C++: Klasy cz. 1

Zajęcia laboratoryjne nr 1 - Metodyka i Techniki Programowania II

Autor instrukcji: dr inż. Andrzej Staniszewski

Obiekty i programowanie obiektowe

Obiekt to przedmiot ze świata rzeczywistego, taki jak książka, plik czy wiertarka lub telefon. Gdy piszesz program to wykorzystuje on **zmienne** (proste i/lub złożone) do przechowywania „przedmiotów" ze świata rzeczywistego, takich jak książki czy pracownicy, opisanych w zmiennych jako zbiór: cech, atrybutów, ilości i czynności.

W programowaniu obiektowym „widzisz" świat rzeczywisty, którego fragment chcesz opisać w programie, jako zbiór przedmiotów tworzących pewien system i operacje, które możesz przeprowadzać na tych przedmiotach. Np. obiekt *__plik__* może mieć operacje **utworzenia**, **modyfikacji**, **wyświetlenia**, **drukowania** i **usuwania**.

Klasy

C++ jako język obiektowy umożliwia definiowanie nowych typów danych, które zawierają nie tylko dane, ale także operacje jakie na tych danych można wykonywać. W C++ obiekty definiuje się poprzez użycie **klasy**. **Klasa** jest uogólnieniem typu, i jest podstawowym pojęciem programowania obiektowego. Liczba możliwych do zdefiniowania klas jest nieograniczona. **Klasa** jest typem (szablonem) **obiektu**. **Klasa** daje możliwość zdefiniowania wszystkich atrybutów (cech) obiektu. Dlatego definicja **klasy** jest bardzo podobna do definicji struktury, bo grupuje składowe reprezentujące **atrybuty** (dane) obiektu. Lecz **klasa** zawiera coś więcej: **funkcje** operujące na danych obiektu. **Funkcje** te zwane są **metodami**.

W definicji klasy wyróżnia się cztery elementy:

1. Zbiór danych składowych, zwanych **polami** (ang. data members),
2. Zbiór funkcji składowych, zwanych **metodami** (ang. member functions),
3. Poziomy dostępu do składowych (ang. access levels),

4.Nazwę klasy (ang. class tag name).

Zatem tak jak dla struktury, definicja klasy w języku C++ musi zawierać niepowtarzalną nazwę, a za nią klamrę otwierającą, dalej składowe: pola (dane) i metody, i klamrę zamykającą:

```
class nazwa_klasy {
        int dana;    //pole
        void wypisz_skladowa (int);   //metoda
};
```

Po zdefiniowaniu **klasy** można zadeklarować **zmienne** typu tej klasy, czyli **obiekty**, następującą deklaracją:

```
nazwa_klasy obiekt_1, obiekt_2, obiekt_3;
```

Program 1 – tworzenie obiektów klasy:

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

// Tworzenie obiektow klasy (tu: pracownik)

class pracownik {
public:
     char imie_nazwisko[64];
     long ident_prac;
     float pensja;
     void inf_o_prac(void) {
     cout << "Imie i nazwisko: " << imie_nazwisko << endl;
     cout << "Identyfikator: " << ident_prac << endl;
     cout << "Pensja: " << pensja << endl;
     };
};

int main(void) {

     pracownik kierownik, sekretarka;

     strcpy (kierownik.imie_nazwisko, "Jan Kowalski");
     kierownik.ident_prac = 101;
     kierownik.pensja = 8400;

     strcpy (sekretarka.imie_nazwisko, "Balbina Wykrot");
     sekretarka.ident_prac = 1234567;
```

```
    sekretarka.pensja = 3000;

    kierownik.inf_o_prac();
    sekretarka.inf_o_prac();
}
```

Program 2 – deklarowanie metody klasy na zewnątrz:

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

//Deklarowanie metody klasy na zewnatrz klasy

class psy {
public:
    char rasa[30];
    int srednia_waga;
    int srednia_wysokosc;
    void inf_o_psie(void);
    };

void psy::inf_o_psie(void) {
    cout << "Rasa: " << rasa << endl;
    cout << "Srednia waga: " << srednia_waga << endl;
    cout << "Srednia wysokosc: " << srednia_wysokosc << endl;
}

int main(void) {

    psy balbina, fred;

    strcpy (balbina.rasa, "Dalmatynczyk");
    balbina.srednia_waga = 58;
    balbina.srednia_wysokosc = 24;

    strcpy (fred.rasa, "Owczarek szetlandzki");
    fred.srednia_waga = 22;
    fred.srednia_wysokosc = 15;

    balbina.inf_o_psie();
    fred.inf_o_psie();
}
```

Składowe publiczne lub prywatne

Składowe klasy mogą być **publiczne** lub **prywatne** – jest to określenie w programie uprawnień dostępu do składowych.

Metody klasy mogą być definiowane wewnątrz klasy. Jednak w przypadku długich i skomplikowanych funkcji będą one zaciemniały definicję klasy i utrudniały czytanie kodu programu. Dlatego metody klasy można definiować na zewnątrz klasy. Ich nazwy muszą być wtedy poprzedzone nazwą klasy i **operatorem widoczności ::**

Metody, które chronią dostęp do danych klasy noszą nazwę **funkcji interfejsu**.

Program 3 – składowe publiczne i prywatne:

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

//Skladowe publiczne i prywatne

class pracownik {
public:
     int wprowadz(char *, long, float);
     void inf_o_prac(void);
     int zmien_pensje(float);
     long podaj_id(void);
private:
     char imie_nazwisko[64];
     long ident_prac;
     float zarobki;
     };

int pracownik::wprowadz(char* nazwisko_prac, long id_prac, float zarobki_prac) {
     strcpy (imie_nazwisko, nazwisko_prac);
     ident_prac = id_prac;
     zarobki = zarobki_prac;
     return(0);
}

void pracownik::inf_o_prac(void) {
     cout << "Imie i nazwisko: " << imie_nazwisko << endl;
     cout << "Identyfikator: " << ident_prac << endl;
     cout << "Zarobki: " << zarobki << endl;
}
```

```
int pracownik::zmien_pensje(float nowa_pensja) {
     if (nowa_pensja < 15000) {
          zarobki = nowa_pensja;
          return(0);
     }
     else
          return(-1);
}

long pracownik::podaj_id(void) {
     return (ident_prac);
}

int main(void) {

     pracownik informatyk;

     if (informatyk.wprowadz("Jan Kowalski", 101101, 4000) == 0){
          cout << "Wartosci przypisane skladowym dla pracownika:" << endl;
               informatyk.inf_o_prac();
          }
    if (informatyk.zmien_pensje(5000.00) == 0) {
          cout << "---------------- \nNowa pensja pracownika" << endl;
          informatyk.inf_o_prac();
          }
     else
          cout << "Nieporawne zarobki!" << endl;

}
```

Konstruktor i destruktor klasy

Konstruktor i destruktor to specjalne metody klasy, które są automatycznie wywoływane podczas tworzenia lub usuwania obiektów.

Konstruktor to metoda klasy, która ułatwia nadawanie wartości początkowych danym klasy. Konstruktor nosi taką samą nazwę jak klasa.

Destruktor to funkcja, która zwalnia pamięć operacyjną zajmowaną przez obiekt. Destruktor nosi nazwę klasy poprzedzoną znakiem ~ (tylda): ~nazwa_klasy. Gdy programy w obiektach przydzielają pamięc, to destruktor staje się wygodną metodą zwalniania pamięci przy usuwaniu obiektów.

Konstruktor i destruktor **nie zwracają** żadnych wartości.

Konstruktor ułatwia utworzenie obiektu. Destruktor umożliwia usunięcie obiektu. Konstruktor i destruktor są funkcjami działającymi na obiektach z którymi są związane.

Program 4 – konstruktor klasy:

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

//Konstruktor klasy

class pracownik {
public:
     pracownik(const char *, long, float);
     void inf_o_prac(void);
     int zmien_pensje(float);
     long podaj_id(void);
private:
     char imie_nazwisko[64];
     long ident_pracownika;
     float zarobki;
     };

pracownik::pracownik(const char* imie_nazwisko, long ident_prac, float zarobki) {
     strcpy (pracownik::imie_nazwisko, imie_nazwisko);
     pracownik::ident_pracownika = ident_prac;
     if (zarobki < 15000)
     pracownik::zarobki = zarobki;
     else
          pracownik::zarobki = 0.0;
}

void pracownik::inf_o_prac(void) {
     cout << "Imie i nazwisko: " << imie_nazwisko << endl;
     cout << "Identyfikator: " << ident_pracownika << endl;
     cout << "Zarobki: " << zarobki << endl;
}


int main(void) {

     pracownik informatyk("Jan Kowalski", 101101, 4000.0);
```

```
    informatyk.inf_o_prac();
}
```

Program 5 – destruktor klasy:

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

//Destruktor

class pracownik {
public:
    pracownik(const char *, long, float);
    ~pracownik(void);
    void inf_o_prac(void);
    int zmien_pensje(float);
    long podaj_id(void);
private:
    char imie_nazwisko[64];
    long ident_pracownika;
    float zarobki;
    };

pracownik::pracownik(const char* imie_nazwisko, long ident_prac, float zarobki) {
    strcpy (pracownik::imie_nazwisko, imie_nazwisko);
    pracownik::ident_pracownika = ident_prac;
    if (zarobki < 15000)
    pracownik::zarobki = zarobki;
    else
        pracownik::zarobki = 0.0;
}

pracownik::~pracownik(void) {
    cout << "Usuwam obiekt: " << imie_nazwisko << endl;
}

void pracownik::inf_o_prac(void) {
    cout << "Imie i nazwisko: " << imie_nazwisko << endl;
    cout << "Identyfikator: " << ident_pracownika << endl;
    cout << "Zarobki: " << zarobki << endl;
}

int main(void) {
```

```
    pracownik informatyk("Jan Kowalski", 101101, 4000.0);
    informatyk.inf_o_prac();

}
```

Programy obiektowe koncentrują się na obiektach i na operacjach na tych obiektach.

Zadania:

1.Zapoznaj się z kodem i przeanalizuj załączone programy. Skompiluj je i zapoznaj się z wynikami ich działania. Powróć do analizy ich kodów.

2.Spróbuj przepisać programy wg własnego projektu. Zacznij chociażby od wprowadzenia komentarzy i zmiany nazw zmiennych.

3.Zastanów się nad możliwością rozbudowy programów. Zacznij chociażby od zwiększenia liczby obiektów na których działa program. Skompiluj program. Czy otrzymałeś wyniki analogiczne do wyników pierwotnego programu?

4.Umieść programy w swoim repozytorium, zachowując konwencje nazw. Pamiętaj o komentarzach w kodzie programu! Nie mogą to być komentarze trywialne, ale merytoryczne.

Literatura:

1.Bjarne Stroustrup, „Język C++”, wydanie dowolne,

2.Stanley B. Lippman, „Podstawy języka C++”, WNT, Warszawa,

3.Kris Jamsa, „Wygraj z C++”, Wyd. Mikom, Warszawa.