Laboratorium 4: „Typy wyliczeniowe i tablice”

mgr inż. Leszek Ciopiński
dr inż. Arkadiusz Chrobot
dr inż. Grzegorz Łukawski

3 listopada 2025

1. Wprowadzenie

Pierwsza część instrukcji zawiera informacje o implementacji typów wyliczeniowych i języku C, natomiast druga część poświęcona jest sposobom tworzenia i korzystania z tablic w tym języku.

2. Tablice

Tablice należą do podstawowych struktur danych, dlatego są dostępne w większości współczesnych języków programowania. Język C nie jest tutaj wyjątkiem. W tym rozdziale zostaną opisane zasady posługiwania się tablicami w tym języku.

2.1. Tworzenie tablic

Tablicę w języku C deklarujemy podając najpierw typ jej elementów, potem jej nazwę, a na końcu, w nawiasach kwadratowych umieszczamy liczbę jej elementów. Deklaracja tablicy kończy się znakiem średnika. Istnieje także możliwość stworzenia tablicy zainicjowanej. W jej deklaracji nawiasy kwadratowe możemy pozostawić puste. Po nich umieszczamy instrukcję przypisania i w nawiasach klamrowych wymieniamy wartości elementów, które rozdzielamy przecinkami. Tablica będzie miała tyle elementów, ile podamy wartości. Przykłady deklaracji tablic podano w listingu 1. Tablice mogą być tworzone zarówno jako zmienne lokalne, jak i globalne.

```c
#define N 10    // Definicja stałej o nazwie N i wartości 10.

int t[N];       // Tablica o dziesięciu elementach typu int.

int tablica[3]; // Tablica o trzech elementach typu int.

double ulamki[] = {0.1, 0.2, 0.3}; // Tablica zainicjowana o trzech elementach
                                   // typu double.

double ulamki2[3] = {0.1, 0.2, 0.3}; // Jak wyżej, ale tym razem podajemy liczbę
                                     // elementów należących do tablicy.
```

Listing 1: Deklaracje tablic

2.2. Dostęp do elementów tablicy

W języku C elementy tablicy indeksowane są zawsze od zera, a indeks ostatniego elementu jest równy liczbie elementów tablicy pomniejszonej o jeden. Należy wiedzieć, że kompilator **nie sprawdza** zakresu wartości indeksów. W języku C nazwa tablicy może być traktowana jako wskaźnik. Dzięki temu dostęp do określonego elementu tablicy może zostać uzyskany dzięki tzw. arytmetyce wskaźników, która w tym wypadku polega na dodaniu do nazwy tablicy indeksu i wykonaniu dereferencji tak uzyskanego wskaźnika. Listing 2 zawiera kilka przykładów.

```c
#define N 10

int a[N];

int i;

for(i=0;i<N;i++)
        scanf("%d",&a[i]);

*(a+3) = 4;

printf("%d\n",*(a+4));
```

Listing 2: Sposoby dostępu do elementów tablicy

2.3. Parametry tablic

Wielkość tablicy, czyli liczbę bajtów zajmowanych przez nią w pamięci możemy wyznaczyć za pomocą operatora `sizeof`. Liczbę elementów tablicy możemy obliczyć dzieląc jej rozmiar przez wielkość pierwszego elementu (jego indeks zawsze wynosi zero). Listing 3 zawiera odpowiednie przykłady. Należy pamiętać, że w przypadku funkcji, które będą opisane w kolejnych instrukcjach, te rozwiązania nie zawsze działają poprawnie.

```c
int a[10];
int b = sizeof(a); // lub 10*sizeof(int);

int c = sizeof(a)/sizeof(a[0]); // Obliczenie liczby elementów tablicy.
```

Listing 3: Wyznaczanie rozmiaru tablic

3. Tablica jako parametr funkcji

W języku C możliwe jest też przekazywanie tablicy jako parametr funkcji. Ponieważ nazwa tablicy jest wskaźnikiem do miejsca, gdzie zaczyna się tablica w pamięci, dlatego możliwych jest kilka sposobów deklaracji tablicy jako parametru. Zostaną one opisane na podstawie Listingu 4.

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

typedef int myTab[5];

void setTab(myTab t)
{
    int i;
    for(i=0; i<5; i++)
        t[i] = i * 3;
}

void printTab(int t[])
{
    int i;
    for(i=0; i<5; i++)
        printf("%d\t", t[i]);
    printf("\n");
}

void printCharTab(char *t)
{
    int i;
    for(i=0; i<sizeof(myTab); i++)
        printf("%d\t", t[i]);
    printf("\n");
}

int main(void)
{
    myTab tablica, tablica2;
    setTab(tablica);
    printTab(tablica);                //Wyświetli:  0     3     6     9     12
    printf("\n");
    memset(tablica2, 7, sizeof(myTab));
    printTab(tablica2);
    printCharTab((char*)tablica2);
    memcpy(tablica2, tablica, sizeof(myTab));
    printTab(tablica2);               //Wyświetli:  0     3     6     9     12
    return 0;
}

```

Listing 4: Różne sposoby użycia tablicy jako parametru funkcji.

3.1. Typedef - nowa nazwa typu

Słowo kluczowe `typedef` służy do definiowania nowej nazwy typu. Nie oznacza to, że tworzony jest całkowicie nowy typ danych. Umożliwia to jednak nadawanie bardziej złożonym strukturom prostszych nazw, co poprawia czytelność kodu i nie wymusza kilkukrotnego podawania długich definicji. Ponadto, jak pokazuje wiersz 4, możliwe jest nadanie nowej nazwy tablicy. Dzięki temu, wszystkie tablice typu `myTab` będą miały ten sam typ bazowy i ten sam rozmiar.

3.2. Deklaracja tablicy jako parametru

Ponieważ nazwa tablicy jest jednocześnie wskaźnikiem miejsca w pamięci, w którym zaczyna się blok danych przechowywanych w danej tablicy, jako parametr funkcji podaje się właśnie nazwę tablicy. Ma to też kolejne konsekwencje. Deklarację tablicy jako parametr można zrealizować na kilka sposobów. W wierszu 6 zaprezentowano wykorzystanie dodatkowej nazwy typu do zaznaczenia, jaki dokładnie rodzaj tablicy będzie przekazywany. Możliwe jest jednak bardziej ogólne określenie. W wierszu 13 zaprezentowano deklarację tablicy. Rozmiar takiej tablicy nie jest jednak automatycznie przekazywany, dlatego przydatne może być przekazanie informacji o rozmiarze tablicy, jako dodatkowy parametr funkcji. Ostatni przykład przedstawiono w wierszu 21. Tablica przekazana jest tu jako zwykły wskaźnik. W niniejszym przykładzie właściwość ta została wykorzystana, aby umożliwić odczytanie danych jako wartości typu `char`, a nie `int`. Należy jednak tego wariantu używać ostrożnie, ponieważ kompilator zaakceptuje przekazanie do takiej funkcji każdego wskaźnika, niekoniecznie tablicy. Ewentualne błędy niewłaściwego użycia mogą się objawiać dopiero w trakcie działania programu.

Inną konsekwencją przekazania tablicy poprzez wskaźnik jest trwałość zmian w niej wykonanych. Tak jak w przypadku typów prostych, przekazanych do funkcji poprzez wskaźniki, tak samo tu, tworzona jest przez funkcję jedynie kopia wskaźnika, a nie wartości, na którą wskazuje. Dlatego jeżeli zmiany wykonane w funkcji miałyby mieć charakter tymczasowy, konieczne jest wykonanie kopii takiej tablicy.

3.3. memset()

```c
#include <string.h>

void *memset(void *s, int c, size_t n);
```

Listing 5: Prototyp `memset()`

Funkcja `memset()` służy do wypełniania tablicy takimi samymi wartościami. Jako parametr `s` przekazywana jest tablica. Jako parametr `c` podawana jest wartość, która ma być wpisana do każdego bajtu pamięci zajmowanego przez tablicę. Należy zauważyć, że chociaż typ parametru `c` to `int`, to wykorzystywane jest tylko 8 najmniej znaczących bitów tej zmiennej. Dlatego, jako ostatni parametr funkcji - `n` należy podać rozmiar tablicy w bajtach, a nie w liczbie elementów.

Dlatego w wyniku wykonania wiersza 36. wartość 7 będzie przypisana do każdego bajtu tablicy, a nie do każdego elementu. W efekcie, funkcja z wiersza 37. wyświetli:

```
117901063 117901063 117901063 117901063 117901063
```

Dopiero wyświetlenie każdego bajtu tablicy z osobna da spodziewany efekt:

```
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
```

Wartością zwracaną przez funkcję `memset()` jest wskaźnik na `s`.

3.4. memcpy()

```c
#include <string.h>

void *memcpy(void *dest, const void *src, size_t n);
```

Listing 6: Prototyp `memcpy()`

Funkcja `memcpy` służy do kopiowania `n` bajtów danych z przestrzeni pamięci określonej jako `src` i zapisywania ich w obszarze pamięci zaczynającym się w miejscu wskazanym przez parametr `dest`. Przestrzenie obydwu pamięci muszą być ciągłe i nie mogą się wzajemnie nachodzić na siebie. Przykładem takich obszarów pamięci są dwie zmienne tablicowe. Dlatego w wierszu 38. zaprezentowano sposób kopiowania jednej tablicy do drugiej.

Wartością zwracaną przez funkcję `memcpy()` jest wskaźnik na `dest`.

4. Typ wyliczeniowy

Typ wyliczeniowy jest używany do stworzenia zmiennych, które posiadają z góry określone wartości. Do zadeklarowania typu wyliczeniowego używamy słowa kluczowego `enum`.

```c
enum Nazwa_Typu{Wartosc1, Wartosc2, Wartosc3};
```

Listing 7: Deklaracja typu wyliczeniowego

Najprostszym przykładem użycia typu wyliczeniowego jest określenie kierunków:

```c
enum Kierunek{GORA, DOL, PRAWO, LEWO};
enum Kierunek krok = DOL;
```

Listing 8: Deklaracja typu wyliczeniowego

W zaprezentowanym przypadku `Kierunek` jest nazwą typu wyliczeniowego. Przy jego użyciu można utworzyć zmienną, której przypisana zostanie jedna z zadeklarowanych wartości. Taka deklaracja może być przydatna do czytelniejszej organizacji kodu, np. w połączeniu z instrukcją `switch`.

```c
switch(ruch) {
   case GORA:
     printf("Góra\n");
     break;
   case DOL:
     printf("Dół\n");
     break;
   default:
     printf("Prawo/Lewo\n");
 }
```

Listing 9: Przykład użycia typu wyliczeniowego.

Nie jest to wymóg kompilatora, ale przyjęto konwencję, według której wartości stałe zapisuje się wielkimi literami. Implementacja typu wyliczeniowego w języku C została oparta o liczby całkowite w zakresie typu `signed int`. Domyślnie, wartościom typu wyliczeniowego nadawane są kolejne liczby całkowite. Możliwe jest jednak zdefiniowanie własnych wartości, jakie mają być im przypisane. Przykłady takiego przyporządkowania przedstawiono na Listingu 10.

```c
#include <stdio.h>

enum Liczby{ZERO, JEDEN, DWA, TRZY=2, CZTERY, PIEC=-2, SZESC, SIEDEM, OSIEM};

int main(void)
{
        printf("%d\n", ZERO);   //Wyświetli: 0
        printf("%d\n", JEDEN);  //Wyświetli: 1
        printf("%d\n", DWA);    //Wyświetli: 2
        printf("%d\n", TRZY);   //Wyświetli: 2
        printf("%d\n", CZTERY); //Wyświetli: 3
        printf("%d\n", PIEC);   //Wyświetli: -2
        printf("%d\n", SZESC);  //Wyświetli: -1
        printf("%d\n", SIEDEM); //Wyświetli: 0
        printf("%d\n", OSIEM);  //Wyświetli: 1

        return 0;
}
```

Listing 10: Różne wartości typu wyliczeniowego.

Warto zauważyć, że kolejne wartości typu wyliczeniowego domyślnie uzyskują wartości liczone od zera. Nie jest jednak problemem, gdy dwie wartości w typie wyliczeniowym uzyskają taką samą wartość liczbową (np. `DWA` i `TRZY`). Przypisanie wartości ujemnej również nie generuje błędów. Daje to duże możliwości programiście, ale wymusza też zachowanie pewnej ostrożności. W przedstawionym przykładzie dla części wartości przypisano takie same liczby. Nie będzie to błędem kompilacji, jednak zachowanie programu może nie być zgodne z oczekiwaniami. Np. gdyby w przykładzie opisującym kierunki, dwie wartości miałyby przypisaną taką samą liczbę, to nie możliwy byłby wybór pomiędzy nimi. Możliwe jest też przypisanie do typu wyliczeniowego wartości liczbowej z zakresu typu `signed int`, która nie ma swojego odpowiednika w typie wyliczeniowym (np. jest poza zakresem). Dlatego dobrą praktyką jest

obsługiwanie typu wyliczeniowego tylko przez zdefiniowane w nim wartości.

5. Liczby pseudolosowe

Jeśli w naszym programie chcemy użyć wartości pseudolosowych, to możemy je otrzymać używając funkcji `srand`() i `rand`(), dostępnych po włączeniu pliku nagłówkowego `stdlib.h`. Pierwsza z nich inicjuje generator liczb pseudolosowych wartością swojego argumentu, czyli ustala pierwszą wartość ciągu pseudolosowego. Zazwyczaj jej argumentem jest liczba zwrócona przez funkcję `time`(). Ta funkcja jest dostępna po włączeniu pliku nagłówkowego `time.h` i również wymaga argumentu, którym może być liczba 0. Funkcja `rand`() zwraca liczby naturalne z przedziału [0, RAND_MAX], gdzie RAND_MAX jest stałą, której wartość zależy od kompilatora i komputera. Zazwyczaj jest to odpowiednio duża liczba. Jeśli chcemy uzyskać liczby mniejsze, to wynik funkcji `rand`() dzielimy z użyciem operatora modulo. Listing 11 demonstruje użycie generatora liczb pseudolosowych. Losowane wartości są liczbami naturalnymi z przedziału [0,9]. Należy zauważyć, że użycie `srand`() i `rand`() nie jest jedynym sposobem na uzyskanie pseudolosowych wartości w języku C. Ponadto, jeśli potrzebne są wartości prawdziwie losowe, np. do zastosowań kryptograficznych, to nie wolno z nich korzystać, tylko należy użyć innych metod.

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main(void)
{
        int a;
        srand(time(0));
        do {
                a=rand()%10;
                printf("%d\n",a);
        } while(a!=5);
        return 0;
}
```

Listing 11: Przykład użycia generatora liczb pseudolosowych

6. Zadania

WSZYSTKIE PROGRAMY NALEŻY NAPISAĆ Z PODZIAŁEM NA FUNKCJE Z PARAMETRAMI!

1. Prześledź działanie programu z listingu 11 przy pomocy debuggera. Sprawdź jak zmienia się wartość zmiennej `a`.
2. Napisz podprogram (funkcję), w której zadeklarujesz typ wyliczeniowy z kilkoma wartościami, przy czym co najmniej 3 będą miały wartości domyślne, kolejne dwie będą miały takie same wartości, a kilka ostatnich będzie miało dowolne wartości zadeklarowane, inne od poprzednich. Wyświetl wartości przyporządkowane tym elementom na ekranie.
3. Napisz program, który zapełni tablicę o $N = 30$ elementach liczbami całkowitymi wylosowanymi z zakresu od -100 do 100, a następnie policzy średnią ($\bar{x}$) z tych liczb. Zawartość tablicy i średnią należy wyświetlić na ekranie.
4. Dopisz do programu z poprzedniego zadania funkcję, tak, aby liczył on odchylenie standardowe (σ), według wzoru:

$$\sqrt{\frac{\sum_{i=0}^{N-1}(x_i - \bar{x})^2}{N}}$$

, gdzie i jest indeksem elementu tablicy, a x jest tablicą.

5. Dopisz do programu z poprzedniego zadania funkcje (lub funkcję), które znajdą wartość maksymalną i minimalną w tej tablicy.

6. Dopisz do programu z poprzedniego zadania funkcję (lub funkcje), która wypisze na ekran wszystkie wartości z tablicy, które mają wartość mniejszą od różnicy $\bar{x} - \sigma$ i większą od sumy $\bar{x} + \sigma$.

7. Napisz program, w którym zadeklarujesz tablicę o 4 elementach typu `int`, a którą wypełni użytkownik. Program powinien potraktować te wartości jako współczynniki wielomianu trzeciego stopnia i policzyć jego wartość, dla zadanej przez użytkownika wartości `x`. Do wyliczania tej wartości użyj reguły Hornera:
$a_0 \cdot x^n + a_1 \cdot x^{n-1} + \ldots + a_{n-1} \cdot x + a_n = (\ldots((a_0 \cdot x + a_1) \cdot x + a_2) \cdot x + \ldots a_{n-1}) \cdot x + a_n$. Zawartość tablicy oraz wynik działania należy wypisać na ekranie.

8. Napisz program, który wypełni tablicę o 5 elementach typu `int` wartościami pobranymi od użytkownika, wyświetli jej zawartość na ekran, a następnie zamieni kolejność tych liczby w tablicy i z powrotem wypisze jej zawartość na ekranie.

9. Napisz program, który wypełni tablicę o 10 elementach liczbami całkowitymi losowanymi z zakresu 0 do 20, a następnie zamieni miejscami „połówki" tej tablicy. Przyład dla tablicy o 4 elementach: $-4, 3, 2, 1 \rightarrow 2, 1, -4, 3$. Zawartość tablicy należy wypisać przed i po wykonaniu zamiany.