Programowanie Obiektowe
Semestr zimowy 2020/21
Materiały z laboratorium i zadania domowe

Przemysław Olbratowski

9 października 2020

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

Spis treści

1 Wskaźniki: 3 i 4 października Wskaźniki, Tablice

1.1 Laboratorium z działu Wskaźniki

1.1.1 Bubble Sort: Sortowanie dowolnego wycinka tablicy

Napisz funkcję `bubble_sort`, która przyjmuje tablicę liczb całkowitych oraz jej długość i sortuje tę tablicę bąbelkowo w kolejności rosnącej. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej.

Przykładowy program

```
int main() {
    int table[] = {13, -2, 21, 5, -8, 5, 7, -10};
    bubble_sort(table + 2, 4);
    for (int *pointer = table; pointer < table + 8;) {
        std::cout << *pointer++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: 13 -2 -8 5 5 21 7 -10
```

Rozwiązanie

```
#include <iostream>

void swap(int *element1, int *element2) {
    int element = *element1;
    *element1 = *element2;
    *element2 = element; }

void bubble_sort(int table[], int size) {
    bool unordered;
    do {
        unordered = false;
        for (int *pointer = table; pointer + 1 < table + size; ++pointer) {
            if (pointer[1] < pointer[0]) {
                swap(pointer, pointer + 1);
                unordered = true; }}}
    while (unordered); }

int main() {
    int table[] = {13, -2, 21, 5, -8, 5, 7, -10};
    bubble_sort(table + 2, 4);
    for (int *pointer = table; pointer < table + 8;) {
        std::cout << *pointer++ << " "; }
    std::cout << std::endl; }
```

1.1.2 Find: Wyszukiwanie elementu o zadanej wartości

Napisz funkcję `find`, która przyjmuje modyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych oraz dowolną wartość całkowitą i zwraca modyfikujący wskaźnik pierwszego wystąpienia tej wartości w zadanym wycinku. Jeżeli poszukiwana wartość w wycinku nie występuje, funkcja zwraca końcowy wskaźnik wycinka. Napisz analogiczną funkcję `find`, która przyjmuje i zwraca wskaźniki niemodyfikujące. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej.

Przykładowy program

```
int main() {
    int table1[] = {13, -2, 21, 5, -8, 5, 7, -10};
    int *pointer1 = find(table1 + 4, table1 + 8, 5);
    const int table2[] = {13, -2, 21, 5, -8, 5, 7, -10};
    const int *pointer2 = find(table2 + 4, table2 + 8, 5);
    std::cout << pointer1 - table1 << " " << pointer2 - table2 << std::endl; }
```

Wykonanie

```
Out: 5 5
```

Rozwiązanie

```
#include <iostream>

int *find(int *begin, int *end, int element) {
    for (; begin < end && *begin != element; ++begin);
    return begin; }

const int *find(const int *begin, const int *end, int element) {
    for (; begin < end && *begin != element; ++begin);
    return begin; }

int main() {
    int table1[] = {13, -2, 21, 5, -8, 5, 7, -10};
    int *pointer1 = find(table1 + 4, table1 + 8, 5);
    const int table2[] = {13, -2, 21, 5, -8, 5, 7, -10};
    const int *pointer2 = find(table2 + 4, table2 + 8, 5);
    std::cout << pointer1 - table1 << " " << pointer2 - table2 << std::endl; }
```

1.2 Zadania domowe z działu Wskaźniki (12, 19, 26 października)

1.2.1 Clock: Zegar analogowy

Napisz funkcję `clock`, która przyjmuje całkowite liczby godzin i minut, na przykład 13 i 23 dla godziny trzynastej dwadzieścia trzy, oraz adresy dwóch zmiennych rzeczywistych i wpisuje do tych zmiennych kąty wychylenia wskazówek wyrażone w stopniach i liczone zgodnie z ruchem wskazówek zegara od położenia pionowo w górę. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double small, large;
    clock(&small, &large, 13, 23);
    std::cout << small << " " << large << std::endl; }
```

Wykonanie

```
Out: 41.5 138
```

1.2.2 DMS: Stopnie, minuty, sekundy

W geografii kąty często wyraża się podając całkowite liczby stopni, minut i sekund. Napisz funkcję `dms`, która przyjmuje kąt w stopniach jako liczbę rzeczywistą oraz adresy trzech zmiennych całkowitych i wpisuje do tych zmiennych liczby stopni, minut oraz sekund. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int degrees, minutes, seconds;
    dms(&degrees, &minutes, &seconds, 123.37);
    std::cout << degrees << " " << minutes << " " << seconds << std::endl; }
```

Wykonanie

```
Out: 123 22 12
```

1.2.3 Fraction: Część całkowita i ułamkowa

Napisz funkcję `fraction`, która przyjmuje nieujemną liczbę rzeczywistą oraz adresy dwóch zmiennych rzeczywistych i wpisuje pod te adresy część całkowitą oraz ułamkową podanej liczby. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `cmath`.

Przykładowy program

```
int main() {
    double integral, fractional;
    fraction(&integral, &fractional, 3.141593);
    std::cout << integral << " " << fractional << std::endl; }
```

Wykonanie

```
Out: 3 0.141593
```

1.2.4 Quadratic: Równanie kwadratowe

Napisz funkcję `quadratic` rozwiązującą równanie kwadratowe $ax^2+bx+c=0$. Funkcja przyjmuje wartości parametrów a, b, c oraz adresy dwóch zmiennych rzeczywistych i zwraca wyróżnik równania. Jeżeli równanie posiada rozwiązania, funkcja wpisuje je pod wskazane adresy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `cmath`.

Przykładowy program

```
int main() {
    double delta, x1, x2;
    delta = quadratic(&x1, &x2, 1.2, 3.7, -4);
    std::cout << delta << std::endl;
    std::cout << x1 << " " << x2 << std::endl; }
```

Wykonanie

```
Out: 32.89
Out: -3.93124 0.847908
```

1.2.5 Swap: Zamiana wartości dwóch zmiennych

Napisz funkcję `swap`, która przyjmuje adresy dwóch zmiennych rzeczywistych i zamienia wartości tych zmiennych. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double a = -3.2, b = 17.1;
    swap(&a, &b);
    std::cout << a << " " << b << std::endl; }
```

Wykonanie

```
Out: 17.1 -3.2
```

1.2.6 Lesser: Zmienna o mniejszej wartości

Napisz funkcję `lesser`, która przyjmuje adresy dwóch zmiennych całkowitych i zwraca adres tej z nich, która ma mniejszą wartość. Jeśli zmienne mają jednakową wartość, funkcja zwraca wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int a = 11, b = 5;
    *lesser(&a, &b) = -7;
    std::cout << a << " " << b << std::endl; }
```

Wykonanie

```
Out: 11 -7
```

1.2.7 Days: Długości miesięcy

Napisz program `days`, który przyjmuje jako argumenty wywołania rok oraz numer miesiąca i wypisuje na standardowe wyjście liczbę dni w tym miesiącu z uwzględnieniem lat przestępnych. Program załącza tylko pliki nagłówkowe `cstdlib` i `iostream`.

Przykładowe wykonanie

```
  Linux: ./days 2000 2
Windows: days.exe 2000 2
Out: 29
```

1.2.8 Beaufort: Skala Beauforta

Skala Beauforta określona jest następującą tabelą:

Beaufort	km/h
0	0 - 0.5
1	0.5 - 6.5
2	6.5 - 11.5
3	11.5 - 19.5
4	19.5 - 29.5
5	29.5 - 39.5
6	39.5 - 50.5
7	50.5 - 62.5
8	62.5 - 75.5
9	75.5 - 87.5
10	87.5 - 102.5
11	102.5 - 117.5
12	117.5 - ∞

Napisz program `beaufort`, który przyjmuje jako argument wywołania prędkość wiatru w kilometrach na godzinę i wypisuje na standardowe wyjście odpowiadającą jej siłę wiatru w skali Beauforta. Program załącza tylko pliki nagłówkowe `cstdlib` i `iostream`.

Przykładowe wykonanie

```
  Linux: ./beaufort 29.2
Windows: beaufort.exe 29.2
Out: 4
```

Uwaga Spróbuj napisać program bez użycia instrukcji warunkowej ani operatora warunkowego.

1.2.9 Nominals: Odliczanie kwoty w nominałach

W pewnym państwie emitowane są nominały 1, 2, 5, 10, 20, 50, 100, 200. Napisz program `nominals`, który przyjmuje jako argument wywołania kwotę bez groszy i wypisuje na standardowe wyjście dające ją w sumie nominały, od największego do najmniejszego. Program odlicza zadaną kwotę w możliwie najmniejszej liczbie emitowanych nominałów dysponując nieograniczoną liczbą monet lub banknotów każdego nominału. Program załącza tylko pliki nagłówkowe `cstdlib` i `iostream`.

Przykładowe wykonanie

```
  Linux: ./nominals 739
Windows: nominals.exe 739
Out: 200 200 200 100 20 10 5 2 2
```

1.2.10 Letters: Zliczanie liter - bitcoin

Napisz program `letters`, który czyta ze standardowego wejścia znaki do napotkania końca pliku, a następnie wypisuje na standardowe wyjście liczby wystąpień kolejnych liter pośród wpisanych znaków. Program nie rozróżnia wielkich i małych liter oraz pomija wszelkie inne znaki. Program załącza tylko pliki nagłówkowe `cctype` i `iostream`.

Przykładowe wykonanie

```
 In: Lorem ipsum dolor sit amet!
Out: 1 0 0 1 2 0 0 0 2 0 0 2 3 0 3 1 0 2 2 2 1 0 0 0 0 0
```

1.2.11 Count: Zliczanie elementów

Napisz funkcję `count`, która przyjmuje niemodyfikujący wskaźnik początkowy i końcowy wycinka tablicy 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 korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table[] = {-1, 7, -1, 12, -5, -1, 10};
    int result = count(table, table + 7, -1);
    std::cout << result << std::endl; }
```

Wykonanie

```
Out: 3
```

1.2.12 Find First Of: Wyszukiwanie jednego z kilku elementów

Napisz funkcję `find_first_of`, która przyjmuje modyfikujący wskaźnik początkowy i końcowy wycinka wektora liczb całkowitych oraz niemodyfikujący wskaźnik początkowy i końcowy innego wycinka. Funkcja zwraca modyfikujący wskaźnik pierwszego wystąpienia którejkolwiek wartości z drugiego wycinka w pierwszym wycinku albo wskaźnik końcowy pierwszego wycinka jeśli żadna wartość z drugiego wycinka w nim nie występuje. Napisz analogiczną funkcję `find_first_of` przyjmującą i zwracającą niemodyfikujące wskaźniki pierwszego wycinka. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie korzystają z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int elements[] = {3, 8, 1};
    int table1[] = {4, 7, 2, 1, 8, 4, 3};
    int *result1 = find_first_of(table1, table1 + 7,
                                 elements, elements + 3);
    const int table2[] = {4, 7, 2, 1, 8, 4, 3};
    const int *result2 = find_first_of(table2, table2 + 7,
                                       elements, elements + 3);
    std::cout << result1 - table1 << " "
              << result2 - table2 << std::endl; }
```

Wykonanie

```
Out: 3 3
```

1.2.13 Adjacent Find: Wyszukiwanie pary równych elementów

Napisz funkcję `adjacent_find`, która przyjmuje modyfikujący wskaźnik początkowy oraz końcowy wycinka tablicy liczb całkowitych i zwraca modyfikujący wskaźnik pierwszego elementu równego swojemu następnikowi. Jeżeli taki element nie istnieje, funkcja zwraca końcowy wskaźnik wycinka. Napisz analogiczną funkcję `adjacent_find` przyjmującą i zwracającą wskaźniki niemodyfikujące. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table1[] = {1, 7, 3, 7, 7, 2};
    int *result1 = adjacent_find(table1, table1 + 6);
    const int table2[] = {1, 7, 3, 7, 7, 2};
    const int *result2 = adjacent_find(table2, table2 + 6);
    std::cout << result1 - table1 << " "
              << result2 - table2 << std::endl; }
```

Wykonanie

```
Out: 3 3
```

1.2.14 Search N: Wyszukiwanie kolejnych elementów o danej wartości

Napisz funkcję `search_n`, która przyjmuje modyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych, dodatnią stałą całkowitą n oraz dowolną wartość całkowitą. Funkcja zwraca modyfikujący wskaźnik początkowy pierwszej n-elementowej sekwencji elementów o zadanej wartości. Jeżeli taka sekwencja nie istnieje, funkcja zwraca końcowy wskaźnik wycinka. Napisz analogiczną funkcję `search_n` przyjmującą i zwracającą wskaźniki niemodyfikujące. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie korzystają z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table1[] = {1, 7, 3, 3, 7, 7, 2};
    int *result1 = search_n(table1, table1 + 7, 2, 7);
    const int table2[] = {1, 7, 3, 3, 7, 7, 2};
    const int *result2 = search_n(table2, table2 + 7, 2, 7);
    std::cout << result1 - table1 << " "
              << result2 - table2 << std::endl; }
```

Wykonanie

```
Out: 4 4
```

1.2.15 Copy: Kopiowanie elementów

Napisz funkcję `copy`, która przyjmuje niemodyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych oraz modyfikujący wskaźnik początkowy innego wycinka i przepisuje wszystkie elementy pierwszego wycinka do drugiego. Funkcja zwraca modyfikujący wskaźnik końcowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table1[] = {-7, 5, 1, 2, 11};
    int table2[5];
    int *result2 = copy(table1, table1 + 5, table2);
    for (const int *pointer2 = table2; pointer2 < result2;) {
        std::cout << *pointer2++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.16 Copy Backward: Kopiowanie wstecz

Napisz funkcję `copy_backward`, która przyjmuje niemodyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych oraz modyfikujący wskaźnik końcowy innego wycinka. Funkcja przepisuje wszystkie elementy pierwszego wycinka do drugiego i zwraca modyfikujący wskaźnik początkowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table1[] = {4, 7, 2, 1, 8};
    int table2[5];
    int *result2 = copy_backward(table1, table1 + 5, table2 + 5);
    std::cout << result2 - table2 << std::endl;
    for (const int *pointer2 = table2; pointer2 < table2 + 5;) {
        std::cout << *pointer2++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.17 Remove: Usuwanie elementów

Napisz funkcję `remove`, która przyjmuje modyfikujący wskaźnik początkowy i końcowy wycinka wektora liczb całkowitych oraz dowolną wartość całkowitą i kolejne elementy nierówne tej wartości przepisuje do kolejnych komórek wycinka począwszy od pierwszej. Funkcja zwraca modyfikujący wskaźnik końcowy wycinka powstałego ze wszystkich przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table[] = {-7, 5, 2, 2, 11, 2, 3};
    int *result = remove(table, table + 7, 2);
    for (const int *pointer = table; pointer < result;) {
        std::cout << *pointer++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.18 Remove Copy: Kopiowanie z usuwaniem

Napisz funkcję `remove_copy`, która przyjmuje niemodyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych, modyfikujący wskaźnik początkowy innego wycinka, oraz dowolną wartość całkowitą. Funkcja przepisuje elementy pierwszego wycinka do drugiego pomijając elementy o podanej wartości i zwraca modyfikujący wskaźnik końcowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table1[] = {4, 7, 2, 1, 8, 2, 2};
    int table2[7];
    int *result2 = remove_copy(table1, table1 + 7, table2, 2);
    for (const int *pointer2 = table2; pointer2 < result2;) {
        std::cout << *pointer2++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.19 Swap Ranges: Zamiana wycinków

Napisz funkcję `swap_ranges`, która przyjmuje modyfikujący wskaźnik początkowy i końcowy wycinka wektora liczb całkowitych oraz modyfikujący wskaźnik początkowy innego wycinka. Funkcja zamienia wartościami pierwszy element pierwszego wycinka z pierwszym elementem drugiego i tak dalej do końca pierwszego wycinka oraz zwraca modyfikujący wskaźnik końcowy wycinka powstałego z elementów wpisanych do drugiego wycinka. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table1[] = {-7, 5, 1, 2, 11}, table2[] = {9, 0, 8, 1, 7};
    int *result2 = swap_ranges(table1, table1 + 5, table2);
    std::cout << result2 - table2 << std::endl;
    for (const int *pointer1 = table1; pointer1 < table1 + 5;) {
        std::cout << *pointer1++ << " "; }
    std::cout << std::endl;
    for (const int *pointer2 = table2; pointer2 < table2 + 5;) {
        std::cout << *pointer2++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.20 Reverse: Odwracanie kolejności elementów

Napisz funkcję `reverse`, która przyjmuje modyfikujący wskaźnik początkowy oraz końcowy wycinka tablicy liczb całkowitych i odwraca kolejność elementów tego wycinka. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table[] = {7, -1, 12, 3, 10, 5, -5};
    reverse(table, table + 7);
    for (int *pointer = table; pointer < table + 7;) {
        std::cout << *pointer++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.21 Reverse Copy: Kopiowanie w odwrotnej kolejności

Napisz funkcję `reverse_copy`, która przyjmuje niemodyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych oraz modyfikujący wskaźnik początkowy innego wycinka. Funkcja przepisuje elementy pierwszego wycinka do drugiego w odwrotnej kolejności i zwraca modyfikujący wskaźnik końcowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table1[] = {4, 7, 2, 1, 8};
    int table2[5];
    int *result2 = reverse_copy(table1, table1 + 5, table2);
    for (int *pointer2 = table2; pointer2 < result2;) {
        std::cout << *pointer2++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.22 Unique: Usuwanie powtórzeń - bitcoin

Napisz funkcję `unique`, która przyjmuje modyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych i wszystkie elementy różne od swoich poprzedników przepisuje na kolejne pozycje wycinka poczynając od początku. Funkcja zwraca modyfikujący wskaźnik końcowy wycinka powstałego z przepisanych elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table[] = {-7, 5, 2, 2, 11, 2, 3, 3};
    int *result = unique(table, table + 8);
    for (const int *pointer = table; pointer < result;) {
        std::cout << *pointer++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.23 Selection Sort: Sortowanie przez wybór

Sortowanie tablicy w kolejności niemalejącej przez wybór przebiega następująco. Znajdujemy w tablicy pierwszy element najmniejszy i zamieniamy go miejscami z pierwszym elementem tablicy. Następnie powtarzamy te czynności dla tablicy bez pierwszego elementu i tak dalej. Napisz funkcję `selection_sort`, która przyjmuje modyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych i sortuje ten wycinek niemalejąco przez wybór. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table[] = {3, 15, -2, 35, 0, -53, 20};
    selection_sort(table, table + 7);
    for (int *pointer = table; pointer < table + 7;) {
        std::cout << *pointer++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: -53 -2 0 3 15 20 35
```

1.2.24 Bubble Sort: Sortowanie bąbelkowe

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

Przykładowy program

```
int main() {
    int table[] = {20, -1, 13, 5, -1, 7, 5, 2, -5, 9};
    bubble_sort(table, table + 10);
    for (int *pointer = table; pointer < table + 10;) {
        std::cout << *pointer++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

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

1.2.25 Includes: Podzbiór

Napisz funkcję `includes`, która przyjmuje niemodyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych oraz niemodyfikujący wskaźnik 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 wycinku powtarza się przynajmniej tyle samo razy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table1[] = {1, 2, 4, 7, 9, 11, 15}, table2[] = {2, 4, 4, 11};
    bool result = std::includes(table1, table1 + 7, table2, table2 + 4);
    std::cout << std::boolalpha << result << std::endl; }
```

Wykonanie

```
Out: false
```

1.2.26 Min Element: Element najmniejszy

Napisz funkcję `min_element`, która przyjmuje modyfikujący wskaźnik początkowy oraz końcowy wycinka tablicy liczb całkowitych i zwraca modyfikujący wskaźnik najmniejszego elementu tego wycinka. Jeżeli takich elementów jest kilka, funkcja zwraca wskaźnik pierwszego z nich. Jeżeli taki element nie istnieje, funkcja zwraca końcowy wskaźnik wycinka. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table1[] = {-1, 7, 5, 1, 12, -3, 8};
    int *result1 = min_element(table1, table1 + 7);
    std::cout << result1 - table1 << " ";
    const int table2[] = {-1, 7, 5, 1, 12, -3, 8};
    const int *result2 = min_element(table2, table2 + 7);
    std::cout << result2 - table2 << std::endl; }
```

Wykonanie

```
Out: 5 5
```

1.2.27 Partial Sum: Sumy częściowe

Napisz funkcję `partial_sum`, która przyjmuje niemodyfikujący wskaźnik początkowy i końcowy wycinka tablicy liczb całkowitych oraz modyfikujący wskaźnik początkowy innego wycinka i do każdej komórki drugiego wycinka wpisuje sumę wszystkich elementów pierwszego o niewiększych indeksach. Funkcja zwraca modyfikujący wskaźnik końcowy wycinka powstałego z wpisanych sum. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table[] = {3, 2, -5, 1, 4};
    int *result = partial_sum(table, table + 5, table);
    for (int *pointer = table; pointer < result;) {
        std::cout << *pointer++ << " "; }
    std::cout << std::endl; }
```

Wykonanie

```
Out: 3 5 0 1 5
```

2 Alokacja: 17 i 18 października Dynamiczna alokacja pamięci

2.1 Laboratorium z działu Alokacja

2.1.1 Getints: Wczytywanie dowolnej ilości danych

Napisz funkcję `getints`, która przyjmuje modyfikujący wskaźnik na zmienną całkowitą, czyta ze standardowego wejścia liczby całkowite do napotkania końca pliku, umieszcza je w dynamicznie zaalokowanej tablicy i zwraca modyfikujący wskaźnik początkowy tej tablicy. Oprócz tego funkcja wpisuje do zmiennej wskazywanej swoim argumentem liczbę wczytanych liczb. Jeżeli ze standardowego wejścia nie da się wczytać żadnej liczby całkowitej, funkcja może nie zaalokować pamięci i zwrócić wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej.

Przykładowy program

```
int main() {
    int size;
    int *table = getints(&size);
    for (int index = 0; index < size;) {
        std::cout << table[index++] << " "; }
    std::cout << std::endl;
    std::free(table); }
```

Przykładowe wykonanie

```
 In: 13 -2 21 5 -8 5 7 -10
Out: 13 -2 21 5 -8 5 7 -10
```

Rozwiązanie

```
#include <cstdlib>
#include <iostream>

int *getints(int *size) {
    *size = 0;
    int capacity = 0;
    int *table = nullptr;
    for (int element; std::cin >> element; ++*size) {
        if (*size == capacity) {
            capacity = (capacity ? 2 * capacity : 1);
            table = (int*)std::realloc(table, capacity * sizeof(int)); }
        table[*size] = element; }
    return table; }

int main() {
    int size;
    int *table = getints(&size);
    for (int index = 0; index < size;) {
        std::cout << table[index++] << " "; }
    std::cout << std::endl;
    std::free(table); }
```

2.1.2 Remove: Kopiowanie z usuwaniem

Napisz funkcję `remove`, która przyjmuje modyfikujący wskaźnik na zmienną całkowitą, stałą tablicę liczb całkowitych oraz jej długość, a także dowolną wartość całkowitą. Funkcja tworzy dynamicznie tablicę liczb całkowitych, przepisuje do niej wszystkie elementy przekazanej tablicy różne od podanej wartości i zwraca modyfikujący wskaźnik początkowy utworzonej tablicy. Oprócz tego funkcja wpisuje liczbę przepisanych elementów do zmiennej wskazywanej pierwszym argumentem. Jeżeli nie przepisano żadnego elementu, funkcja może nie zaalokować pamięci i zwrócić wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej.

Przykładowy program

```
int main() {
    const int table1[] = {7, 3, 8, 4, 3, 3, 9, 2};
    int size2;
    int *table2 = remove(&size2, table1, 8, 3);
    for (int index2 = 0; index2 < size2; ++index2) {
        std::cout << table2[index2] << " "; }
    std::cout << std::endl;
    std::free(table2);  }
```

Wykonanie

```
Out: 7 8 4 9 2
```

Rozwiązanie

```
#include <cstdlib>
#include <iostream>

int *remove(int *size2, const int table1[], int size1, int element) {
    *size2 = 0;
    int *table2 = nullptr;
    if (size1) {
        table2 = (int*)std::malloc(size1 * sizeof(int));
        for (int index1 = 0; index1 < size1; ++index1) {
            if (table1[index1] != element) {
                table2[(*size2)++] = table1[index1]; }}}
    return table2; }

int main() {
    const int table1[] = {7, 3, 8, 4, 3, 3, 9, 2};
    int size2;
    int *table2 = remove(&size2, table1, 8, 3);
    for (int index2 = 0; index2 < size2; ++index2) {
        std::cout << table2[index2] << " "; }
    std::cout << std::endl;
    std::free(table2);  }
```

2.2 Zadania domowe z działu Alokacja (26 października, 9, 16 listopada)

2.2.1 Combination: Losowe kombinacje z powtórzeniami

Napisz program `combination`, który przyjmuje jako argumenty wywołania dowolną liczbę nieujemnych liczb całkowitych i drukuje na standardowe wyjście w losowej kolejności tyle jedynek, ile wynosi pierwszy argument, tyle dwójek, ile wynosi drugi i tak dalej. Program załącza tylko pliki nagłówkowe `cstdlib`, `ctime` i `iostream`.

Przykładowe wykonanie

```
  Linux: ./combination 3 0 5 2
Windows: combination.exe 3 0 5 2
Out: 1 3 1 3 4 3 3 1 4 3
```

2.2.2 Eratostenes: Sito Eratostenesa - bitcoin

Wszystkie liczby pierwsze mniejsze od dodatniej liczby całkowitej n można wyznaczyć następującą metodą zwaną sitem Eratostenesa. Spośród liczb większych od 1 i mniejszych od n wykreślamy wszystkie wielokrotności 2 poczynając od 4, następnie wszystkie wielokrotności 3 poczynając od 6 i tak dalej. Pozostałe na koniec niewykreślone liczby to wszystkie liczby pierwsze mniejsze od n. Napisz program `eratostenes`, który przyjmuje jako argument wywołania dodatnią liczbę całkowitą i wypisuje na standardowe wyjście wszystkie mniejsze od niej liczby pierwsze w kolejności rosnącej. Program załącza tylko pliki nagłówkowe `cstdlib` i `iostream`.

Przykładowe wykonanie

```
  Linux: ./eratostenes 100
Windows: eratostenes.exe 100
Out: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

2.2.3 Pascal: Trójkąt Pascala

Napisz program `pascal`, który przyjmuje jako argument wywołania liczbę naturalną n i drukuje na standardowe wyjście n pierwszych wierszy trójkąta Pascala sformatowanych jak poniżej. Program załącza tylko pliki nagłówkowe `cstdlib`, `iomanip` i `iostream`.

Przykładowe wykonanie

```
  Linux: ./pascal 6
Windows: pascal.exe 6
Out:           1
Out:         1   1
Out:       1   2   1
Out:     1   3   3   1
Out:   1   4   6   4   1
Out: 1   5  10  10   5   1
```

2.2.4 Permutations: Wyznaczanie wszystkich permutacji

Napisz program `permutation`, który przyjmuje jako argument wywołania liczbę naturalną n i wypisuje na standardowe wyjście wszystkie permutacje liczb od 1 do n. Program załącza tylko pliki nagłówkowe `cstdlib` i `iostream`.

Przykładowe wykonanie

```
  Latex: ./permutation 3
Windows: permutation.exe 3
Out: 1 2 3
Out: 1 3 2
Out: 2 1 3
Out: 2 3 1
Out: 3 1 2
Out: 3 2 1
```

2.2.5 Relay: Przesiadka

Kierowca planuje podróż z miasta początkowego przez kilka pośrednich do miasta końcowego. Trasa jest na tyle długa, że wymaga po drodze jednego noclegu. Kierowca chce tak wybrać miasto na nocleg, aby kilometraże pierwszego i drugiego dnia podróży jak najmniej się różniły. Napisz program `relay`, który czyta ze standardowego wejścia odległości między kolejnymi miastami do napotkania końca pliku i wypisuje na standardowe wyjście numer miasta, w którym wypada nocleg, przy czym miasto początkowe ma numer zero. Program załącza tylko pliki nagłówkowe `cmath`, `cstdlib` i `iostream`.

Przykładowe wykonanie

```
 In: 15.2 23.1 2.5 7.3 11 5.3
Out: 2
```

2.2.6 Variation: Losowe wariacje bez powtórzeń

Napisz program `variation`, który przyjmuje jako argumenty wywołania nieujemną liczbę całkowitą k oraz liczbę naturalną n i drukuje na standardowe wyjście k niepowtarzających się naturalnych liczb losowych z przedziału od 1 do n włącznie. Jeśli to niemożliwe, program nic nie drukuje. Program załącza tylko pliki nagłówkowe `cstdlib`, `ctime` i `iostream`.

Przykładowe wykonanie

```
  Linux: ./variation 10 100
Windows: variation.exe 10 100
Out: 78 15 23 50 49 87 2 35 98 69
```

3 Napisy: 7 i 8 listopada
Napisy w stylu języka C

4 Klasy: 21 i 22 listopada
Klasy, metody, składowe statyczne, funkcje zaprzyjaźnione

5 Operatory: 5 i 6 grudnia Przeciążanie operatorów

6 Metody: 19 i 20 grudnia
Specjalne metody klas

7 Dane: 16 i 17 stycznia
Dynamiczne struktury danych

8 Wyjątki: 30 i 31 stycznia
Obsługa sytuacji wyjątkowych