Politechnika Wrocławska

Techniki programowania

INP001002Wl

rok akademicki 2018/19

semestr letni

Wykład 3

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

A-1 p. 411B

Politechnika Wrocławska

Plan prezentacji

- Abstrakcja funkcyjna
- Struktury
- Klasy
  - hermetyzacja
  - konstruktor
  - destruktor
  - operator przypisania
  - konstruktor kopiujący

Na podstawie:

- A. Allain, *Przewodnik dla początkujących C++*
- S. Prata, *Szkoła programowania C++*

Politechnika Wrocławska

Unikanie powtórzeń

- W celu uniknięcia powtórzeń fragmentów kodu wygodnie jest wydzielać funkcje implementujące algorytm lub jego fragment

```
okrag(10, 10, 5)
wypelnijOkrag(10, 10, CZERWONY)

rysujPocisk(10,10)
```

Politechnika Wrocławska

Unikanie powtórzeń

- Powtórzenia mogą dotyczyć nie tylko algorytmów, ale także struktur danych.

```
enum BierkaSzachowa = {PUSTE_POLE,
BIALY_PION, BIALA_WIEZA, /*i pozostałe
bierki*/};

//...mnóstwo kodu

for(int i = 0; i < 8; i++)
      szachownica[i][1] = BIALY_PION;
```

Politechnika Wrocławska

Unikanie powtórzeń

- Powtórzenia mogą dotyczyć nie tylko algorytmów, ale także struktur danych.

```
//...mnóstwo kodu

if( szachownica[0][0] == BIALY_PION ){
    /* zrób coś */
}
```

Politechnika Wrocławska

Abstrakcja funkcyjna

- Szczegóły związane ze strukturą danych można ukryć, wykorzystując abstrakcję funkcyjną.

```
BierkaSzachowa pobierzBierke(int x, int y){
      return szachownica[x][y];
}
```

Politechnika Wrocławska

Abstrakcja funkcyjna

- Funkcja określa, jakie dane wejściowe pobiera oraz jakie dane wyjściowe zwraca, ale nic nie mówi o tym jak jest implementowana.

```
BierkaSzachowa pobierzBierke(int x, int y){
        return szachownica[x][y];
}
```

Politechnika Wrocławska

Abstrakcja funkcyjna

- Zalety stosowania abstrakcji funkcyjnej:
  - nie musisz pamiętać implementacji, możesz korzystać z gotowej funkcji,
  - jeśli odnajdziesz błąd w implementacji, to wystarczy go naprawić w jednym miejscu,
  - zapewniasz sobie elastyczność w implementacji – możesz zacząć od sposobu łatwiejszego w implementacji, ale być może mniej efektywnego, a później zastąpić go wydajniejszym.

Politechnika Wrocławska

Ukrywanie reprezentacji struktur danych

- Nie jest ważne, jak przechowujesz dane, ale co z nimi robisz
- Przykład klasa **`string`**
  - odczytywanie długości łańcucha,
  - odczytywanie i zapis znaków,
  - wyświetlanie na ekran,
  - nie jest istotne jak są przechowywane dane (np. tablice znakowe, listy powiązane).

Użycie funkcji w celu ukrycia układu struktury

```
enum BierkaSzachowa = {PUSTE_POLE,
BIALY_PION, BIALA_WIEZA, /*i pozostałe
bierki*/};
enum KolorGracza = {KG_BIALY, KG_CZARNY};

struct Szachownica{
    BierkaSzachowa plansza[8][8];
    KolorGracza czyj_ruch;
}
```

Użycie funkcji w celu ukrycia układu struktury

```
BierkaSzachowa pobierzBierke( const
Szachownica* w_plansza, int x, int y ){
    return w_plansza->plansza[x][y];
}

KolorGracza pobierzRuch(const Szachownica*
w_plansza){
    return w_plansza->czyj_ruch;
}
```

Użycie funkcji w celu ukrycia układu struktury

```
void wykonajRuch( Szachownica* w_plansza, int
z_x, int z_y, int na_x, int na_y ){
    //tu sprawdzilibyśmy dopuszczalność ruchu
    w_plansza->plansza[na_x][na_y] = \
        w_plansza->plansza[z_x][z_y];
    w_plansza->plansza[z_x][z_y] = \
        PUSTE_POLE;
}
```

Użycie funkcji w celu ukrycia układu struktury

```
Szachownica b;
//inicjalizacja szachownicy
pobierzRuch(&b);
wykonajRuch(&b, 0, 0, 1, 0); //przesuń bierkę
z (0, 0) na (1, 0)
```

- Funkcje są połączone ze strukturą, ponieważ pobierają ją jako argument.

Politechnika Wrocławska

Funkcja jako składowa struktury

- Metoda jest funkcją zadeklarowaną jako część struktury
- Wywołanie metody nastąpi dla instancji struktury

Funkcja jako składowa struktury

```
struct Szachownica{
    BierkaSzachowa plansza[8][8];
    KolorGracza czyj_ruch;

    BierkaSzachowa pobierzBierke( int x, int y ){
        return plansza[x][y];
    }
    KolorGracza pobierzRuch(){
        return czyj_ruch;
    }
    void wykonajRuch(int z_x, int z_y, int na_x, int na_y ){
        //tu sprawdzilibyśmy dopuszczalność ruchu
        plansza[na_x][na_y] = plansza[z_x][z_y];
        plansza[z_x][z_y]   = PUSTE_POLE;
    }
}
```

Funkcja jako składowa struktury

```
Szachownica b;
//inicjalizacja szachownicy
b.pobierzRuch();
b.wykonajRuch(0, 0, 1, 0); //przesuń bierkę z (0, 0) na (1, 0)
```

- Wywołanie metody wygląda niemal jak odwołanie się do pola struktury.

Politechnika Wrocławska

Zewnętrzna definicja metody

```
struct Szachownica{
    BierkaSzachowa plansza[8][8];
    KolorGracza czyj_ruch;

    BierkaSzachowa pobierzBierke( int x, int y );
    KolorGracza pobierzRuch();
    void wykonajRuch(int z_x, int z_y, int na_x, int na_y );
}
```

Politechnika Wrocławska

Zewnętrzna definicja metody

```
BierkaSzachowa Szachownica::pobierzBierke( int x, int y ){
    return plansza[x][y];
}

KolorGracza Szachownica::pobierzRuch(){
    return czyj_ruch;
}

void Szachownica::wykonajRuch( \
    int z_x, int z_y, int na_x, int na_y ){
        //tu sprawdzilibyśmy dopuszczalność ruchu
        plansza[na_x][na_y] = plansza[z_x][z_y];
        plansza[z_x][z_y]   = PUSTE_POLE;
    }
}
```

Politechnika Wrocławska

Klasa

- Klasa przypomina strukturę, ale jest wzbogacona o możliwość określenia, które metody i dane należą do wewnętrznej implementacji klasy, a które są przeznaczone dla jej użytkowników.

Politechnika Wrocławska

Klasa

- W języku C++ można zabronić metodom, które nie należą do danej klasy, korzystania z wewnętrznych danych klasy.
- Ułatwia to pielęgnację oprogramowania – można modyfikować wewnętrzną implementację klasy bez zmiany kodu korzystającego z tej klasy.
- Dostęp do składowych klasy może być prywatny (**`private`**) lub publiczny (**`public`**).

Politechnika Wrocławska

Klasa

```
class Szachownica{
  public:
    BierkaSzachowa pobierzBierke( int x, int y );
    KolorGracza pobierzRuch();
    void wykonajRuch(int z_x, int z_y, int na_x, int na_y );

  private:
    BierkaSzachowa _plansza[8][8];
    KolorGracza _czyj_ruch;
};
```

Politechnika Wrocławska

Klasa

```
BierkaSzachowa Szachownica::pobierzBierke( int x, int y ){
    return _plansza[x][y];
}

KolorGracza Szachownica::pobierzRuch(){
    return _czyj_ruch;
}

void Szachownica::wykonajRuch( \
    int z_x, int z_y, int na_x, int na_y ){
        //tu sprawdzilibyśmy dopuszczalność ruchu
        _plansza[na_x][na_y] = _plansza[z_x][z_y];
        _plansza[z_x][z_y]   = PUSTE_POLE;
    }
}
```

Politechnika Wrocławska

Klasa

```
Szachownica b;
//inicjalizacja szachownicy
b.pobierzRuch();
b.wykonajRuch(0, 0, 1, 0);
//przesuń bierkę z (0, 0) na (1, 0)
```

- Zmienną określonej klasy nazywamy obiektem.

Politechnika Wrocławska

Klasa

- Tworzenie klasy to tworzenie nowego typu danych.
- W pierwszej kolejności określ, co klasa powinna robić – określ interfejs.
- Następnie określ jakich składowych (pól i metod) będziesz potrzebować. Wszystkie metody, które nie muszą być publiczne powinny być prywatne.
- Zalecenie: nie definiuj pól klasy jako publicznych. Jeśli potrzebujesz udostępnić określone pole, stwórz metody, które je udostępniają (setter i getter).

Politechnika Wrocławska

Klasa

- Pola prywatne są przechowywane w pamięci tak samo jak pola publiczne, zwykle tuż obok.
- Pola prywatne nie gwarantują bezpieczeństwa prywatnych danych w przypadku działania złośliwego programu.

Politechnika Wrocławska

Hermetyzacja

- Hermetyzacja (enkapsulacja) oznacza ukrywanie implementacji, dzięki czemu użytkownicy klasy mogą pracować wyłącznie z określonym zbiorem metod, które stanowią interfejs klasy.

Politechnika Wrocławska

Cykl życia klasy

- Istnieją trzy operacje, które (najprawdopodobniej) będzie realizować każda klasa:
  - inicjalizacja,
  - czyszczenie pamięci (lub innych zasobów),
  - kopiowanie samej siebie.

Politechnika Wrocławska

Konstruktor

```
Szachownica plansza;
```

- Konstruktor pozwala na zainicjalizowanie pól obiektu w momencie deklaracji.

Politechnika Wrocławska

Konstruktor

```
class Szachownica{
  public:
    Szachownica(); //konstruktor brak zwracanej wartości
    BierkaSzachowa pobierzBierke( int x, int y );
    KolorGracza pobierzRuch();
    void wykonajRuch(int z_x, int z_y, int na_x, int na_y );

  private:
    BierkaSzachowa _plansza[8][8];
    KolorGracza _czyj_ruch;
};
```

Politechnika Wrocławska

Konstruktor

```
Szachownica::Szachownica(){
    _czyj_ruch = KG_BIALY;

    //czyścimy planszę
    for(int i=0; i<8; i++){
        for(int j=0; j<8; j++){
            _plansza[i][j] = PUSTE_POLE;
        }
    }

    //pozostały kod ustawiający bierki
};
```

Politechnika Wrocławska

Konstruktor

- Konstruktor jest wywoływany przy deklaracji obiektu

```
Szachownica plansza;
```

- Jest także wywoływany podczas alokowania pamięci

```
Szachownica *plansza = new Szachownica;
```

- Jeśli deklarujesz wiele obiektów, to konstruktory wywoływane są w kolejności deklaracji.

```
Szachownica a;
Szachownica b;
```

Politechnika Wrocławska

Konstruktor

- Konstruktory mogą przyjmować argumenty, można je także przeciążać.

```
class Szachownica{
  public:
    Szachownica();
    Szachownica(int rozmiar_planszy);
  //pozostałe składowe klasy
};

Szachownica plansza(8);
//8 jest argumentem konstruktora

Szachownica *plansza = new Szachownica(8);
```

Politechnika Wrocławska

Konstruktor

```
//tak nie można
Szachownica plansza();

//ale to zadziała
Szachownica *plansza = new Szachownica();
```

Politechnika Wrocławska

Konstruktor domyślny

- Jeśli nie utworzysz konstruktora, zostanie stworzony konstruktor domyślny.
- Konstruktor domyślny inicjalizuje wszystkie pola klasy wywołując ich konstruktory (zmienne podstawowe nie zostaną zainicjalizowane).

Politechnika Wrocławska

Konstruktor domyślny

- Jeśli zadeklarujesz (jakikolwiek) konstruktor, nie będzie tworzony konstruktor domyślny.

Politechnika Wrocławska

Inicjalizacja składowych klasy

```
class Szachownica{
  public:
    Szachownica();

    string pobierzRuch();
    BierkaSzachowa pobierzBierke( int x, int y );
    void wykonajRuch(int z_x, int z_y, int na_x, int na_y );

  private:
    BierkaSzachowa _plansza[8][8];
    string _czyj_ruch;
};
```

Politechnika Wrocławska

Inicjalizacja składowych klasy

- Konstruktor

```
Szachownica::Szachownica(){
    _czyj_ruch = "bialy";
    //dalszy kod
}
```

- Konstruktor z listą inicjalizacyjną

```
Szachownica::Szachownica()
    : _czyj_ruch("bialy"){
    //dalszy kod
}
```

Politechnika Wrocławska

Inicjalizacja składowych klasy

```
class Szachownica{
  public:
    Szachownica();

    string pobierzRuch();
    BierkaSzachowa pobierzBierke( int x, int y );
    void wykonajRuch(int z_x, int z_y, int na_x, int na_y );

  private:
    BierkaSzachowa _plansza[8][8];
    string _czyj_ruch;
    int _licznik_ruchow;
};
```

Politechnika Wrocławska

Inicjalizacja składowych klasy

- Konstruktor z listą inicjalizacyjną

```
Szachownica::Szachownica()
     : _czyj_ruch("bialy")
     , _licznik_ruchow( 0 ){
     //dalszy kod
}
```

Inicjalizacja składowych klasy

- Jeśli pole klasy jest zadeklarowane jako stałe, to musi zostać umieszczone na liście inicjalizacyjnej

```
class ListaStalych{
    public:
        ListaStalych(int wart);

    private:
       const int _wart;
};

ListaStalych::ListaStalych(int wart)
    : _wart(wart){
}
```

Politechnika Wrocławska

Inicjalizacja składowych klasy

- Podobnie pole, które jest referencją musi być inicjalizowane na liście

Politechnika Wrocławska

Niszczenie obiektu

- Jeśli konstruktor alokuje pamięć, to zwolnienie jej powinno nastąpić w destruktorze.

Politechnika Wrocławska

Niszczenie obiektu

```
struct WezelListyPowiazanej{
    int wart;
    WezelListyPowiazanej *w_następny;
};

class ListaPowiazana{
  public:
    ListaPowiazana(); //konstruktor
    void wstaw(int wart); //dodanie węzła

  private:
    WezelListyPowiazanej *_w_glowa;
};
```

Politechnika Wrocławska

Niszczenie obiektu

```
class ListaPowiazana{
  public:
    ListaPowiazana(); //konstruktor
    ~ListaPowiazana(); //destruktor
    void wstaw(int wart); //dodanie węzła

  private:
    WezelListyPowiazanej *_w_glowa;
};
```

Politechnika Wrocławska

Niszczenie obiektu

```
ListaPowiazana::~ListaPowiazana(){
    WezelListyPowiazanej *w_itr = _w_glowa;
    while( w_itr != NULL )
    {
        WezelListyPowiazanej *w_tymcz = w_itr->w_następny;
        delete w_itr;
        w_itr = w_tymcz;
    }
}
```

Politechnika Wrocławska

Niszczenie obiektu

```
class WezelListyPowiazanej{
  public:
    ~WezelListyPowiazanej();
    int wart;
    WezelListyPowiazanej *w_nastepny;
};

WezelListyPowiazanej::~WezelListyPowiazanej(){
    delete w_nastepny;
}

ListaPowiazana::~ListaPowiazana(){
    delete _w_glowa;
}
```

Politechnika Wrocławska

Niszczenie obiektu

- Destruktor jest wywoływany gdy:
  - usunięto wskaźnik do tego obiektu,
  - obiekt wyszedł poza zasięg,
  - obiekt należy do klasy, której destruktor wywołano.

Politechnika Wrocławska

Niszczenie obiektu

```
ListaPowiazana *w_lista = new ListaPowiazana;
delete w_lista;
```

Politechnika Wrocławska

Niszczenie obiektu

```
if(1){
    ListaPowiazana lista;
}//tu jest wywołany destruktor

{
    ListaPowiazana a;
    ListaPowiazana b;
}//tu są wywołane destruktory
//najpierw niszczony jest obiekt b
//później a
```

Politechnika Wrocławska

Niszczenie obiektu

```
class NazwaOrazEmail{
    //metody klasy

  private:
    string _nazwa;
    string _email;
};
```

Politechnika Wrocławska

Kopiowanie obiektów

```
ListaPowiazana lista_jeden;
ListaPowiazana lista_dwa;

lista_dwa = lista_jeden;
ListaPowiazana lista_trzy = lista_dwa;
```

Politechnika Wrocławska

Kopiowanie obiektów

- Istnieje domyślny operator przypisania
- Wykorzystanie domyślnego operatora przypisania może prowadzić do powstania płytkiej kopii

```
lista_dwa = lista_jeden;
//lista_dwa._w_glowa = lista_jeden._w_glowa;
```

Politechnika Wrocławska

Kopiowanie obiektów

```
lista_dwa = lista_jeden;
```

- Implementacja operatora przypisania

```
ListaPowiazana& operator= (ListaPowiazana&
lewa, const ListaPowiazana& prawa);
```

Politechnika Wrocławska

Kopiowanie obiektów

- Implementacja operatora przypisania wewnątrz klasy

```
class ListaPowiazana{
  public:
    ListaPowiazana(); //konstruktor
    ~ListaPowiazana(); //destruktor
    ListaPowiazana& operator= (const ListaPowiazana& inna);

    void wstaw(int wart); //dodanie węzła

  private:
    WezelListyPowiazanej *_w_glowa;
};
```

Politechnika Wrocławska

Kopiowanie obiektów

```
ListaPowiazana& ListaPowiazana::operator= (const
                                        ListaPowiazana& inna)
{
   if(this == &inna){
      return *this;
   }

   delete _w_glowa;
   _w_glowa = NULL;

   WezelListyPowiazanej *w_itr = inna._w_glowa;
   while (w_itr != NULL){
      wstaw( w_itr->wart );
      w_itr = w_itr->w_następny;
   }
   return *this;
};
```

Politechnika Wrocławska

Konstruktor kopiujący

```
ListaPowiazana lista_jeden;
ListaPowiazana lista_dwa(lista_jeden);
```

Politechnika Wrocławska

Kopiowanie obiektów

- Implementacja operatora przypisania wewnątrz klasy

```
class ListaPowiazana{
  public:
    ListaPowiazana(); //konstruktor
    ~ListaPowiazana(); //destruktor
    ListaPowiazana& operator= (const ListaPowiazana& inna);
    ListaPowiazana (const ListaPowiazana& inna);

    void wstaw(int wart); //dodanie węzła

  private:
    WezelListyPowiazanej *_w_glowa;
};
```

Politechnika Wrocławska

Kopiowanie obiektów

```
ListaPowiazana::ListaPowiazana (const ListaPowiazana& inna)
     : _w_glowa(NULL)
{
    WezelListyPowiazanej *w_itr = inna._w_glowa;
    while (w_itr != NULL)
    {
        wstaw( w_itr->wart );
        w_itr = w_itr->w_następny;
    }
};
```

Politechnika Wrocławska

Metody generowane automatycznie

1. Domyślny konstruktor
2. Domyślny destruktor
3. Operator przypisania
4. Konstruktor kopiujący

Politechnika Wrocławska

Podsumowanie

- Abstrakcja funkcyjna
- Struktury
- Klasy
  - hermetyzacja
  - konstruktor
  - destruktor
  - operator przypisania
  - konstruktor kopiujący