Wskaźniki

Wygodnie jest wyobrażać sobie pamięć operacyjną jako coś w rodzaju wielkiej tablicy bajtów. W takiej strukturze każdy element (nazwiemy go komórką) powinien dać się jednoznacznie identyfikować poprzez swój indeks. I tutaj rzeczywiście tak jest – numer danego bajta w pamięci nazywamy jego adresem.
W ten sposób dochodzimy też do pojęcia wskaźnika:

Wskaźnik (ang. pointer) jest adresem pojedynczej komórki pamięci operacyjnej.

Jest to więc w istocie liczba, interpretowana jako unikalny indeks danego miejsca w pamięci. Specjalne znaczenie ma tu jedynie wartość zero, interpretowana jako wskaźnik pusty (ang. null pointer), czyli nie odnoszący się do żadnej konkretnej komórki pamięci.
Wskaźniki służą więc jako łącza do określonych miejsc w pamięci operacyjnej; poprzez nie możemy odwoływać się do tychże miejsc.

Wskaźniki na zmienne

Wskaźnik jest przede wszystkim liczbą - adresem w pamięci, i w takiej też postaci istnieje w programie. Język C++ ma ponadto ścisłe wymagania dotyczące kontroli typów i z tego powodu każdy wskaźnik musi mieć dodatkowo określony typ, na jaki wskazuje. Innymi słowy, kompilator musi znać odpowiedź na pytanie: „Jakiego rodzaju jest zmienna, na którą pokazuje dany wskaźnik?”. Dzięki temu potrafi zachowywać kontrolę nad typami danych w podobny sposób, w jaki czyni to w stosunku do zwykłych zmiennych.

Zobaczmy teraz elementarny przykład deklaracji oraz użycia wskaźnika:

```
// deklaracja zmiennej typu int oraz wskaźnika na zmienne tego typu
int zmienna = 10;

int* wskaznik; // nasz wskaźnik na zmienne typu int

// przypisanie adresu zmiennej do naszego wskaźnika i użycie go do
// wyświetlenia jej wartości w konsoli
wskaznik = &zmienna; // wskaznik odnosi się teraz do zmienna
cout << *wskaznik; // otrzymamy 10, czyli wartość zmiennej
```

Dobra wiadomość jest taka, iż mimo prostoty ilustruje on większość zagadnień związanych ze wskaźnikami na zmiennej.

Oczywiście najpierw mamy deklarację zmiennej (z inicjalizacją), lecz nas interesuje bardziej sposób zadeklarowania wskaźnika, czyli:

```
int* wskaznik;
```

Poprzez dodanie gwiazdki (*) do nazwy typu int informujemy kompilator, że oto nie ma już do czynienia ze zwykłą zmienną liczbową, ale ze wskaźnikiem przeznaczonym do przechowywania adresu takiej zmiennej. `wskaznik` jest więc wskaźnikiem na zmienne typu int, lub, krócej, wskaźnikiem na (typ) int.

A zatem mamy już zmienną, mamy i wskaźnik. Przydałoby się zmusić je teraz do współpracy: niech `wskaznik` zacznie odnosić się do naszej zmiennej! Aby tak było, musimy pobrać jej adres i przypisać go do wskaźnika - o tak:

```
wskaznik=&zmienna;
```

Zastosowany tutaj operator & służy właśnie w tym celu - do uzyskania adresu miejsca w pamięci, gdzie egzystuje zmienna. Potem rzecz jasna zostaje on zapisany w `wskaznik`; odtąd wskazuje on więc na zmienną zmienna.

Na koniec widzimy jeszcze, że za pośrednictwem wskaźnika możemy dostać się do zmiennej i użyć jej w ten sam sposób, jaki znaliśmy dotychczas, choćby do wypisania jej wartości w oknie konsoli:

```
cout << *wskaznik;
```

Z pewnością domyślamy się, że operator * nie dokonuje tutaj mnożenia, lecz podejmuje wartość zmiennej, z którą połączony został wskaznik; nazywamy to dereferencją wskaźnika. W jej wyniku otrzymujemy na ekranie liczbę, którą oryginalnie przypisaliśmy do zmiennej `zmienna`. Bez zastosowania wspomnianego operatora zobaczyliśmy wartość wskaźnika (a więc adres komórki w pamięci), nie zaś wartość zmiennej, na którą on pokazuje. To oczywiście wielka różnica.

Zaprezentowana próbka kodu faktycznie realizuje zatem zadanie wyświetlenia wartości zmiennej zmienna w iście okrężny sposób. Zamiast bezpośredniego przesłania jej do strumienia wyjścia posługujemy się w tym celu dodatkowym pośrednikiem w postaci wskaźnika.
Samo w sobie może to budzić wątpliwości co do sensowności korzystania ze wskaźników. Pomyślmy jednak, że mając wskaźnik możemy umożliwić dostęp do danej zmiennej z jakiegokolwiek miejsca programu - na przykład z funkcji, do której przekażemy go jako parametr (w końcu to tylko liczba!). Potrafimy wtedy zaprogramować każdą czynność (algorytm) i zapewnić jej wykonanie w stosunku do dowolnej ilości zmiennych, pisząc odpowiedni kod tylko raz.

Deklaracje wskaźników

Stwierdziliśmy, że wskaźniki mogą z powodzeniem odnosić się do zmiennych - albo ogólnie mówiąc, do danych w programie. Czynią to poprzez przechowywanie numeru odpowiedniej komórki w pamięci, a zatem pewnej wartości. Sprawia to, że wskaźniki są w rzeczy samej także zmiennymi.

Wskaźniki w C++ to zmienne należące do specjalnych typów wskaźnikowych. Taki typ łatwo poznać po obecności przynajmniej jednej gwiazdki w jego nazwie. Jest nim więc choćby i`nt* - typ zmiennej wskaznik` z poprzedniego przykładu. Zawiera on jednocześnie informację, na jaki rodzaj danych będzie nasz wskaźnik pokazywał - tutaj jest to `int`. Typ wskaźnikowy jest więc typem pochodnym, zdefiniowanym na podstawie jednego z już wcześniej istniejących.

Deklarowanie wskaźników jest zatem niczym innym, jak tylko wprowadzeniem do kodu nowych zmiennych - tyle tylko, iż mają one swoiste przeznaczenie, inne niż reszta ich licznych współbraci. Czynność ich deklarowania, a także same typy wskaźnikowe zasługują przeto na szersze omówienie.

Dowiedzieliśmy się już, że pisząc gwiazdkę po nazwie jakiegoś typu, uzyskujemy odpowiedni wskaźnik na ten typ. Potem możemy użyć go, deklarując właściwy wskaźnik; co więcej, możliwe jest uczynienie tego aż na cztery sposoby:

```
int* wskaznik;
int *wskaznik;
int*wskaznik;
int * wskaznik;
```

Widać więc, że owa gwiazdka „nie trzyma się” kurczowo nazwy typu (tutaj `int`) i może nawet oddzielać go od nazwy deklarowanej zmiennej, bez potrzeby użycia w tym celu spacji.

Zadanie 1.
Stworzyć plik `wskaznik1.cpp` i wpisać w nim następujący kod:

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


int main()
{
   int liczba = 10;   // deklaracja zmiennej i przypisanie wartości w znany do tej pory sposob
   int *wsk;         // deklaracja zmiennej wskaźnikowej o nazwie wsk
   wsk = &liczba;    //przypisanie do zmiennej wsk adresu zmiennej liczba
```

```
   cout << "Zadeklarowana liczba: " << liczba << endl;
   cout << "Jej adres (zapisany kodem heksadecymalnym) w pamieci komputera to: " << wsk << endl;
   cout << "Jej adres (rozkodowany) w pamieci komputera to: " << (unsigned long)wsk << endl;
   cout << "wyluskana za pomoca wskaznika wartość liczby to " << *wsk <<endl;
   system("pause");
   return 0;
}
```

następnie skompilować plik oraz uruchomić program.

Zadanie 2.
Zmodyfikować powyższy program następująco:

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

int main()
{
   int liczba = 10;
   int *wsk;
   wsk = &liczba;
   cout << "Zadeklarowana liczba: " << liczba << endl;
   *wsk = 15;
   cout << "Po zmianie mamy: " << endl;
   cout << "Zadeklarowana liczba: " << liczba << endl;
   cout << "Liczba wyluskana wskaznikiem : " << *wsk << endl;
   system("pause");
   return 0;
}
```

następnie skompilować plik oraz uruchomić program.

Zalety stosowania wskaźników.

Rozważmy następujący program:

```
#include <iostream>
using namespace std;
void zmien_wartosc(int a)
  {
    a=5;
  }
int main()
{
   int liczba = 10;
   cout << "Zadeklarowana liczba: " << liczba << endl;
```

```
   zmien_wartosc(liczba);
   cout << "Po wywolaniu funkcji mamy: " << endl;
   cout << "Zadeklarowana liczba: " << liczba << endl;
   system("pause");
   return 0;
}
```

Niestety nie otrzymaliśmy tego czego oczekiwaliśmy. Liczba 10 nie została zamieniona na liczbę 5. Pytanie dlaczego tak się stało. Odpowiedź na to pytanie wiąże się ze sposobem przekazywania argumentu do funkcji. W naszym przypadku do funkcji `zmien_wartosc` przekazywana jest wartość zmiennej `liczba`, czyli 10 (nazywamy tę operację przekazywaniem przez wartość), a dokładniej kopia tej zmiennej. A zatem wszystko co się dzieje wewnątrz naszej funkcji dotyczy kopii, a nie oryginału. Powoduje to, że funkcja zmienia wartość kopii zmiennej, a nie oryginału. Dlatego też wartość oryginalna zmiennej `liczba` to nadal 10 i taka wartość powtórnie zostanie wyświetlona na ekranie monitora.
Co zatem zrobić, aby nasz program działał poprawnie. Domyślamy się, trzeba w jakiś inny sposób przekazać argument do funkcji. Z pomocą przychodzą tutaj wskaźniki. Wówczas działamy wewnątrz funkcji już na adresie zmiennej, a nie na jej wartości, czyli odwołujemy się do konkretnej zmiennej (zmienna ma przypisany jeden adres). W naszym przypadku odwołamy się do zmiennej `liczba` (nie do kopii) poprzez jej adres, `i` zmienimy jej wartość na 5. W wywołaniu funkcji należy użyć operatora `&`. Zrealizujemy to w następujący sposób:

```
#include <iostream>
using namespace std;
void zmien_wartosc(int *a)
  {
    *a=5;
      }
int main()
{
   int liczba = 10;
   cout << "Zadeklarowana liczba: " << liczba << endl;
   zmien_wartosc(&liczba);
   cout << "Po wywolaniu funkcji mamy: " << endl;
   cout << "Zadeklarowana liczba: " << liczba << endl;
   system("pause");
   return 0;
}
```

Kolejną zaletą stosowania wskaźników jest możliwość zwracania (ale bez użycia słowa `return`) przez funkcję więcej niż jednej wartości. Rozważmy następujący kod:

```
#include <iostream>
using namespace std;
void suma_iloczyn(int a, int b, int *suma, int *iloczyn)
  {
    *suma = a + b;
    *iloczyn = a * b;
      }
int main()
```

```
{
   int liczba1 = 10;
   int liczba2 = 6;
   int suma=0;
   int iloczyn=0;
   suma_iloczyn(liczba1, liczba2, &suma, &iloczyn);
   cout << "Suma liczb wynosi " << suma <<" , a iloczyn "<< iloczyn << endl;
   system("pause");
   return 0;
}
```

Widzimy, że funkcja `suma_iloczyn` oblicza sumę i iloczyn liczb a i b przy użyciu wskaźników. Należy w definicji funkcji zadeklarować wskaźnik na zmienne `suma, iloczyn`, a następnie wyłuskać wartość sumy i iloczynu za pomocą operatora *. Gdybyśmy parametry `suma` oraz `iloczyn` przekazali do funkcji poprzez wartość, to nie uzyskalibyśmy takiego efektu (znów działalibyśmy na kopii, a nie na oryginale, czyli w wyniku otrzymalibyśmy wartości oryginalne sumy i iloczynu, czyli 0).

Zadanie 3.
Napisz funkcję, która oblicza pierwiastki trójmianu kwadratowego i wypisuje je na ekranie monitora.