Podstawy programowania: Laboratorium nr 5
Referencje. Obsługa plików w trybie tekstowym.

2017-2018

mgr inż. Przemysław Walkowiak
dr inż. Michał Ciesielczyk

Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisuj wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;`,
4. umieszczeniu wszystkich rozwiązań w jednym pliku, chyba, że w poleceniu napisano inaczej.
5. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie).

Wprowadzenie

Referencje

Język C++ umożliwia również tworzenie zmiennych typu referencyjnego, które same w sobie nie zajmują żadnej pamięci, lecz są nowymi nazwami, aliasami, na istniejące obiekty. Typ referencyjny uzyskuje się poprzez dodanie modyfikatora typu '&' w czasie deklarowania zmiennej. Ponadto w przeciwieństwie do zwykłych zmiennych, zmienna referencyjna zawsze musi być zainicjalizowana w czasie deklaracji.

```
int wiek;
int &ref_wiek = wiek;
std::string nazwisko;
std::string &ref_nazwisko = nazwisko;
std::string imie;
auto &ref_imie = imie; // typ zostanie automatycznie wydedukowany
```

Odwołując się do referencji (zmiennej typu referencyjnego), posługujemy się tak jak każdą inną zmienną. Np.:

```
ref_wiek = 5;
ref_nazwisko = "Kowalski";
std::cin >> ref_imie; // powiedzmy, że podamy imię "Jan"

std::cout << ref_imie << " " << ref_nazwisko; // Wyjście: "Jan Kowalski"
std::cout << imie << " " << nazwisko; // Wyjście "Jan Kowalski"
```

Stałe referencje

Do typu referencyjnego można również zaaplikować kolejny modyfikator, **const**. Spowoduje on, że referencji nie będzie można zmienić, jednakże zmienna do której się odwołuje może ulec zmianie. Np.:

```
std::string nazwisko{"Kowalski"};

auto &ref_nazwisko = nazwisko;
ref_nazwisko = "Nowak";
std::cout << ref_nazwisko << " " << nazwisko; // Wyświetli się "Nowak
    Nowak"

nazwisko = "Sienkiewicz";
std::cout << ref_nazwisko << " " << nazwisko; // Wyświetli się
                                              // "Sienkiewicz Sienkiewicz"

const std::string &const_ref_imie = imie;
const auto &const_ref_nazwisko = nazwisko;
const_ref_nazwisko = "Mickiewicz"; // błąd kompilacji
                                   // - nie można zmienić wartości

nazwisko = "Wybicki"; // to już mozna zrobić
std::cout << const_ref_nazwisko << " " << nazwisko; // Wyświetli się
                                                    // "Wybicki Wybicki"
```

Przekazywanie argumentów do funkcji przez referencję

Jednym z najczęstszych miejsc, gdzie wykorzystuje się typy referencyjne jest przekazywanie argumentów do funkcji. W tradycyjnym sposobie przekazywania przez wartość (tj. `int sum(std::vector<int> v)`) zawartość wektora jest kopiowana (co można było zaobserwować przy okazji poprzednich laboratoriów). W przypadku małych obiektów, np. typy proste (**int**, **char**), przekazywanie przez wartość jest wręcz preferowane. Dla dużych typów, np. długi napis typu `std::string`, tablice: `std::vector`, `std::array`, przekazywanie przez wartość — kopiowanie — nie jest efektywnym rozwiązaniem. W takich przypadkach wykorzystuje się typ referencyjny.

Aby funkcja mogła otrzymywać referencję do zmiennej jako argument, podobnie jak przy deklaracji zmiennej należy dodać modyfikator '&' za typem, a przed nazwą argumentu.

```
void fillVector(std::vector<int> &v, int value) {
    for (auto & x : v) {
        x = value;
    }
}

void resizeMatrix(std::vector<std::vector<int>> &mat, int m, int n) {
    mat.resize(m);
```

```
    // do other things
}
```

Zaletą przekazywania argumentu przez wartość był fakt, że podczas modyfikacji zmiennej wewnątrz funkcji, zmienna na zewnątrz pozostawała niezmieniona — jednakże kosztem dodatkowego kopiowania.

Za pomocą referencji można uzyskać ten sam efekt jednakże bez ponoszenia dodatkowych kosztów. Do zakazywania modyfikacji służy modyfikator typu **const**.

```
void printVector(const std::vector<int> &v) {
    for (auto &x : v) {
        std::cout << x;
    }
}
void doSomething(const std::strng &text) {
    text = "Pusty napis"; // błąd kompilacji - mamy stałą referencję
}

std::vector<int> sum(const std::vector<int> &v1, const std::vector<int> &v1) {
    std::vector<int> result;
    // ...
    return result;
}
```

Referencje — kiedy stosować?

- Typ referencyjny powinno się wykorzystywać zawsze, gdy jest to możliwe. Dzięki niemu jesteśmy w stanie uniknąć kopiowania dużych obiektów oszczędzając zarówno pamięć jak i czas procesora.
- W przypadku funkcji, powinniśmy zawsze "domyślnie" wykorzystywać referencję z modyfikatorem **const**. Dopiero wtedy, gdy potrzebujemy modyfikować daną zmienną możemy pomyśleć o usunięciu tego modyfikatora.
- Należy również uważać zwracając referencję z funkcji, gdyż referencja zakłada, że dany obiekt do którego się ona odwołuje będzie istniał dłużej niż ona sama. Jeżeli referencja odwołuje się do zmiennej lokalnej funkcji, może to prowadzić do tak zwanego niezdefiniowanego działania.
- Kiedy chcemy wykorzystać argumenty funkcji do zwrócenia jakiejś wartości. Jednakże należy się zastanowić wpierw, czy nie można po prostu wykorzystać normalnej wartości zwracanej.

Obsługa plików tekstowych

Aby móc korzystać z obsługi plików tekstowych z biblioteki standardowej C++ należy załączyć bibliotekę `<fstream>`, tj. dodać do programu dyrektywę: `#include <fstream>`.

Przykładowy zapis do pliku tekstowego:

```
ofstream output;         // deklaracja strumienia wyjściowego
output.open("plik.txt");    // otwieranie do zapisu pliku o wskazanej
    nazwie
if (output) {            // sprawdzenie czy plik został poprawnie otwarty
    output << "Hello";  // zapis tekstu do pliku
}
output.close();      // zamykanie pliku
```

Przykładowy odczyt z pliku tekstowego:

```
ifstream input;      // deklaracja strumienia wejściowego
input.open("plik.txt"); // otwieranie do odczytu pliku o wskazanej nazwie
if (input) {             // sprawdzenie czy plik został poprawnie otwarty
    string buffer;
    while(input >> buffer) {    // odczyt z pliku kolejnego słowa
        cout << buffer; // wyświetlanie wczytanej informacji
    }
}
input.close();           // zamykanie pliku
```

Więcej informacji: `http://www.cplusplus.com/reference/fstream/`.

Zadania

Zadanie 1

Skopiuj definicje funkcji `print`, `sum`, `average` oraz `minmax` z poprzednich zajęć a następnie zmodyfikuj je w taki sposób aby zmienna typu `std::vector` przekazywana była przez referencję. Zastanów się, czy powinna być to stała (**const**) referencja czy też nie – umieść odpowiedź w komentarzu. Przetestuj swoją implementację

Zadanie 2

Zaimplementuj następujące funkcje dla wektorów liczb całkowitych:

a) `equals` – przyrównującą dwa wektory,
b) `add` oraz `subtract` – odpowiednio dodającą i odejmującą dwa wektory,
c) `multiply` – mnożącą podany wektor przez skalar (również liczbę całkowitą).

W każdym przypadku zastanów się w jaki sposób powinny być przekazywane argumenty funkcji oraz zwracane wyniki – przez wartość, referencję, a może stałą referencję?

Następnie, przetestuj działanie swojej implementacji w następujący sposób:

a) wygeneruj n-wymiarowy wektor liczb pseudolosowych i przyrównaj go z samym sobą;
b) wygeneruj dwa n-wymiarowe wektory liczb pseudolosowych i przyrównaj je wzajemnie;
c) wygeneruj dwa n-wymiarowe wektory liczb pseudolosowych i wyświetl ich sumę;
d) wygeneruj n-wymiarowy wektor liczb pseudolosowych i mnożąc go razy liczbę podaną przez użytkownika.

Dodatkowe informacje:

- Równość wektorów
- Dodawanie i odejmowanie wektorów
- Mnożenie przez skalar

Zadanie 3

Zdefiniuj funkcję:

```
void writeVector(const vector<int>& vec, const string& fileName)
```

zapisującą zawartość tablicy dynamicznej do pliku tekstowego. W przypadku, gdy wskazany plik już istnieje, nowa funkcja powinna go nadpisać.

Następnie wygeneruj pseudolosowy wektor 100-elementowy (funkcja `randomVector()`) oraz zapisz go do wybranego pliku. Korzystając z dowolnej aplikacji notatnika, sprawdź poprawność zapisanego przez Ciebie pliku – czy jesteś w stanie jednoznacznie odczytać jego zawartość?

Zadanie 4

Zdefiniuj funkcję `vector<int> readVector(const string& fileName)` odczytującą zawartość tablicy dynamicznej ze wskazanego pliku tekstowego. Następnie, odczytaj plik zapisany w poprzednim zadaniu i wyświetl na ekranie zawartość wektora (funkcja `print`).

Zadanie 5

Zaimplementuj funkcję `bubbleSort` sortującą podaną tablicę z wykorzystaniem algorytmu sortowania bąbelkowego. Zastanów się, czy podawany wektor powinien być przekazywany przez wartość, referencję, czy też stałą referencję – wyjaśnienie umieść w komentarzu.

Następnie, utwórz jednowymiarową tablicę o długości n, zainicjalizuj ją wartościami pseudolosowymi (od -999 do 999) i posortuj ją z wykorzystaniem funkcji `bubbleSort`. Zapisz wynik – posortowaną tablicę – w pliku tekstowym i sprawdź poprawność jego zawartości.

Dodatkowe informacje:

- Algorytm sortowania bąbelkowego :

function BUBBLESORT(A : tablica n-elementowa)
  repeat
    for $0 \leqslant i < n-1$ **do**
      if $A_i > A_{i+1}$ **then**
        swap(A_i, A_{i+1})
      end if
    end for
    $n \leftarrow n-1$
  until $n > 1$
end function

Zadanie 6

Skopiuj definicje aliasu `IntMatrix` oraz funkcji do obsługi macierzy liczb całkowitych z porzednich zajęć (`createMatrix`, `randomMatrix` oraz `print`). W jednej z funkcji macierz może być przekazywana przez stałą referencję. W której?

Zmodyfikuj swoją implementację (sposób przekazywania macierzy) i ponownie przetestuj swoją implementację, np.:

```
IntMatrix A = createMatrix(m, n);
printMatrix(A);

IntMatrix B = randomMatrix(m, n, -10, 15);
printMatrix(B);
```

Zadanie 7

Zdefiniuj funkcje `add` oraz `subtract`, które wykonują odpowiednio operacje dodawania ($C = A + B$) oraz odejmowania ($C = A - B$) macierzy liczb całkowitych (typu `IntMatrix`, czyli `vector<vector<int>>`). Funkcje powinny zwracać nową macierz wynikową C.

Utwórz dwie macierze losowe A i B o wartościach z przedziału $\langle -16; -4 \rangle$ i wymiarach $m \times n$, a następnie przetestuj swoją implementację wykonując operacje $A + B$ i $A - B$. Wartość m, n wczytaj od użytkownika.

Zadanie 8

Zdefiniuj funkcję `multiply`, które zwraca wynik operacji mnożenia dwóch macierzy liczb całkowitych A oraz B.

Utwórz dwie macierze losowe A i B o wartościach z przedziału $\langle 1; 14 \rangle$ i wymiarach odpowiednio $m \times n$ oraz $n \times p$, a następnie przetestuj swoją implementację wykonując operację $A * B$. Wartości m, n, p wczytaj od użytkownika.

Dodatkowe informacje:

- Mnożenie macierzy – `http://pl.wikipedia.org/wiki/Mno%C5%BCenie_macierzy`

Zadanie 9

Zdefiniuj dwie funkcje:

a) `double inv(double x)` – zwracającą odwrotność podanej liczby ($\frac{1}{x}$), oraz
b) `double inv_square(double x)` – zwracającą kwadrat odwrotności podanej liczby ($\frac{1}{x^2}$).

Następnie, zaimplementuj funkcję `forEach`, która przyjmuje jako argumenty macierz liczb rzeczywistych (tablicę dwuwymiarową) oraz funkcję, która powinna zostać wykonana dla każdego elementu macierzy. Deklaracja funkcji `forEach` mogłaby wyglądać następująco:

```
void forEach(vector<vector<double>>& m, double (&fun)(double))
```

Przetestuj swoją implementację funkcji `forEach`, wykorzystując funkcję `sqrt` (zwracającą pierwiastek podanej liczby) oraz zdefiniowane przez Ciebie funkcje `inv` i `inv_square`. Sprawdź efekt działania każdej z tych funkcji.

Przykład:

```
vector<vector<double>> m;
// inicjalizacja macierzy
// ...
forEach(m, sqrt);
forEach(m, inv);
forEach(m, inv_square);
```

Na następne zajęcia

- Struktury – `http://cplusplus.com/doc/tutorial/structures/`
- Pliki nagłówkowe – `http://www.cplusplus.com/articles/Gw6AC542`