

HWO

DANIEL LEVI

315668129

Topic 1 – smart pointers in C++

1. מצביעים חכמים הם אובייקטים אשר מתנהגים כמו מצביעים רגילים, אך בנוסף לכך מתוכננים לניהול הגישה לאובייקטים שאנחנו מקצים, ומתי או היכן לשחרר אותם מהזיכרון. מצביעים חכמים מוחקים באופן אוטומטי את האובייקט אותו הם מנהלים כאשר אין לאובייקט שימוש יותר, עקב חוסר במשתמשים לאובייקט ויציאה מקטע הקוד שמשמש בו.

a. תועלת משימוש במצביעים חכמים-

- המנגנון לוקח מהמתכנת את האחריות על שחרור הזיכרון, ובכך מונע דליפות זיכרון.
- המנגנון יודע לנהל בעלות על אובייקט, כך שהאובייקט ישוחרר רק כאשר כל הבעלויות עליו מסתיימות. זה עוזר לנו להימנע משגיאה בזמן ריצה בשל גישה לזיכרון ששוחרר וכו'.

עלויות בשימוש במצביעים חכמים-

- מנגנון הניהול של המצביעים החכמים מוסיף תקורה לתוכנית. התקורה אמנם קטנה יחסית, אך עשויה להיות משמעותית בתחומים מסוימים (HPC).
- על ידי שימוש במנגנון החדש של מצביעים חכמים, אנחנו פוגעים בתאימות לאחור. למשל, ספריות שרוצות לבצע שימוש במצביעים חכמים יפגעו ביכולת שלהן להיות חלק מפרויקט שנכתב בסטנדרט קודם.

b. מימוש למצביע חכם-

```
template <typename T>
class SmartPointer
{
private:
    T* ptr; // Is used for getting access to the memory
    address
public:
    SmartPointer(T* ptrToStore) : ptr(ptrToStore) { }

    ~SmartPointer()
    {
        delete ptr;
    }

    T& operator*()
    {
        return *ptr;
    }

    T* operator->()
    {
        return ptr;
    }
};
```

2. נשתמש במצביע רגיל כאשר אנחנו צריכים לחסוך בתקורה. למשל במערכות קטנות שמוגבלות מאוד במשאבים. השימוש במצביעים חכמים גורם לתקורה שיכולה להיות משמעותית במערכות שכאלה.
- a. דוגמה בה התקורה של המצביעים החכמים יכולה לבלוט, היא תוכנית בה נקצה מצביע בצורה מחזורית לאורך מספר רב של פעמים. למשל, ניקח תוכנית בה מוגדרת המחלקה של Date, וקריאות לפונקציות של הקצאת מצביע לאובייקט החדש, הדפסת התאריך ושחרור המצביע. את הפונקציות נריץ מספר רב של פעמים (1,000,000) ונראה הצטברות של תקורה לאורך שימוש רב במצביעים חכמים.
- Date מחלקת

```
class Date {
    int day;
    int month;
    int year;
public:
    Date(int day, int month, int year) : day(day), month(month), year(year) {}
    ~Date() {}

    int getDay() { return day; }
    int getMonth() { return month; }
    int getYear() { return year; }
};
```

פונקציות להקצאת ושחרור מצביעים של מחלקת Date, ולהדפסת התאריך-

```
void allocateRaw()
{
    int day = 20, month = 4, year = 2023;
    Date* rawPointer = new Date(day, month, year);
    cout << "The date today - " << rawPointer->getDay() << "/" <<
        rawPointer->getMonth() << "/" << rawPointer->getYear() << std::endl;

    delete rawPointer;
}

void allocateSmart()
{
    int day = 20, month = 4, year = 2023;
    unique_ptr<Date> smartDate(new Date(day, month, year));
    cout << "The date today - " << rawPointer->getDay() << "/" <<
        rawPointer->getMonth() << "/" << rawPointer->getYear() << std::endl;
}
```

```
int main()
{
    for (int i = 0; i < 1000000; i++)
    {
        allocateRaw();
    }
    for (int i = 0; i < 1000000; i++)
    {
        allocateSmart();
    }

    return 0;
}
```

פונקציית ה-main-

b. שימוש ב-gprof על התוכנית שכתבתי תציג פער ביצועים לרעת השימוש במצביעים חכמים-

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative	seconds	self seconds	calls	self ns/call	total ns/call	name
25.04	0.02	0.02	1000000	20.03	70.11		allocateSmart()
12.52	0.03	0.01	2000000	5.01	5.01		std:: Head base<0ul, Date*, false>:: M head(std:: Head base<0ul, Date*,
12.52	0.04	0.01	2000000	5.01	10.02		Date*& std:: get helper<0ul, Date*, std::default_delete<Date> >(std::
12.52	0.05	0.01	1000000	10.02	10.02		allocateRaw()
12.52	0.06	0.01	1000000	10.02	10.02		std:: Head base<0ul, Date*, false>:: Head base()
12.52	0.07	0.01	1000000	10.02	20.03		std::unique_ptr<Date, std::default_delete<Date> >:: M ptr()
12.52	0.08	0.01	1000000	10.02	20.03		std:: Tuple impl<0ul, Date*, std::default_delete<Date> >:: Tuple impl()
0.00	0.08	0.00	3000000	0.00	0.00		std::unique_ptr<Date, std::default_delete<Date> >::get() const
0.00	0.08	0.00	3000000	0.00	0.00		std::unique_ptr<Date, std::default_delete<Date> >::operator->() const
0.00	0.08	0.00	3000000	0.00	0.00		std:: unique_ptr impl<Date, std::default_delete<Date> >:: M ptr() const
0.00	0.08	0.00	3000000	0.00	0.00		std:: Head base<0ul, Date*, false>:: M head(std:: Head base<0ul, Date*,
0.00	0.08	0.00	3000000	0.00	0.00		std:: Tuple impl<0ul, Date*, std::default_delete<Date> >:: M head(std::
0.00	0.08	0.00	3000000	0.00	0.00		Date* const& std:: get helper<0ul, Date*, std::default_delete<Date> >()
0.00	0.08	0.00	3000000	0.00	0.00		std::tuple element<0ul, std::tuple<Date*, std::default_delete<Date> > >
0.00	0.08	0.00	2000000	0.00	0.00		Date::getDay()
0.00	0.08	0.00	2000000	0.00	0.00		Date::getYear()
0.00	0.08	0.00	2000000	0.00	0.00		Date::getMonth()
0.00	0.08	0.00	2000000	0.00	0.00		Date::Date(int, int, int)
0.00	0.08	0.00	2000000	0.00	0.00		Date::~Date()
0.00	0.08	0.00	2000000	0.00	5.01		std:: Tuple impl<0ul, Date*, std::default_delete<Date> >:: M head(std::
0.00	0.08	0.00	2000000	0.00	10.02		std:: unique_ptr impl<Date, std::default_delete<Date> >:: M ptr()
0.00	0.08	0.00	2000000	0.00	10.02		std::tuple element<0ul, std::tuple<Date*, std::default_delete<Date> > >
0.00	0.08	0.00	1000000	0.00	0.00		std::default_delete<Date>::operator()(Date*) const
0.00	0.08	0.00	1000000	0.00	0.00		std:: Head base<lul, std::default_delete<Date>, true>:: M head(std:: Head
0.00	0.08	0.00	1000000	0.00	0.00		std:: Head base<lul, std::default_delete<Date>, true>:: Head base()
0.00	0.08	0.00	1000000	0.00	0.00		std::unique_ptr<Date, std::default_delete<Date> >::get deleter()
0.00	0.08	0.00	1000000	0.00	30.05		std::unique_ptr<Date, std::default_delete<Date> >::unique_ptr<std::defau
0.00	0.08	0.00	1000000	0.00	0.00		std:: Tuple impl<lul, std::default_delete<Date> >:: M head(std:: Tuple
0.00	0.08	0.00	1000000	0.00	0.00		std:: Tuple impl<lul, std::default_delete<Date> >:: Tuple impl()
0.00	0.08	0.00	1000000	0.00	0.00		std:: unique_ptr impl<Date, std::default_delete<Date> >:: M deleter()
0.00	0.08	0.00	1000000	0.00	30.05		std:: unique_ptr impl<Date, std::default_delete<Date> >:: unique_ptr impl
0.00	0.08	0.00	1000000	0.00	20.03		std::tuple<Date*, std::default_delete<Date> >::tuple<Date*, std::default
0.00	0.08	0.00	1000000	0.00	0.00		std::default_delete<Date>& std:: get helper<lul, std::default_delete<Da
0.00	0.08	0.00	1000000	0.00	0.00		std::tuple element<lul, std::tuple<Date*, std::default_delete<Date> > >

3.

a. ההבדל בין struct ב-C וב-C++ נעוץ בעובדה ש-C++ מאפשרת תכנות מונחה עצמים. כך לדוגמה, ב-C++ ניתן להוסיף פונקציות של ה-struct בתוכו, וכך לנהל אובייקט שנבנה בעזרת ה-Constructor ומשתחרר בעזרת ה-Destructor, מנוהל על ידי אופרטורים שמותאמים לו(בעזרת העמסת אופרטורים בתוך ה-struct) ולנהל את הגישה החיצונית לשדות האובייקט בעזרת מילות המפתח- public, private ו-protected. סימן בולט לכך הוא אופן יצירת אובייקט מסוג ה-struct. ב-C נגדיר אובייקט שכזה על ידי- struct Name object; בעוד שב-C++ נגדיר אובייקט שכזה על ידי Name object; כאשר Name הוא סוג האובייקט, ו-object הוא אובייקט חדש מהסוג שלו.

מכך ניתן לראות את הדגש שניתן ב-C++ לתכנות מונחה עצמים בעזרת struct ו-class.

b. ההבדל המהותי בין struct ל-class ב-C++ הוא העובדה שהשדות ב-struct הם באופן דיפולטיבי בעלי הרשאת גישה שהיא public, בעוד השדות ב-class הם באופן דיפולטיבי בעלי הרשאת גישה שהיא private.

c. כאשר ננסה לשחרר את הרשימה, ייתכן שבזמן שחרור ה-root, ההורס הדיפולטיבי של המחלקה יקרא להורס של unique_ptr, כך שהאיבר הבא ברשימה גם הוא ישוחרר, וכך בשרשרת כל הצמתים ברשימה ישוחררו. מצד שני, ייתכן שנבצע שחרור כפול של זיכרון עקב התנהלות לא נכונה במצביעים וכך נגרום לשגיאת זמן ריצה.

d. ולגרינד- מקרה ראשון- שחרור הזיכרון כמו שצריך לקרות. אין שגיאות ולגרינד.

```
student@student:~/ParallelComputing/HW0/LinkedList$ valgrind --leak-check=yes ./a.out
==3817== Memcheck, a memory error detector
==3817== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3817== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==3817== Command: ./a.out
==3817==
==3817== HEAP SUMMARY:
==3817==   in use at exit: 0 bytes in 0 blocks
==3817==   total heap usage: 3 allocs, 3 frees, 72,736 bytes allocated
==3817==
==3817== All heap blocks were freed -- no leaks are possible
==3817==
==3817== For counts of detected and suppressed errors, rerun with: -v
==3817== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

- מקרה שני- שחרור זיכרון כפול. יש שגיאה.

```
student@student:~/ParallelComputing/HW0/LinkedList$ valgrind --leak-check=yes ./a.out
==4274== Memcheck, a memory error detector
==4274== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4274== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==4274== Command: ./a.out
==4274==
==4274== Invalid read of size 8
==4274== at 0x108956: std::unique_ptr<Node, std::default_delete<Node>>::~unique_ptr() (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== by 0x108931: Node::~Node() (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== by 0x108866: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== Address 0x5b7fc80 is 8 bytes inside a block of size 16 free'd
==4274== at 0x4C33238: operator delete(void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4274== by 0x108855: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== Block was alloc'd at
==4274== at 0x4C3217F: operator new(unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4274== by 0x10870B: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274==
==4274== Invalid write of size 8
==4274== at 0x108983: std::unique_ptr<Node, std::default_delete<Node>>::~unique_ptr() (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== by 0x108931: Node::~Node() (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== by 0x108866: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== Address 0x5b7fc80 is 8 bytes inside a block of size 16 free'd
==4274== at 0x4C33238: operator delete(void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4274== by 0x108855: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== Block was alloc'd at
==4274== at 0x4C3217F: operator new(unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4274== by 0x10870B: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274==
==4274== Invalid free() / delete / delete[] / realloc()
==4274== at 0x4C33238: operator delete(void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
```

```
==4274== Invalid free() / delete / delete[] / realloc()
==4274== at 0x4C33238: operator delete(void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4274== by 0x108873: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== Address 0x5b7fc80 is 0 bytes inside a block of size 16 free'd
==4274== at 0x4C33238: operator delete(void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4274== by 0x108855: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274== Block was alloc'd at
==4274== at 0x4C3217F: operator new(unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4274== by 0x10870B: main (in /home/student/ParallelComputing/HW0/LinkedList/a.out)
==4274==
==4274== HEAP SUMMARY:
==4274==   in use at exit: 0 bytes in 0 blocks
==4274==   total heap usage: 3 allocs, 4 frees, 72,736 bytes allocated
==4274==
==4274== All heap blocks were freed -- no leaks are possible
==4274==
==4274== For counts of detected and suppressed errors, rerun with: -v
==4274== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

Topic 2 – Lambda functions

1. מימוש פונקציית ה-lambda על ידי העברת הווקטור by value

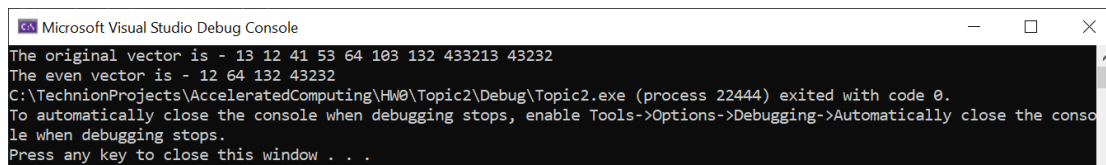
```
int main()
{
    auto even_vector = [](std::vector<int> nums_vector)
    {
        std::vector<int> new_vector;
        for (size_t i = 0; i < nums_vector.size(); i++)
        {
            if (nums_vector.at(i) % 2 == 0)
            {
                new_vector.insert(new_vector.end(), nums_vector.at(i));
            }
        }
        return new_vector;
    };

    std::vector<int> original_vector{ 13, 12, 41, 53, 64, 103, 132, 433213, 43232 };
    std::vector<int> new_vector = even_vector(original_vector);
    std::cout << "The original vector is - ";
    for (auto i : original_vector)
        std::cout << i << ' ';

    std::cout << std::endl << "The even vector is - ";
    for (auto i : new_vector)
        std::cout << i << ' ';

    return 0;
}
```

תוצאת ההרצה שנקבל היא כפי שציפינו-



```
Microsoft Visual Studio Debug Console
The original vector is - 13 12 41 53 64 103 132 433213 43232
The even vector is - 12 64 132 43232
C:\TechnionProjects\AcceleratedComputing\HW0\Topic2\Debug\Topic2.exe (process 22444) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

כעת נראה מימוש על ידי העברת הווקטור by reference

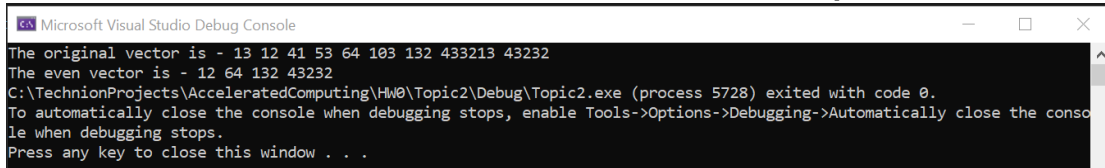
```
int main()
{
    auto even_vector = [](std::vector<int>& nums_vector)
    {
        std::vector<int> new_vector;
        for (size_t i = 0; i < nums_vector.size(); i++)
        {
            if (nums_vector.at(i) % 2 == 0)
            {
                new_vector.insert(new_vector.end(), nums_vector.at(i));
            }
        }
        return new_vector;
    };

    std::vector<int> original_vector{ 13, 12, 41, 53, 64, 103, 132, 433213, 43232 };
    std::vector<int> new_vector = even_vector(original_vector);
    std::cout << "The original vector is - ";
    for (auto i : original_vector)
        std::cout << i << ' ';

    std::cout << std::endl << "The even vector is - ";
    for (auto i : new_vector)
        std::cout << i << ' ';

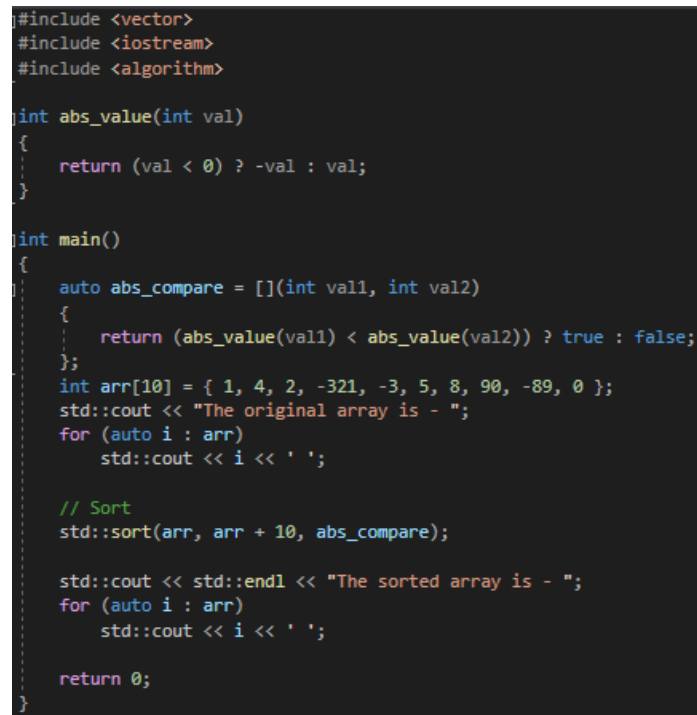
    return 0;
}
```

גם כעת, התוצאה שנקבל תהיה נכונה-



```
Microsoft Visual Studio Debug Console
The original vector is - 13 12 41 53 64 103 132 433213 43232
The even vector is - 12 64 132 43232
C:\TechnionProjects\AcceleratedComputing\HW0\Topic2\Debug\Topic2.exe (process 5728) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

2. מימוש lambda להשוואת מספרים לפי הערך האבסולוטי שלהם, כולל מיון מערך על פי השוואה זו-



```
#include <vector>
#include <iostream>
#include <algorithm>

int abs_value(int val)
{
    return (val < 0) ? -val : val;
}

int main()
{
    auto abs_compare = [](int val1, int val2)
    {
        return (abs_value(val1) < abs_value(val2)) ? true : false;
    };

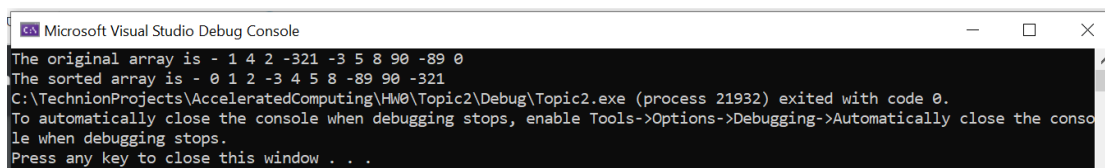
    int arr[10] = { 1, 4, 2, -321, -3, 5, 8, 90, -89, 0 };
    std::cout << "The original array is - ";
    for (auto i : arr)
        std::cout << i << ' ';

    // Sort
    std::sort(arr, arr + 10, abs_compare);

    std::cout << std::endl << "The sorted array is - ";
    for (auto i : arr)
        std::cout << i << ' ';

    return 0;
}
```

התוצאה אכן יוצאת כנדרש-



```
Microsoft Visual Studio Debug Console
The original array is - 1 4 2 -321 -3 5 8 90 -89 0
The sorted array is - 0 1 2 -3 4 5 8 -89 90 -321
C:\TechnionProjects\AcceleratedComputing\HW0\Topic2\Debug\Topic2.exe (process 21932) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

```

#include <vector>
#include <algorithm>
#include <iostream>

int main()
{
    int comparison_counter = 0;
    std::vector<int> vec{ 0,5,2,9,7,6,1,3,4,8 };
    auto sort_lambda = [&comparison_counter](int val1, int val2)
    {
        comparison_counter++;
        return (val1 < val2) ? true : false;
    };
    std::cout << "The original vector is - ";
    for (auto i : vec)
    {
        std::cout << i << ' ';
    }

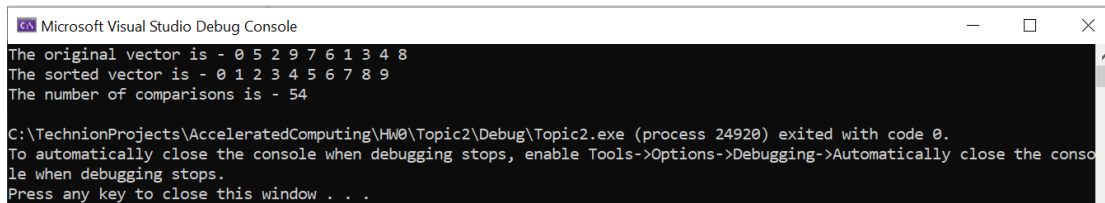
    // sort
    std::sort(vec.begin(), vec.end(), sort_lambda);

    std::cout << std::endl << "The sorted vector is - ";
    for (auto i : vec)
    {
        std::cout << i << ' ';
    }

    std::cout << std::endl << "The number of comparisons is - " << comparison_counter << std::endl;
    return 0;
}

```

התשובה המודפסת עבור הדוגמה הנתונה היא-



```

Microsoft Visual Studio Debug Console
The original vector is - 0 5 2 9 7 6 1 3 4 8
The sorted vector is - 0 1 2 3 4 5 6 7 8 9
The number of comparisons is - 54

C:\TechnionProjects\AcceleratedComputing\HW0\Topic2\Debug\Topic2.exe (process 24920) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

אלגוריתם המיון שנמצא בשימוש ה-STL הוא IntroSort. אלגוריתם מיון שמורכב ממעין קומבינציה של שלושת אלגוריתמי המיון, QuickSort, HeapSort, ו-InsertionSort.

Topic 3 – Functors in C++

.1

```
class Accumulator {  
    float sum;  
public:  
    Accumulator() { sum = 0; }  
    ~Accumulator() {}  
  
    float operator()(float to_add)  
    {  
        this->sum += to_add;  
        return sum;  
    }  
};
```

```
int main()  
{  
    // Q1.a  
    std::vector<float> pdf_vector = { 0.1f,0.05f,0.4f,0.15f,0.2f,0.1f };  
  
    // Q1.b  
    float vec_sum = 0;  
    for (float i : pdf_vector)  
    {  
        vec_sum += i;  
    }  
    assert(vec_sum == 1);  
  
    // Q1.c  
    Accumulator acc;  
    std::vector<float> cdf_vector;  
    for (float i : pdf_vector)  
    {  
        cdf_vector.push_back(acc(i));  
    }  
  
    // Prints  
    std::cout << "The pdf vector is - ";  
    for (float i : pdf_vector)  
    {  
        std::cout << i << " ";  
    }  
  
    std::cout << std::endl << "The cdf vector is - ";  
    for (float i : cdf_vector)  
    {  
        std::cout << i << " ";  
    }  
  
    return 0;  
}
```

תוצאות הדפסה-

```
Microsoft Visual Studio Debug Console
The pdf vector is - 0.1 0.05 0.4 0.15 0.2 0.1
The cdf vector is - 0.1 0.15 0.55 0.7 0.9 1
C:\TechnionProjects\AcceleratedComputing\HW0\Topic3\Debug\Topic3.exe (process 19532) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

שימוש ב-functor אפשר לנו קוד יותר ברור, קריא וגמיש. במקום לבצע קוד שייתן לנו סכימה בכל פעם מחדש, אנחנו מקבלים מעין פונקציה עם מטרה ברורה (accumulator), וכאשר נצטרך לשנות במעט את הפעולה אז לא נצטרך לעבור על פני כל הקוד. נוכל פשוט לשנות את הצורה בה ה-functor עובד במחלקה. בנוסף, כאשר נצטרך להשתמש בפונקציה הזו שוב, אז במקום לשכפל שוב את המנגנון עם עוד משתנה גלובאלי מחוץ לפונקציית ה-main, נוכל פשוט ליצור עוד אובייקט מסוג המחלקה ולהשתמש בו. זה מקטין את המקום לטעויות בקוד.

2. Functors

- יתרונות- יכולים לשמור מצב מסוים. למשל, בתרגיל הקודם כתבנו Accumulator ששמר בכל רגע נתון את המצב של ה-Functor. זה יכול להיות מאוד שימושי במצבים מהסוג הזה. השימוש בהם גם מוסיף לנו גמישות רבה ביכולת שלנו לכתוב פונקציות. השימוש במחלקה כאובייקט פונקציונלי יכול להביא לנו אופציות רבות לתכנות פונקציונלי.
- חסרונות- כתיבת Functors והשימוש בהן יכול להיות פחות אינטואיטיבי וברור למתכנתים. השימוש בהם הוא לא מובן מאליו וקצת חורג מצורת התכנות המסורתית שרבים הורגלו אליה.
- דוגמה למקרה רלוונטי לשימוש ב-Functors- בתרגיל לפני, השימוש ב-Accumulator נתן לנו ניצול טוב של יתרונות השיטה.

Lambda Functions

- יתרונות- נכתבות בצורת inline ומאפשרות לחסוך בעוד פונקציות נפרדות ומנגנונים ארוכים כמו Functors. השימוש בהן מאוד גמיש ומאפשר פעולות גם על משתנים בתוך הבלוק בו הן מוגדרות. דבר שכזה עשוי לעזור לנו במקרים מסוימים בהן יש צורך לשנות את המשתנים בתוך הבלוק כתוצאה מפונקציה כלשהי.
- חסרונות- פחות קריא מאפשרויות אחרות, עשוי לגרום לכפילויות קוד עקב שימוש בבלוקים שונים ובחלקי קוד שונים. בנוסף, פונקציות שכאלה גוררות גם תקורה בזמן השימוש בהן.
- דוגמה למקרה רלוונטי לשימוש ב-Lambda Functions- למשל ב-Topic 2 שאלה 3, ניצלנו טוב את השימוש של ה-Lambda בתוך בלוק מסוים. היכולת שלנו לשנות את ה-comparison_counter בתוך ה-Lambda עזרה לנו לחסוך בפעולות מיותרות והעברת מצביע/רפרנס עם התנהלות מסורבלת יותר של הקאונטר.

-Function Pointers

- יתרונות- מאפשר התנהלות גמישה יותר. נוכל להשתמש בהן בכדי ליצור פונקציות גנריות, ובכך למנוע כפילויות קוד.
- חסרונות- מוריד את קריאות הקוד. השימוש בהן ממש מסובך מעצם העובדה שההתנהלות איתם היא בעזרת מצביעים. ההתנהלות הזאת כמעט תמיד מבלבלת ועלולה לגרום לשגיאות לא צפויות עקב ניהול שגוי של מצביעים(למשל קריאה מתוך NULL וכד').
- דוגמה למקרה רלוונטי לשימוש ב-Function Pointers

```
void perform_operation(int x, int y, int (*operation)(int, int)) {  
    std::cout << "Performing operation on " << x << " and " << y << ": ";  
    std::cout << operation(x, y) << std::endl;  
}
```

השימוש במצביע לפונקציה עוזר לנו ליצור גנריות בקוד. כאן הגנריות מתבטאת ביכולת לבצע פעולות על שני משתנים מסוג integer. זה יאפשר לנו באותה צורה להגדיר למשל פונקציה בשם "calculator", ולהשתמש ב-API מאוד דומה שירכז לנו את כל הפעולות "תחת אותה קורת גג".