przykl/>

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