Podstawy Programowania
Semestr letni 2022/23
Materiały z laboratorium i zadania domowe

Przemysław Olbratowski

3 marca 2023

Slajdy z wykładu są dostępne w serwisie UBI. Informacje organizacyjne oraz formularz do uploadu prac domowych znajdują się na stronie `info.wsisiz.edu.pl/~olbratow`. Przy zadaniach domowych w nawiasach są podane terminy sprawdzeń.

12.2 Zadania domowe z działu Iteratory (14, 21, 28 czerwca)

12.2.1 Adjacent Find: Wyszukiwanie pary równych elementów

Napisz funkcję `adjacent_find`, która przyjmuje modyfikujący iterator początkowy oraz końcowy wycinka wektora liczb całkowitych i zwraca modyfikujący iterator pierwszego elementu równego swojemu następnikowi. Jeżeli taki element nie istnieje, funkcja zwraca końcowy iterator wycinka. Napisz analogiczną funkcję `adjacent_find` przyjmującą i zwracającą iteratory niemodyfikujące. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie używają indeksów i korzystają tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {1, 7, 3, 7, 7, 2};
    auto result1 = adjacent_find(vector.begin(), vector.end());
    auto result2 = adjacent_find(vector.cbegin(), vector.cend());
    std::cout << result1 - vector.begin() << " "
              << result2 - vector.cbegin() << std::endl; }
```

Wykonanie

```
Out: 3 3
```

12.2.2 Bubble Sort: Sortowanie bąbelkowe

Bąbelkowe sortowanie wektora przebiega następująco. Porównujemy pierwszy element z drugim i jeśli są w niewłaściwej kolejności, to zamieniamy je wartościami. Następnie porównujemy drugi element z trzecim i tak dalej, do końca wektora. Jeżeli w takim pojedynczym przebiegu musieliśmy wykonać choć jedną zamianę, to powtarzamy wszystko od początku. W przeciwnym razie wektor jest już posortowany. Napisz funkcję `bubble_sort`, która przyjmuje modyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych i sortuje ten wycinek bąbelkowo w kolejności niemalejącej. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z plików nagłówkowych `utility` i `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {20, -1, 13, 5, -1, 7, 5, 2, -5, 9};
    bubble_sort(vector.begin(), vector.end());
    for (auto iterator = vector.cbegin(); iterator < vector.cend();) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: -5 -1 -1 2 5 5 7 9 13 20
```

12.2.3 Copy: Kopiowanie elementów

Napisz funkcję `copy`, która przyjmuje niemodyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz modyfikujący iterator początkowy innego wycinka i przepisuje wszystkie elementy pierwszego wycinka do drugiego. Funkcja zwraca modyfikujący iterator końcowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    const std::vector<int> vector1 {-7, 5, 1, 2, 11};
    std::vector<int> vector2(5);
    auto result = copy(vector1.cbegin(), vector1.cend(), vector2.begin());
    for (auto iterator = vector2.cbegin(); iterator < result;) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: -7 5 1 2 11
```

12.2.4 Copy Backward: Kopiowanie wstecz

Napisz funkcję `copy_backward`, która przyjmuje niemodyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz modyfikujący iterator końcowy innego wycinka. Funkcja przepisuje wszystkie elementy pierwszego wycinka do drugiego i zwraca modyfikujący iterator początkowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    const std::vector<int> vector1 {4, 7, 2, 1, 8};
    std::vector<int> vector2(vector1.size());
    auto result = copy_backward(vector1.begin(), vector1.end(), vector2.end());
    std::cout << result - vector2.begin() << std::endl;
    for (auto iterator = vector2.cbegin(); iterator < vector2.cend();) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: 0
Out: 4 7 2 1 8
```

12.2.5 Count: Zliczanie elementów

Napisz funkcję `count`, która przyjmuje niemodyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz dowolną wartość całkowitą i zwraca liczbę wystąpień tej wartości w zadanym wycinku. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    const std::vector<int> vector {3, 7, -1, 7, 10};
    int result = count(vector.cbegin(), vector.cend(), 7);
    std::cout << result << std::endl; }
```

Wykonanie

```
Out: 2
```

12.2.6 Find: Wyszukiwanie elementu

Napisz funkcję `find`, która przyjmuje modyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz dowolną wartość całkowitą i zwraca modyfikujący iterator pierwszego wystąpienia tej wartości w wycinku albo końcowy iterator wycinka, jeżeli ta wartość w nim nie występuje. Napisz analogiczną funkcję `find` przyjmującą i zwracającą iteratory niemodyfikujące. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie używają indeksów i korzystają tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {3, -1, -5, 7, 10};
    auto result1 = find(vector.begin(), vector.end(), 7);
    auto result2 = find(vector.cbegin(), vector.cend(), 7);
    std::cout << result1 - vector.begin() << " "
              << result2 - vector.cbegin() << std::endl; }
```

Wykonanie

```
Out: 3 3
```

12.2.7 Find First Of: Wyszukiwanie jednego z kilku elementów

Napisz funkcję `find_first_of`, która przyjmuje modyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz niemodyfikujący iterator początkowy i końcowy innego wycinka. Funkcja zwraca modyfikujący iterator pierwszego wystąpienia którejkolwiek wartości z drugiego wycinka w pierwszym albo iterator końcowy pierwszego wycinka, jeśli żadna wartość z drugiego w nim nie występuje. Napisz analogiczną funkcję `find_first_of` przyjmującą i zwracającą niemodyfikujące iteratory pierwszego wycinka. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie używają indeksów i korzystają tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector1 {4, 7, 2, 1, 8, 4, 3}, vector2 {3, 8, 1};
    auto result1 = find_first_of(vector1.begin(), vector1.end(),
                                 vector2.cbegin(), vector2.cend());
    auto result2 = find_first_of(vector1.cbegin(), vector1.cend(),
                                 vector2.cbegin(), vector2.cend());
    std::cout << result1 - vector1.begin() << " "
              << result2 - vector1.cbegin() << std::endl; }
```

Wykonanie

```
Out: 3
```

12.2.8 Includes: Podzbiór

Napisz funkcję `includes`, która przyjmuje niemodyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz niemodyfikujący iterator początkowy i końcowy innego wycinka. Oba wycinki są posortowane niemalejąco. Funkcja zwraca prawdę, jeśli każdy element drugiego wycinka znajduje się również w pierwszym, albo fałsz w przeciwnym razie. Jeśli jakiś element występuje w drugim wycinku kilka razy, to funkcja zwraca prawdę jedynie jeśli w pierwszym powtarza się przynajmniej tyle samo razy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector1 {1, 2, 4, 7, 9, 11, 15}, vector2 {2, 4, 4, 11};
    bool result = includes(vector1.cbegin(), vector1.cend(),
                           vector2.cbegin(), vector2.cend());
    std::cout << std::boolalpha << result << std::endl; }
```

Wykonanie

```
Out: false
```

12.2.9 Min Element: Element najmniejszy - grupowo

Napisz funkcję `min_element`, która przyjmuje modyfikujący iterator początkowy oraz końcowy wycinka wektora liczb całkowitych i zwraca modyfikujący iterator najmniejszego elementu tego wycinka. Jeżeli takich elementów jest kilka, funkcja zwraca iterator pierwszego z nich. Jeżeli taki element nie istnieje, funkcja zwraca końcowy iterator wycinka. Napisz analogiczną funkcję `min_element` przyjmującą i zwracającą iteratory niemodyfikujące. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie używają indeksów i korzystają tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {7, 5, 1, 12, 8};
    auto result1 = min_element(vector.begin(), vector.end());
    auto result2 = min_element(vector.cbegin(), vector.cend());
    std::cout << result1 - vector.begin() << " "
              << result2 - vector.cbegin() << std::endl; }
```

Wykonanie

```
Out: 2 2
```

12.2.10 Partial Sum: Sumy częściowe - indywidualnie

Napisz funkcję `partial_sum`, która przyjmuje niemodyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz modyfikujący iterator początkowy innego wycinka i do każdej komórki drugiego wycinka wpisuje sumę wszystkich elementów pierwszego o niewiększych indeksach. Funkcja zwraca modyfikujący iterator wycinka powstałego z wpisanych sum. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {3, 2, -1, 3, 4};
    auto result = partial_sum(vector.cbegin(), vector.cend(), vector.begin());
    for (auto iterator = vector.cbegin(); iterator < result;) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: 3 5 4 7 11
```

12.2.11 Remove Copy: Kopiowanie z usuwaniem

Napisz funkcję `remove_copy`, która przyjmuje niemodyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych, modyfikujący iterator początkowy innego wycinka, oraz dowolną wartość całkowitą. Funkcja przepisuje elementy pierwszego wycinka do drugiego pomijając elementy o podanej wartości i zwraca modyfikujący iterator końcowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    const std::vector<int> vector1 {4, 7, 2, 1, 8, 2, 2};
    std::vector<int> vector2(vector1.size());
    auto result = remove_copy(vector1.begin(), vector1.end(), vector2.begin(), 2);
    std::cout << result - vector2.begin() << std::endl;
    for (auto iterator = vector2.cbegin(); iterator < vector2.cend();) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: 4
Out: 4 7 1 8 0 0 0
```

12.2.12 Reverse: Odwracanie kolejności elementów

Napisz funkcję `reverse`, która przyjmuje modyfikujący iterator początkowy oraz końcowy wycinka wektora liczb całkowitych i odwraca kolejność elementów tego wycinka. Funkcja powinna być przystosowana do użycia w przykładowym progrmie poniżej. Funkcja nie używa indeksów i korzysta tylko z plików nagłówkowych `utility` oraz `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {7, -1, 12, 3, 10, 5};
    reverse(vector.begin(), vector.end());
    for (auto iterator = vector.cbegin(); iterator < vector.cend();) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: 5 10 3 12 -1 7
```

12.2.13 Reverse Copy: Kopiowanie w odwrotnej kolejności

Napisz funkcję `reverse_copy`, która przyjmuje niemodyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz modyfikujący iterator początkowy innego wycinka. Funkcja przepisuje elementy pierwszego wycinka do drugiego w odwrotnej kolejności i zwraca modyfikujący iterator końcowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector1 {4, 7, 2, 1, 8};
    std::vector<int> vector2(vector1.size());
    auto result = reverse_copy(vector1.cbegin(), vector1.cend(), vector2.begin());
    std::cout << result - vector2.begin() << std::endl;
    for (auto iterator = vector2.cbegin(); iterator < vector2.cend();) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: 5
Out: 8 1 2 7 4
```

12.2.14 Search N: Wyszukiwanie kolejnych elementów o danej wartości

Napisz funkcję `search_n`, która przyjmuje modyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych, dodatnią stałą całkowitą n oraz dowolną wartość całkowitą. Funkcja zwraca modyfikujący iterator początkowy pierwszej n-elementowej sekwencji elementów o zadanej wartości. Jeżeli taka sekwencja nie istnieje, funkcja zwraca końcowy iterator wycinka. Napisz analogiczną funkcję `search_n` przyjmującą i zwracającą itertory niemodyfikujące. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie używają indeksów i korzystają tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {1, 7, 3, 3, 7, 7, 2};
    auto result1 = search_n(vector.begin(), vector.end(), 2, 7);
    auto result2 = search_n(vector.cbegin(), vector.cend(), 2, 7);
    std::cout << result1 - vector.begin() << " "
              << result2 - vector.cbegin() << std::endl; }
```

Wykonanie

```
Out: 4 4
```

12.2.15 Swap Ranges: Zamiana wycinków

Napisz funkcję `swap_ranges`, która przyjmuje modyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych oraz modyfikujący iterator początkowy innego wycinka i zamienia wartościami pierwsze elementy tych wycinków, potem drugie i tak dalej, do końca pierwszego wycinka. Funkcja zwraca modyfikujący iterator końcowy drugiego wycinka. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector1 {-7, 5, 1, 2, 11}, vector2 {9, 0, 8, 1, 7};
    auto result = swap_ranges(vector1.begin(), vector1.end(), vector2.begin());
    std::cout << result - vector2.begin() << std::endl;
    for (auto iterator = vector1.cbegin(); iterator < vector1.cend();) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl;
    for (auto iterator = vector2.cbegin(); iterator < vector2.cend();) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: 5
Out: 9 0 8 1 7
Out: -7 5 1 2 11
```

12.2.16 Unique: Usuwanie powtórzeń

Napisz funkcję `unique`, która przyjmuje modyfikujący iterator początkowy i końcowy wycinka wektora liczb całkowitych i wszystkie elementy różne od swoich następników przepisuje na kolejne pozycje wycinka, poczynając od początku. Funkcja zwraca modyfikujący iterator końcowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów i korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {-7, 5, 2, 2, 11, 2, 3, 3};
    auto result = unique(vector.begin(), vector.end());
    for (auto iterator = vector.cbegin(); iterator < result;) {
        std::cout << *iterator++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: -7 5 2 11 2 3
```