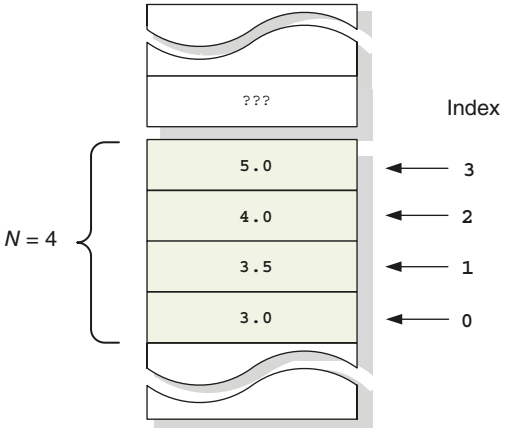
Do zapamiętania

- Wybieraj sensowne i czytelne nazwy (identyfikatory) dla obiektów (zmiennych, stałych, funkcji, klas itd.).
- Wybieraj typy danych odpowiednie do obliczeń.
- Staraj się używać typów ze znakiem.
- Zawsze inicjalizuj zmienne. W przypadku inicjalizacji wartością zero wystarczy dodać { }.

3.2. Przykładowy projekt – zbieranie ocen studentów

Jak dotąd poznaliśmy sposób definiowania i korzystania z pojedynczych zmiennych i stałych różnego typu. Dosyć często jednak musimy przechowywać wiele takich obiektów jeden po drugim. Czasami nie wiemy, ile takich obiektów będziemy potrzebować. Aby to zrobić, możemy skorzystać ze struktury danych nazywanej *wektorem* (lub *tablicą*)[10]. Wektor zawierający cztery obiekty numeryczne został przedstawiony na rysunku 3.2.



Rysunek 3.2. Wektor składa się z pewnej liczby obiektów zajmujących ciągłą przestrzeń w pamięci. Tutaj mamy $N = 4$ obiektów, które mogą przechowywać dane zmiennoprzecinkowe. Dostęp do każdej komórki wektora możemy uzyskać poprzez podanie jej indeksu. Pierwszy element ma indeks 0, a ostatni dostępny jest pod indeksem $N - 1$

Zacznijmy od tego, że wektor jest strukturą danych o następujących cechach:

- Zawiera obiekty tego samego typu.
- Obiekty są rozmieszczone w pamięci w sposób zwarty, tj. są one ułożone jeden obok drugiego (bez odstępów).

[10] Termin *tablica* zarezerwowany jest zwykle dla struktur o stałym rozmiarze (podrozdział 3.10), natomiast termin *wektor* używany jest dla obiektów, których rozmiar dynamicznie się zmienia.

- Dostęp do każdego obiektu w wektorze możemy uzyskać poprzez dostarczenie jego *indeksu* do *operatora indeksu* `[]`. Prawidłowe indeksy to wartości od 0 do $N - 1$, gdzie N oznacza liczbę elementów w wektorze.

W języku C++ wektory możemy tworzyć na wiele sposobów, jednak najbardziej elastycznym jest użycie obiektu `std::vector` z biblioteki standardowej, który:

- Może być na początku pusty.
- Może zostać uzupełniony nowymi obiektami, gdy staną się one dostępne.
- Zawiera wiele przydatnych funkcji.

Zanim powiemy sobie szczegółowo, co może zrobić dla nas `std::vector`, zacznijmy od prostego programu przykładowego z listingu 3.3. Jego celem jest zebranie i przechowanie ocen studentów, a następnie obliczenie ich zaokrąglonej średniej.

Listing 3.3. Program do zbierania i obliczania średniej ocen studentów (w *StudentGrades*, *main.cpp*)

```
#include <vector>       // Nagłówek pozwalający użyć std::vector
#include <iostream>     // Nagłówki dla wejścia i wyjścia
#include <iomanip>      // oraz formatowania wyjścia
#include <cmath>        // Dla funkcji matematycznych

// Wprowadzamy poniższe, aby móc pisać vector zamiast std::vector
using std::cout, std::cin, std::endl, std::vector;


int main()
{

  cout << "Enter your grades" << endl;

  vector< double >    studentGradeVec; // Pusty wektor wartości double

  // Zbierz oceny studentów
  for( ;; )
  {
    double grade {};

    cin >> grade;

    // Jeśli OK, wstaw nową ocenę na końcu wektora
    if( grade >= 2.0 && grade <= 5.0 )
      studentGradeVec.push_back( grade );


    cout << "Enter more? [y/n] ";
    char ans {};
    cin >> ans;

    if( ans == 'n' || ans == 'N' )
      break;    // sposób na wyjście z pętli
  }
```