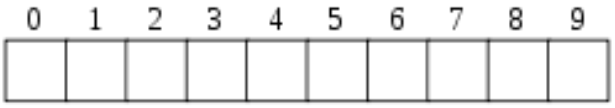
W rozdziale Zmienne w C dowiedziałeś się, jak przechowywać pojedyncze liczby oraz znaki. Czasami zdarza się jednak, że potrzebujemy przechować kilka, kilkanaście albo i więcej zmiennych jednego typu. Nie tworzymy wtedy np. dwudziestu osobnych zmiennych. W takich przypadkach z pomocą przychodzi nam tablica.



Rysunek 1: tablica 10-elementow

Tablica to ciąg zmiennych jednego typu. Ciąg taki posiada jedną nazwę a do jego poszczególnych elementów odnosimy się przez numer (indeks).

Wstęp

Sposoby deklaracji tablic

Tablicę deklaruje się w następujący sposób:

```
typ nazwa_tablicy[rozmiar];
```

Listing 1: Deklaracja

gdzie rozmiar oznacza ile zmiennych danego typu możemy zmieścić w tablicy. Zatem aby np. zadeklarować tablicę, mieszczącą 20 liczb całkowitych możemy napisać tak:

```
int tablica[20];
```

Listing 2: Deklaracja tablicy 20 int

Podobnie jak przy deklaracji zmiennych, także tablicy możemy nadać wartości początkowe przy jej deklaracji. Odbywa się to przez umieszczenie wartości kolejnych elementów oddzielonych przecinkami wewnątrz nawiasów klamrowych:

```
int tablica[3] = {0,1,2};
```

Listing 3: Inicjalizacja

Niekoniecznie trzeba podawać rozmiar tablicy, np.:

```
int tablica[] = {1, 2, 3, 4, 5};
```

Listing 4: Inicjalizacja bez rozmiaru

W takim przypadku kompilator sam ustali rozmiar tablicy (w tym przypadku - 5 elementów).

Rozpatrzmy następujący kod:

```
#include <stdio.h>
#define ROZMIAR 3
int main()
{
  int tab[ROZMIAR] = {3,6,8};
  int i;
  puts ("Druk tablicy tab:");

  for (i=0; i<ROZMIAR; ++i) {
    printf ("Element numer %d = %d\n", i, tab[i]);
  }
  return 0;
```

```
}
```

Listing 5: Printowanie tablicy

Wynik:

```
Druk tablicy tab:
Element numer 0 = 3
Element numer 1 = 6
Element numer 2 = 8
```

Listing 6: Wyniki

Jak widać, wszystko się zgadza.

W powyżej zamieszczonym przykładzie użyliśmy stałej do podania rozmiaru tablicy. Jest to o tyle pożądany zwyczaj, że w razie potrzeby zmiany rozmiaru tablicy, zmieniana jest tylko wartość w jednej linijce kodu przy #define, w innym przypadku musielibyśmy szukać wszystkich wystąpień rozmiaru rozsianych po kodzie całego programu.

Odczyt/zapis wartości do tablicy

Tablicami posługujemy się tak samo jak zwykłymi zmiennymi. Różnica polega jedynie na podawaniu indeksu tablicy. Określa on, z którego elementu (wartości) chcemy skorzystać spośród wszystkich umieszczonych w tablicy. Numeracja indeksów rozpoczyna się od zera, co oznacza, że pierwszy element tablicy ma indeks równy 0, drugi 1, trzeci 2, itd.

Spróbujmy przedstawić to na działającym przykładzie. Przeanalizuj następujący kod:

```
int tablica[5] = {0};
int i = 0;
tablica[2] = 3;
tablica[3] = 7;
for (i=0;i!=5;++i) {
  printf ("tablica[%d]=%d\n", i, tablica[i]);
}
```

Listing 7: Fragment

Jak widać, na początku deklarujemy 5-elementową tablicę, którą od razu zerujemy. Następnie pod trzeci i czwarty element (liczone począwszy od 0) podstawiamy liczby 3 i 7. Pętla ma za zadanie wyprowadzić wynik naszych działań.

Tablice znaków

Tablice znaków, tj. typu char oraz unsigned char, posiadają dwie ogólnie przyjęte nazwy, zależnie od ich przeznaczenia:

- bufory - gdy wykorzystujemy je do przechowywania ogólnie pojętych danych, gdy traktujemy je jako po prostu ćiągi bajtów"(typ char ma rozmiar 1 bajta, więc jest elastyczny do przechowywania np. danych wczytanych z pliku przed ich przetworzeniem)
- napisy - gdy zawarte w nich dane traktujemy jako ciągi liter; jest im poświęcony osobny rozdział Napisy.

Przykład:

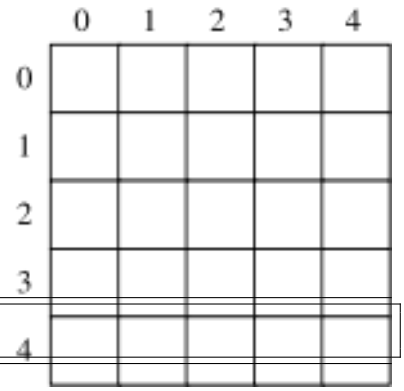
```
 /*
 http://joequery.me/code/snprintf-c/


  gcc a.c -Wall
 ./a.out

012345678
hello th\0
turtle\078
2222\05678

*/
#include<stdio.h>
#define BUFSIZE 9




void init_buf(char *buf, size_t size){
    int i;
    for(i=0; i<size; i++){
        buf[i] = i + '0'; // int to char conversion
    }
}

void print_buf(char *buf){
    int i;
    char c;
    for(i=0; i<BUFSIZE; i++){
        c = buf[i];
        if(c == '\0'){
            printf("\\0");

        }
        else{
            printf("%c", buf[i]);
        }
    }
    printf("\n");
}


int main(){
    char buf[BUFSIZE];
    init_buf(buf, BUFSIZE);
    print_buf(buf);

    // hello there! == 12 characters, > BUFSIZE
    init_buf(buf, BUFSIZE);
    snprintf(buf, BUFSIZE, "hello there!");
    print_buf(buf);
```

```
    // turtle == 6 charaters, < BUFSIZE
    init_buf(buf, BUFSIZE);
    snprintf(buf, BUFSIZE, "turtle");
    print_buf(buf);

    // 2222220 == 7 charaters, > 5
    init_buf(buf, BUFSIZE);
    snprintf(buf, 5, "%d", 222222 * 10);
    print_buf(buf);

    return 0;
}
```

Listing 8: Fragment kodu

Tablice wielowymiarowe

Rozważmy teraz konieczność przechowania w pamięci komputera całej macierzy o wymiarach 10 x 10. Można by tego dokonać tworząc 10 osobnych tablic jednowymiarowych, reprezentujących poszczególne wiersze macierzy. Jednak język C dostarcza nam dużo wygodniejszej metody, która w dodatku jest bardzo łatwa w użyciu. Są to tablice wielowymiarowe, lub inaczej "tablice tablic". Tablice wielowymiarowe definiujemy podając przy zmiennej kilka wymiarów, np.:



Rysunek 2: tablica 10-elementow

```
float macierz[10][10];
```

Listing 9: Macierz

Tak samo wygląda dostęp do poszczególnych elementów tablicy:

```
macierz[2][3] = 1.2;
```

Listing 10: Macierz

Jak widać ten sposób jest dużo wygodniejszy (i zapewne dużo bardziej ńaturalny") niż deklarowanie 10 osobnych tablic jednowymiarowych. Aby zainicjować tablicę wielowymiarową należy zastosować zagłębianie klamr, np.:

```
float macierz[3][4] = {
  { 1.6, 4.5, 2.4, 5.6 },  /* pierwszy wiersz */
  { 5.7, 4.3, 3.6, 4.3 },  /* drugi wiersz */
  { 8.8, 7.5, 4.3, 8.6 }   /* trzeci wiersz */
};
```

Listing 11: Macierz

Dodatkowo, pierwszego wymiaru nie musimy określać (podobnie jak dla tablic jednowymiarowych) i wówczas kompilator sam ustali odpowiednią wielkość, np.:

```
float macierz[][4] = {
  { 1.6, 4.5, 2.4, 5.6 },  /* pierwszy wiersz */
```

```
   { 5.7, 4.3, 3.6, 4.3 },  /* drugi wiersz */
   { 8.8, 7.5, 4.3, 8.6 },  /* trzeci wiersz */
   { 6.3, 2.7, 5.7, 2.7 }  /* czwarty wiersz */
 };
```

Listing 12: Macierz

Innym, bardziej elastycznym sposobem deklarowania tablic wielowymiarowych, jest użycie wskaźników. Opisane to zostało w następnym rozdziale.

Kolejność głównych wierszy

Kolejność głównych wierszy (ang. Row Major Order = ROM [1])

W C tablica wielowymiarowa A[n][m] :

- jest przechowywana wierszami
- numeracja indeksów rozpoczyna się od zer

```
A[0][0], A[0][1], ..., A[0][m-1], A[1][0], A[1][1],..., A[n-1][m-1]
```

Listing 13: Tablica

Przykładowy program :

```
/*
 http://stackoverflow.com/questions/2151084/map-a-2d-
 array-onto-a-1d-array-c/2151113

*/
  #include <stdio.h>

   int main(int argc, char **argv) {
   int i, j, k;
   int arr[5][3];
   int *arr2 = (int*)arr;

       for (k=0; k<15; k++) {
          arr2[k] = k;
          printf("arr[%d] = %2d\n", k, arr2[k]);
       }

       for (i=0; i<5; i++) {
         for (j=0; j< 3; j++) {
            printf("arr2[%d][%d] = %2d\n", i, j ,arr[i][j]);
         }
       }
    }
```

Listing 14: Kod

Ograniczenia tablic

Pomimo swej wygody tablice statyczne mają ograniczony, z góry zdefiniowany rozmiar, którego nie można zmienić w trakcie działania programu. Dlatego też w niektórych zastosowaniach tablice statyczne zostały wyparte tablicami dynamicznymi, których rozmiar może być określony w trakcie działania programu. Zagadnienie to zostało opisane w następnym rozdziale.

Wystarczy pomylić się o jedno miejsce (tzw. błąd off by one) by spowodować, że działanie programu zostanie nagle przerwane przez system operacyjny (błąd przy uruchamianiu) :

```
int foo[100];
int i;

for (i=0; i<=100; i+=1) /* powinno być i<100 */
  foo[i] = 0;

/* program powinien zakończyć się błędem */
```

Listing 15: Bład