

01. Przypomnienie elementów języka C++

1 Wskaźniki

Operator adresu & pozwala odczytać adres pamięci pod jakim znajduje się dowolna zmienna:

```
1 int x = 123;
2 float y = 1.23f;
3 std::cout << &x << " " << &y;
```

Rozmiar w bajtach, jaki zajmuje zmienna danego typu można sprawdzić przy pomocy operatora sizeof:

```
1 std::cout << sizeof(char) << std::endl;
2 std::cout << sizeof(int) << std::endl;
3 std::cout << sizeof(double) << std::endl;
```

Powyższy adres można przypisać do zmiennej specjalnego typu nazywanej *wskaźnikiem* (ang. *pointer*) przy użyciu operatora *:

```
1 int x = 123;
2 int *ptr = &x; // utworzenie zmiennej wskaźnikowej
3 std::cout << &x << " = " << ptr;
```

Przypisanie wskaźnikowi wartości NULL lub 0 oznacza, że jest to wskaźnik pusty, który nie wskazuje na żaden obszar pamięci.

Dostęp do wartości wskazywanej przez wskaźnik odbywa się poprzez operator *dereferencji* * (operator *wyłuskania*):

```
1 std::cout << x << " = " << *ptr;
```

Należy pamiętać, że przypisanie wskaźnikowi adresu pewnej zmiennej x, a następnie zmiana wartości przez niego wskazywanej przy pomocy dereferencji, skutkuje zmianą zmiennej x:

```
1 std::cout << x; // x wynosi 123
2 *ptr = 321;
3 std::cout << x; // x wynosi 321 bez bezpośredniej zmiany
```

Wskaźniki często wykorzystuje się do takiego przekazywania argumentów do funkcji, aby mogła ona modyfikować przekazywane zewnętrzne zmienne, np.:

```
1 void inc(int* a) { // wskaźnik jako argument
2     (*a)++; // wyluskanie wartosci i jej zwiekszenie
3 }
4
5 int main() {
6     int x = 123;
7     inc(&x); // nalezy przekazac adres zmiennej
8     std::cout << x; // wartosc to 124
9     return 0;
10 }
```

Zadanie 01 Uruchom kod źródłowy dołączony do ćwiczeń. Zdefiniuj procedurę `cube2`, która będzie działała podobnie jak `cube`, z tą różnicą, że nie będzie zwracała nowej liczby, lecz modyfikowała zmienną przekazaną jako argument. Wykorzystaj wskaźniki. Przetestuj działanie funkcji.

2 Referencje

Referencję (ang. *reference*) można rozumieć jako alias (inną nazwę) dla zmiennej. Tworzy się ją za pomocą *operatora referencji* `&`:

```
1 int x = 123;
2 int &y = x;
3 std::cout << x << " = " << y;
```

Referencje zachowują się jak wskaźniki (tzn. zmiana zmiennej będącej referencją pociąga zmianę zmiennej początkowej), ale korzysta się z nich jak ze zwykłych zmiennych.

Przykład przekazywania argumentów funkcji przez referencję:

```
1 void inc(int& a) { // przekazanie argumentu przez referencje
2     a++;          // zwiększenie wartosci bez wyluskiwania
3 }
4
5 int main() {
6     int x = 123;
7     inc(x);       // przekazanie zmiennej, nie jej adresu
8     std::cout << x; // wartosc to 124
9     return 0;
10 }
```

Zadanie 02 Zdefiniuj analogiczną do `cube2` procedurę `cube3` wykorzystując przekazywanie argumentu przez referencję.

3 Tablice a wskaźniki

W językach C/C++ nazwa tablicy jest wskaźnikiem na pierwszy jej element. Dlatego do przekazania tablicy jako argumentu funkcji można wykorzystać wskaźniki:

```
1 int sum(int* tab, // argumentem jest wskaźnik
2         int size) { // rozmiar tablicy
3     // ...
4 }
5
6 int main() {
7     int tab[3] = {5, 10, 15};
8     sum(tab, 3); // nazwa tablicy to wskaźnik więc OK
9 }
```

Na wskaźnikach można również wykonywać operacje arytmetyczne. Dodanie do wskaźnika (czyli adresu) liczby całkowitej powoduje jego przesunięcie w pamięci o odpowiednią liczbę bloków pamięci (tj. liczbę bajtów odpowiadającą obszarowi pamięci zajmowanemu przez dany typ).

Przykład wykorzystania arytmetyki wskaźników w połączeniu z tablicami:

```
1 int tab[3] = {5, 10, 15};
2
3 for (int* w = tab; // wskaźnik na pierwszy element tablicy
4     w != tab + 3 // dopoki iterujesz po adresie we wnętrzu tablicy
5     ++w) { // zwiększ adres
6     std::cout << *w << " ";
7 }
```

Zadanie 03 Napisz funkcję, która wypisze na ekran podaną jako argument tablicę elementów po elemencie oddzielonymi przecinkiem i spacją, np. 1, 2, 3, 4, 5. Drugim argumentem tej funkcji powinien być rozmiar tablicy. Przetestuj działanie utworzonej funkcji.

Następnie spróbuj zmodyfikować ją tak, aby iterować po tablicy z wykorzystaniem wskaźnika zamiast indeksu.

Zadanie 04 Napisz funkcję, która zwróci wskaźnik na element w tablicy liczb całkowitych o największej wartości. Następnie wywołaj na tym elemencie odpowiednią funkcję obliczającą sześcian i wypisz całą tablicę.

4 Dynamiczna alokacja pamięci

Dynamiczną alokację pamięci (ang. *dynamic memory allocation*) w trakcie wykonywania programu umożliwiają operatory `new` (alokacja pamięci dla pojedynczej zmiennej) oraz `new[]` (dla tablicy):

```
1 int *p = new int; // wskaźnik wskazuje teraz na obszar, którego
2                 // nie zajmuje żadna z utworzonych zmiennych
3 *p = 123;
```

Dzięki temu możliwe jest stworzenie tablicy o nieznanym początkowo rozmiarze:

```
1 // ... wczytanie n od uzytkownika
2 int *tab = new int[n];
3 // ... wczytanie n elementow tablicy
```

Konieczne jest własnoręczna *dealokacja pamięci* (zwolnienie), w przeciwnym wypadku pamięć będzie zajmowana nawet po zakończeniu wykonywania programu (tzw. *wyciek pamięci*).

```
1 delete p;
2 delete[] tab;
```

Zadanie 05 Napisz fragment kodu, w którym tworzona będzie dynamiczna tablica liczb całkowitych. Zarówno liczby, jak i rozmiar tablicy powinny być podawane przez użytkownika w trakcie działania programu. Przetestuj działanie kodu wypisując wczytaną tablicę. Pamiętaj o dealokacji pamięci.