PĘTLE

Pętle *while* **i** *for*

```
while (wyrazenie)
  instrukcja
```

Pętla

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

jest równoważna rozwinięciu

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

Przykład:

```
#include <ctype.h>
/*  atoi:  zamienia  s  na liczbe calkowita  */
int atoi(char s[])
{
  int i,n,sign;
  for (i=; isspace(s[i]); i++)  /*  przeskocz    */
     ;                          /*  biale znaki  */
  sign=(s[i]=='-') ? -1:1;
  if (s[i] == '+' || s[i] == '-') /* przeskocz   */
    i++;                          /* znak liczby */
  for (n=0; isdigit(s[i]); i++)
    n=10*n+(s[i]-'0');
  return sign*n;
}
```

W pętli `for` często stosuje się operator `,` . Przykład:

```
#include <string.h>
/*  funkcja odwraca tekst  s  w miejscu  */
void reverse(char s[])
{
   int c,i,j;
   for (i=0, j=strlen(s)-1; i<j; i++,j--) {
      c=s[i];
      s[i]=s[j];
      s[j]=c;
   }
}
```

Wyrażenia oddzielone przecinkiem oblicza się od lewej do prawej strony, a typem i wartością wyniku jest typ i wartość prawego argumentu.

Pętla *do-while*

```
do
  instrukcja
while (wyrazenie);
```

Najpierw wykonuje się instrukcję, a potem oblicza wyrażenie. Pętla zostanie zatrzymana wtedy, gdy (*wyrażenie*) staje sie fałszywe.

Przykład:

```
#include <stdio.h>
void main()
{
   int i,n,sum=0;
   for (i=0; i<4; i++) {
      printf("Podaj liczbe calkowita: ");
      scanf("%d", &n);
      sum+=n;
   }
   printf("Suma: %d\n",sum);
}
```

Ten sam program z użyciem pętli **while**:

```
#include <stdio.h>
void main()
{
   int i=0,n,sum=0;         /* Nie zapominac o */
   while (i<4) {            /* inicjalizacji   */
      printf("Podaj liczbe calkowita: ");
      scanf("%d", &n);
      sum+=n;          /* ani o inkrementacji  */
      i++;
   }
   printf("Suma: %d\n",sum);
}
```

Ten sam program z użyciem pętli **do–while**:

```
#include <stdio.h>
void main()
{
   int i=0,n,sum=0;      /* Nie zapominac o */
   do {                  /* inicjalizacji   */
      printf("Podaj liczbe calkowita: ");
      scanf("%d", &n);
      sum+=n;
      i++;          /* ani o inkrementacji  */
   } while (i<4);
   printf("Suma: %d\n",sum);
}
```

Instrukcja *switch*

```
switch (wyrazenie) {
  case  wyrazenie-stale: instrukcje
  case  wyrazenie-stale: instrukcje
  default: instrukcje
}
```

Przykład (program zlicza wystąpienia litery l, spacji i pozostałych znaków):

```
#include <stdio.h>
void main()
{  int letterl=0,spaces=0,rest=0;
   while ((c=getch()) != EOF) {
      switch (c) {
         case 'l' : letterl++;
                    break;
         case ' ' : spaces++;
                    break;
         default:   rest++;
                    break;
      }
```

```
   }
   printf("%d  %d  %d\n", letterl, spaces, rest);
}
```

Instrukcje *break* i *continue*

break – powoduje natychmiastowe wyjście z najbardziej zagnieżdżonej pętli lub instrukcji *switch*, w której występuje.

continue – powoduje przerwanie bieżącego kroku pętli i wykonanie od początku następnego kroku. Dla pętli *while* i *do* oznacza to natychmiastowe sprawdzenie warunku zatrzymania, natomiast w pętli *for* powoduje przekazanie sterowania do części przyrostowej.

Przykłady:

```
#include <stdio.h>     /* Liczymy wystapienia*/
char *Ref="H e l l o"; /* spacji */
void main()
{  int Count;
   for (Count=0; *Ref; Ref++) {
      if (*Ref != ' ')
        continue;
      Count++;
   }
   printf("%d\n", Count);
}

-------Drugi przyklad---------------
#define MAXL 1000
#include <stdio.h>
void main()  /* Usun koncowe odstepy i znaki tab */
{  int n;    char line[MAXL];
   while ((n=getline(line, MAXL)>0) {
      while (--n >= 0)
         if (line[n] != ' '  &&
             line[n] != '\t' &&
             line[n] != '\n')
           break;
      line[n+1]='\0';
      printf("%s\n", line);
   }
}
```

Instrukcja **goto** i etykiety

Najczęściej jest stosowana do wyjścia z głęboko zagnieżdżonych pętli, np.

```
for ( ... )
   for ( ... ) {
      ...
      if (failed)
        goto err:  /* skocz do obslugi bledow */
   }
...
err:  /*  popraw lub wypisz komunikat */
```

Wskaźniki i adresy

```
x        -  zmienna
px = &x;  -  przypisanie zmiennej px adresu zmiennej x

px=&x;  y=*px;   jest rownowazne   y=x;

Deklaracje:     int x,y;
                int *px;
                float *pz;

Od wskaznika wymaga sie wskazywania obiektu
okreslonego rodzaju!!
Wyjatek  -  wskaznik do  void !!

Rownowazne zapisy:
*ip+=1;         ++*ip;        (*ip)++;

W trzecim przypadku niezbedne nawiasy, gdyz operacje
okreslone przez jednoargumentowe  *  i  ++  sa
wykonywane od prawej strony do lewej.
```

Wskaźniki i tablice

```
int a[10];
int *pa,x;
pa=&a[0];    x=*pa;    jest rownowazne  x=a[0];

*(pa+1)    odnosi sie do  a[1]
pa+i       adres do a[i]
*(pa+i)    rowne a[i]

nazwa tablicy   ==  wskaznik do zerowego elementu

a == &a[0]    zatem
pa=&a[0];   rownowazne   pa=a;
a[i]  rownowazne  *(a+i)
```

Różnica między nazwą tablicy i zmienną wskaźnikową

```
pa=a;     pa++;    O.K.

a=pa;   a++;   p=&a;     Zle!!  Niedopuszczalne!!

   Jezeli  adresy  p  q  odnosza sie do tej samej
tablicy, to relacje  <, <=, > >=  dzialaja poprawnie.
   Grozne moze byc porownywanie adresow do
roznych tablic!
```

Przykłady:

```
/*  WERSJA  I  */
strlen(char *s)  /*  podaj dlugosc tekstu  */
{
   int n;
   for (n=0; *s != '\0'; s++)
      n++;
   return n;
}


/*  WERSJA  II  */
strlen(char *s)  /*  podaj dlugosc tekstu  */
{
   char *p=s;
   while (*p != '\0')
      p++;
   return (p-s);
}


/*  WERSJA  III  */
strlen(char *s)  /*  podaj dlugosc tekstu  */
{
   char *p=s;
   while (*p)
      p++;
   return (p-s);
}
```

```
/*  WERSJA  IV  */
strlen(char *s)  /*  podaj dlugosc tekstu  */
{
   char *p=s;
   while (*p++)
      ;
   return (p-s);
}
```

Arytmetyka na adresach

Typowe operacje:

```
p++;   - przesuniecie do nastepnego elementu
p+=i;  - przesuniecie do elementu oddalonego
         o  i  pozycji od aktualnego
```

Poprawnymi operacjami wskaźnikowymi są:

- przypisanie wskaźników do obiektów tego samego typu
- dodawanie lub odejmowanie wskaźnika i liczby całkowitej
- odejmowanie bądź porównywanie dwóch wskaźników do elementów tej samej tablicy
- przypisanie wskaźnikowi wartości zero lub przyrównanie wskaźnika do zera

Nie wolno:

- dodawać do siebie dwóch wskaźników ani ich mnożyć, dzielić, przesuwać albo składać z maskami, ani też dodawać do nich liczb zmiennoprzecinkowych (**wskaźniki nie są liczbami całkowitymi**)
- wskaźnikowi do obiektów jednego typu przypisać bez rzutowania wskaźnika do obiektów innego typu (z wyjątkiem typu `void *`).

Argumenty - przekazywanie przez wartość

W języku C wszystkie argumenty funkcji są przekazywane przez "wartość". Oznacza to, że funkcja otrzymuje kopie argumentów i na nich pracuje. Wywołana funkcja nie może bezpośrednio zmienić wartości zmiennej w funkcji wywołującej.

Przykład:

```
/*  Funkcja podnosi argument  base
    do potegi  n                    */
int power(int base, int n)
{
   int p;
   for (p=1; n > 0; n--)
      p=p*base;
   return p;
}
```

Zmiana wartości zmiennej n wewnątrz funkcji nie wpływa na wartość argumentu, z którym została wywołana.