Wprowadzenie do programowania obiektowego w języku C++

Spis treści

W pracy opisano podstwowe mechanizmy związane z programowaniem obiektowym w języku C++. Opis ma na celu zobrazowanie wybranych możliwości i problemów przy zastosowaniu uproszczeń, które siłą rzeczy prowadzą do braku ścisłości.

Przyjęto założenie biegłości czytelnika w zakresie programowania w języku C.

Programowanie w języku C++ jest szczególnie rozpowszechnione wszędzie tam, gdzie realizowane są duże projekty, a w pracach bierze udział grupa programistów. Język ten, będący rozszerzeniem języka C, jest zorientowany obiektowo.

1 Klasy

Podstawowym pojęciem programowania obiektowego jest klasa (`class`). Przypomina ona strukturę (`struct`) stosowaną w języku C ale może grupować nie tylko zmienne, ale również funkcje (zwane metodami) należące do klasy.

Klasę o nazwie `name` deklaruje się w następujący sposób:

```
class name{
  ...
};
```

2 Metody

Funkcje należące do klasy, przetwarzające dane należące do klasy nazywane są metodami. Metody deklaruje się i czasami definiuje w bloku deklaracji klasy.

Przykład:

Krzywe elementarne służące do kreślenia figur geometrycznych zawsze zawierają jeden punkt charakterystyczny. Klasa zawierająca dane dotyczące pojedynczego punktu może być zadeklarowana w sposób następujący:

```
class point{
  int _x,_y;
  public:
    void set_x(int x){_x=x;}
    void set_y(int y){_y=y;}
    int  get_x(){return(_x);}
    int  get_y(){return(_y);}
};
```

Dostęp do zmiennych `_x` i `_y` uzyskuje się wyłącznie za pośrednictwem metod `set_x`, `set_y`, `get_x` i `get_y`. Podobnie jak zmienne, metody podlegają ochronie przed dostępem. Wszystkie przedstawione w przykładzie metody są dostępne z poziomu programu głównego, ponieważ zostały poprzedzone instrukcją `public`.

Zasady ochrony opisano w rozdziale 'Ochrona danych'.

3 Konstruktor

W przedstawionym przed chwilą przykładzie pominięto bardzo ważny problem. Przed pierwszym użyciem zadeklarowane zmienne `_x` i `_y` zawierają nieznaną, przypadkową wartość.

Język C++ pozwala na nadanie wartości zmiennym podczas tworzenia klasy. Służy temu celowi konstruktor, czyli specyficzna metoda, której nazwa jest tożsama z nazwą klasy.

```
class point{
  int _x,_y;
  public:
    void set_x(int x){_x=x;}
    void set_y(int y){_y=y;}
    int  get_x(){return(_x);}
    int  get_y(){return(_y);}
         point(){_x=_y=0;}
};
```

Konstruktor jest wywoływany po utworzeniu obiektu.

4 Destruktor

Po użyciu obiektu, gdy nie jest on już więcej potrzebny należy zwolnić zajmowane przezeń zasoby. Dokonuje się tego za pomocą destruktora. Zwolnienie pamięci następuje po wykonaniu wszelkich instrukcji zawartej wewnątrz funkcji. W związku z tym, że destruktor jest bezwarunkowo wywoływany podczas likwidacji obiektu, nie może on przyjmować argumentów.

Destruktor dla przedstawionego wcześniej przykładu może wyglądać następująco:

```
...
```

```
  ~point(){}
...
```

5 Ochrona danych

Ochrona danych, inaczej hermetyzacja (*ang.* encapsulation), pozwala na ograniczanie dostępu do określonych danych i metod. Zmienne i metody umieszczone w bloku deklaracji klasy nie są domyślnie dostępne dla innych klas (również klas potomnych) i funkcji. Można to zmienić używając słów kluczowych: `public`, zezwalając na swobodny dostęp, `protected` zezwalając wyłącznie na dostęp z klas pochodnych, albo `private` przywracając ochronę danych.

Niżej przedstawiona deklaracja ukazuje w jaki sposób należy używać odpowiednich poleceń:

```
class name{
  public:
    int x;
  protected:
    int y;
  private:
    int z;
};
```

6 Dziedziczenie

Dziedziczenie (*ang.* inheritance) pozwala na tworzenie nowych obiektów w oparciu o obiekty już istniejące. Nowe obiekty 'dziedziczą' zmienne i metody określone jako `public` albo `protected`. Dziedziczenie jest oznaczane znakiem dwukropka ':'.

Przykładem dziedziczenia może być utworzenie obiektu opisującego okrąg. Okrąg może być opisany trzema wartościami: współrzędnymi punktu centralnego oraz długością promienia okręgu. Wykorzystany więc zostanie wcześniej zdefiniowany obiekt opisujący punkt `point`.

```
class circle : public point{
  int _r;
  public:
    void set_r(int r){_r=r;}
    int  get_r(){return(_r);}
         circle(){_r=0;}
};
```

Utworzenie obiektu `circle` powoduje jego inicjalizację, tak więc wszystkie zmienne `_x`, `_y` i `_r` od tej chwili zawierają prawidłowe wartości. Nowe wartości można przypisać tym zmiennym wykorzystując funkcje `set_x`, `set_y` i `set_r`.

Obiekt może również dziedziczyć metody i zmienne po kilku obiektach. W takim przypadku nazwy obiektów bazowych rozdziela się przecinakami, np.

```
class circle : public point, public colour{
  ...
};
```

7 Polimorfizm

Polimorfizm, czyli wielopostaciowość (*ang.* polymorphism) pozwala na tworzenie metod, których zachowanie jest zależne od sposobu wywołania. Na podstawie typów i liczby argumentów przekazywanych do funkcji kompilator decyduje, której 'odmiany' metody należy użyć w danej sytuacji.

Niżej przedstawiony kod stanowi odmianę wcześniej zastosowanego, przy czym w tym przypadku wartości zmiennych odczytuje się oraz przypisuje się za pomocą metod o jednakowej nazwie.

```
class point{
  int _x,_y;
  public:
    void xvalue(int x){_x=x;}
    int  xvalue(){return(_x);}
    void yvalue(int y){_y=y;}
    int  yvalue(){return(_y);}
};
```

8 Tworzenie i usuwanie obiektów

W zależności od potrzeb obiekty mogą być tworzone na dwa różne sposoby:

8.1 Metoda statyczna

```
point p;
```

Dostęp do obiektu uzyskuje się w sposób następujący:

```
p.set_x(1);
```

8.2 Metoda dynamiczna

Drugi sposób jest bardziej elastyczny. Stosuje się do tego celu instrukcję `new`. Adres obiektu zapamiętywany jest we wskaźniku do obiektu:

```
point * p=new point;
```

Dostęp do metod uzyskuje się w sposób następujący:

```
p->set_x(1);
```

Po wykorzystaniu obiekt można usunąć z pamięci stosując instrukcję `delete`:

```
delete p;
```

9 Kompilacja programu

Programy zawierające kod przedstawiony w niniejszej pracy kompilowano w systemie NetBSD z użyciem instrukcji

```
g++ -ansi -pedantic -Wall program.cc -o program
```

10 Podsumowanie

Programowanie zorientowane obiektowo wymaga zachowania odpowiedniej dyscypliny i skłania do pisania programów dobrze udokumentowanych. Nowe cechy języka C++, jego nowe mechanizmy trudne do uzyskania w języku C, pozwalają na zmniejszenie liczby występujących błędów.

Znaczna kompatybilność języka C++ z C pozwala na programowanie mieszane obiektowo - strukturalne, a więc m. in. na wykorzystanie bibliotek obiektowych w programach strukturalnych.

Zalety języka C++ są okupione znacznie większym czasem kompilacji i często widoczną ociężałością pracy programu wynikowego.

2001, Igor Skalski