Struktury

Struktura jest elementem języka, służącym do grupowania informacji. Aby rozważyć celowość stosowania struktur przeanalizujmy następujący przykład. Spis telefonów zawiera dla każdej osoby (prywatnej) następujące dane: imię, nazwisko, skrót adresu oraz numer telefonu. Aby zapamiętać w komputerze dane z książki telefonicznej potrzebowalibyśmy kilku tablic: tablicy imion, nazwisk, adresów oraz numerów telefonu. Dogodniejsze jest jednak zgrupowanie informacji odnośnie imienia, nazwiska, adresu i numeru telefonu jednej osoby w zwarty blok - strukturę. Możemy wtedy utworzyć pojedynczą tablicę struktur, celem zebrania informacji o wszystkich wymaganych do zapamiętania osobach. Stosowanie struktur czyni bardziej przejrzystą konstrukcję programu, pozwala również uniknąć pewnych błędów.

Przeanalizujmy następujący program (plik *struktury.cpp*)

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

// Deklaracje struktur:

// Ta struktura ma nazwe, ale nie definiuje swoich zmiennych.
// Zrobimy to pozniej i tak najczesciej sie robi
struct karta
{
   int figura;
   int kolor;
};          // Tu KONIECZNIE musi byc srednik

// Struktura moze nie miec nazwy, ale musi wtedy miec zdefiniowane zmienne.
struct
{
   char imie[50];
   char nazwisko[50];
} rozdajacy = {"Wielki", "Szu"}; // Lista definicji KONCZY sie SREDNIKiem

// Mozliwy jest rowniez wariant mieszany: podana nazwa I definicje zmiennych

// Prototypy funkcji uzytych w programie
// UWAGA!!! prototypy sa umieszczone PO deklaracjach struktur!!!
// W deklaracjach funkcji korzystamy ze struktury karta, podczas deklaracji
// funkcji kompilator musi juz znac deklaracje "karty"
void wypisz(karta k);
karta najnizsza();

int main()
{
// Definicje zmiennych
karta k1 = {2, 3}, k2;

   cout<<"Program ilustruje operacje na strukturach: "<<endl;

   // Odczytanie pola struktury
   cout<<"Dziesiejszym rozdajacym jest " << rozdajacy.imie <<
            " " << rozdajacy.nazwisko << endl << endl;

   cout<<"Zmodyfikujemy karte k2... "<<endl;
   // Wpisanie wartosci do pola struktury
   k2.figura = 14;
   k2.kolor = 4;

   // Wypiszmy wartosci kart k1 oraz k2
   cout<<"Karta k1:"<<endl;
   wypisz(k1);
   cout<<"Karta k2:"<<endl;
   wypisz(k2);
   cout<<endl;

   cout<<"Mozliwe sa przypisania struktur jako calosci..."<<endl;
   // Przypisanie jednej strukturze drugiej
   k2 = k1;

   cout<<"Karta k1:"<<endl;
```

```
    wypisz(k1);
    cout<<"Karta k2:"<<endl;
    wypisz(k2);
    cout<<endl;

    cout<<"Mozliwe jest przypisanie wyniku funkcji strukturze..."<<endl;
    // Przypisanie strukturze wyniku funkcji (musi zachodzic zgodnosc typow)
    k2 = najnizsza();
    wypisz(k2);
    cout<<endl;
}

// funkcja moze przyjac strukture jako swoj parametr
void wypisz(karta k)
{
    if( (k.figura>=2) && (k.figura<=10)) cout<<k.figura<<" ";
    else
    switch(k.figura)
    {
            case 11:  cout<<"Walet ";
                              break;

            case 12:  cout<<"Dama ";
                              break;

            case 13:  cout<<"Krol ";
                              break;

            case 14:  cout<<"As ";
                              break;

            default:  cout<<"Bledna karta (zla figura) ";
    }

    switch(k.kolor)
    {
            case 1:   cout<<"Pik"<<endl;
                      break;

            case 2:   cout<<"Trefl"<<endl;
                      break;

            case 3:   cout<<"Karo"<<endl;
                      break;

            case 4:   cout<<"Kier"<<endl;
                      break;

            default:cout<<"Bledna karta (zly kolor)"<<endl;
    }
}

// Funkcja moze zwrocic strukture
karta najnizsza()
{
karta wynik;

    wynik.kolor = 1;
    wynik.figura = 2;
    return wynik;
}
```

Powyższy program ilustruje sposoby używania struktur. W programie została zdefiniowana struktura o nazwie *„karta“*, opisująca kartę do gry. Każda z kart w talii ma dwie cechy: figurę (czy jest to król, as, 10 itd.) oraz kolor (pik, trefl, karo lub kier). Nasuwa się zatem pomysł, aby zmienne zawierające informacje o tych dwóch cechach karty zgrupować ze sobą. Przyjrzyjmy się definicji struktury *„karta“*:

```
struct karta
{
    int figura;
    int kolor;
```

```
};
```

Deklarując strukturę tworzymy nowy, złożony typ danych języka. Deklaracja struktury rozpoczyna się obowiązkowym słowem kluczowym *struct*. Po nim może nastąpić nazwa tworzonego typu strukturalnego, następnie w bloku, ograniczonym nawiasami klamrowymi następują deklaracje zmiennych składowych struktury, które będą służyły do przechowywania informacji. Po nawiasie zamykającym mogą pojawić się definicje zmiennych naszego nowo zadeklarowanego typu strukturalnego (lub typu pochodnego np. wskaźnika do tworzonego typu strukturalnego), oddzielonych przecinkami, po definicjach musi pojawić się średnik. Podanie nazwy tworzonego typu strukturalnego oraz definicji zmiennych nie jest obowiązkowe, jednak co najmniej jeden z tych elementów musi się pojawić: nazwa typu strukturalnego lub definicja co najmniej jednej zmiennej. Najczęściej stosowanym rozwiązaniem jest podanie nazwy typu strukturalnego i pominięcie definicji zmiennych – zmienne zostają utworzone w dalszej części programu.

Aby zdefiniować zmienną typu strukturalnego, należy podać nazwę typu strukturalnego, a następnie nazwę zmiennej. Przykładem może być następujący fragment powyższego programu:

```
karta k1 = {2, 3}, k2;
```

Struktura *k1* jest typu *karta*, inicjalizowana jest wartościami dla poszczególnych składowych struktury, podanymi w kolejności ich występowania w deklaracji typu strukturalnego (tutaj: figura=1, kolor=2), inicjalizację zmiennej można pominąć (patrz definicja struktury *k2*).

Podstrawową operacją na zmiennej typu strukturalnego jest odwołaniue się do pojedynczej składowej struktury. Służy do tego operator . (kropka), którego używa się następująco:

zmienna_typu_strukturalnego.nazwa_składowej

np. ***k2.figura = 14;*** lub ***k2.figura = k1.figura;***

Istotne informacje na temat użytkowania struktur:

- można dokonać przypisania jednej struktury drugiej operatorem =
- funkcja może przyjąć strukturę jako swój argument.
- funkcja może zwrócić strukturę jako swoją wartość.
- dwie poprzednie właściwości wymagają znajomości typu strukturalnego przez kompilator, dlatego deklaracje funkcji, używających danego typu strukturalnego muszą pojawić się w tekście programu po deklaracji tego typu.

Rozważmy teraz kolejny listing (plik *fizycy.cpp*)

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

struct fizyk
{
   char imie[50];
   char nazwisko[50];
   char dziedzina[200];
};

int main()
{
// Tablica z danymi
fizyk tablica[4]= { "Albert", "Einstein", "teorie wzglednosci",
                    "Richard" , "Feynman", "elektrodynamika kwantowa",
                    "Erwin", "Schrodinger", "mechanika kwantowa"};

   cout<<"Program demonstruje zastosowanie struktur w tablicach"<<endl<<endl;
   cout<<"Dopiszmy sobie jeszcze jednego fizyka do tablicy:"<<endl;

   cout<<"Podaj imie : ";
   cin.getline( tablica[3].imie , 50);
   cout<<"Podaj nazwisko : ";
   cin.getline( tablica[3].nazwisko, 50);
   cout<<"Podaj specjalnosc : ";
   cin.getline(tablica[3].dziedzina, 200);

   cout<<endl<<"Wypiszmy dane z bazy informacji o fizykach:"<<endl<<endl;
```

```
    // zmienna i beda indeksowane komorki tablicy
    int i;

    for(i=0; i<4; i++)
    {
            cout << "*** rekord nr " << i << " ***" <<endl;
            cout << "Imie : " << tablica[i].imie << endl;
            cout << "Nazwisko : " << tablica[i].nazwisko << endl;
            cout << "Specjalizacja : " << tablica[i].dziedzina << endl;

            // no i przejscie do nowej linii
            cout << endl;
    }
}
```

Celem programu jest zaprezentowanie sposobu używania tablic struktur. Rozważmy następujący zapis:

tablica[i].nazwisko

Wyrażenie *tablica[i]* oznacza „element tablicy o indeksie *i*“. Elementem tym jest struktura. Celem uzyskania dostępu do składowej tej struktury, stawiamy kropkę po nawiasie zamykającym i dopisujemy nazwę składowej. Zatem rozważany zapis w całości oznacza „pole *nazwisko* w strukturze, będącej elementem tablicy o indeksie *i*“.

W programie pojawia się jeszcze jeden nowy element. Do tej pory do wczytywania wartości, podawanych przez użytkownika, używaliśmy zawsze instrukcji *cin>>zmienna*. Ten sposób realizacji pobierania wartości od użytkownika ma jedną wadę: jeśli zmienna będzie tablicą znaków, to ze strumienia wejściowego odczytany będzie jedynie pierwszy wyraz, gdyż strumień potraktuje spacje jako separatory kolejnych wartości, wprowadzonych przez użytkownika. Aby uniknąć tej niedogodności użyta została instrukcja:

cin.getline(tablica[3].nazwisko, 50);

Z jednej strony instrukcja ta przypomina wywołanie funkcji, oprócz tego pojawia się kropka, związana ze strukturami oraz strumień *cin*. Do wyjaśnienia znaczenia powyższej instrukcji potrzebne jest powiedzenie paru zdań na temat programowania obiektowego. Polega ono na definiowaniu klas, będących zgrupowaniem zmiennych, zwanych *„polami“* (podobnie jak w strukturze) i *„metod“* – funkcji przeznaczonych do operowania na tych zmiennych. Dla przykładu klasa o nazwie „kwadrat“, mająca reprezentować tę figurę w programie do rysowania np. CorelDRAW, może mieć następujące pola: współrzędne narożników, typ wypełnienia i typ obramowania. Przykładowe metody takiej klasy to „rysuj“ – rysująca kwadrat na ekranie, „przesuń“ – przesuwająca kwadrat o pewien wektor, czy „obróć“ – obracającą kwadrat o zadany kąt. Metody mogą dokonywać modyfikacji pól, mogą je odczytywać itd. Część metod udostępniona jest do wywoływania z zewnątrz klasy. Gdy mamy zdefiniowaną klasę, możemy tworzyć „obiekty“, które można traktować jako „zmienne typu klasy“. Strumień *cin* jest obiektem klasy standardowej o nazwie *„istream“* – strumień wejściowy. Operator kropki język *C++* przejął z języka C. Musiało wydawać się naturalne rozszerzenie jego znaczenia na wybranie jednej z metod klasy, podobnie jak składowej w strukturze. Z drugiej strony „metoda“ w programowaniu obiektowym to pełnoprawna funkcja, więc wywołanie metody powinno wyglądać tak, jak wywołanie funkcji. Stąd w rozważanym zapisie występuje zarówno kropka, jak i nawiasy, oznaczające wywołania funkcji. Powyższa instrukcja służy zatem wywołaniu metody *getline* na rzecz obiektu *cin*. Zadaniem metody *getline* jest pobranie z klawiatury linii tekstu. Jest to metoda trzyargumentowa: pierwszy argument to tablica, do której ma być wczytany tekst (dokładniej: wskaźnik do typu znakowego, wskazujący tę tablicę), drugi to liczba znaków, jakie może pomieścić tablica znakowa, w której chcemy przechowywać tekst, zaś trzeci parametr to znak końca wprowadzanego tekstu – jest on domniemany jako znak końca linii. Jeśli użytkownik wpisze zbyt długi tekst, metoda *getline* obetnie nadmiarowe znaki, uwzględniając odliczenie od podanego rozmiaru tablicy jednej pozycji na znak *NULL*.

Zadanie

Proszę napisać program, który wczyta do tablicy struktur o rozmiarze nie przekraczającym 10 pozycji dane: imię, nazwisko i numer telefonu (rodzaj mini-książki telefonicznej), a następnie wypisze te dane na ekran.

Proszę wykonać kserokopię tabeli priorytetów operatorów języka C++ z książki „*Język C++*”, autorstwa Bjarne Stroustrup (tabela „Podsumowanie operatorów”, str. 134-135)