TYP REFERENCYJNY – PRZEZWISKO ZMIENNEJ

zmienne tego typu nie zajmują nowego miejsca w pamięci,
służą do reprezentacji innych zmiennych w programie.

nazwa_typu nazwa_zmiennej; // utworzenie zwykłej zmiennej
nazwa_typu & nazwa_zmiennej_referencyjnej = nazwa_zmiennej;
// jest to zdefiniowanie aliasu - innej nazwy dla tej samej zmiennej

```
int  wzrost = 5;
int  sredni_wzrost = wzrost;
int& kontr_wzrost  = wzrost;    // utworzenie zmiennej referencyjnej
                                // związanej z tym samym obszarem
                                // pamięci co zmienna wzrost
kontr_wzrost = kontr_wzrost - 1;// równoważne: wzrost = wzrost + 1

cout << wzrost;
cout << sredni_wzrost;
cout << kontr_wzrost;
```

RÓŻNE STYLE PROGRAMOWANIA

```
#include<iostream>
using namespace std;
// ---------------------------
int a, b, w;
// ---------------------------
void srednia(void)
{
  w = (a + b) / 2.0;
}
// ---------------------------
int main(void)
{
  cout<<"Podaj dwie liczby\n";
  cin >> a >> b;
  srednia();
  cout << "srednia =" << w;
}
```

```
#include<iostream>
using namespace std;
// ------------------------------
void srednia(int a, int b, double& w)
{
  w = (a + b) / 2.0;
}
// ------------------------------
int main(void)
{
  int a, b;
  double w;
  cout << "Podaj dwie liczby \n";
  cin >> a >> b;
  srednia(a, b, w);
  cout << "srednia =" << w;
}
```

```
#include<iostream>
using namespace std;
// ---------------------------
double srednia(int a, int b)
{
  return 0.5*(a + b);
}
// ---------------------------
int main(void)
{
  int a, b;
  double w;
  cout<<"Podaj dwie liczby\n";
  cin >> a >> b;
  w = srednia(a, b);
  cout << "srednia =" << w;
}
```

```
a = 3;
b = 5;
srednia();
double w1 = w;
a = 8;
b = 5;
srednia();
double w2 = w;
a = 3;
b = 8;
srednia();
double w3 = w;
double w = (w1 - w2) / w3;
```

```
double w1, w2, w3;
srednia(3, 5, w1);
srednia(8, 5, w2);
srednia(3, 8, w3);
double w = (w1 - w2) / w3;
```

```
double w = (srednia(3, 5) - srednia(8, 5)) / srednia(3, 8);
```

WIELOKROTNE RETURN

```cpp
#include<iostream>
using namespace std;

int max(int x, int y)
{
  if (x>y) return x;
  else    return y;
}

int main(void)
{
  int a, b, c, m;
  cout << "Podaj trzy liczby A,B,C";
  cin >> a >> b >> c;
  m = max(a, b);
  cout << "Maksimum z liczb A i B ="
       << m << endl;
  cout << "Maksimum z liczb B i C ="
       << max(b, c) << endl;
  cout << "Maksimum z A, B, C ="
       << max(a, max(b, c)) << endl;
}
```

```cpp
#include<iostream>
using namespace std;

int poszukaj(int t[10], int a)
{
  for (int i = 0; i<10; i++)
    if (t[i] == a)
      return i;
  return -1;
}

int main(void)
{
  int t[10] = { 12, 67, 23, 67, 34, 11 };
  int ind = poszukaj(t, 23);

  if (ind >= 0)
    cout << "23==t[" << ind + 1 << "]";
  else
    cout << "23 nie zawiera sie w tablicy t";
}
```

FUNKCJA REKURENCYJNA

```
#include<iostream>
using namespace std;

int silnia(int n)
{
     int w;
     if (n <= 1) w = 1;
     else        w = n*silnia(n - 1);
     return w;
}
// ----------------------------------------
int silniaV2(int n)
{
     return (n <= 1 ? 1 : n*silniaV2(n - 1));
}
// ----------------------------------------
int main(void)
{
     int a = silnia( 4);
     cout << "4!=" << a;
     cout << "5!=" << silniaV2(5);
}
```

```
//  int a=silnia(4);
//  int a=4*silnia(3);
//  int a=4*3*silnia(2);
//  int a=4*3*2*silnia(1);
//  int a=4*3*2*1;
//  int a=24;
```

```
//  int a = silnia(4);
//          int w = 4*silnia(3);
//                  int w = 3*silnia(2);
//                          int w = 2*silnia(1);
//                                  int w = 1;
//                          int w = 2*1;
//                  int w = 3*2*1;
//          int w = 4*3*2*1;
//  int a = 4*3*2*1;
```

DEKLARACJA A DEFINICJA FUNKCJI

```
double pole(double a, double b)              // definicja (i zarazem deklaracja) funkcji
{
    return a*b;
}
// ------------------------------------------------
double kubatura(double a, double b, double c);     // deklaracja (prototyp) funkcji
// double kubatura( double, double, double);
// ------------------------------------------------
int main(void)
{
    // ...
    p = pole(dlugosc, szerokosc);
    v = kubatura(dlugosc, szerokosc, wysokosc);
    // ...
}
// ------------------------------------------------
double kubatura(double a, double b, double c)
{
    double obj;
    obj = a*b*c;
    return obj;
}
```

// **definicja funkcji**
// znajdować się ona może w tym
// samym lub innym pliku

PARAMETRY DOMYŚLNE FUNKCJI

```
#include<iostream>
using namespace std;

void Zdanie(char Nazw[20], char Imie[20] = "Jan", int wiek = 80, double zarobki = 100.0)
{
     cout << Nazw <<" "<< Imie <<", bedac w wieku "<< wiek <<" lat, zarabia "<< zarobki <<" zl.\n";
}
// ------------------------------------------------------------------------------------------
//void ZdanieDrugie(char Nazw[20], char Imie[20] = "Jan", int wiek = 80);
 void ZdanieDrugie( char[20],  char [20]="Anna", int =80);
// ------------------------------------------------------------------------------------------
int main(void)
{
     char nazwisko[5][20] = { "Nowak", "Kowalski", "Gawron", "Kaczmarek" };
     char imie[5][20] = { "Jolanta", "Aneta", "Grzegorz" };
     double zarobki = 1222.00;
     int wiek = 32;

     Zdanie(nazwisko[0], imie[1], wiek, zarobki);    // Nowak, Aneta, 32, 1222.0
     Zdanie(nazwisko[0], imie[1], wiek);             // Nowak, Aneta, 32, 100.0
     Zdanie(nazwisko[0], imie[1]);                   // Nowak, Aneta, 80, 100.0
     Zdanie(nazwisko[0]);                            // Nowak, Jan, 80, 100.0
     // Zdanie();
     Zdanie(nazwisko[2], "Amanda", 44);              // Gawron, Amanda, 44, 100.0
     // Zdanie( nazwisko[2], "Amanda", , 850.00);

     system("pause");
}
// ------------------------------------------------------------------------------------------
void ZdanieDrugie(char Nazw[20], char Imie[20], int wiek)
{
     cout << Nazw << " " << Imie << ", ma " << wiek << " lat.\n";
}
```

PRZECIĄŻANIE FUNKCJI - POLIMORFIZM

Zamiast funkcji o różnych nazwach możemy użyć funkcje o tej samej nazwie, lecz różniących się ilością parametrów lub/i ich typem. Parametry wywołania decydują o wybraniu właściwej funkcji.

```
#include<iostream>
using namespace std;

int wyswietl(int a)           { cout << "fun. pierwsza:" << a << endl;              return 1; }
void wyswietl(double a)       { cout << "fun. druga   :" << a << endl;                return; }
void wyswietl(int a, int b)   { cout << "fun. trzecia :" << a << " " << b << endl;   return;  }
void wyswietl(int a, double b){ cout << "fun. czwarta :" << a << " " << b << endl; return;    }
//double  wyswietl( int a)    { cout<<"fun. niedopuszczona:"<<a<<endl;  return 2.0;           }

int main(void)
{
    int a = 33;
    double x = 0.5, y = 42;
    wyswietl(a);              // fun. pierwsza
    wyswietl(x);              // fun. druga
    wyswietl(y);              // fun. druga
    wyswietl(0.23);           // fun. druga
    wyswietl(32.);            // fun. druga
    wyswietl(31);             // fun. pierwsza
    wyswietl(int(0.34));      // fun. pierwsza
    wyswietl(x, 1);           // fun. trzecia
    wyswietl(x, y);           // fun. czwarta
}
```

WKAŹNIKI / ADRESY

wskaźnik - jest zmienną, która zawiera **adres** (wskazanie) innej zmiennej
lub adres dowolnego obszaru w pamięci komputera,
(np. może być to adres obszaru danych lub adres kodu programu)

Ogólna postać definicji wskaźnika:

typ_danych* identyfikator_wskaźnika ;

Taki wskaźnik zawiera informację o:

- **adresie** zmiennej w pamięci komputera
- **typie** danych przechowywanych w tej zmiennej

```
//Przykłady definicji :
int * wskaznik;          // wskaźnik na zmienną całkowitą
double * wsk_liczby;     // wskaźnik na zmienną rzeczywistą
char * wsk_znaku;        // wskaźnik na pojedynczy znak
char * tekst;            // wskaźnik na początek łańcucha znaków
                         // (na pierwszy znak tego łańcucha)
```

WKAŹNIKI / ADRESY - CD

Ze wskaźnikami i adresami związane są dwa operatory:

- operator **adresu** (referencji) **&** zwracający adres zmiennej podanej po prawej stronie tego operatora.
- operator **wyłuskania** (dereferencji) * identyfikujący obszar zmiennej wskazywanej przez wskaźnik podany po prawej stronie tego operatora.

```
int liczba = 5;
int *wskaznik;
wskaznik = &liczba;  // przypisanie zmiennej wskaźnik
                     // adresu zmiennej liczba
*wskaznik = 10;      // przypisanie wartości 10 do zmiennej
                     // wskazywanej przez wskaźnik
                     // ( tutaj jest to równoważne liczba = 10 )
```

Można również korzystać ze wskaźników „niezdefiniowanych” (anonimowych).
Taki wskaźnik zawiera tylko informację o **adresie** początku obszaru pamięci
(bez określenia typu wskazywanych danych). Definicja takiego wskaźnika ma postać:

void * **identyfikator_wskaźnika ;**

jest to wskaźnik na „dowolny” ciąg bajtów danych.

WKAŹNIKI / ADRESY - PRZYKŁAD

```
int a = 7;
cout << a << endl;
cout << &a << endl;

int *w;
w = &a;
//int *w = &a;
cout << w << endl;
cout << *w << endl;
```

```
7
003BEC5C
003BEC5C
7
```

```
int t[3] = { 2, 4, 6 };

cout << t[0] << endl;
cout << t[1] << endl;
cout << t[2] << endl;

cout << &t[0] << endl;
cout << &t[1] << endl;
cout << &t[2] << endl;
```

```
int *w1=&t[0], *w2=&t[1],
    *w3=&t[2];
cout << w1 << endl;
cout << w2 << endl;
cout << w3 << endl;

cout << *w1 << endl;
cout << *w2 << endl;
cout << *w3 << endl;
```

```
2
4
6
003BEC70
003BEC74
003BEC78
003BEC70
003BEC74
003BEC78
2
4
6
```

```
double x = 2.1;
double y = 7.1;
double z = 9.1;

double *w = &x;

cout << &x;    // 003B2C11
cout << w;     // 003B2C11
cout << *w;    // 2.1
cout << &w;    // 003BEDEE

w = &y;
cout << *w;          // 7.1

*w = z;
cout << x << y << z; // 2.1 9.1 9.1

*w = 88;
cout << x << y << z; // 2.1 88 9.1

w = &x;
*w += 2;
cout << x << y << z;  // 4.1 88 9.1
```

REFERENCJA ORAZ WSKAŹNIK JAKO PARAMETR FUNKCJI

```
void sprobuj_zam(int x, int y)// void sprobuj_zam( int x = a, int y = b)
{                                    // zmienne a b nie ulegną zmianie
     int tmp = x;
     x = y;
     y = tmp;
}
// ------------------------------------
void zamien_ref(int &x, int &y) // void zamien_ref( int &x = c, int &y = d)
{
     int tmp = x;
     x = y;
     y = tmp;
}
// ------------------------------------
void zamien_wsk(int *x, int *y) // void zamien_wsk( int *x = &e, int *y = &f)
{
     int tmp = *x;
     *x = *y;
     *y = tmp;
}
// ------------------------------------
```

```
int main(void)
{
     int a = 2, b = 8,c = 3, d = 5e = 9, f = 1;

     sprobuj_zam(a, b);
     cout << u << v;

     zamien_ref(c, d);
     cout << c << d;

     zamien_wsk(&e, &f);
     cout << x << y;

     //zamien_ref(7, a);  // blad
}
```