Instrukcje pętli

Pętla for

for ([wyrażenie$_1$]; [wyrażenie$_2$]; [wyrażenie$_3$])
instrukcja

1. **Obliczenie *wyrażenia$_1$* . Typ – dowolny. Liczy się tylko jeden raz przed pierwszą iteracją. Określa początkowy stan pętli.**

2. **Obliczenie *wyrażenia$_2$* . Typ – arytmetyczny lub wskaźnik. Liczy się przed każdą iteracją. Jest warunkiem zakończenia pętli.**
   - **Jeżeli *wyrażenie$_2$* != 0 – wykonana *instrukcja* .**
   - ***wyrażenie$_2$* == 0 – sterowanie będzie przekazane pierwszej instrukcji poza pętlą.**

3. **Obliczenie *wyrażenie$_3$* (liczy się po każdej iteracji). Typ – dowolny. Jest często jest „licznikiem pętli”, chociaż i nie koniecznie. Goto 2.**

Instrukcja pętli powoduje wielokrotne wykonanie *instrukcji* .

Przykłady:

```
for(i=0; i<I_MAX; i++)
{
   a = 5*i;
}

.....

for(j=J_MAX; j>=0; j--)
{
   a += 5*j;
}
```

Przykłady:

```
int i=5;
double a = 0;
for( ; a < 45.0; ++i)
{
    a = 3.14*i;
}

i = 0;
for( ;  ;  ; )
{
    a = 12.6+5.4*i;
    if(a > 1034.46)
        break;
    i+= 2;
}
```

Pętla *while*

while (wyrażenie)
instrukcja

1. **Obliczenie wartości *wyrażenie* . Typ – arytmetyczny lub wskaźnik.**
   Liczy się przed każdym (i przed pierwszym) obrotem pętli.
   Decyduje o zakończeniu albo wykonaniu pętli.
   - **Jeżeli *wyrażenie != 0* – wykonanie *instrukcja* .**
   - ***wyrażenie == 0* – wykonanie pierwszej instrukcji poza ciałem bloku *while.***
2. **Goto 1**

Przykład:

```
while (1)
{
.....................................
//to jest nieskończona pętla
}
```

```
int i = 100;
double a = 1.0;
.....................................
while (i  > 10)
{
   a *= (double)(i+1);
   i--;
}
```

Instrukcja pętli
do ... while

do *instrukcja* while *(wyrażenie)*

1. **Wykonanie *instrukcja* .**
2. **Obliczenie *wyrażenie* . Typ – arytmetyczny lub wskaźnik. Decyduje o zakończeniu albo wykonaniu pętli.**

   Jeśli *wyrażenie != 0* – goto 1.

   wyrażenie* == *0* – wykonanie pierwszej instrukcji poza ciałem bloku *while.

***instrukcja* będzie wykonana co najmniej jeden raz.**

Przykład:

```
int i = 100;
double a = 1.0;
.....................................
do
{
    a *= (double)(i+1);
    i--;
} while(i > 10);
```

Operatory += -= *= /=
++ --

```
x += a;          //x = x + a
z -= b;          // z = z – b
y *= 10.0;       //y = y*10.0;
tp /= a1;        //tp = tp/a1;

int i, j;
i++;             //i = i+1
++i ;            //i = i+1
.....................................
i = 1;
j = i++;         //i = 2, j = 1
.....................................
i = 1;
j = ++i;         //i = 2, j = 2
```

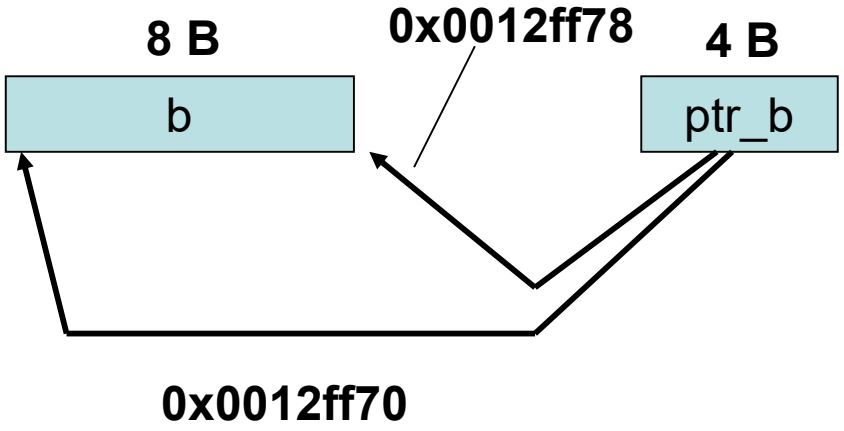
```
int i, j;
i--;                    //i = i-1
--i;                    //i = i-1
.....................................
i = 1;
j = i--;                //i = 0, j = 1
.....................................
i = 1;
j = --i;                //i = 0, j = 0
……..……..……..……..……..

i = ij = 0;
x[i++] = a[ij++];  //x[0] = a[0];  i = 1;  ij = 1;
……..……..……..……..……..

i = ij = 0;
x[++i] = a[++ij];  //x[1] = a[1];  i = 1; ij = 1;
```

Inkrementowanie wskaźnika

```
double b = 1.0;
double *ptr_b = NULL;
ptr_b = &b;        //ptr_b = 0x0012ff70
ptr_b++;           //ptr_b = 0x0012ff78 – ptr_b+sizeof(double)
```



Instrukcja *break* – przerywa pętle nie zależnie od wartości, którą przyjmuje *wyrażenie* – warunek zakończenia pętli.

Instrukcja *return* - przerywa pętle nie zależnie od wartości, którą przyjmuje wyrażenie – warunek zakończenia pętli. Powoduje wyjście z ciała funkcji.

Instrukcja

goto label;
……
label: instrukcja

Przekazuje sterowanie instrukcji, która jest pierwszą po etykietce *label*
Jeśli *goto label* stoi w pętle, a etykietka *label:* **stoi poza blokiem pętli, to pętla ta pozostanie przerwaną.**

Instrukcja

continue;

Powoduje przejście na nową iteracje pętli. Część kodu w instrukcje pętli po instrukcji *continue* nie będzie wykonana. Sterowanie przekazuje się na obliczenie wartości wyrażenia, decydującego o zakończenie pętli. Znaczenie $wyrażenie_3$ dla pętli *for* będzie ustawione na następną iteracje.

Przykłady:

```
for(i=0;i<i_max; i++)
{
   x = f(i);
   if(x < 0)
         break;    //przerwać pętle
   y=x*x-10.0;
}

for(i=0;i<i_max; i++)
{
   x = f(i);
   if(x < 0)
      continue;    //przekazać sterowanie na początek pętli
   y=x*x-10.0;
}
```

```
void ff()
{
……….
   i = 0;
   while( z > 0 )
   {
      x = f(i);
      if(x < 0)
         return;        //przerywanie nie tylko pętli for, a i funkcji ff
      y = x*x-10.0;
      i++;
   }
……….

}
```

```
for(i=0;i<i_max; i++)
{
   x = f(i);
   if(x < 0)
       goto lab_1;    //przekazać sterowanie na pierwszą instrukcje poza
                      //etykietką lab_1
   y=x*x-10.0;
}

……..
lab_1:    z = x*x-t;
```

Etykietki i operator *goto label:* trzeba używać ostrożnie: wszędzie, gdzie tylko jest możliwość, unikać ich zastosowania, ponieważ komplikują oni kolejność wykonania instrukcji programu, dodając skoki w wykonaniu instrukcji kodu. To się zatrudnia debagowanie i czytelność programu oraz obniża wydajność aplikacji.

Przeczcież istnieją sytuacji, kiedy zastosowanie tego operatora może być bardzo skuteczne.

Typ struct FILE *ptr_file;

1. Definicja wskaźnika na obiekt typu FILE:

FILE *ptr_file = NULL;

2. Otwieranie pliku tekstowego:

ptr_file = fopen(“nazwa pliku”, “wt”);

3. Funkcje obsługi pliku.

fprintf(ptr_file, “spec. format”, lista_argumentów);

fscanf(ptr_file, “spec. format”, lista_argumentów);

rewind(ptr_file); //przestawia wskaźnik pozycji pliku na
//początek

fflush(ptr_file); //zapisuje zawartość buforu w plik

4. Zamykanie pliku.

```
fclose(ptr_file);
```

5. Kasowanie pliku.

```
remove(“nazwa pliku”);
```

Otwieranie pliku:

prototyp: FILE *fopen(const char **filename*, const char **mode*);

***filename* – ścieżka i nazwa pliku**
“C:\\Document and Settings\\MyDocs\\my_plik.txt”
lub
“my_plik.txt” (plik będzie otwarty w katalogu bieżącym)
***mode* – tryb otwarcia:**

”r” – tylko na czytanie. Zwraca NULL, jeśli: plik nie istnieje lub nie w stanie być odnaleziony; plik pozostał wcześniej otwarty i nie zamknięty bieżącą aplikacją; plik jest otwarty inną aplikacją i nie jest rozdzielany na odczyt.

”w” – otwiera pusty plik na zapis. Jeśli plik o takiej nazwie już istnieje, jego zawartość pozostanie wykasowaną. Jeżeli nie istnieje, to będzie stworzony nowy plik.

”a” – dopisuje w koniec istniejącego pliku; jeśli plik nie istnieje, tworzy nowy plik

”r+” – otwiera plik na zapis i czytanie. Plik mysi istnieć.

”w+” – otwiera pusty plik na zapis i czytanie. Jeśli plik już istnieje, jego zawartość pozostanie wykasowaną (truncated).

”a+” – otwiera plik na czytanie i na dopisanie. Jeśli plik nie istnieje, tworzy nowy plik

W dodatek za trybem mode może być dopisany symbol *t* lub *b* .
***t* – plik będzie otwarty w trybie tekstowym**
***b* - w trybie binarnym**

Zwykle jeśli *t* lub *b* pominięte, plik się otwiera w trybie tekstowym.

Fragment otwarcia pliku:

```
FILE *ptr_f = NULL;     //definicja wskaźnika na obiekt typu FILE
……….
ptr_f = fopen(“my_file.txt”, “r”);

//to jest koniecznie!
If(!ptr_f)        // lub if(ptr_f == NULL)
   exit(1);


……….
fclose(ptr_f);    //zamknij plik!
ptr_f = NULL;   //muszę wiedzieć, że plik już jest zamknięty
……….
```

Funkcji fprintf, fscanf są bardzo podobne do funkcji printf, scanf. Jedyną różnicą jest to, że pierwszy argument – to jest wskaźnik do pliku:

int fprintf(FILE *ptr_f, “specyfikacje formatu” [, argument]...);

int fscanf(FILE *ptr_f, “specyfikacje formatu” [, argument]...);

Zamknięcie pliku powoduje dopisanie danych, które znajdują się w buforze systemowym, a również ciąg działań systemu operacyjnego. Każdy plik otwarty koniecznie mysi być zamknięty.

int fclose(FILE * ptr_f);

Zwraca 0, gdy plik jest skutecznie zamknięty, EOF – błąd przy zamykaniu.

Wyładować zawartość bufora systemowego w plik

```
int fflush( FILE * ptr_f );
```

Zwraca 0 przy sukcesie, EOF – przy błędnym zakończeniu.

Ustawienie pozycji pliku na początek:

```
void rewind( FILE * ptr_f );
```

Przykład W4_1

Pliki binarne.

Zapis:
size_t fwrite(const void *buffer, size_t size, size_t count, FILE *stream);

buffer – adres danych do zapisu,
size – sizeof jednostki zapisu,
count – ilość elementów w zapisie (słów)
stream – wskaźnik do pliku

Zwraca ilość poprawnie zapisanych elementów (słów) z bufora buffer.

Pliki binarne.

Odczyt:

size_t fread(void *buffer, size_t size, size_t count, FILE *stream);

buffer – adres danych do odczytu,
size – sizeof jednostki zapisu,
count – ilość elementów w zapisie (słów)
stream – wskaźnik do pliku

Zwraca ilość poprawnie odczytanych elementów (słów) w buffer.

Pliki binarne.

int fseek(FILE *stream, long offset, int origin);

int _fseeki64(FILE *stream, __int64 offset, int origin);

Przestawia wskaźnik pozycji pliku stream na offset bytów licząc od:

origin = SEEK_CUR - bieżącej pozycji

origin = SEEK_END - końca pliku

origin = SEEK_SET - początku pliku

Jeśli OK, fseek, _fseeki64 zwracają 0.

long ftell(FILE *stream);

__int64 _ftelli64(FILE *stream);

Zwraca wartość wskaźnika pozycji pliku, licząc od początku.

Przykład W4_2