פקולטה: מדעי הטבע
מחלקה: מדעי המחשב
שם הקורס: תכנות מערכות ב
קוד הקורס: 7023010-2 כל הקבוצות
מועד __**ב**___ סמסטר: ___**ב**___ שנה: ___ה'תשפ"א
תאריך הבחינה: ה' באב ה'תשפ"א, 14/07/2021
משך הבחינה: שעתיים וחצי – 150 דקות
המרצים: אראל סגל-הלוי, ערן קאופמן, חרות סטרמן.
המתרגלים: יבגני נייטרמן, אריה יטיב, אור אביטל.

אנא קיראו היטב את כל ההנחיות והשאלות לפני שתתחילו לכתוב.

יינתן מענק של 2 נקודות לסטודנטים שיכתבו את הפתרון באופן ברור קריא וקל לבדיקה, בפרט:

- השאלות פתורות במחברת הבחינה לפי הסדר שבשאלון;
- כל שאלה מתחילה בעמוד נפרד;
- הכתב ברור וקריא, בלי מחיקות קשקושים חיצים וטקסט מיותר.

אסור להשתמש בכל חומר עזר.

יש לענות על כל השאלות במחברת הבחינה.
אין צורך להעתיק את השאלון למחברת - השאלון יתפרסם לאחר הבחינה.

יש לענות תשובות מלאות, להסביר כל תשובה בפירוט, ולכתוב תיעוד לקוד.
יש לענות תשובות ממוקדות - לא יינתנו נקודות על תשובות עם טקסט מיותר שאינו קשור לנושא.

אם אתם לא זוכרים, לא בטוחים או לא מבינים משהו בשאלה כלשהי - נסו לפתור את השאלה כמיטב יכולתכם, הסבירו מה הבנתם ולמה התכוונתם, והמשיכו לשאלה הבאה. אל "תיתקעו" בשאלה אחת. מומלץ להשקיע עד 20 דקות בשאלה.

בהצלחה!!!

שאלה 0 [עד 64 נק']

[ציון על מטלות + מענקים]

שאלה 1 [9 נק'] - תור דו-כיווני

א. כתבו פונקציה בשם printKMax, המקבלת כקלט וקטור של מספרים ממשיים, ומספר שלם k, ומדפיסה את האיבר הגדול ביותר בכל תת-וקטור רציף באורך k. חתימת הפונקציה:

```
void printKMax(vector<double> input, int k) { ... }
```

תוכנית ראשית לדוגמה:

```
int main() {
     printKMax({11,22,33,44,33,22,11}, 3);
          // prints 33 44 44 44 33. Explanation:
          // The maximum of 11,22,33 is 33
          // The maximum of 22,33,44 is 44
          // The maximum of 33,44,33 is 44
          // The maximum of 44,33,22 is 44
          // The maximum of 33,22,11 is 33
     printKMax({55,44,33,22,11,22,33,44,55}, 4);
          // prints 55 44 33 33 44 55
}
```

שימו לב: **המימוש צריך להיות יעיל ולעבוד מהר גם על וקטורים ארוכים מאד וכש-k גדול מאד**. לשם כך יש להשתמש בתור דו-כיווני – deque. בנספח לבחינה זו, נמצא דף ההסבר על deque מאתר cppreference.

ב. הדגימו את פעולת הפונקציה שכתבתם על הקלט הראשון מהתוכנית הראשית (11,22,33,44,33,22,11), חלון בגודל 3. תארו את מצבם של מבני-הנתונים והמשתנים הפנימיים שהשתמשתם בהם בכל שלב.

שאלה 2 [9 נק'] - זיכרון

נתונה המחלקה הבאה:

```
class Vector {
    int m_size=0;                    // default initialization
    int* m_arr=new int [10];  // default initialization
    Vector(int size) { m_size=size; m_arr = new int[size]; } // custom size initialization
        ~Vector(){ delete[] m_arr;};
};
```

א. המחלקה עלולה לגרום לזליגת זיכרון. הסבר מדוע, והסבר מה צריך לשנות בבנאי בשורה 4 כדי שלא תהיה זליגת זיכרון ממנו (יש לכתוב את הבנאי לאחר השינוי; אין לבצע שינויים נוספים).

ב. מה ראוי להוסיף למחלקה כדי שתעבוד כמו שצריך, בהתאם ל"כלל השלושה" (Rule of Three) שלמדנו בהרצאות? ענה בכתיבת הקוד הרלוונטי.

ג. הוספתי תוכנית main כדלקמן אך התוכנית לא מתקמפלת. תקן את השגיאות במחלקה (ייתכן יותר מאחת) כדי שהתוכנית תתקמפל.

```
int main() {
    Vector matrix[10];
}
```

הערה : ניתן לפתור את שלושת הסעיפים ע״י כתיבת כל המחלקה בבת אחת ובתוספת הערות.

שאלה 3 [9 נק'] - מספר עם יחידות

הוסיפו למחלקה NumberWithUnits, שכתבתם במטלה 3, **אופרטור חילוק (/)**. האופרטור צריך לוודא שלשני הארגומנטים שלו יש יחידות תואמות (כגון: שניהם אורך, שניהם משקל וכו' – לפי קובץ היחידות). אם היחידות לא תואמות – יש לזרוק חריגה. אם היחידות תואמות – יש להחזיר את היחס ביניהן **כמספר ממשי**.
מצורפת תוכנית ראשית לדוגמה.

```
#include <iostream>
using namespace std;

#include "NumberWithUnits.hpp"
using namespace ariel;

int main() {
      ifstream units_file{"units.txt"};
      NumberWithUnits::read_units(units_file);

      NumberWithUnits a{2, "km"}; // 2 kilometers
      NumberWithUnits b{500, "m"}; // 500 meters
      cout << (a/b) << endl; // prints 4, since 2 km = 4 * 500 m.
      cout << (b/a) << endl; // prints 0.25, since 500 m = 0.25 * 2 km.

      NumberWithUnits c{2, "kg"}; // 2 kilograms
      cout << (b/c) << endl; // throws an exception, e.g.
         //"units [km] and [kg] do not match."
      return 0;
}
```

א. כתבו את הקוד שצריך להוסיף לקובץ הכותרת NumberWithUnits.hpp.

ב. כתבו את הקוד שצריך להוסיף לקובץ המימוש NumberWithUnits.cpp. מותר לכם להשתמש בפונקציות שמימשתם במטלה עצמה – אין צורך לממש אותן מחדש – אבל יש להסביר בפירוט מה הפונקציות האלו עושות.

שאלה 4 [6 נק'] – לינוקס

נתונה תיקיה במערכת לינוקס, ובה הקבצים הבאים:

```
drwxr-xr-x 2    4096 Jul  3 22:39 .
drwxrwxr-x 3    4096 Jul  3 22:27 ..
-rw-rw-r-- 1      91 Jul  3 22:39 hello.cpp
-rw-rw-r-- 1      39 Jul  3 22:39 run.sh
-rw-rw-r-- 1      39 Jul  3 22:39 string.txt
```

תוכן הקבצים:

hello.cpp:

```
#include <iostream>
using namespace std;
int main() {
      cout << "Enter a string: ";
      string s;
      cin >> s;
      cout << endl << "The string you entered is " << s << endl;
}
```

run.sh:

```
clang++-9 -std=c++2a $1
./a.out
```

string.txt:

```
abcdefghij
```

כשמריצים בשורת הפקודה:

```
$ ./run.sh hello.cpp
```

מתקבלת הודעת שגיאה:

```
bash: ./run.sh: Permission denied
```

א. הסבירו את הודעת השגיאה וממה היא נובעת.

ב. הסבירו איך לתקן את השגיאה - מה צריך לעשות כדי שאותה שורה בדיוק "**run.sh hello.cpp/.**" תקמפל את התוכנית הנמצאת בקובץ hello.cpp ותריץ אותה. הסבירו מה צריך לשנות בקבצים – אם צריך, ואיזה פקודות לינוקס צריך לבצע – אם צריך. **יש לכתוב בפירוט את כל הקוד הדרוש.**

ג. אחרי התיקון של סעיף ב, כשמריצים "**run.sh hello.cpp/.**", התוכנית עוצרת ומחכה לקלט מהמשתמש. כתבו פקודה ב-bash שתריץ את התוכנית כך שלא תחכה לקלט מהמשתמש, אלא תקבל קלט מהקובץ string.txt. למשל, כשמריצים עם תוכן הקובץ הנוכחי, יתקבל הפלט:

```
The string you entered is abcdefghij
```

שאלה 5 [9 נק'] – פרמוטציות

בקורס "תיכנות מערכות ג++", סכום ציוני המטלות בקורס נקבע באופן הבא: עוברים על המטלות לפי הסדר ומחברים את הציונים שלהן, עד שהסכום מגיע ל-48 או יותר; המטלות הבאות לא נחשבות. לדוגמה, אם ציוני המטלות הם:

```
10  20  30  40  50
```

אז הסכום יהיה 30+20+10 = 60.

לעומת זאת, אם ציוני המטלות הם:

```
40  50  30  20  10
```

אז הסכום יהיה 40 + 50 = 90.

סטודנטים רבים התלוננו שזה לא הוגן, ששני אנשים קיבלו אותם ציונים בדיוק רק בסדר אחר, והציון הסופי שלהם יהיה שונה. לכן הוחלט לאפשר לכל סטודנט לסדר את המטלות שלו כרצונו. עליכם לכתוב פונקציה שתעזור לסטודנטים לסדר את המטלות באופן שייתן להם את הציון הגבוה ביותר האפשרי ע"פ הכללים. הפונקציה צריכה לעבור על כל הדרכים האפשריות לסדר את המטלות, לחשב את הסכום בכל סדר אפשרי, ולהחזיר את הסכום הגבוה ביותר.

העזרו בפונקציות הבאות מהספריה התקנית:

```
template< class RandomIt >
    void sort( RandomIt first, RandomIt last );

template< class BidirIt >
    bool next_permutation( BidirIt first, BidirIt last );
```

שימו לב: בנספח תוכלו למצוא את הדף המלא של הפונקציה `next_permutation` מתוך אתר `cppreference`.

כותרת הפונקציה שאתם צריכים לכתוב:

```
int best_sum(vector<int> grades) {..}
```

תוכנית ראשית לדוגמה:

```
int main() {
      cout << best_sum({10,20,30,40,50}) << endl; // 90
      cout << best_sum({50,40,30,20,10}) << endl; // 90
      cout << best_sum({40,50,30,20,10}) << endl; // 90
      cout << best_sum({10,10,32,10,10,33,10,10}) << endl; // 75=32+33+10
}
```

שאלה 6 [8 נק'] – ירושה

נתון הקוד הבא:

```
1 . #include <iostream>
2 . using namespace std;
3 .
4 . class A
5 . {
6 . public:
7 .     virtual void f(const int i) { cout << "A::f" << endl; }
8 .     void g(const int i) { cout << "A::g" << endl; }
9 .     void h(const int i) { cout << "A::h" << endl; }
10.     virtual void i(const int i) { cout << "A::i" << endl; }
11. };
12.
13. class B : public A
14. {
15. public:
16.     void f() { cout << "B::f" << endl; }
17.     void g() { cout << "B::g" << endl; }
18. };
19.
20. class C : public B
21. {
22. public:
23.     void f(const int i) { cout << "C::f" << endl; }
24.     void i(const int i) { cout << "C::i" << endl; }
25. };
26.
27. class D : public C
28. {
29. public:
30.     void f(const int i) { cout << "D::f" << endl; }
31.     void g(const int i) { cout << "D::g" << endl; }
32.     void h(const int i) { cout << "D::h" << endl; }
33.     void i() { cout << "D::i" << endl; }
34. };
35.
36. int main()
37. {
38.     D d;
39.     A* pa = &d;
40.     B* pb = &d;
41.     C* pc = &d;
42.     D* pd = &d;
43.
44.     pa->f(1);
45.     pb->f(1);
46.     pc->f(1);
```

```
47.     pc->f();
48.     pd->f(1);
49.
50.     pa->g(1);
51.     pb->g(1);
52.     pc->g(1);
53.     pd->g(1);
54.
55.     pa->h(1);
56.     pb->h(1);
57.     pc->h(1);
58.     pd->h(1);
59.
60.     pa->i(1);
61.     pd->i(1);
62.
63.     A a, *p = &a;
64.     p->f(1);
65. }
```

עבור שורות 44-64 (לא כולל שורה 63 ולא כולל שורות ריקות) יש לכתוב האם השורה מתקמפלת או לא.
אם השורה מתקמפלת, יש לכתוב מה הפלט של השורה.

שאלה 7 [2 נק']

יינתן מענק של 2 נקודות על כתיבה מסודרת לפי הפירוט הבא:

- השאלות פתורות במחברת הבחינה לפי הסדר שבשאלון;
- כל שאלה מתחילה בעמוד נפרד;
- הכתב ברור וקריא, בלי מחיקות קשקושים חיצים וטקסט מיותר.

std::next_permutation

Defined in header `<algorithm>`

template< class BidirIt > bool next_permutation(BidirIt first, BidirIt last);	(1)	(until C++20)
template< class BidirIt > constexpr bool next_permutation(BidirIt first, BidirIt last);		(since C++20)
template< class BidirIt, class Compare > bool next_permutation(BidirIt first, BidirIt last, Compare comp);	(2)	(until C++20)
template< class BidirIt, class Compare > constexpr bool next_permutation(BidirIt first, BidirIt last, Compare comp);		(since C++20)

Permutes the range `[first, last)` into the next permutation, where the set of all permutations is ordered lexicographically with respect to `operator<` or `comp`. Returns true if such a "next permutation" exists; otherwise transforms the range into the lexicographically first permutation (as if by `std::sort(first, last, comp)`) and returns false.

Parameters

first, last - the range of elements to permute

comp - comparison function object (i.e. an object that satisfies the requirements of *Compare*) which returns true if the first argument is *less* than the second.
The signature of the comparison function should be equivalent to the following:
bool cmp(const Type1 &a, const Type2 &b);
While the signature does not need to have const &, the function must not modify the objects passed to it and must be able to accept all values of type (possibly const) `Type1` and `Type2` regardless of value category (thus, Type1 & is not allowed, nor is Type1 unless for `Type1` a move is equivalent to a copy (since C++11)).
The types Type1 and Type2 must be such that an object of type BidirIt can be dereferenced and then implicitly converted to both of them.

Type requirements
-
`BidirIt` must meet the requirements of *ValueSwappable* and *LegacyBidirectionalIterator*.

Return value

true if the new permutation is lexicographically greater than the old. false if the last permutation was reached and the range was reset to the first permutation.

Exceptions

Any exceptions thrown from iterator operations or the element swap.

Complexity

At most N/2 swaps, where N = std::distance(first, last). Averaged over the entire sequence of permutations, typical implementations use about 3 comparisons and 1.5 swaps per call.

Possible implementation

```
template<class BidirIt>
bool next_permutation(BidirIt first, BidirIt last)
{
    if (first == last) return false;
    BidirIt i = last;
    if (first == --i) return false;

    while (true) {
        BidirIt i1, i2;

        i1 = i;
        if (*--i < *i1) {
            i2 = last;
            while (!(*i < *--i2))
                ;
            std::iter_swap(i, i2);
            std::reverse(i1, last);
            return true;
        }
        if (i == first) {
            std::reverse(first, last);
            return false;
        }
    }
}
```

Example

The following code prints all three permutations of the string "aba"

Run this code

```
#include <algorithm>
#include <string>
#include <iostream>

int main()
{
    std::string s = "aba";
    std::sort(s.begin(), s.end());
    do {
        std::cout << s << '\n';
    } while(std::next_permutation(s.begin(), s.end()));
}
```

Output:

```
aab
```

```
aba
baa
```

std::deque

Defined in header <deque>

template<

class T,
class Allocator = std::allocator<T> (1)

> class deque;

namespace pmr {

template <class T>
using deque = std::deque<T, std::pmr::polymorphic_allocator<T>>; (2) (since C++17)

}

`std::deque` (double-ended queue) is an indexed sequence container that allows fast insertion and deletion at both its beginning and its end. In addition, insertion and deletion at either end of a deque never invalidates pointers or references to the rest of the elements.

As opposed to std::vector, the elements of a deque are not stored contiguously: typical implementations use a sequence of individually allocated fixed-size arrays, with additional bookkeeping, which means indexed access to deque must perform two pointer dereferences, compared to vector's indexed access which performs only one.

The storage of a deque is automatically expanded and contracted as needed. Expansion of a deque is cheaper than the expansion of a std::vector because it does not involve copying of the existing elements to a new memory location. On the other hand, deques typically have large minimal memory cost; a deque holding just one element has to allocate its full internal array (e.g. 8 times the object size on 64-bit libstdc++; 16 times the object size or 4096 bytes, whichever is larger, on 64-bit libc++).

The complexity (efficiency) of common operations on deques is as follows:

- Random access - constant O(1)
- Insertion or removal of elements at the end or beginning - constant O(1)
- Insertion or removal of elements - linear O(n)

`std::deque` meets the requirements of *Container*, *AllocatorAwareContainer*, *SequenceContainer* and *ReversibleContainer*.

Template parameters

T	-	The type of the elements.

	T must meet the requirements of *CopyAssignable* and *CopyConstructible*.	(until C++11)
	The requirements that are imposed on the elements depend on the actual operations performed on the container. Generally, it is required that element type is a complete type and meets the requirements of *Erasable*, but many member functions impose stricter requirements.	(since C++11)
Allocator	- An allocator that is used to acquire/release memory and to construct/destroy the elements in that memory. The type must meet the requirements of *Allocator*. The behavior is undefined (until C++20)The program is ill-formed (since C++20) if Allocator::value_type is not the same as T.	

Iterator invalidation

This section is incomplete

There are still a few inaccuracies in this section, refer to individual member function pages for more detail

Operations	Invalidated
All read only operations	Never
swap, std::swap	The past-the-end iterator may be invalidated (implementation defined)
shrink_to_fit, clear, insert, emplace, push_front, push_back, emplace_front, emplace_back	Always
erase	If erasing at begin - only erased elements If erasing at end - only erased elements and the past-the-end iterator Otherwise - all iterators are invalidated (including the past-the-end iterator).
resize	If the new size is smaller than the old one : only erased elements and the past-the-end iterator If the new size is bigger than the old one : all iterators are invalidated Otherwise - none iterators are invalidated.
pop_front	Only to the element erased
pop_back	Only to the element erased and the past-the-end iterator

Invalidation notes

- When inserting at either end of the deque, references are not invalidated by insert and emplace.
- push_front, push_back, emplace_front and emplace_back do not invalidate any references to elements of the deque.

- When erasing at either end of the deque, references to non-erased elements are not invalidated by erase, pop_front and pop_back.
- A call to resize with a smaller size does not invalidate any references to non-erased elements.
- A call to resize with a bigger size does not invalidate any references to elements of the deque.

Member types

Member type		Definition
`value_type`		`T`
`allocator_type`		`Allocator`
`size_type`		Unsigned integer type (usually std::size_t)
`difference_type`		Signed integer type (usually std::ptrdiff_t)
`reference`		value_type&
`const_reference`		const value_type&
`pointer`		
`Allocator::pointer`	(until C++11)	
std::allocator_traits<Allocator>::pointer	(since C++11)	
`const_pointer`		
`Allocator::const_pointer`	(until C++11)	
std::allocator_traits<Allocator>::const_pointer	(since C++11)	
`iterator`		*LegacyRandomAccessIterator* to `value_type`
`const_iterator`		*LegacyRandomAccessIterator* to `const value_type`
`reverse_iterator`		std::reverse_iterator<iterator>
`const_reverse_iterator`		std::reverse_iterator<const_iterator>

Member functions

(constructor)	constructs the `deque` (public member function)
(destructor)	destructs the `deque` (public member function)
operator=	assigns values to the container (public member function)
assign	assigns values to the container (public member function)
get_allocator	returns the associated allocator

	(public member function)

Element access

at	access specified element with bounds checking (public member function)
operator[]	access specified element (public member function)
front	access the first element (public member function)
back	access the last element (public member function)

Iterators

begin cbegin (C++11)	returns an iterator to the beginning (public member function)
end cend (C++11)	returns an iterator to the end (public member function)
rbegincrbegin (C++11)	returns a reverse iterator to the beginning (public member function)
rendcrend (C++11)	returns a reverse iterator to the end (public member function)

Capacity

empty	checks whether the container is empty (public member function)
size	returns the number of elements (public member function)
max_size	returns the maximum possible number of elements (public member function)
shrink_to_fit (C++11)	reduces memory usage by freeing unused memory (public member function)

Modifiers

clear	clears the contents (public member function)
insert	inserts elements (public member function)
emplace (C++11)	constructs element in-place (public member function)
erase	erases elements (public member function)
push_back	adds an element to the end (public member function)
emplace_back (C++11)	constructs an element in-place at the end (public member function)
pop_back	removes the last element (public member function)
push_front	inserts an element to the beginning (public member function)

emplace_front (C++11)	constructs an element in-place at the beginning (public member function)
pop_front	removes the first element (public member function)
resize	changes the number of elements stored (public member function)
swap	swaps the contents (public member function)

Non-member functions

operator==operator!=operator<operator<=operator>operator>=operator<=> (removed in C++20)(removed in C++20)(removed in C++20)(removed in C++20)(removed in C++20)(C++20)	lexicographically compares the values in the deque (function template)
std::swap(std::deque)	specializes the std::swap algorithm (function template)
erase(std::deque)erase_if(std::deque) (C++20)	Erases all elements satisfying specific criteria (function template)

Deduction guides(since C++17)

Example

Run this code

```
#include <iostream>
#include <deque>

int main()
{
    // Create a deque containing integers
    std::deque<int> d = {7, 5, 16, 8};

    // Add an integer to the beginning and end of the deque
    d.push_front(13);
    d.push_back(25);

    // Iterate and print values of deque
    for(int n : d) {
        std::cout << n << '\n';
    }
}
```

Output:

```
13
7
5
16
8
25
```