01. Przypomnienie elementów języka C++

1 Wskaźniki

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

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

Odczytanie rozmiaru w bajtach, jaki zajmuje zmienna danego typu umożliwia operator `sizeof`:

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

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

```
int x = 123;         // utworzenie zwyklej zmiennej
int *ptr = &x;       // utworzenie zmiennej wskaznikowej
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*):

```
std::cout << x << " = "
          << *ptr;  // wypisanie wartosci pod wskaznikiem
```

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

```
std::cout << x;      // x == 123
*ptr = 321;
std::cout << x;      // x == 321 bez bezposredniej zmiany
```

Wskaźniki wykorzystuje się m.in. do umożliwienia takiego przekazywania argumentów do funkcji, które pozwala na modyfikację zmiennej zewnętrznej wewnątrz ciała funkcji (pamiętaj, że domyślnie argumenty funkcji w języku C++ przekazywane są *przez wartość*, a więc tworzona jest kopia zmiennej przekazywanej jako argument). Przykład:

```
void inc(int* a) {  // wskaznik jako argument
  (*a)++;           // wyluskanie wartosci i jej zwiekszenie
}

int main() {
  int x = 123;
  inc(&x);          // nalezy przekazac adres zmiennej
  std::cout << x;   // wartosc to 124
  return 0;
}
```

Zadanie 01 Uruchom kod źródłowy dołączony do ćwiczeń. Zdefiniuj procedurę `cube_ptr`, która będzie działała podobnie jak `cube`, z tą różnicą, że nie będzie zwracała nowej liczby, lecz modyfikowała zmienną przekazaną jako argument. W implementacji rozwiązania 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* `&`:

```
int x = 123;
int &y = x;
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:

```
void inc(int& a) {  // przekazanie argumentu przez referencje
  a++;              // zwiekszenie wartosci bez wyluskiwania
}

int main() {
  int x = 123;
  inc(x);           // przekazanie zmiennej, nie jej adresu
  std::cout << x;   // wartosc to 124
  return 0;
}
```

Zadanie 02 Zdefiniuj analogiczną do `cube_ptr` procedurę `cube_ref` 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:

```
int sum(int* tab,   // argumentami funkcji sa wskaznik
        int size) { // oraz rozmiar tablicy
  // ...
}

int main() {
  int tab[3] = {5, 10, 15};
  sum(tab, 3);      // nazwa tablicy to wskaznik, wiec OK
}
```

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:

```
int tab[3] = {5, 10 15};

for (int* w = tab;  // wskaznik na pierwszy element tablicy
     w != tab + 3   // dopoki iterujesz po adresie wewnatrz tablicy
     ++w) {         // zwieksz adres
  std::cout << *w << " ";
}
```

Zadanie 03 Napisz funkcję `print_tab`, która wypisze na ekran podaną jako argument tablicę. Elementy powinny być oddzielone przecinkiem i spacją, np. `1, 2, 3, 4, 5`. Pamiętaj, że drugim argumentem tej funkcji powinien być rozmiar tablicy. Przetestuj działanie utworzonej funkcji.

Następnie zmodyfikuj stworzoną funkcję w taki sposób, 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

W języku C++ *dynamiczną alokację pamięci* (ang. *dynamic memory allocation*) w trakcie wykonywania programu umożliwiają operatory `new` (alokacja pamięci dla pojedynczej zmiennej) oraz `new[]` (alokacja ciągłego obszaru pamięci dla tablicy):

```
int *p = new int; // wskaznik wskazuje teraz na obszar, ktorego
                  // nie zajmuje zadna z utworzonych zmiennych
*p = 123;
```

Dzięki temu możliwe jest stworzenie tablicy o nieznanym początkowo rozmiarze:

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

Po przydzieleniu pamięci dynamicznie, konieczna jest własnoręczna jej *dealokacja* (zwolnienie). W przeciwnym wypadku pamięć będzie zajmowana nawet po zakończeniu wykonywania programu (tzw. *wyciek pamięci*).

```
delete p;
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.