Wykład 5

5. Instrukcje iteracyjne

5.1. Pętla *for* („dla”)

5.2. Pętla *while* („dopóki”)

5.3. Pętla *do .. while* („wykonuj dopóki”)

5.4. Instrukcje sterujące przebiegiem programu

- *break*
- *continue*.

5. Instrukcje iteracyjne

Iteracja – cykliczne wykonywanie sekwencji rozkazów.

Aby zaprojektować iterację należy określić:

- ciąg instrukcji, które mają być powtarzane,
- warunek kontynuowania lub zakończenia tego ciągu.

Pętle są istotne dla wykonywania cyklicznych zadań takich jak powtarzanie sekwencji instrukcji lub dostęp do elementów tablicy.

W ogólnym przypadku można wyróżnić trzy ogólne rodzaje pętli:

- o ustalonej ilości przebiegów – powtarzają swoje instrukcje wstępnie określoną liczbę razy;
- warunkowe – powtarzane tak długo, dopóki sprawdzany warunek jest prawdziwy, lub do momentu, gdy warunek stanie się prawdziwy;
- nieskończone – pętle otwarte, powtarzane w nieskończoność (posiadają mechanizm opuszczenia pętli).

5.1. Pętla *for*

Instrukcja *for* tworzy pętlę o następującej postaci:

for (wyr1 ; wyr2 ; wyr3) { <sekwencja instrukcji>; }

for (wyr1 ; wyr2 ; wyr3) instrukcja;

Każde z trzech wyrażeń jest opcjonalne (może go nie być), ale nawiasy i średniki są obowiązkowe.

Funkcje wyrażeń:

wyr1 – inicjacja zmiennych sterujących pętlą (może być kilka zmiennych sterujących pętlą);

wyr2 – warunek kontynuacji pętli (określa czy pętla powinna wykonać kolejną iterację);

wyr3 – zwiększanie lub zmniejszenie zmiennych sterujących pętlą.

Działanie pętli:

1. Obliczane jest wyrażenie wyr1 inicjujące liczniki pętli (jeśli występuje).
2. Wyliczane jest wyrażenie warunkowe wyr2. Jeżeli jego wartość jest równa 0, to instrukcja for kończy działanie. Jeżeli jest ono niezerowe lub gdy wyr2 jest pominięte, to wykonywane są instrukcje (instrukcja) związana z for.
3. Wyliczane jest wyrażenie modyfikujące liczniki pętli wyr3 (jeśli występuje). Działanie jest wznawiane od kroku 2.

Np.

```
for (int i = 0; i < 10; i++) cout << i << ‘  ‘;          // deklaracja i

for (i = 0; i < 10; i+=3) cout << i << ‘  ‘;    // przyrost i o 3

for (i = 10;  i >=0;  i--) cout << i << ‘  ‘;

for (i = 10; i >=0; i-=3) cout << i << ‘  ‘;    // zmniejszenie i o 3

char litera;          float x, s;           long w;

for (litera = ‘A’; litera <= ‘Z’; litera++)  cout << litera;

for (i=0,  s = 0.0,  x = 0.1;  i < 10; i++) { s = s + x * i;  cout << s<< endl; }

for (i=1, w = 1;  i < 10; i++) w = w * i;      \\ iloczyn
```

Wyrażenie wyr2 jest obliczane na początku pętli dlatego jeśli nie będzie ono spełnione, to pętla nie wykona się.

```
i=5;
for (  ;   i < 5;   i++) cout << i << endl;

// pętla nie zostanie wykonana; wartość i nie zmienia się

cout << i << endl;          //    i = 5
```

Pętla for może być pusta (kończy się średnikiem). Wykonanie takiej pętli prowadzi jedynie do modyfikacji zmiennych sterujących.

```
for (i=5;  i < 10;   i++)  ;          // pusta pętla for

// wykonanie pętli powoduje zmianę wartości zmiennej i

cout << i << endl;          //    i = 10
```

Wyrażenia pętli for mogą być rozbudowane. Kolejny przykład pokazuje możliwość wczytywania danych (scanf) za pomocą wyr3. Dwa kolejne Enter (‘\n’) kończą działanie pętli.

```
char z=0;
for (printf("Wczytaj znak! Enter - koniec: \n"); z!='\n'; scanf("%c%c", &z))
{
   if (z) cout << "Wczytano "<< z << endl;   }
```

Można również wykorzystać scanf() do sprawdzenia, czy w pętli wczytano liczbę.

```
printf("Wczytaj liczbe! Dowolny znak - koniec \n");

 for (long t=0, licz=0; scanf("%li", &t)==1; licz++)
   { cout << "Wczytano "<< t << "   Licznik = " << licz << endl; }
```

Warunek zakończenia pętli może być testowany również dla wyrażeń typu rzeczywistego.

```
for (double xa=2.0, y=0; y < 8024; xa++)
  { y=pow(xa,3); cout << "x = " << xa << ";  x do 3 =  " << y << endl; }
```

Przykład 5.1. Obliczanie n! .

```
void main(void)
{
  int i,n;
  double silnia = 1.0;

 clrscr();

 cout << "\nPodaj n z zakresu [1..20]:" << endl;
 cin >> n;

 if (n >0 && n <=20)
 {
   for (i=1; i<=n; i++) silnia*=(double) i;

   cout << n << "! = " << silnia << "\n";
 }
   else
   cout << "Liczba n jest spoza zakresu " << endl;

   getch();
}
```

- Pętla for może być wielokrotnie zagnieżdżona.

Przykład 5.2. Wyprowadzanie liczb na ekran wierszami.

```
void main(void)
{
  int a, i, j;

  clrscr();   randomize();

  // wyprowadzanie wierszami; jedna pętla

  for (i=1; i<33; i++)
  {  a = random(100);
     if (i%8) printf("%3d", a); else printf("%3d\n", a);
  }
  cout << endl;
```

```
// wyprowadzanie wierszami;  2 pętle

 for (i=1; i<5; i++)
  {
  for (j=1; j<9; j++)

   { a = random(100);
     printf("%3d", a);
    }
       printf("\n");
  }
   getch();
}
```

Przykład 5.3. Wyprowadzanie figur o wysokości i podstawie h.

```
                X                X
          h    XX               XX    h
               XXX              XXX
```

```
void main(void)
{
  int i,j, x,y, h;

  clrscr();    x=1; y=1;   h = 7;

  for (i=0; i<h; i++)
  {  gotoxy(x,y+i);
     for (j=1; j<i+1; j++) cout << '*';
   }

  x=20;  y=1;

  for (i=0; i<h; i++)
  {  gotoxy(x-i,y+i);
     for (j=1; j<i+1; j++) cout << '*';
   }
   getch();
}
```

Przykład 5.4. Pętla nieskończona.

```
void main(void)
{
  char zn; double x,y;

  // realizacja petli nieskonczonej za pomocą for

  clrscr();

  for (; ;)
  {
  cout << "\nPodaj liczbe: ";
  cin >> x;  y = 2*x;
  cout << "2 * "<<x << " = " << y << "\n\n";
  cout << "Kolejne obliczenia (T/N)?";
  zn = getche();
  if (zn !='t' && zn !='T') exit(0);
  }
    getch();
}
```

5.2. Pętla *while*

Pętla warunkowa while ma następującą składnię:

while (wyrażenie_warunkowe) { <sekwencja instrukcji>; }

while (wyrażenie_warunkowe) instrukcja;

Wyrażenie warunkowe jest sprawdzane przed wykonaniem instrukcji pętli. Jeśli wyrażenie jest fałszywe, to instrukcje pętli while nie zostaną wykonane.

Np.

```
        while ( (zn = getche() ) != 'k')
        {  instrukcje; }
```

```
int i =0;

while (i < 10)                    //wyprowadzanie liczb od 0 do 9
{
   cout << i << ' ';  i++;
}

i = 0;
while (i < 10)
{
   cout << i << ' ';   i+=3;              // przyrost i o 3
}
```

Przykład 5.5. Wprowadzanie znaków i liczb w pętli

```
void main(void)
{
  char zn; int koniec = 0;

  // wprowadzanie znakow w petli
  clrscr();
  while (! koniec)
  {
    cout << "\nWpisz znak (N - koniec) :";
    cin >> zn;
    if (zn == 'n' || zn == 'N') koniec = 1;
    else cout << zn;
  }

  double x=0;

  while (x<5.0 || x>20.0)
  {
     cout << "\nPodaj liczbe z przedzialu [5,20]: ";
     cin >> x;
  }
   getche();
}
```

5.3. Pętla *do-while*

Pętla warunkowa do-while ma następującą składnię:

```
do   {

    <sekwencja instrukcji>;

 }  while (wyrażenie_warunkowe);


do   instrukcja;  while (wyrażenie_warunkowe);
```

Pętla jest wykonywana tak długo, aż wyrażenie osiągnie wartość FALSE.
Instrukcja jest stosowana w tych przypadkach, gdy chcemy,
aby wykonana została co najmniej jedna iteracja.

```
int i =0;

do                                  // wyprowadzanie liczb od 0 do 9
{    cout << i << ‘  ‘;  i++;

} while (i < 10);


i = 0;
do
{
   cout << i << ‘  ‘;   i+=3;              // przyrost i o 3
}  while (i <10);

double s = 0.0;
i=0;

do
{
       s = s + i;
       i++;
}  while (i <10);
```

Przykład. 5.6. Obliczanie pierwiastka za pomocą algorytmu Newtona.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>

const double EPS = 1E-9;

void main(void)
{
 int koniec = 0;
 double x, px, d;       // obliczanie pierwiastka x

 clrscr();
 do {
      cout << "\nPodaj liczbe dodatnia: ";
      cin >> x;
      cout << endl;

      if (x > 0.0) {
      // wartosc poczatkowa pierwiastka
      px = x*0.5;
      // iteracyjne znajdowanie kolejnych przyblizen
      do {
           px = (x/px + px)/2.0;
           d = x - px*px;
      } while ( !(fabs(d) < EPS) );

      cout << "Pierwiastek (" << x << ") = ";
      printf("%.17lf  %.17lf", px, sqrt(x));
      koniec = 1;
      }
      else {
            cout << "Wprowadziles liczbe ujemna \n";
            koniec = 0;  }
 } while (!koniec);

  getche();
 }
```

Przykład. 5.7. **Dla ustalonego n obliczyć sumę szeregu.**
S(n) = (1 +1/2 + 1/3) – (1/4 + 1/5 + 1/6) + ... ± (... +1/n).

```
#include <conio.h>  #include <stdio.h>

void main()
{
int n = 4, i;   double z = 1.0;   double s;    s = 0.0;

  for (i=1; i<=n; i++)  {  s = s + z/i;   if (! (i%3) ) z=-z; }

  printf("\nSuma s(%d) = %lf", n, s);
  getch();
}
```

Przykład. 5.8. Obliczyć sin(x) ze wzoru S(x)=x – x^3/3! + x^5/5! – x^7/7! +... . Obliczenia zakończyć, gdy odległość pomiędzy kolejnymi wyrazami szeregu nie przekracza EPS = 1E-7.

```
#include <conio.h>  #include <stdio.h>  #include <math.h>

const double EPS = 1E-7;

void main(void)
{   double x, s, wn, wp;     long i;

  clrscr();
  do {
      printf("Podaj argument sinusa [0,2*3.14]\n");
      scanf("%lf",&x);
       }   while (x < 0.0 || x>2*M_PI);

  i=1;   wp=x+2*EPS;   wn=x;   s=x;

  while (fabs( fabs(wn)-fabs(wp) ) > EPS)
  {
    wp=wn;    i+=2;   wn=-wn*x*x/(i*(i-1));    s=s+wn;
  }
    printf("\nSinus argumentu wynosi %.17lf",s);
    printf("\nFunkcja standardowa daje %.17lf",sin(x));
    getch();
}
```

Przykład 5.9. Prosty kalkulator z dwoma pętlami.

```
void main(void)
{
 char oper, zn;
 double a, b, wynik;
 int pom;

 clrscr();
 do {
  cout << "\nPodaj pierwszy argument : ";
  cin >> a;
  cout << "\nPodaj drugi argument : ";
  cin >> b;
  cout << endl;

   cout << "\nWybierz operacje [+, -, /, *]: ";
   pom = 1;  // wybrana operacja jest poprawna
 switch (oper = getche()) {
 case '+': wynik = a + b; break;
 case '-': wynik = a - b; break;
 case '*': wynik = a * b; break;
 case '/': if (b != 0.0) wynik = a/b;
           else
           { cout << "\nDzielenie przez zero"; pom=0; }
           break;
 default:  { cout << "\nWybrano zla operacje"; pom=0; }
}   //switch

 if (pom) // wyswietlenie wynikow
 { cout << "\nWynik: ";
   cout << a << ' ' << oper << ' ' << b << " = " << wynik << endl;
 }
 do {
     cout << "\nCzy wykonac kolejne obliczenia (T/N)?";
     zn = getche();
     zn =_toupper(zn);
     cout << endl;
    } while( !(zn == 'T' || zn == 'N') );

  } while (zn == 'T');
}
```

Przykład 5.10. Obliczanie n! (z while i zmniejszaniem i--).

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <ctype.h>

void main(void)
{
 char zn;
 double silnia;
 unsigned int i,n;

 clrscr();

 do {
  cout << "\nPodaj liczbe, ktorej silnie chcesz obliczyc: ";
  cin >> n;
  cout << endl;
  if (n >=1 && n <=20) {
   silnia = 1.0;
   i = n;

   while (i>1) silnia *= (double) i--;

   cout << n << "! = " << silnia << endl;
  }
  else
    cout << "\nLiczba spoza zakresu \n";
  do {
      cout << "\nKolejne obliczenia (T/N)?";
      zn = getche();
      zn =_toupper(zn);
      cout << endl;
     } while( !(zn == 'T' || zn == 'N') );

  } while (zn == 'T');
}
```

5.4. Instrukcje sterujące przebiegiem programu

- *break*
- *continue*.

Instrukcja *break* umożliwia przerwanie procesu iteracji.
Może być stosowana dla każdego typu iteracji.

```
while  (wyrażenie)
{
        <sekwencja_instrukcji_1>;

        if (warunek_zakończenia_pętli) break;

        <sekwencja_instrukcji_2>;
}

        <sekwencja_instrukcji_3>;
```

Spełnienie warunku zakończenia pętli powoduje przerwanie procesu Iteracji i przejście do wykonania trzeciej sekwencji instrukcji.

Instrukcja *continue* umożliwia pominięcie instrukcji wykonywanych w pętli. Może być stosowana dla każdego typu iteracji.

```
while  (wyrażenie)
{
        <sekwencja_instrukcji_1>;

        if (warunek_skoku) continue;

        <sekwencja_instrukcji_2>;
}
```

Spełnienie warunku skoku powoduje pominięcie drugiej sekwencji instrukcji i przejście do sprawdzenia warunku zakończenia pętli.

Przykład 5.11. Wykorzystanie break.

```
void main(void)
{
 int i;
 clrscr();
 randomize();

 while (1)                          // zakończenie pętli, gdy
 {     i = random(100);             // pojawi się liczba z przedziału [20,30]
       if (i>=20 && i<=30)
       { cout << "\nLiczba z przedzialu [20,30]: " << i;
          break; }
       cout << i << ' ';
 }
      cout << endl << endl;

 for (; ;)
 {     i = random(100);
       if (i>=10 && i<=20)
       { cout << "\nLiczba z przedzialu [10,20]: " << i;
          break; }
       cout << i << ' ';
 }
 getch();
}
```

Przykład 5.12. Wykorzystanie continue.

```
void main(void)
{
 int i, n, licz;
 clrscr();
 randomize();

 i=1;  n=10;  licz = 0;

 while (1)
 {     i = random(100);
       if (i%3) continue;           // wyprowadzanie 10 liczb
       printf("%3d", i );           // podzielnych przez 3
       licz++;
       if (licz >= n) break;
 }
      cout << endl << endl;

 licz = 0;

 for (; licz<n; )                   // to samo z pętlą for
 {     i = random(100);
       if (i%3) continue;
       printf("%3d", i);
       licz++;
 }
 getch();
}
```