1. FUNKCJE

Funkcje można rozumieć jako podprogramy w programie. Dzięki temu rozwiązaniu możemy napisać jedną funkcję, po czym wywołać ją w programie wiele razy. Choć wszystkie działania jakie wykonuje funkcja możliwe są do realizacji bez ich użycia, to funkcje oszczędzają czas, zapobiegają powtarzalności kodu (efektywność) i umożliwiają zmianę kodu funkcji w jednym miejscu (jej definicji).

Funkcja:

1) Definicja funkcji.
2) Prototyp funkcji (to nazwa funkcji z zestawem argumentów poprzedzonych typem danych).
3) Wywołanie funkcji.

Budowa funkcji zwracającej wartość (słowo kluczowe RETURN i typ zwracanej wartości)

```
typ_zwracanej_wartości nazwa (typ_argumentu nazwa_argumentu)
{
        return zwracana_wartość
}
```

Budowa funkcji, która nie zwraca wartości (słowo kluczowe VOID)

```
void to_jest_moja_funkcja()
{
        ciało_funkcji
}
```

Każda funkcja posiada właściwości:

- zwraca dane (lub nie, w przypadku funkcji typu void)
- posiada nazwę
- może posiadać dowolną liczbę argumentów wejściowych (lub żadnego)

Wywołanie funkcji

```
nazwa_funkcji (wartość_argumentu …. )
```

Funkcje, które nie zwracają wartości wywołujemy podając ich nazwę. Funkcje, które podają wartość przypisujemy do zmiennych lub kierujemy wyjście do pliku lub na ekran.

Przykład 1. Podstawy funkcji – prototypy i wywołania

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

//Deklaracja samych prototypów funkcji przed funkcją main()
void funkcja();     //prototyp funkcji

int main()
{
    funkcja();      //wywołanie funkcji nr 1
    funkcja();      //wywołanie funkcji nr 2
}

//definicja
void funkcja()
{
    //ciało funkcji
    cout << "Funkcja wykonała się.\n";
}
```

Przykład 2. Podstawy funkcji – definicja bez prototypu (częstsze użycie)

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

//definicja funkcji wraz z jej ciałem
void funkcja ()
{
  //ciało funkcji
   cout << "Funkcja wykonała się.\n";
}

int main()
{
    funkcja();      //wywołanie funkcji (pierwsze)
    funkcja();      //wywołanie funkcji (drugie)
}
```

Przykład 3. Funkcje nie zwracające wyniku / zwracające wynik

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

// Definicja funkcji
// Funkcja ma 2 argumenty a i b typu całkowitego i zwraca wartość całkowitą (int)
int suma(int a, int b)
{
  //deklaracja dodatkowej zmiennej wynik typu int
  int wynik;
  //obliczenie wartości zmiennej instrukcją przypisania
  wynik = a + b;
  //zwracanie wartości funkcji
  return wynik;
}

// Definicja funkcji
// Funkcja ma 2 argumenty a i b typu całkowitego, nie zwraca wartości
void roznica(int a, int b)
{
  //deklaracja dodatkowej zmiennej wynik typu int
  int wynik;
  //obliczenie wartości zmiennej instrukcją przypisania (użycie argumentów)
  wynik = a + b;
  //funkcja nie zwraca wartości (w definicji jest słowo void), jedynie wyświetla
wynik
  cout <<"\n roznica = " << wynik;
}

// Definicja funkcji
// Funkcja ma 1 argument typu całkowitego, nie zwraca wartości
void pierwiastek_kwadratowy(int x)
{
  //wynik będzie liczbą całkowitą
  int wynik;
  wynik = x * x;
  cout <<"\n pierwiastek kwadratowy z" << x << " wynosi: " << wynik;
}

// Definicja funkcji
// Funkcja ma 1 argument typu całkowitego, zwraca argument
int pole_kwadratu(int x)
{
  //wynik będzie liczbą całkowitą
  int wynik;
  wynik = x * x;
  return wynik;
}
```

```
//program główny
int main()
{
  cout << "\nUżycie funkcji nr 1 - suma:";
  // wyświetlenie sumy liczb 10 i 2
  cout << "\n suma = " << suma(10, 2);
  cout << "\nUżycie funkcji nr 2 - roznica:";
  // wyświetlenie roznicy liczb 10 i 5
  roznica(10, 5);
  cout << "\nUżycie funkcji nr 3 - pierwiastek kwadratowy:";
  pierwiastek_kwadratowy(2);
  cout << "\nUżycie funkcji nr 4 - pole kwadratu:";
  cout << "\n pole kwadratu = " << pole_kwadratu(2);
}
```

Przykład 4. Funkcja licząca średnią

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

int a, b, c, d;
float srednia(int x, int y, int z, int d); // prototyp funkcji (informacja, że
funkcja istnieje

//Obliczenie średniej arytmetycznej czterech podanych liczb
int main()
{
    cout << "Prosze wprowadzic kolejno cztery liczby calkowite:" << endl;
    cout << "Podaj pierwsza liczbe: ";
    cin >> a;
    cout << "Podaj druga liczbe: ";
    cin >> b;
    cout << "Podaj trzecia liczbe: ";
    cin >> c;
    cout << "Podaj czwartą liczbe: ";
    cin >> d;

    cout << "Srednia wynosi: " << srednia(a, b, c, d);   //wywołanie funkcji
}

float srednia(int x, int y, int z, int d)  // definicja funkcji
{
    return (x + y + z + d)/4;
}
```

Przykład 5. Funkcja sprawdzająca, która liczba jest większa

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

double ktora_wieksza (double a, double b);
void witaj();

int main()
{
  //wywołanie funkcji ktora_wieksza
  cout<<"Wywołanie ktora_wieksza: "<<max(5.2, 7.3)<<endl;
  //wywołanie funkcji witaj
  witaj();
  cout<<endl;
}

double ktora_wieksza (double a, double b)
{
  if(a>b)
    return a;
  return b;
}

void witaj()
{
  cout<<"Witaj użytkowniku!!!";
}
```

Przykład 6. Funkcja - argumenty przesyłane przez wartość

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

void funkcja(int a)
{
  cout << "Wartosc zmiennej w funkcji =" << a << endl;
  a++;
  cout << "Po zmodyfikowaniu wartosc zmiennej =" << a << endl;
}

int main()
{
  int a = 5;
  cout << "Wartość zmiennej przed wywołaniem funkcji: " << a << endl;
  funkcja(a);
  cout << "Wartość zmiennej po wywołaniu funkcji: " << a << endl;
}
```

Przykład 7. Funkcja - argumenty przesyłane przez wskaźnik

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

void funkcja(int* w)
{
  cout << "Użycie funkcji ===============" <<endl;
  cout << "Adres wskaznika w funkcji to " << &w << endl;
  cout << "Wskaznik w funkcji to " << w << endl;
  cout << "Wartosc zmiennej przed modyfikacją = " << (*w) << endl;
  (*w)++;
  cout << "Wartość zmiennej po modyfikacji = " << (*w) << endl;
}

int main()
{
  int a = 5;
  cout << "Wartosc zmiennej przed użyciem funkcji " << a << endl;
  cout << "Adres zmiennej przed użyciem funkcji " << &a << endl;
  int* w = &a;
  funkcja(w);
}
```

Przykład 8. Przeciążanie funkcji – funkcje o tej samej nazwie, różniące się zwracanym typem i argumentami

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

float pole(float r);        //pole koła
int pole(int a);            //pole kwadratu
int pole(int a, int b);     //pole prostokąta

int main()
{
    float promien;
    int a, b;

    cout << "Podaj promien koła: ";
    cin >> promien;
    cout << "Pole koła o promieniu " << promien << " wynosi: " << pole(promien) <<
endl;

    cout << "Podaj długość boku kwadratu: ";
    cin >> a;
    cout << "Pole kwadratu o boku " << a << " wynosi: " << pole(a) << endl;

    cout << "Podaj długość pierwszego boku prostokąta: ";
    cin >> a;
    cout << "Podaj długość drugiego boku prostokąta: ";
    cin >> b;
```

```
    cout << "Pole prostokąta o bokach " << a << " i " << b << " wynosi: " <<
pole(a,b);

    return 0;
}

float pole(float r)
{
    return 3.14 * r * r;
}

int pole(int a)
{
    return a * a;
}

int pole(int a, int b)
{
    return a * b;
}
```

Przykład 9. Rekurencja – wywołanie funkcji w niej samej

```
#include <iostream>

using namespace std;

int naj_wspolny_dzielnik(int a, int b)
{
return b == 0 ? a : naj_wspolny_dzielnik(b, a % b);
}

int main()
{
int x, y;

x = 5; y = 15;
cout << "Największy wspólny dzielnik (" << x << "," << y << ") = "<<
naj_wspolny_dzielnik(x,y) << endl;

x = 732442; y = 1270;
cout << "Największy wspólny dzielnik (" << x << "," << y << ") = "<<
naj_wspolny_dzielnik(x,y) << endl;
}
```

Przykład 10. Rekurencja – wywołanie funkcji w niej samej

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

int silnia( int n )
{
    if( n <= 1 )
         return 1;
    return n * silnia( n - 1 );
}

int main()
{
  cout << silnia(4);
}
```

Tablice możemy przekazywać do funkcji:
- void sposob1 (int tablica[123]);
- void sposob2 (int tablica[]);
- void sposob3 (int *tablica);

Przykład 11. Tablice – przekazywanie tablic jednowymiarowych do funkcji

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

void wypelnijTab( int t[], int iRozmiar )
{
    cout << "Podaj " << iRozmiar << " liczb:" << endl;
    int i = 0;
    do
    {
      cin >> t[ i ];
      i++;
    } while (i < iRozmiar);
}

int main()
{
    int tab[ 5 ];
    wypelnijTab( tab, 5 );

    int i = 0;
    cout <<"Wypisuję tablicę: ";
    for (int i = 0; i < 5; i++)
    {
      cout<< tab[i] <<" ";
    }
}
```

Przykład 12. Tablice – przekazywanie tablic jednowymiarowych do funkcji

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

void wczytajNapis(string t[], int n)
{
  for(int i=0;i<n;i++)
  {
    cout << "Podaj " << i <<"-ty element: ";
    cin >> t[i];
  }
}

void wypiszNapis(string imie[], string nazwisko[], int n)
{
  for(int i=0;i<n;i++)
    cout << imie[i] << " " << nazwisko[i] << endl;
}

int main()
{
  string imie[3];
  string nazwisko[3];

  cout << "Podaj kolejno imiona\n";
  wczytajNapis(imie,3);
  cout << "Podaj kolejno nazwiska\n";
  wczytajNapis(nazwisko,3);

  wypiszNapis(imie,nazwisko,3);
  return 0;
}
```

Przykład 13. Tablice – zamiana elementów tablicy

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

void zamiana(int tab[])
{
        for(int i=0; i<10; i++)
                tab[i] = tab[i] * tab[i];
}

int main()
{
        //inicjacja tablicy
        int tablica[10] = {0, 3, 4, 3, 6, 7, 11, -5, -10, 87};

        //wypisanie elementów tablicy
        for(int i=0;i<10;i++)
                cout<<tablica[i]<<" ";

        cout<<endl; //wstawienie znaku końca linii (enter)

        //wykonanie polecenia
        zamiana(tablica); //przekazując tablicę, podajemy tylko jej nazwę

        //ponowne wypisanie elementów tablicy
        for(int i=0;i<10;i++)
                cout<<tablica[i]<<" ";
}
```

Przykład 14. Tablice – zamiana elementów tablicy (wskaźnik na tablicę)

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

void zamiana(int *tab) //podajemy jedynie nazwę
{
  for(int i=0; i<10; i++)
    tab[i] = tab[i] * tab[i];
}

int main()
{
        //inicjacja tablicy
        int tablica[10] = {0, 3, 4, 3, 6, 7, 11, -5, -10, 87};

        //wypisanie elementów tablicy
        for(int i=0;i<10;i++)
                cout<<tablica[i]<<" ";
```

```
        cout<<endl; //wstawienie znaku końca linii (enter)

        //wykonanie polecenia
        zamiana(tablica); //przekazując tablicę, podajemy tylko jej nazwę

        //ponowne wypisanie elementów tablicy
        for(int i=0;i<10;i++)
                cout<<tablica[i]<<" ";
}
```

Przykład 15. Tablice – przekazywanie tablicy wielowymiarowej do funkcji

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

void zamiana(int tab[3][3])
{
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      tab[i][j] = tab[i][j] * tab[i][j];
}

int main()
{
  //inicjacja tablicy dwuwymiarowej
  int tablica[3][3] = {{1, 2, 3}, {4, 5, 6}, {-1, -2, -3}};

  //wypisanie elementów tablicy
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      cout<<tablica[i][j]<<" "; //lub tablica[i][j] = tablica[i][j]*tablica[i][j];

  cout<<endl; //wstawienie znaku końca linii (enter)

  //Użycie funkcji
  zamiana(tablica); //przekazując tablicę, podajemy tylko jej nazwę

  //ponowne wypisanie elementów tablicy
  cout<<"Tablica po zmianie elementów: ";
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      cout<<tablica[i][j] <<" "<< tablica[i][j]*tablica[i][j];
}
```

Przykład 16. Tablice – przekazywanie tablicy wielowymiarowej do funkcji (wskaźnik na tablicę)

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

void zamiana(int **tab)
{
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      tab[i][j] = tab[i][j] * tab[i][j];
}

int main()
{
  srand(time(NULL)); //ustawienie ziarna losowania

  //deklarujemy tablicę dynamicznie
  int **tablica = new int *[3];

  for(int i=0;i<3;i++)
    tablica[i] = new int [3];

  //przypisanie do tablicy losowych liczb z przedziału [0..9]
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      tablica[i][j] = rand()%10;

  //wypisanie elementów tablicy
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      cout<<tablica[i][j]<<" ";

  cout<<endl; //wstawienie znaku końca linii (enter)

  //wykonanie polecenia
  zamiana(tablica); //przekazując tablicę, podajemy tylko jej nazwę

  //ponowne wypisanie elementów tablicy
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      cout<<tablica[i][j]<<" ";
}
```

2. Typy wyliczeniowe

Typ wyliczeniowy jest typem zdefiniowanym przez zbiór (zwykle niewielki) nazwanych stałych całkowitych. Zmienne wyliczeniowe są typu całkowitego. Przechowują one wartość będącą elementem pewnego zbioru. Poszczególne elementy zbioru są identyfikowane przez nazwę.

Przykład typu wyliczeniowego o nazwie dni:
enum dni {pon, wto, sro, czw, pia, sob, nie};

Mówiąc prościej, zmienna powyższego typu będzie przybierać JEDYNIE siedem możliwych wartości (dni tygodnia). W nawiasie wąsiastym widoczne są nazwy wartości jakie może przybierać typ wyliczeniowy. Istnieje możliwość ustawienia własnych wartości kolejnych elementów danego typu (domyślnie numeracja jest od 0 z krokiem o 1).

Przykład wartości:
enum dzien {pon = 1, wto, sro, czw, pia, sob, nie};
element PON ma wartość 1, nadaną przez użytkownika, WT ma wartość 2, z kolei NIE ma wartość 7

Możemy jawnie określić typ stałych wyliczeniowych (musi to być typ całkowitoliczbowy).

Przykłady:

```
enum mojTyp {a, b, c, d, e, f, g}; // wartości: 0, 1, 2, 3, 4, 5, 6
enum mojTyp {a=3, b, c, d, e, f, g}; // wartości: 3, 4, 5, 6, 7, 8, 9
enum mojTyp {a, b, c, d=0, e, f, g}; // wartości: 0, 1, 2, 0, 1, 2, 3
enum mojTyp {a=3, b, c=3, d, e=2, f, g=7}; // wartości: 3, 4, 3, 4, 2, 3, 7
enum mojTyp {a=3, b=3, c=3, d=3, e=3, f=3, g=3}; // wartości: 3, 3, 3, 3, 3, 3, 3
enum mojTyp {a=60, b=0, c, d, e, f, g}; // wartości: 60, 0, 1, 2, 3, 4, 5
```

Przykład 1. program korzystający z typu wyliczeniowego bez nazwy

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

enum
{
    pomarancz,
    jablko,
    truskawka = 8,
    banan,
    salatka = pomarancz + jablko  //typ wyliczeniowy enum jest typem arytmetycznym
};

int main()
{
    cout << jablko; //wypisze 1, jabłko to nazwa!
    cout << endl;
    cout << truskawka; //wypisze 8
}
```

Przykład 2. Typ wyliczeniowy i przekazywanie wartości do funkcji

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

enum
{
    kwadrat,
    kolo,
    prostokat = 8,
    trojkat,
    figury = kolo + prostokat
};

void funkcja(int liczba) {
    cout << liczba <<endl;
}

int main()
{
    funkcja(kwadrat);
    funkcja(kolo);
    funkcja(figury);
}
```

Przykład 3. Typ wyliczeniowy - funkcja

```
#include <iostream>
#include <cstdio>
#include <cstdlib>

using namespace std;
enum obiad {OMLETY, PIEROGI, SHAKE, BURGER, PIZZA};

void restauracja (obiad coJemy)
{
    if ( coJemy == OMLETY )
        cout<<"Na obiad są omlety!"<<endl;
    if ( coJemy == PIEROGI )
        cout<<"Ruskie pierogi. Smacznego!"<<endl;
    if ( coJemy == SHAKE )
        cout<<"Tylko shake!"<<endl;
    if ( coJemy == BURGER )
        cout<<"Burger!"<<endl;
    if ( coJemy == PIZZA )
        cout<<"Pizza na obiad!"<<endl;
}

int main ()
{
    obiad dzisiajNaObiad; //deklaracja zmiennej typu obiad (enum)
    dzisiajNaObiad = PIZZA;    //przypisanie wartości zmiennej
```

```
    restauracja(dzisiajNaObiad); //uzycie funkcji restauracja

    dzisiajNaObiad = PIEROGI;
    restauracja(dzisiajNaObiad);

//Jeśli wypiszemy wartość zmiennej bezpośrednio, wówczas pojawi się wartość
liczbowa, czyli np. 1.
    cout<<PIEROGI;
}
```

3. Struktury i zmienne strukturalne

Struktury są konstrukcją pozwalającą przechowywać różne typy danych. Tworzy się je za pomocą kluczowego słowa **struct**. Służy ono do tworzenia zdefiniowanych przez programistę typów.

```
struct nazwaStruktury {
  typ nazwa_elementu;
  typ nazwa_drugiego_elementu;
  typ nazwa_trzeciego_elementu;
  ...
};
```

```
struct osoba {
    string imie;
    string nazwisko;
    int wiek;
} ania, kasia, jan;

//powyżej, prócz deklaracji struktury utworzono również jej obiekty (ania, kasia, jan)
```

Do elementów składowych struktury odwołujemy się za pomocą kropki, podając wcześniej nazwę zmiennej strukturalnej. Po kropce podajemy nazwę składowej struktury.

Przykład 1. Struktura i odwołanie do jej pól

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

// deklaracja struktury
struct osoba
{
    string imie;
    string nazwisko;
    int wiek;
};

int main()
{
    osoba ania; // tworzenie obiektu struktury o nazwie ania
```

```
    ania.imie = "Ania";
    ania.nazwisko = "Kowalska";
    ania.wiek = 43;
    cout << "Twoje imie to: " << ania.imie << endl;
}
```

Istnieje również możliwość utworzenia tablicy struktur. Przydaje się to do tworzenia prostych programów bazodanowych. Tablice struktur tworzy się podobnie jak tablice typów podstawowych.

Przykład 2. Struktura i tablica

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

// deklaracja struktury
struct osoba {
    string imie;
    string nazwisko;
    int wiek;
};

int main()
{
    osoba grupa[5]; // tablica ze struktury
    for (int i = 0; i<5; i++)
    {
        // pobieramy dane
        cout << "Podaj imie osoby numer " << i+1 << endl;
        cin >> grupa[i].imie;
    }

    cout<<endl;

    for (int i = 0; i<5; i++)
    {
        // wyświetlamy tablicę
        cout << grupa[i].imie << endl;
    }
}
```

Struktury mogą być również tworzone w sposób dynamiczny (przy użyciu wskaźników). Przy czym należy pamiętać o zwalnianiu pamięci. W strukturze statycznej używamy kropki odwołując się do pól struktury, natomiast w strukturze dynamicznej używamy operatora strzałki.

Przykład 3. Struktura i wskaźnik

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

struct osoba {
    string imie;
    int wiek;
};

int main()
{
    osoba * pracownik1 = new osoba; // wskaźnik *pracownik1 na strukturę
    pracownik1->imie = "Andrzej";
    cout << pracownik1->imie << endl;
    delete pracownik1; // usuwamy obiekt
}
```

Przykład 4. Struktura i użycie w funkcji

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

//struktura o nazwie kalkulator
struct kalkulator
{
  int suma;
  int roznica;
  long int iloczyn;
  double iloraz;
};

//funkcja zwraca strukturę, więc jest struct kalkulator i nazwa funkcji (oblicz).
Funkcja przyjmuje dwa argumenty - x i y.
struct kalkulator oblicz(int x, int y)
{
  struct kalkulator o;
  o.suma = x+y;
  o.roznica = x-y;
  o.iloczyn = x*y;
  o.iloraz = (double) x/y;  //konwertujemy int na double
  return o;
}

int main()
{
 int  x, y;
 struct kalkulator wynik;
```

```
 cout <<"Podaj x i y: "<<endl;
 cin >>x>>y;
 cin.ignore();

 wynik =  oblicz (x,y); // używamy funkcji oblicz

 cout <<"Wynik dodawania: "<<wynik.suma<<endl;
 cout <<"Wynik odejmowania: "<<wynik.roznica<<endl;
 cout <<"Wynik mnozenia: "<<wynik.iloczyn<<endl;
 cout <<"Wynik dzielenia: "<<wynik.iloraz<<endl;
}
```

Przykład 5. Struktura jako baza danych o pracownikach (struktura w tablicach)

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

struct Pracownik
{
  string imie;
  string nazwisko;
  float pensja;
} pracownicy[30];
// definiujemy strukture Pracownik, jednoczesnie zadeklarowalismy tablice 30-
elementowa

int main()
{
  char odpowiedz;
  int biezacy = 0;

  do
  {
    cout << "Wprowadzamy dane o pracownikach\n";
    cout << "podaj imie:";
    cin >> pracownicy[biezacy].imie;
    cout << "podaj nazwisko:";
    cin >> pracownicy[biezacy].nazwisko;
    cout << "podaj pensje:";
    cin >> pracownicy[biezacy].pensja;
    cout << "Wprowadzona osoba nr " << biezacy << " to:\n";
    cout << pracownicy[biezacy].imie
         << " " << pracownicy[biezacy].nazwisko
         << ", pensja: " << pracownicy[biezacy].pensja
         << endl;
    biezacy++;
    if(biezacy >= 30)
      break;
    cout << "Wprowadzamy dalej (T/N)?";
```

```
    cin >> odpowiedz;
  } while(odpowiedz=='t' or odpowiedz=='T');
}
```

Tablicę struktur tworzymy i odwołujemy się w ten sam sposób co zwykłe tablice prostych zmiennych.
nazwa_struktury nazwa_tablicy [liczba_elementów];

Przykład 6. Tablica struktur

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

struct punkty{
  int x, y;
  char nazwa;
};

int main()
{
  punkty tab[1000];

  //przypisanie wartosci do pierwszej komórki tablicy
  tab[0].x = 2;
  tab[0].y = 4;
  tab[0].nazwa = 'A';

  //przypisanie do ostatniej komórki tablicy
  tab[999].x = 1;
  tab[999].y = 5;
  tab[999].nazwa = 'X';

  //odwoływanie się do elementów tablicy
  cout<<"Dane pierwszego punktu: "<<tab[0].x<<" "
  <<tab[0].y<<" "<<tab[0].nazwa<<endl;
}
```

Przykład 7. Struktura w strukturze

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

//struktura wewnetrzna
struct Data
{
    int dzien;
    int miesiac;
    int rok;
};

//deklaracja struktury
struct DanePersonalne
{
    string nazwisko;
    string imie;
    int pensja;
    string plec;
    Data data;
};

//utworzenie tablicy 100 elementowej
int main()
{
    DanePersonalne tab[ 100 ];;
    tab[ 0 ] = { "Kowalski", "Jan", 2000, "m", { 1, 2, 1980 } };
    tab[ 1 ] = { "Kazimierz", "Pawel", 999, "m", { 31, 1, 1996 } };

    for( int i = 0; i < 2; ++i )
    {
      cout<<tab[ i ].imie<<" ";
      cout<<tab[ i ].nazwisko<<" ";
      cout<<tab[ i ].pensja<<" zł, ";
      cout<<tab[ i ].plec<<" , ";
      cout<<tab[ i ].data.dzien<<" ";
      cout<<tab[ i ].data.miesiac<<" ";
      cout<<tab[ i ].data.rok<<" ";
      cout<<endl;
    }
}
```

Struktura 8. Struktura – wskaźnik (dwie możliwości zapisu)

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

struct Pracownik
{
  string imie;
  string nazwisko;
  float pensja;

};

int main()
{
  Pracownik p;
  Pracownik* wsk = &p;
  Pracownik& referencja = p;

  p.imie = "Krzysztof";
  wsk->nazwisko = "Nowak";
  referencja.pensja = 6543.21;

  cout << "Imie: " << (*wsk).imie
       << "\nNazwisko: " << (*wsk).nazwisko
       << "\nPensja: " << (*wsk).pensja << endl
       //LUB PROSTSZY ZAPIS!
       << "Imie: " << wsk->imie
       << "\nNazwisko: " << wsk->nazwisko
       << "\nPensja: " << wsk->pensja;
}
```

4. Widoczność zmiennych – globalna i lokalna

Zmienne globalne: są zadeklarowane poza blokiem głównym programu (i jakimkolwiek innym). Czas życia zmiennej to okres od utworzenia jej przez system, do zakończenia działania aplikacji. **Jeśli nie nastąpi przysłonięcie, zmienna jest widoczna dla całego programu.**

Zmienne lokalne: są tworzone wewnątrz konkretnego bloku { }. Ich czas życia to czas wykonania instrukcji bloku w którym jest wywołana. Jeśli jest to blok główny programu, to zmienna jest widoczna w całym programie. Kompilator po napotkaniu bloku ze zmienną generuje kod, który alokuje pamięć dla wszystkich zmiennych lokalnych. Widoczność zmiennej (poza sytuacją zasłonięcia) to czas do końca wykonania bloku.

Struktura stworzona przed funkcją main() będzie strukturą globalną, czyli każdy podprogram będzie mógł z niej korzystać. Jeśli stworzymy ją wewnątrz jakiegoś bloku, będzie lokalną i widoczna tylko w tym miejscu.

Przykład 9. Widoczność globalna zmiennej – struktury

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


//struktura globalna widoczna w całym programie (w tym poza funkcją int main)
struct pracownik
{
  char imie[20]; //elementy struktury
  char nazwisko[20];
  int wiek;
  double pensja;
};

int main()
{
  pracownik p1 = {"Ania","Kowalska",44,2000};
}
```

Przykład 10. Widoczność lokalna zmiennej – struktury

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

//struktura widoczna jedynie w funkcji int main

int main()
{
  //struktura lokalna widoczna tylko w main()
  struct samochod{
    char marka[20]; //elementy struktury
    char model[20];
    int rok_produkcji;
    double pojemnosc;
  };

  samochod peugeot = {"Peugeot","407",2013,2.7};
}
```

Przykład 11. Struktury i funkcje

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <sstream>
using namespace std;

struct movies_t //utworzenie struktury
{
  string title;
  int year;
} mine, yours; //utworzenie zmiennych - struktura

void printmovie (movies_t movie); //prototyp funkcji

int main ()
{
  string mystr;

  mine.title = "Odyseja Kosmiczna";
  mine.year = 1968;

  cout << "Wpisz tytuł: ";
  getline (cin,yours.title);
  cout << "Wpisz rok: ";
  cin >> yours.year;

  cout << "Mój ulubiony film:\n ";
  printmovie (mine); //użycie funkcji
  cout << "Twój ulubiony film:\n ";
  printmovie (yours); //użycie funkcji
}

void printmovie (movies_t movie) //funkcja
{
  cout << movie.title;
  cout << " (" << movie.year << ")\n";
}
```