Struktury i pary

#struktura #struct
#.
#konstruktor
#para #pair

Często się zdarza, że musimy połączyć w jednym obiekcie dane różnych typów. Wtedy powinniśmy skorzystać ze struktury danych, która nazywa się po prostu *struktura* (ang. **structure**, wymawiaj: *strakczer*) i jest obecna w języku C od najdawniejszych czasów. Szczególnym przypadkiem struktury składającej się tylko z dwóch elementów jest *para* (ang. **pair**, wymawiaj: *per*). Obydwa te kontenery omówimy w tym podrozdziale.

Pojęcie struktury zostało na przestrzeni lat znacznie rozwinięte i stanowi kanwę programowania obiektowego. W zasadzie struktury w obecnym ich rozumieniu to nic innego, jak klasy o wszystkich elementach publicznych.

Struktury

Strukturę można traktować po prostu jako specyficzny złożony typ danych, zdefiniowany przez programistę. Załóżmy, że chcielibyśmy utworzyć taki typ o nazwie `person` (z angielskiego: *osoba*) i zawrzeć w nim informację o imieniu (ang. **first name**, wymawiaj: *ferst nejm*), nazwisku (ang. **last name**, wymawiaj: *last nejm*) oraz roku urodzenia (ang. **year**, wymawiaj: *jer*).

Dwie pierwsze składowe* struktury to zmienne typu `string`, a trzecia to liczba całkowita (`int`). Definicja naszej struktury powinna wyglądać następująco:

```
struct person
{
  string first_name, last_name;
  int year;
};
```

Proszę zwrócić uwagę, że po deklaracji struktury jest znak średnika `;`. Deklaracje poszczególnych składowych wyglądają jak deklaracje zwykłych zmiennych. Aha, jedna uwaga: definicja struktury powinna być umieszczona poza jakąkolwiek funkcją czy procedurą, no i oczywiście przed pierwszym użyciem.†

Identyfikatora `person` używamy dokładnie tak samo, jak nazwy typu. Oto przykład deklaracji zmiennej tego typu i nadania wartości jej składowym:

*W programowaniu obiektowym użylibyśmy raczej nazwy „pola”.
†W programowaniu obiektowym zamiast słowa kluczowego `struct` występuje słowo kluczowe `class`.

```
person x;
x.first_name = "Jan";
x.last_name = "Kowalski";
x.year = 2001;
```

Warto zwrócić uwagę na operator kropki `.` umożliwiający dostęp do składowych struktury.

Niby jest OK, ale nie wygląda to na najwygodniejszą metodę nadawania wartości. Chciałoby się, aby wartości składowych podać bezpośrednio w deklaracji zmiennej, na przykład:

```
person y("Ewa", "Nowak", 2004);
```

Żeby to zadziałało, musimy zdefiniować we wnętrzu struktury funkcję zwaną *konstruktorem*, której zadaniem jest nadanie wartości składowym. Nazwa konstruktora jest zawsze taka sama, jak nazwa struktury.[‡] Oto poprawiona definicja struktury `person`:

```
struct person
{
  string first_name, last_name;
  int year;

  person(string fn, string ln, int yr)
  {
    first_name = fn;
    last_name = ln;
    year = yr;
  }
};
```

Co ciekawe (ale skądinąd logiczne, dlaczego?) nie podaje się typu zwracanej wartości, choć konstruktor jest funkcją.

W danej strukturze można zdefiniować dowolną ilość konstruktorów – wedle potrzeb programisty. Wszystkie mają taką samą nazwę, lecz muszą różnić się zestawem argumentów.

Pary

Para to kontener zawierający tylko dwa pola – dowolnych typów. Deklaracja zmiennej tego typu przypomina nieco deklaracje omawianych już kontenerów, gdyż zawiera nawiasy kątowe. Aby używać tego kontenera należy dodać dyrektywę `#include <utility>` na początku programu.

Oto przykład deklaracji pary o pierwszym elemencie typu `string`, a drugim typu `int`:

```
pair<string, int> p;
```

Zmienna *p* ma dwie składowe (dwa pola) o nazwie `first` (wymawiaj: *ferst*) oraz `second` (wymawiaj: *sekend*). Odwołujemy się do nich tak samo, jak do elementów struktury, na przykład:

[‡]Lub klasy, jeśli mówimy o programowaniu obiektowym.

```
p.first = "Paragraph";
p.second = 22;
```

Tym razem mamy gotowy konstruktor, zatem możemy wartości składowych pary nadawać bezpośrednio w deklaracji:

```
pair<string, int> q("Slaughterhouse", 5);
```

Do nadania wartości składowym pary można także użyć funkcji `make_pair()`, na przykład:

```
pair<string, int> r = make_pair("King Matt", 1);
```

Pary są często wykorzystywanymi kontenerami, choćby dlatego, że są gotowe do użycia. Ponadto większe kontenery zawierające pary łatwo jest posortować – w przypadku struktur trzeba się odrobinę postarać, co opiszemy w następnym podrozdziale.