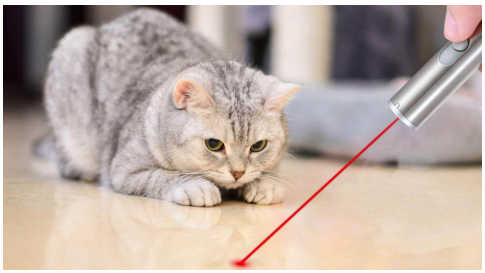
Wskaźniki i iteratory



#wskaźnik
#iterator

W tym podrozdziale omówimy specjalne typy zmiennych, których zadaniem jest wskazywanie adresów (położenia) innych zmiennych. Od początku w języku C istniały *wskaźniki* i pełniły bardzo ważną rolę przy tworzeniu programów. Wraz z programowaniem obiektowym pojawiły się *iteratory*, będące rozwinięciem idei wskaźników. Dodane zostały specyficzne ograniczenia w ich funkcjonalności, które powodują, że unika się dotkliwych błędów, jak to bywało w przypadku niefrasobliwego używania klasycznych wskaźników. Mimo to, zdaniem Kocurra, warto chyba kilka słów o nich powiedzieć, aby wprowadzić Czytelników we właściwy klimat.

Wskaźniki

Wskaźnik (ang. **pointer**) to zmienna, której wartością jest adres innej zmiennej w pamięci komputera. W deklaracji zmiennej wskaźnikowej przed identyfikatorem zmiennej pojawia się symbol gwiazdki `*`, na przykład poniżej mamy deklarację wskaźnika, który może wskazywać na zmienne typu `int`:*

```
int *p;
```

Ten wskaźnik nie wskazuje jeszcze na nic sensownego (na ogół będzie miał losową wartość), ale już poniżej zacznie wskazywać na zmienną n (symbol `&` oznacza adres zmiennej):†

```
int n = 5;
p = &n;
```

Od tej pory symbol $*p$ oznacza to samo, co n, na przykład:

```
cout << *p << endl;  // pojawi się 5
n++;
cout << *p << endl;  // pojawi się 6
(*p)++;
cout << n << endl;   // pojawi się 7
```

Nawiasy otaczające $*p$ są konieczne, bo operator inkrementacji `++` ma bardzo wysoki priorytet.

Wspominaliśmy, że zmienne tablicowe są zarazem wskaźnikami do swojego początkowego elementu, tak jest w istocie, jeśli mamy deklarację:

*Wskaźniki i iteratory wskazujące na zmienne różnych typów są także różnych typów.
†Ampersand `&` ma kilka znaczeń w języku C++: wspomniany adres zmiennej, referencję oraz iloczyn bitowy.

```
int a[10];
```

to wtedy $*a$ to jest to samo, co $a[0]$. Ponadto w języku C mamy tak zwaną arytmetykę wskaźników, to znaczy, że $a+1$ pokazuje na kolejny element tablicy, czyli że $*(a+1)$ znaczy to samo, co $a[1]$.[‡] Można nawet spotkać się z twierdzeniem, że tablice i wskaźniki to jedno i to samo. Nie do końca tak jest, bo zmiennej tablicowej (w powyższym przykładzie a) nie możemy nadać nowej wartości, bo jest ona wskaźnikiem stałym (ale oczywiście możemy dowolnie zmieniać zawartość jej komórek), natomiast zmienna wskaźnikowa nie ma tych ograniczeń.

Wskaźniki wymagają pewnej ostrożności w stosowaniu, bo nie działają tu żadne reguły poprawności, więc łatwo wygenerować wskaźnik, który pokazuje gdzieś w hektary, nie wiadomo gdzie.

Iteratory

Młodszymi braćmi wskaźników są *iteratory* działające na kontenerach definiowanych w bibliotece STL. Jak ich sama nazwa wskazuje, generalnie służą one do iterowania, czyli spacerowania po danej strukturze. W zasadzie są one bardzo podobne do wskaźników, ale mają wbudowane rozmaite ograniczenia (zależnie od rodzaju iteratora).

Już poznaliśmy dwa iteratory: `begin()` oraz `end()`. Wskazują one odpowiednio na początkowy element kontenera, albo na następne miejsce za jego ostatnim elementem. (To ostatnie określenie jest raczej umowne.) Szukając analogii ze wskaźnikami, możemy stwierdzić, że na przykład dla wektora:

```
vector<int> V(5);
```

wyrażenie $*(V.begin())$ oznacza to samo, co $V[0]$, natomiast element $*(V.end())$ nie istnieje.

Zawartość wektora do tej pory wypisywaliśmy w ten sposób:

```
for(int i = 0; i < V.size(); i++)
  cout << V[i] << endl;
```

A jak wyglądałoby takie wypisanie z użyciem iteratora? Jakiego w ogóle jest on typu? Tu niestety zaczynają się małe schody, no powiedzmy: schodki. Typ iteratora jest związany z typem kontenera (tutaj: `vector`), ale także zależy od typu zapisanego w nawiasach kątowych. Jeśli chcemy zadeklarować zmienną p, która mogłaby być iteratorem dla powyższego wektora, powinno wygladać to tak:

```
vector<int>::iterator p;
```

Tajemnicze dwa dwukropki oznaczają przynależność do klasy `vector` (z tym dodatkiem w nawiasach kątowych).

Zmienna p powinna zacząć od wartości $V.begin()$, ale nie powinna osiągnąć wartości $V.end()$, ponieważ wskazuje ona poza zawartość wektora. Do przechodzenia do następnego elementu

[‡]Iteratory mają (na ogół) tę samą własność.

użyjemy operatora `++` (pominiemy tutaj implementacyjne szczegóły, ważne że działa). Z kolei element wskazywany przez iterator p to $*p$ i to właśnie należy wypisać:

```
for(p = V.begin(); p != V.end(); p++)
  cout << *p << endl;
```

Rezultat będzie dokładnie taki sam, jak w poprzednim przypadku.

Kontenery z biblioteki STL mają to do siebie, że powyższa konstrukcja działa bez zmian dla każdego z nich,[§] tylko zamiast `vector` trzeba użyć `deque`, `set`, `map` i tak dalej.

W standardzie C++11 (lub nowszym) przeglądanie kontenerów jest istotnie uproszczone i bardzo wygodne. Ponadto wprowadzono tam kontener `array`, aby nawet na zwykłych tablicach o stałym rozmiarze można było operować w taki sam sposób. Zajmiemy się tym już niedługo.

[§]Poza kontenerem `pair`.