Programowanie w języku C++ (EAR1S03006)

Politechnika Białostocka - Wydział Elektryczny

Automatyka i Robotyka, semestr III, studia stacjonarne I stopnia

Rok akademicki 2021/2022

Zajęcia nr 2 (13.10.2021)

dr inż. Jarosław Forenc

Struktury - deklaracja

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};
```

```
struct zesp
{
    float Re, Im;
};
```

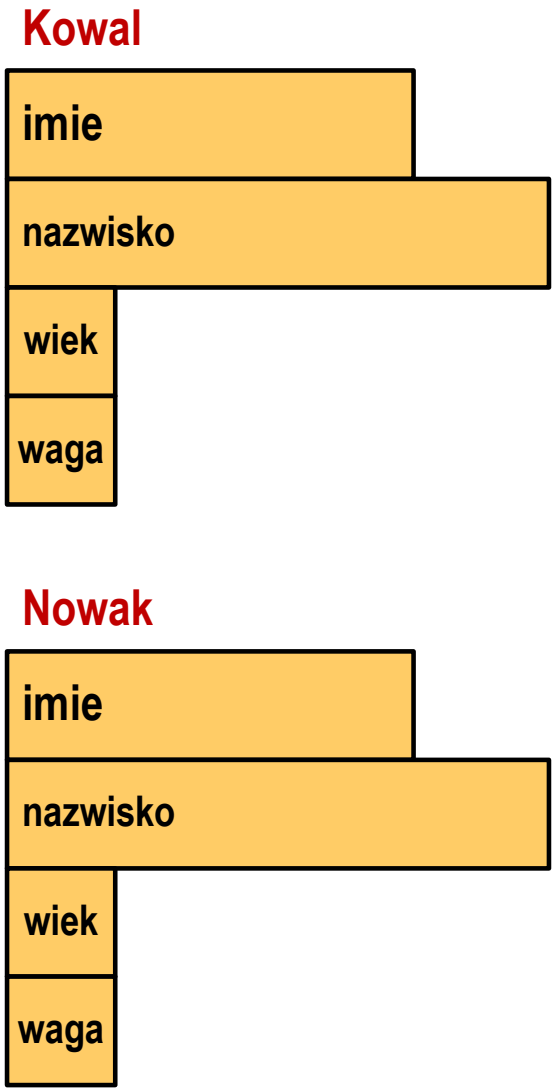
- Deklarując strukturę tworzymy nowy typ danych (struct osoba, struct zesp), którym można posługiwać się tak samo jak każdym innym typem standardowym
- Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci)
- Zapisanie danych do struktury wymaga zdefiniowania zmiennej strukturalnej

Struktury - deklaracja zmiennej strukturalnej

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
} Kowal ;

int main(void)
{
    struct osoba Nowak ;
    ...
}
```



- Kowal, Nowak - zmienne typu struct osoba

Struktury - odwołania do pól struktury

- Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_struktury.nazwa_pola
```

- Zapisanie wartości do pól zmiennej Nowak ma postać

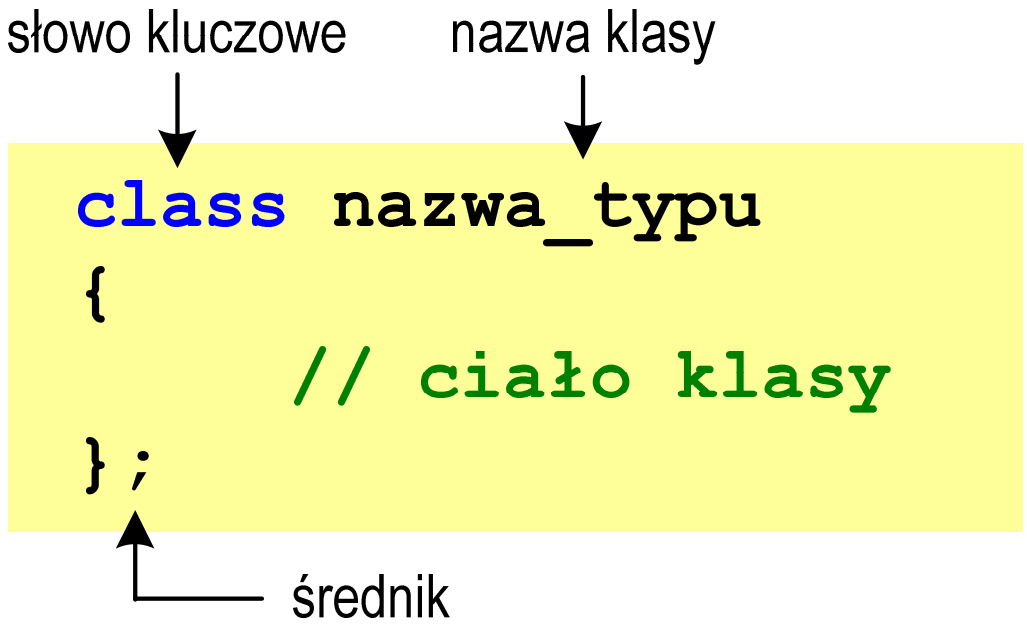
```
Nowak.wiek = 25;
strcpy(Nowak.imie,"Jan");
```

- Gdy zmienna strukturalna jest wskaźnikiem, to do odwołania do pola struktury używamy operatora pośredniego wyboru pola (**->**)

```
wskaźnik_do_struktury -> nazwa_pola
```

```
struct osoba Nowak, *Nowak1 = &Nowak;
Nowak1 -> wiek = 25;
```

Definicja klasy



- zmienne typu nazwa_typu nazywa się obiektami
- utworzenie obiektu wymaga, podobnie jak przy deklaracji innych zmiennych, podania nazwy typu i nazwy obiektu:

  nazwa_typu x; - deklaracja obiektu x klasy nazwa_typu

  nazwa_typu *y; - deklaracja wskaźnika y do obiektów typu nazwa_typu

Składniki klasy - dane

- dane (dane składowe, pola, składniki) - oznaczają to samo co pola w strukturach

```
class osoba
{
 public:
    char imie[20];
    char nazwisko[30];
    int  wiek;
};
```

- do danych w klasie odwołujemy się w taki sam sposób jak do pól struktury:

obiekt.dana
wskaźnik->dana

```
osoba x;
x.wiek = 15;

osoba *y;
y = &x;
y->wiek = 20;
```

mówimy:
„x jest obiektem klasy osoba”
„y jest wskaźnikiem do obiektu klasy osoba”

Składniki klasy - funkcje

- funkcje (funkcje składowe, metody) - są to funkcje operujące na danych składowych klasy

```
class osoba
{
    char imie[20];
    char nazwisko[30];
    int  wiek;
 public:
    void zapisz(char *i, char *n, int w);
};
```

mówimy:
„wywołanie funkcji zapisz na rzecz obiektu x klasy osoba”

- do funkcji w klasie odwołujemy się w taki sam sposób jak do jej danych:

  obiekt.funkcja(argumenty)
  wskaźnik->funkcja(argumenty)

```
osoba x;
x.zapisz("Jan","Nowak",30);

osoba *y = &x;
y->zapisz("Adam","Nowak",25);
```

Składniki klasy - funkcje

- funkcje (funkcje składowe, metody) - są to funkcje operujące na danych składowych klasy

```
class osoba
{
    char imie[20];
    char nazwisko[30];
    int  wiek;
 public:
    void zapisz(char *i, char *n, int w);
};
```

- deklaracje danych i funkcji mogą być umieszczane w klasie w dowolnej kolejności
- niezależnie od miejsca zdefiniowania składnika wewnątrz klasy - składnik znany jest w całej definicji klasy

Prawa dostępu do składników klasy

private (prywatne)

- oznacza, że funkcje i dane klasy dostępne są tylko z wnętrza klasy
- dla danych oznacza to, że tylko funkcje będące składnikami klasy (oraz funkcje zaprzyjaźnione) mogą te dane odczytywać lub do nich coś zapisywać
- dla funkcji oznacza to, że mogą one zostać wywołane tylko przez inne funkcje składowe tej klasy (oraz funkcje zaprzyjaźnione)

public (publiczne)

- komponenty publiczne są ogólnie dostępne, można się do nich odwoływać z wnętrza klasy lub spoza klasy

protected (zabezpieczone)

- dostęp jest taki sam jak dla private, ale dodatkowo są one dostępne dla klas wywodzących się od tej klasy (dziedziczenie)

Prawa dostępu do składników klasy

- etykiety private, public, protected można umieszczać w dowolnej kolejności, mogą one powtarzać się

```
class osoba
{
   char imie[20];
   char nazwisko[30];
   int  wiek;
 public:
   int ocena;
   void zapisz(char *i, char *n);
 private:
   float wzrost;
   float waga;
 public:
   void drukuj();
};
```

- domyślnie (bez podania praw dostępu) wszystkie składowe są prywatne
- funkcje składowe klasy mają dostęp do wszystkich jej danych i funkcji (niezależnie od praw dostępu)

Definiowanie funkcji składowych klasy

```
class nazwa
{
    typ funkcja(parametry)
    {
        kod
    }
};
```

wewnątrz klasy

deklaracja
i definicja funkcji

```
class nazwa
{
    typ funkcja(parametry);
};
```

poza klasą

deklaracja funkcji

```
typ nazwa::funkcja(parametry)
{
    kod
}
```

definicja funkcji

Definiowanie funkcji składowych wewnątrz klasy

```
class osoba
{
   char imie[20];
   char nazwisko[30];
   int  wiek;
 public:
   void zapisz(char *i, char *n, int w)
   {
      strcpy(imie,i);
      strcpy(nazwisko,n);
      wiek = w;
   }
   void drukuj(void)
   {
      cout << imie << " " << nazwisko;
      cout << " " << wiek << endl;
   }
};
```

- funkcja zdefiniowana wewnątrz klasy jest funkcją inline
- podczas kompilacji, w miejscu wywołania funkcji, wstawiany jest jej kod
- ciało funkcji wewnątrz klasy nie powinno mieć więcej niż dwie/trzy linijki kodu
- częste wywołania długich funkcji mogą prowadzić do dużego wzrostu wielkości pliku wynikowego

Definiowanie funkcji składowych poza klasą

```
class osoba
{
    char imie[20];
    char nazwisko[30];
    int  wiek;
 public:
    void zapisz(char *i, char *n, int w);
    void drukuj(void);
};

void osoba::zapisz(char *i, char *n, int w)
{
    strcpy(imie,i);
    strcpy(nazwisko,n);
    wiek = w;
}

void osoba::drukuj(void)
{
    cout << imie << " " << nazwisko;
    cout << " " << wiek << endl;
}
```

deklaracje funkcji

:: - operator zakresu, pokazuje do jakiej klasy należy funkcja

Przykład: klasa osoba

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

class osoba
{
   private:
     char imie[20];
     char nazwisko[30];
     int  wiek;
   public:
     void zapisz(char *i, char *n, int w);
     void drukuj();
};

void osoba::zapisz(char *i, char *n, int w)
{
    strcpy(imie,i);
    strcpy(nazwisko,n);
    wiek = w;
}
```

Przykład: klasa osoba

```
void osoba::drukuj()
{
    cout << imie << " " << nazwisko;
    cout << " " << wiek << endl;
}

int main(void)
{
    osoba os1, os2;

    os1.zapisz("Jan","Kowalski",30);
    os2.zapisz("Anna","Nowak",25);

    os1.drukuj();
    os2.drukuj();

    return 0;
}
```

```
Jan Kowalski 30
Anna Nowak 25
```

Obiekty w pamięci komputera

- definicja klasy nie definiuje obiektu, a więc nie przydziela pamięci

```
class osoba
{
 private:
   char imie[20];
   char nazwisko[30];
   int  wiek;
 public:
   void zapisz(char *i,char *n,int w);
   void drukuj();
};

int main(void)
{
   osoba os1, os2;
   ...
}
```

- definiując kilka obiektów danej klasy w pamięci przydzielane jest miejsce dla wszystkich danych, natomiast funkcje są w pamięci tylko jeden raz
- w definicji klasy nie można inicjować danych(*)

Wskaźnik this

- funkcje wywoływane są zawsze na rzecz konkretnego obiektu
- do wnętrza funkcji przekazywany jest niejawnie wskaźnik do tego obiektu - tym adresem funkcja inicjalizuje swój wskaźnik zwany this
- w rzeczywistości funkcja drukuj():

```
void osoba::drukuj()
{
    cout << imie << " " << nazwisko;
    cout << " " << wiek << endl;
}
```

ma następującą postać:

```
void osoba::drukuj()
{
    cout << this->imie << " " << this->nazwisko;
    cout << " " << this->wiek << endl;
}
```

Modyfikator const

```
class osoba
{
 private:
   char imie[20];
   char nazwisko[30];
   int  wiek;
 public:
   void zapisz(char *i,char *n,int w);
   void drukuj() const;
};

void osoba::drukuj() const
{
    cout << imie << " " << nazwisko;
    cout << " " << wiek << endl;
}
```

- modyfikator const zapewnia, że funkcja nie będzie mogła zmieniać wartości danych składowych klasy
- wskaźnik this przekazywany do takiej funkcji traktowany jest jako stały
- modyfikator const umieszcza się w deklaracji i definicji funkcji

Funkcje zaprzyjaźnione z klasą

- funkcja zaprzyjaźniona z klasą to funkcja, która nie będąc składnikiem klasy ma dostęp do wszystkich (także prywatnych) składników klasy

```
class osoba
{
  private:
    char imie[20];
    char nazwisko[30];
    int  wiek;
  public:
    void zapisz(char *i,char *n,int w);
    friend void funkcja(osoba Nowak);
};

void funkcja(osoba Nowak)
{
    Nowak.wiek = 20;
}
```

- funkcja może „przyjaźnić się” z więcej niż jedną klasą
- nie ma znaczenia, w którym miejscu w klasie pojawia się deklaracja przyjaźni (sekcja private, protected, public)
- funkcja zaprzyjaźniona może być funkcją składową innej klasy
- przyjaźń nie jest dziedziczona