Język C – zajęcia nr 6

Typy pochodne

1. Tablice

Tablica to **ciąg elementów jednakowego typu**. Jeżeli **`t`** jest tablicą, a **`n`** ma wartość typu całkowitego, to **`t[n]`** jest elementem numer **`n`** tej tablicy. Generalnie elementy wszystkich tablic są numerowane od **zera**. Początkowym elementem tablicy **`t`** jest **`t[0]`**. W deklaracji tablicy podawany jest jej rozmiar w formie **stałego** wyrażenia o wartości znanej w momencie kompilacji.

```
int t[20];   // Definicja 20-elementowej tablicy typu int
             // o elementach t[0], t[1], ... t[19]
```

UWAGA. Nazwa tablicy jest równocześnie stałym **wskaźnikiem** na jej początkowy element.

Inicjalizacja tablic

Do tablic stosowana jest inicjalizacja zbiorcza **wszystkich elementów**, przy czym inicjalizator ma postać**:**

{ lista wyrażeń }

W przypadku listy krótszej niż rozmiar tablicy reszta elementów tablicy jest inicjalizowana wartością **zero**. Tablica może mieć postać tablicy **otwartej** (bez podanego rozmiaru), jeżeli jej rozmiar wynika z postaci inicjalizatora.

```
int p[4]={6,8,3,5};  // p[0]=6, p[1]=8, p[2]=3, p[3]=5
int q[4]={6,8};      // q[0]=6, q[1]=8, q[2]=0, q[3]=0
int s[]={6,8};    // s[0]=6, s[1]=8, Tablica 2-elementowa
```

Inicjalizacja tablic znakowych. Wyjątkowo w przypadku tablic typu **char** inicjalizator może mieć formę stałej łańcuchowej.

```
char s1[]={'K','o','t','e','k'};  // 5 elementów
char s2[]="Kotek";                // 6 elementów, s2[5]=0
char s3[4]={'P','i','e','s'};     // 4 elementy
char s4[4]="Pies";  // Błąd, inicjalizator ma 5 elementów
char s5[1]={'A'};   // Tablica jednoelementowa
char s6[1]={"A"};   // Błąd, inicjalizator dwuelementowy
char s7[]="";       // Tablica jednoelementowa, s7[0]=0
```

Uwaga: Niektóre kompilatory nie sygnalizują błędu w przypadku inicjalizowania tablicy o długości równej liczbie niezerowych znaków łańcucha inicjalizującego (nie umieszczają w tablicy końcowego znaku NULL). W przypadku inicjalizowania tablicy **otwartej** znak NULL kończący łańcuch znakowy jest wliczany do jego długości.

W przypadku, gdy zmienna (także element tablicy) nie jest inicjalizowana w sposób jawny, otrzymuje wartość początkową **zero**. Wyjątek stanowią zmienne **automatyczne**, które otrzymują **nieokreślone** wartości początkowe.

Dostęp do elementów tablicy umożliwia **operator indeksowania**. Wartością wyrażenia:

tab [k]

jest wartość elementu o numerze **k** tablicy **tab**.

Tablice wielowymiarowe

Język C umożliwia posługiwanie się tablicami wielowymiarowymi.

Tablica dwuwymiarowa jest w istocie tablicą jednowymiarową, której elementami są tablice. Przykład deklaracji:

```
int tab[2][12];
```

Elementy tablicy dwuwymiarowej są umieszczane w pamięci wierszami, tzn. jeśli elementy takiej tablicy są obsługiwane w kolejności ich położenia w pamięci to skrajnie prawy indeks zmienia się najszybciej.

Przykład inicjalizacji:

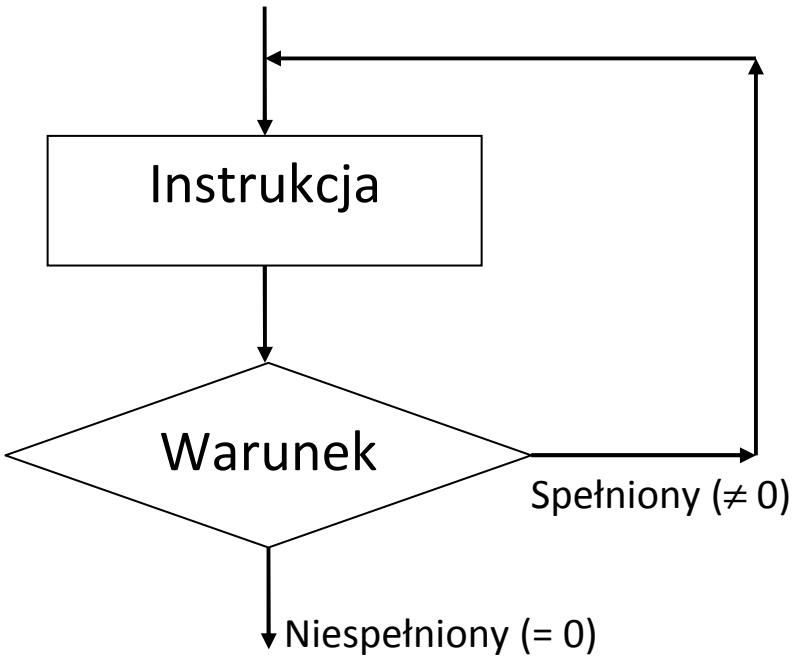
```
int tab[2][7] =
{
     {1, 3, 5, 7, 9, 11, 13},
     {2, 4, 6, 8, 10, 12, 14}
};
```

Dostęp do elementów: **tab [i] [j]**

Pozostałe typy pochodne zostaną omówione w dalszej części kursu.

Instrukcje języka C – ciąg dalszy

Pętla warunkowa ze sprawdzaniem warunku na końcu pętli



Instrukcja iteracji `do-while`

`do` ***inst*** **`while (`*wyr*`);`**

Instrukcja podobna do instrukcji **`while`**, służąca do organizowania pętli. Wykonywane są czynności:

1. Wykonywana jest instrukcja ***inst***.
2. Wyliczana jest wartość wyrażenia ***wyr***. Jeżeli jego wartość jest **równa zero**, to wykonywanie instrukcji **`do-while`** zostaje zakończone.
3. Czynności powtarzane są od kroku 1.

Pętla **`do-while`** różni się od pętli **`while`** tym, że warunek zakończenia w pętli **`while`** sprawdzany jest **przed wykonaniem** instrukcji stanowiącej ciało pętli, a w pętli **`do-while`** **po wykonaniu** ciała pętli. Ciało pętli **`do-while`** wykonywane jest minimum jeden raz, a ciało pętli **`while`** może nie być wykonywane ani razu.

Uwaga: Jeżeli instrukcja ***inst*** jest instrukcją złożoną, to występująca wewnątrz instrukcja **`break`** lub **`return`** może zakończyć wykonywanie instrukcji **`do-while`**.

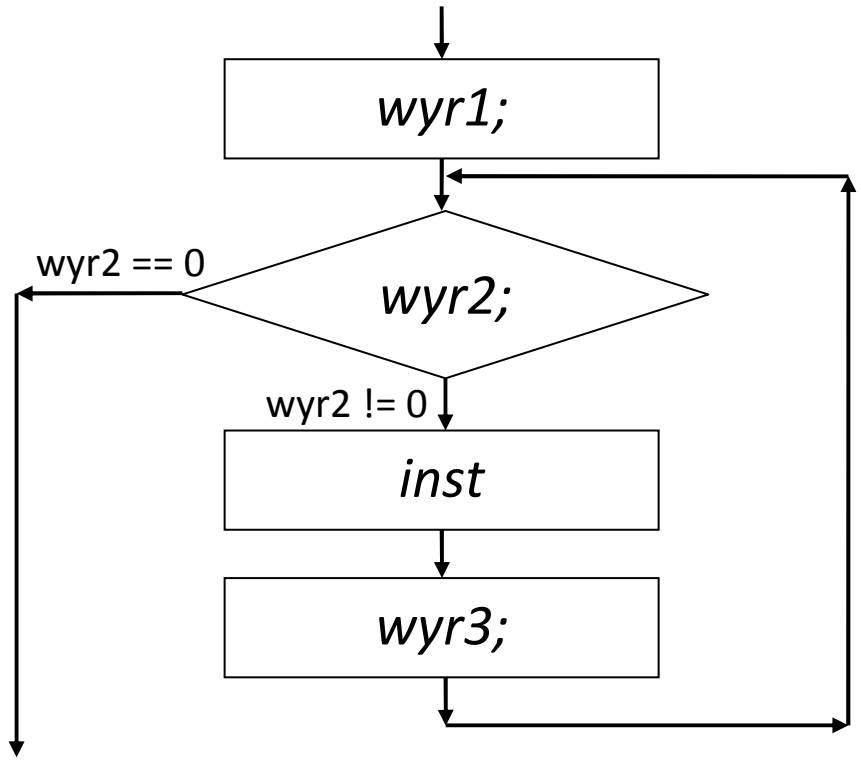
Instrukcja iteracji `for`

`for` **(** ***wyr1*** **;** ***wyr2*** **;** ***wyr3*** **)** ***inst***

Instrukcja **`for`** służy ***najczęściej*** **(choć niekoniecznie)** do realizacji *pętli z licznikiem* (zob. materiały C05). Przeznaczeniem instrukcji for jest organizowanie pętli przez powtarzalne wykonywanie instrukcji ***inst*** w następujący sposób:

1. Wyliczona zostaje wartość wyrażenia ***wyr1***.
2. Wyliczana jest wartość wyrażenia ***wyr2*** i jeżeli wartość ta jest **równa zero**, to wykonywanie instrukcji **`for`** zostaje zakończone. Jeżeli wartość wyrażenia ***wyr2*** jest **różna od zera**, to wykonywana jest instrukcja ***inst***.
3. Wyliczana jest wartość wyrażenia ***wyr3*** i czynności powtarzane są od kroku 2.

Instrukcja **`for`** **(** ***wyr1*** **;** ***wyr2*** **;** ***wyr3*** **)** ***inst*** jest wykonywana wg schematu:



czyli instrukcja **`for`** **(** ***wyr1*** **;** ***wyr2*** **;** ***wyr3*** **)** ***inst*** jest równoważna sekwencji:

```
wyr1;
while (wyr2)
{
      inst
      wyr3;
}
```

Najczęściej (choć niekoniecznie) wyrażenie ***wyr1*** nadaje zmiennej zwanej licznikiem wartość początkową, wyrażenie ***wyr2*** sprawdza czy licznik nie przekroczył wartości końcowej, a wyrażenie ***wyr3*** modyfikuje wartość licznika.

Wprowadź i uruchom program obliczający 6!, zinterpretuj kod źródłowy:

```
#include <stdio.h>
int main()
{
   int s,k;
   s=1;
   for(k=1; k<=6; k++) s=s*k;
   printf("%d",s);
   getch(); return 0;
}
```

```
720
```

W powyższym przykładzie *zmienna* k nazywana jest *licznikiem*. W przypadku pętli z licznikiem instrukcja stanowiąca ciało pętli (prosta lub złożona) wykonywana jest określoną liczbę razy.

Uwaga: Jeżeli instrukcja ***inst*** jest instrukcją złożoną, to występująca wewnątrz instrukcja **`break`** lub **`return`** może zakończyć wykonywanie instrukcji **`for`**.

Uwaga: Każde z wyrażeń ***wyr1***, ***wyr2***, ***wyr3*** można pominąć. Brak wyrażenia ***wyr2*** traktowany jest tak, jakby wartość tego wyrażenia była zawsze różna od zera, co powoduje, że taka instrukcja **`for`** przekształca się w nieskończoną pętlę. Najprostsza nieskończona pętla ma postać:

`for(;;)` ***instr***

Wykonywanie takiej pętli powinno być przerwane poprzez warunkowe użycie instrukcji **`break`** lub **`return`** wewnątrz instrukcji złożonej ***instr***.

Instrukcja ***inst*** występująca w pętli **`for`** może być również instrukcją pętli. Umożliwia to wielokrotne zagnieżdżenie pętli. Na jedno wykonanie instrukcji stanowiącej ciało pętli zewnętrznej składa się wykonanie wszystkich powtórzeń instrukcji będącej ciałem pętli wewnętrznej.

Wprowadź i uruchom program, zinterpretuj kod źródłowy:

Tabliczka mnożenia

```
#include <stdio.h>
int main()
{
   int tab[10][10];
   int i,k;
   for(i=1;i<=9;i++)          //Petla zewnetrzna
      for(k=1;k<=9;k++)       //Petla wewnetrzna
         tab[i][k]=i*k;
   for(i=1;i<=9;i++)
   {
      for(k=1;k<=9;k++)
         printf("%3d",tab[i][k]);
      printf("\n");
   };
   getch(); return 0;
}
```

```
 1  2  3  4  5  6  7  8  9
 2  4  6  8 10 12 14 16 18
 3  6  9 12 15 18 21 24 27
 4  8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
 7 14 21 28 35 42 49 56 63
 8 16 24 32 40 48 56 64 72
 9 18 27 36 45 54 63 72 81
```

Zarówno instrukcja **`while`**, jak i **`do-while`** może zostać użyta do konstruowania pętli zagnieżdżonych podobnie jak w przypadku instrukcji **`for`**.

UWAGA: Wykonanie instrukcji **`break`** powoduje natychmiastowe przerwanie wykonywania instrukcji **`switch`**, **`for`**, **`while`** lub **`do-while`**, w której ta instrukcja **`break`** występuje.

Jeżeli instrukcja złożona, w której występuje **`break`**, jest zagnieżdżona w innej instrukcji **`switch`**, **`for`**, **`while`** lub **`do-while`**, to przerwanie wykonania dotyczy najbardziej wewnętrznej instrukcji **`switch`**, **`for`**, **`while`** lub **`do-while`**.

Instrukcja `continue`

`continue ;`

Instrukcja może występować wewnątrz pętli **`for`**, **`while`** lub **`do-while`**. Jej wykonanie powoduje zaniechanie wykonywania dalszych instrukcji składających się na instrukcję złożoną stanowiącą ciało pętli i rozpoczęcie następnego obiegu pętli.

```
for(k=0;k<n;k++)
{
    if(tab[k]>tab[k+1])continue;   //Nastepny obieg petli
    x=tab[k];
    tab[k]=tab[k+1];
    tab[k+1]=x;
}
```

Zadania – napisz i uruchom programy:

1. C06-1. Wczytaj z klawiatury ciąg *n* liczb rzeczywistych (zakładamy że *n* jest wcześniej podane przez użytkownika i *n* <=100) , po czym wypisz te liczby w kolumnie w kolejności odwrotnej do wczytywania.
   Wskazówka. Wykorzystaj tablicę. Zastosuj pętle:

```
for (i=0; i<n; i++) …          // przy wczytywaniu liczb
for (i=n-1; i>=0; i--) …       // przy wypisywaniu liczb
```

2. C06-2.
   a) Wczytaj macierz *A* liczb całkowitych (max. dwucyfrowych) o *4* wierszach i *5* kolumnach i wydrukuj ją (zachowując wyrównanie liczb w kolumnach).
   b) Dla każdego elementu macierzy *A* wyznacz sumę jego sąsiednich elementów (istnieją max. 4 elementy sąsiednie: u góry, u dołu, z lewej i z prawej). *Uwaga: dla skrajnych elementów sumowaniu podlegają 3, a niekiedy 2 elementy sąsiednie, nie wolno odwoływać się do nieistniejących elementów tablicy, programista jest odpowiedzialny za sprawdzenie i nie wykraczanie poza tablicę!*
   c) Wydrukować otrzymaną macierz 4 x 5, zawierającą ww. sumy.

   Przykład:

   Macierz oryginalna:

22	*-5*	*10*	*-1*	*56*
4	*7*	*7*	*99*	*1*
5	*-5*	*11*	*75*	*0*
34	*-2*	*8*	*14*	*99*

   Macierz sum sasiednich elementow:

-1	*39*	*1*	*165*	*0*
34	*1*	*127*	*82*	*155*
33	*21*	*85*	*124*	*175*
3	*37*	*23*	*182*	*14*

Kody źródłowe programów C06-1 i C06-2 wraz z oknami zawierającymi przykładowe efekty uruchomienia tych programów, należy zapisać w **jednym** dokumencie (**format Word**) o nazwie **C06.doc** i przesłać za pośrednictwem platformy Moodle w wyznaczonym terminie.