# Język C – zajęcia nr 7

## Instrukcje języka C – ciąg dalszy

### *Instrukcja* `return`

**`return`** ***wyr*** **`;`**

Instrukcja powoduje powrót z aktualnie wykonywanej funkcji ze zwracaną wartością równą wartości wyrażenia ***wyr***. W przypadku funkcji, która nie zwraca wartości (jest typu **`void`** ), instrukcja powrotu ma postać:

**`return ;`**

### *Instrukcja* `goto`

**`goto`** ***nazwa*** **`;`**

Wśród instrukcji zdefiniowanych w języku C jest instrukcja skoku **`goto`**, ale nie zaleca się jej wykorzystywania w programach z wyjątkiem tych kilku przypadków, gdy jej użycie jest naprawdę uzasadnione. Występująca w instrukcji ***nazwa*** musi być nazwą jednej z etykiet występujących w obszarze funkcji, w której użyta została instrukcja **`goto`**.

Wykonanie instrukcji **`goto`** powoduje bezpośrednie przejście do instrukcji opatrzonej etykietą o nazwie podanej w instrukcji skoku.

> **Użycie instrukcji `goto` nigdy nie jest absolutnie konieczne, zawsze można uzyskać taki sam efekt stosując w odpowiedni sposób pozostałe instrukcje języka C.**

Jednym z uzasadnionych zastosowań instrukcji **`goto`** jest spowodowanie wyjścia na poziom zewnętrzny z wnętrza zagnieżdżonej pętli lub instrukcji wyboru.

**Wprowadź i uruchom program, zinterpretuj wszystkie elementy kodu źródłowego:**

```
#include <stdio.h>
int main()
{
   int tab[10][10];
   int i,k;
   for(i=1;i<=9;i++)
   {
      for(k=1;k<=9;k++)
      {
         tab[i][k]=20-i*k;
         printf("%4d",tab[i][k]);
      }
      printf("\n");
   }
   for(i=1;i<=9;i++)
      for(k=1;k<=9;k++)
         if(tab[i][k]==0) goto sukces;
   printf("\nZerowy element nie wystepuje");
   getch(); return 0;
sukces:
   printf("\nZerowy element znaleziono na pozycji (%d,%d)",i,k);
   getch(); return 1;
}
```

```
  19  18  17  16  15  14  13  12  11
  18  16  14  12  10   8   6   4   2
  17  14  11   8   5   2  -1  -4  -7
  16  12   8   4   0  -4  -8 -12 -16
  15  10   5   0  -5 -10 -15 -20 -25
  14   8   2  -4 -10 -16 -22 -28 -34
  13   6  -1  -8 -15 -22 -29 -36 -43
  12   4  -4 -12 -20 -28 -36 -44 -52
  11   2  -7 -16 -25 -34 -43 -52 -61

Zerowy element znaleziono na pozycji (4,5)
```

## Uwagi dotyczące stylu programowania

Program można pisać w sposób mniej lub bardziej porządny i systematyczny. Przejrzyste programy pozwalają na znacznie łatwiejszą ich analizę i ewentualne modyfikacje w przyszłości.

Programując warto stosować następujące zalecenia, które służą zwiększeniu czytelności programu:

- **nazwy poszczególnych obiektów programu powinny oddawać ich znaczenie,**
- **nie należy używać wielu podobnych, niewiele różniących się pomiędzy sobą nazw,**
- **najlepiej jest umieszczać po jednej instrukcji w jednym wierszu tekstu programu,**
- **nie warto poświęcać czytelności programu dla niewielkiego zwiększenia jego efektywności,**
- **nie należy ulepszać programu bez wyraźnej potrzeby,**
- **należy umieszczać w programie konieczne komentarze, a unikać zbędnych,**
- **stosowanie literałów w wyrażeniach powinno być ograniczone do minimum.**

Równie istotne jest przyjęcie ustalonych konwencji edycyjnych tekstu źródłowego programu. Obok merytorycznej zawartości programu ważna jest również jego forma. Dla kompilatora istotna jest jedynie treść analizowanego tekstu programu z punktu widzenia składni języka. Dla programisty może być istotna także redakcja tekstu. Dobrze zredagowany tekst może znacznie zwiększyć czytelność i ułatwić zrozumienie działania programu.

Spory efekt w zakresie zwiększania czytelności programu uzyskać można stosunkowo niewielkim kosztem dobierając i stosując odpowiednie spacjowanie poziome.

**Spacjowanie poziome:**

Odstępy poziome (znaki spacji i tabulacji) są umieszczane w celu rozdzielenia występujących obok siebie identyfikatorów lub słów kluczowych. W pozostałych przypadkach odstępy w tekście służą zwiększeniu wyrazistości notacji algorytmu reprezentowanego przez program.

Przykładowo następująca lista wyrażeń:

```
(a+b*c,d/e-f,g*(h/i+j)-k,m-n)
```

staje się znacznie bardziej czytelna po uzupełnieniu jej spacjami:

```
(a+b*c, d/e-f, g*(h/i+j)-k, m-n)
```

Spacjami można również otaczać niektóre operatory dla uzyskania bardziej przejrzystej struktury złożonych wyrażeń. Wyrażenie:

```
a*b+c*d/e-f/g
```

zyskuje na przejrzystości, gdy występujące w nim operatory sumy i różnicy zostaną odseparowane spacjami:

```
a*b + c*d/e - f/g
```

Dalsze porządkowanie tekstu źródłowego programu uzyskać można stosując właściwe spacjowanie pionowe.

**Spacjowanie pionowe:**

Bardziej złożone instrukcje często zapisywane są w kilku kolejnych liniach tekstu. Bez odpowiedniego podziału na poszczególne linie tekst programu staje się zupełnie nieczytelny.

Spróbujmy ustalić do czego służy następujący program:

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

Nie jest łatwo stwierdzić, jakie obliczenia wykonuje ten program (jeżeli w ogóle jest to coś sensownego). Główną przyczyną jest niewłaściwe **spacjowanie pionowe**. Dokładnie ten sam tekst programu podzielony w sposób przemyślany na poszczególne linie pozwala rozpoznać program wyznaczający tabliczkę mnożenia:

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

Obok właściwego spacjowania tekstu źródłowego programu dużą rolę w jego czytelności odgrywa przyjęcie i konsekwentne stosowanie odpowiednich **wcięć tekstu**.

**Wcięcia tekstu:**

Najczęściej problem wcinania tekstu dotyczy instrukcji złożonych. **Istotne jest w takim przypadku rozmieszczenie logicznych nawiasów `{` oraz `}`** .

Pierwszy sposób poleca umieszczać je w pozycji zgodnej z pozycją instrukcji sterującej wcinając jedynie zawartość nawiasów:

```
k=0;
while (k<n-1)
{
   k=k+1;
   if (tab[k-1]>tab[k])
   {
      x=tab[k-1];
      tab[k-1]=tab[k];
      tab[k]=x;
   }
};
```

Drugi ze sposobów umieszczania logicznych nawiasów polega na wyrównywaniu ich z instrukcjami składającymi się na blok utworzony przez te nawiasy:

```
k=0;
while (k<n-1)
   {
   k=k+1;
   if (tab[k-1]>tab[k])
      {
      x=tab[k-1];
      tab[k-1]=tab[k];
      tab[k]=x;
      }
   };
```

Trzeci sposób jest bardziej zwięzły i zaleca umieszczanie nawiasu **{** bezpośrednio po instrukcji sterującej, a odpowiadającego mu nawiasu **}** w osobnej linii w pozycji początku instrukcji. Podobnie jak wyżej, zawartość bloku jest wcinana:

```
k=0;
while (k<n-1) {
   k=k+1;
   if (tab[k-1]>tab[k]) {
      x=tab[k-1];
      tab[k-1]=tab[k];
      tab[k]=x;
   }
};
```

Programista powinien wybrać jeden z zaprezentowanych sposobów wcinania tekstu lub też być może opracować inny, własny sposób wcinania poszczególnych linii programu i stosować go **konsekwentnie** we wszystkich pisanych przez siebie programach.

**Zadania – napisz i uruchom programy:**

C07-1. Wczytać liczbę naturalną *n* (*n* <= 30000) i wydrukować ją w zapisie o podanej podstawie *p*, gdzie $16 \geq p \geq 2$.

C07-2. Wyszukać i wydrukować kolejne liczby naturalne *n* ($1 \leq n \leq 999$) równe sumie sześcianów swoich cyfr w zapisie dziesiętnym.

**Wskazówki:**

Ad. C07-1

**a)** Po wczytaniu liczby naturalnej *n* i podstawy systemu *p*, obliczać w pętli ilorazy *n*/*p* i reszty z dzielenia *n*%*p*, zapisując te reszty w tablicy typu *int* (reszty to cyfry w zapisie o podstawie *p*); **zobacz też materiały dodatkowe „Zapis liczb";**

b) Niezerowe elementy tablicy drukować od końca, stosując specyfikator **X** (zob. opis funkcji **printf**).

Ad. C07-2

- aby zidentyfikować cyfry danej liczby *n* w zapisie dziesiętnym, należy postępować jak w Ad. C07-1 pkt a) przy *p* = 10 (wystarczy zidentyfikować trzy cyfry);
- każdą ze znalezionych trzech wartości cyfr podnieść do trzeciej potęgi, a następnie zsumować; sumę tą porównać z daną liczbą *n*.

Kody źródłowe programów C07-1 i C07-2 wraz z oknami zawierającymi efekty uruchomienia tych programów, należy zapisać w **jednym** dokumencie (**format Word 2003**) o nazwie **C07.doc** i przesłać za pośrednictwem platformy Moodle w wyznaczonym terminie.