Wskaźniki

Wyobraźmy sobie pamięć komputera, jako taką schematyczną kratownicę. W polach kratownicy zapisywane są zmienne, które zostały zdefiniowane w programie. Każda kratka ma swój unikalny adres, dzięki któremu możemy odczytać wartość zmiennej. Wskaźniki przechowują właśnie te adresy.

Definicja wskaźnika jest podobna do definicji zmiennej:

```
int *wsk; // definicja wskaznika
```

Jest to wskaźnik o nazwie *wsk* który wskazuje na adresy komórek w których są zapisane zmienne typu int. Różnica polega na tym, że przy nazwie wskaźnika jest gwiazdka *. Ona odróżnia definicję zmiennej od nazwy wskaźnika.
Zdefiniujmy sobie teraz zmienną typu int o nazwie numer:

```
int numer;
```

Ta zmienna ma zarezerwowany w pamięci pewien obszar, posiadający unikalny adres. Na razie nie znamy tego adresu, ale dzięki wskaźnikowi już niedługo go poznamy. Wskaźniki przechowują adresy, czyli do wskaźnika należy przypisać adres zmiennej numer. Robimy to w następujący sposób:

```
wskaznik = &numer; // ustawiamy wskaźnik na komórkę ze zmienną numer
```

Za wskaźnik o nazwie *wskaznik* podstaw ADRES zmiennej o nazwie numer. Jeżeli się dobrze wpatrzymy to znak & będzie nam przypominał literę a jak adres. Teraz nasz wskaźnik wskazuje na komórkę pamięci, w której zapisana jest zmienna numer. Wypiszmy ten adres:

```
cout << "Adres zmiennej numer to " << wskaznik; // wypisujemy adres komórki
```

Tak właśnie wygląda adres komórki pamięci, teraz pojawia się pytanie: znam adres komórki pamięci, w której jest zapisana zmienna numer…tylko po co mi to wiedzieć… Jeżeli znamy adres komórki, to możemy wykonywać operacje na tej zmiennej. Aby dowiedzieć się ile wynosi zmienna, zapisana pod danym adresem wystarczy napisać:

```
int *wskaznik; // definicja wskaznika
	int numer =5;
	wskaznik = &numer; // ustawiamy wskaźnik na komórkę ze zmienną numer
	cout << "Adres zmiennej numer to " << wskaznik<<endl; // wypisujemy adres komórki
	cout << "Wartosc zmiennej wynosi " << *wskaznik<<endl; // przed nazwą wskaźnika dajemy gwiazdke
```

Aby poznać wartość zmiennej kryjącej się pod adresem na który wskazuje wskaźnik, należy użyć gwiazdki *.

```
	int numer;
	int *wskaznik;

	cout << "Podaj wartosc zmiennej numer: ";
	cin >> numer;
```

```
	wskaznik = &numer; // ustawiamy wskaźnik na komórkę ze zmienną numer
	cout << "Adres zmiennej numer to " << wskaznik << endl; // wypisujemy adres
komórki
	cout << "Wartosc zmiennej wynosi " << *wskaznik<<endl; // przed nazwą wskaźnika
dajemy gwiazdkę.
```

Zobaczmy co się stanie ze zmienną, jeżeli do wyrażenia **wskaznik* dodamy 1.

```
	int numer;
	int *wskaznik;

	cout << "Podaj wartosc zmiennej numer: ";
	cin >> numer;

	wskaznik = &numer; // ustawiamy wskaźnik na komórkę ze zmienną numer

	cout << "Adres zmiennej numer to " << wskaznik << endl; // wypisujemy adres
komórki
	cout << "Wartosc zmiennej wynosi " << *wskaznik << endl; // przed nazwą
wskaźnika dajemy gwiazdkę.
	cout << "Teraz dodamy jeden do wyrazenia *wskaznik :" << endl << endl;

	(*wskaznik)++; // całość musi być w nawiasie

	cout << "Adres zmiennej numer to " << wskaznik << endl; // wypisujemy adres
komórki
	cout << "Wartosc zmiennej wynosi " << *wskaznik << endl; // przed nazwą
wskaźnika dajemy gwiazdkę.
	cout << "Wartosc zmiennej wynosi " << numer << endl; // wypisujemy zmienną numer.
```

Adres zmiennej numer nie zmienił się, ale jej wartość tak. Dlaczego? Ponieważ wskaźnik wskazuje na komórkę pamięci, a my zwiększyliśmy o jeden wartość zmiennej na której adres wskazuje wskaźnik.

Poprzez wskaźnik możemy modyfikować wartość zmiennej, która jest zapisana pod adresem wskazywanym przez wskaźnik.

Przy użyciu wskaźników możemy przeprowadzać takie same operacje jak przy użyciu zmiennych:

```
	float liczba1, liczba2, suma;
	float *wsk1, *wsk2; // wskaźniki typu float

	wsk1 = &liczba1; // ustawiamy wskaźnik wsk1 na adres zmiennej liczba1
	wsk2 = &liczba2; // ustawiamy wskaźnik wsk2 na adres zmiennej liczba2
	cout << "Podaj pierwsza liczbe: "; // prosimy o liczby
	cin >> liczba1;
	cout << "Podaj druga liczbe: ";
	cin >> liczba2;

	suma = (*wsk1) + (*wsk2); // obliczamy sumę
	cout << "Suma podanych liczb wynosi " << suma;
```

Wyrażenia *wsk1 i *wsk2 są w nawiasach. Gdybyśmy nie dali nawiasów to nie zwiększylibyśmy wartości zmiennej lecz adres, czyli przesunęlibyśmy się do kolejnej komórki pamięci. Bez nawiasów, wyrażenie *wsk1++ traktowane jest jako wsk1++. Jak widać, zwiększamy adres a nie wartość.

Wskaźnik na tablice

Napiszmy program, w którym zadeklarujemy tablicę i spróbujemy wypisać na ekran jej nazwę:

```
	float liczby[10];
	cout << liczby;
```

Jest to adres tablicy. Z tego wnioskujemy, że nazwa tablicy jest jednocześnie wskaźnikiem do pierwszego elementu tej tablicy. Jest to wskaźnik stały. Przekonamy się o tym, wypisując wartość zmiennej, na którą wskazuje wskaźnik. Jest to wskaźnik stały. To znaczy, że nie można na wskaźnik liczby podstawić wartości liczby+1.

```
	float liczby[10];

	liczby[0] = 5; // za pierwszy element podstawiamy 5
	cout << *liczby; // piszemy wartość zmiennej, na którą wskazuje wskaźnik liczby
```

Zapiszmy do tablicy kilka elementów i zwiększmy o i adres na który wskazuje wskaźnik liczby.

```
	float liczby[10];
	int i;

	for (i = 0; i<5; i++) // wczytujemy 5 liczb wsk
	{
		cout << "podaj liczbe: ";
		cin >> liczby[i];
	}

	cout << "adresy:" << endl << endl;
	for (i = 0; i<5; i++) // wypisujemy 5 adresów
	{
		cout << liczby + i << endl; // bo liczby jest wskaźnikiem stałym.
Wypisujemy adresy
	}
```

Wyświetliły się kolejne adresy komórek pamięci. Są to komórki, w których zapisane są kolejne elementy tablicy. Aby się o tym przekonać, przeanalizuj przykład:

```
float liczby[10];
	int i;

	for (i = 0; i<5; i++) // wczytujemy 5 liczb
	{
		cout << "podaj liczbe: ";
		cin >> liczby[i];
	}

	cout << "wartosci:" << endl << endl;
	for (i = 0; i<5; i++) // wypisujemy 5 adresów
	{
		cout << *(liczby + i) << endl; // bo liczby jest wskaźnikiem stałym.
Wypisujemy wartości
	}
```

Dynamiczna alokacja pamięci:

Często podczas działania programu pojawia się problem dynamicznego zarządzania pamięcią. Problem można zaprezentować na przykładzie tablicy o różnej ilości elementów. Z pomocą przychodzą wskaźniki oraz operatory *new* i *delete*. Najpierw tworzymy wskaźnik określonego typu a następnie

zmienną o dowolnej wielkości. Potem przydzielamy wskaźnikowi adres nowej zmiennej utworzonej na stercie:

```
int main()
{
	int* zmienna;
	zmienna = new int[3];
	zmienna[0] = 111;

	cout << *zmienna << endl;

	delete zmienna;
}
```

Wskaźniki i funkcje

W języku C++ przekazujemy argumenty do funkcji poprzez tzw. przekazywanie przez wartość. W języku C oraz C++ możemy przekazywać wartości do funkcji poprzez przekazywanie przez wskaźnik. Wskaźnik będzie wtedy argumentem funkcji.

```
// funkcja przyjmuje jako argument wskaźnik
void ZwiekszLiczbe(int* liczba)
{
	*liczba += 5;
}

int main()
{
	int numer = 5;
	int* wsk = &numer;
	ZwiekszLiczbe(wsk); //przekazujemy wskaźnik (bez operatorów)
	cout << numer << endl;
	ZwiekszLiczbe(&numer); //przekazujemy bezpośrednio adres zmiennej (operator &)
	cout << numer << endl;
}
```

Należy zwrócić uwagę, że do funkcji której argumentem jest wskaźnik, przekazujemy adres zmiennej za pomocą operatora pobrania adresu (&) a nie samą wartość.

Przekazywanie argumentów przez wskaźnik

Przekazywanie argumentów przez wskaźnik jest możliwe zarówno w języku C++ jak i C. Użycie wskaźników razem z funkcjami jest bardzo wygodne i ekonomiczne. Zdarzają się sytuacje, kiedy skorzystanie ze wskaźników jest niezbędne do osiągnięcia odpowiedniego efektu.

Przekazując argumenty przez wartość, funkcja może zwrócić tylko jeden parametr za pomocą return. Oznacza to, że możliwe jest zmodyfikowanie wartości tylko jednej zmiennej występującej poza wywoływaną funkcją. Co w przypadku gdy podczas wywołania jednej funkcji chcemy zmienić kilka zmiennych? – tutaj niezbędne są wskaźniki. Jest to najlepszy przykład na zobrazowanie tego, dlaczego wskaźniki są przydatne.
Podsumowując przekazywanie przez wskaźnik:

- argumenty wskaźnikowe wskazują na zmienne z poza funkcji, więc wewnątrz funkcji nie są tworzone kopie
- funkcja może zmodyfikować wiele zmiennych z poza funkcji (bez użycia return)

- wskaźniki także przekazujemy przez wartość, wynika to z faktu że wskaźnik też jest typem zmiennej (zmienna wskaźnikowa), jednak mimo że wskaźnik (adres miejsca w pamięci) zostanie skopiowany to wartość wyłuskana ze wskaźnika nie jest już kopią

```
void zwieksz_kilka(int* dl, int* wys, int* waga)
{
	// zmienna '*dl', '*wys' i '*waga' nie są kopiami
	// operowanie na nich zmienia ich wartość w "całym" programie
	// funkcja nie zwraca nic - bo nie ma sensu

	*dl = *dl * 2;
	*wys = *wys * 2;
	*waga = *waga * 2;
}

int main()
{
	// zmienne
	int dlugosc = 125;
	int wysokosc = 300;
	int waga = 20;

	// wskaźniki do zmiennych
	int* wsk_dlugosc = &dlugosc;
	int* wsk_wysokosc = &wysokosc;
	int* wsk_waga = &waga;

	// wywołanie funkcji
	zwieksz_kilka(wsk_dlugosc, wsk_wysokosc, wsk_waga);

	// wyświetlenie nowych wartości
	cout << dlugosc << endl;
	cout << wysokosc << endl;
	cout << waga << endl;

}
```

Zwróć uwagę, argumenty funkcji to zmienne wskaźnikowe czyli „adresy do zmiennych”. Gwiazdka w argumentach nie jest operatorem wyłuskania, informuje ona kompilator że zmienna jest wskaźnikiem (czyli „adresem”). Jak można wykorzystać ten fakt? Nie trzeba deklarować zmiennych wskaźnikowych aby przekazywać argument funkcji przez wskaźnik. Można przekazać bezpośrednio adres za pomocą operatora pobrania adresu (&).

```
void zwieksz_kilka(int* dl, int* wys, int* waga)
{
	// zmienna '*dl', '*wys' i '*waga' nie są kopiami
	// operowanie na nich zmienia ich wartość w "całym" programie
	// funkcja nie zwraca nic - bo nie ma sensu

	*dl = *dl * 2;
	*wys = *wys * 2;
	*waga = *waga * 2;
}

int main()
{
	// zmienne
	int dlugosc = 125;
	int wysokosc = 300;
```

```
        int waga = 20;

        // wywołanie funkcji z (&) (tak jak byśmy przekazywali wskaźniki)
        zwieksz_kilka(&dlugosc, &wysokosc, &waga);

        // wyświetlenie nowych wartości
        cout << dlugosc << endl;
        cout << wysokosc << endl;
        cout << waga << endl;
}
```

Zadania do samodzielnego rozwiązania:

Wykorzystując wskaźniki należy zaproponować implementację do poniższych zadań.

1. Napisz program, który:
   - wypisze na ekran adres zadeklarowanej zmiennej,
   - pobierze wartość zmiennej, wypisze na ekran jej adres oraz wartość,
   - przy użyciu wskaźników obliczy różnicę dwóch liczb,
   - przy użyciu wskaźników obliczy średnią trzech liczb,
2. Napisz program, w którym zadeklarujesz tablicę a następnie:
   - Wypisz na ekran adres jej pierwszego elementu.
   - Wypisz na ekran adres jej czwartego elementu.
   - Napisz program, w którym wylosujesz wartości do tablicy z przedziału podanego przez użytkownika i wypiszesz ich adresy.
3. Napisz program, w którym wczytasz elementy tablicy, obliczysz ich średnią oraz wpiszesz elementy większe od średniej.
4. Napisz funkcję, która podnosi podaną liczbę do kwadratu. Parametr do funkcji przekaż poprzez wskaźnik.
5. Napisz funkcję, która umożliwi policzenie podanej potęgi dla podanej liczby. Parametry przekaż poprzez wskaźnik. Funkcja powinna zwrócić wskaźnik do wyniku.
6. Napisz funkcję, która wypisuje podany znak podaną liczbę razy. Parametry przekaż poprzez wskaźniki. Na zakończenie funkcja powinna zmniejszać o 1, parametr odpowiadający za ilość powtórzeń.
7. Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu int, która:
   - zwraca jako wartość mniejszą z liczb wskazywanych przez argumenty.
   - zwraca jako wartość wskaźnik na zmienną przechowującą mniejszą z liczb wskazywanych przez argumenty.
8. Napisz funkcję otrzymującą jako argumenty referencje do dwóch zmiennych typu int, która zamienia ze sobą wartości zmiennych, do których referencje dostaliśmy w argumentach.