תכנות מכוון עצמים ו- C++
יחידה 12
STL – Standard Templates Library

קרן כליף

ביחידה זו נלמד:

- סקירת ה- STL
- סוגים של conatiner'ים
- איטרטור
- Object Functions
- אלגוריתמים
- המחלקה string
- תכנות פונקציונלי / Lambda Expression

קישור לזיפ עם כל הדוגמאות <<

STL – Standard Template Library

- ה- STL הוא אוסף מימושים למבני-נתונים ואלגוריתמים בשפת C++
- מימושים אלו מתבססים על templates
- בין מבני-הנתונים הממומשים ניתן למצוא:
  - מערך
  - רשימה מקושרת
  - מפה
  - קבוצה
- בין האלגוריתמים הממומשים ניתן למצוא:
  - חיפוש
  - מיון
  - מינימום ומקסימום

STL מבני הנתונים שנסקור במצגת זו

- מבני נתונים שהם Sequences Container (פניה לאיבר לפי מיקומו)
  - vector – מימוש למערך שיודע להגדיל את עצמו
  - list – מימוש לרשימה מקושרת דו-כיוונית

- מבני נתונים שהם Associative Container (פניה לאיבר לפי מפתח)
  - set – מימוש לקבוצה בלי כפילות איברים. הנתונים שמורים בעץ בינארי
  - multiset – כנ"ל, אך מאפשר כפילות איברים
  - map – מימוש למפה: אוסף של זוגות: key+value. לכל מפתח יהיה ערך יחיד
  - multimap – כנ"ל, אבל לכל מפתח יכולים להיות כמה ערכים

Sequences Container

vector דוגמת שימוש

```
#include <vector>
using namespace std;
int main()
{
    vector<int> numbers;

    cout << "Is collection empty? " << numbers.empty() << endl;
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;

    numbers.push_back(4);
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
    numbers.push_back(8);
    numbers.push_back(2);
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
    cout << "Is collection empty? " << numbers.empty() << endl;
    // values in the vector: 4 8 2

    int firstValue = numbers.front();
    cout << "first value is " << firstValue << endl;

    numbers.pop_back();
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
    numbers.reserve(10);
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
    numbers.clear();
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
}
```

גודל לוגי

גודל פיזי

מוסיפה איבר לסוף

ניתן לראות שהגודל הפיזי גדל כל פעם ב- 1

```
Is collection empty? 1
size=0 capacity=0
size=1 capacity=1
size=3 capacity=3
Is collection empty? 0
first value is 4
size=2 capacity=3
size=2 capacity=10
size=0 capacity=10
```

מסיר את האיבר האחרון

ניתן לראות שלא מקטינים את הגודל הפיזי

מקצה איברים לגודל הפיזי

דוגמת שימוש list

```
#include <list>

int main()
{
    list<int> numbers;

    cout << "Is collection empty? " << numbers.empty() << endl;
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;

    numbers.push_back(4);
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
    numbers.push_back(8);
    numbers.push_back(2);
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
    cout << "Is collection empty? " << numbers.empty() << endl;
    // values in the vector: 4 8 2

    int firstValue = numbers.front();
    cout << "first value is " << firstValue << endl;

    numbers.pop_back();
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
    numbers.reserve(10);
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
    numbers.clear();
    cout << "size=" << numbers.size() << " capacity=" << numbers.capacity() << endl;
}
```

vector דוגמה לאוסף המכיל אובייקטים

```
class Person
{
    char name[10];
public:
    Person(const char* name) {setName(name);}
    Person(const Person& other)
    {
        cout << "In Person(copy) " << other.name << "\n";
        *this = other;
    }
    ~Person() { cout << "In Peron::~Person " << name << "\n"; }
    void setName(const char* name) {strcpy(this->name, name);}
};

int main()
{
    vector<Person> persons;
    vector<Person*> personsPtr;

    Person p1("gogo"), p2("momo");

    persons.push_back(p1);
    cout << "---------------\n";
    persons.push_back(p2);

    personsPtr.push_back(&p1);
    personsPtr.push_back(&p2);

    p1.setName("gogogo");
}
```

הגדלת המערך, ולכן שיכפול איבריו!

```
In Person(copy) gogo
---------------
In Person(copy) gogo
In Peron::~Person gogo
In Person(copy) momo
In Peron::~Person momo
In Peron::~Person gogogo
In Peron::~Person gogo
In Peron::~Person momo
```

איברי המערך נהרסים לפי סדרם הסידורי

Watch 1

Name	Value
p1	{name=0x0012ff10 "gogogo" }
p2	{name=0x0012fefc "momo" }
persons	[2]({name=0x00345fa8 "gogo" },{name=0x00345fb2 "momo" })
[0]	{name=0x00345fa8 "gogo" }
[1]	{name=0x00345fb2 "momo" }
personsPtr	[2](0x0012ff10 {name=0x0012ff10 "gogogo" },0x0012fefc {nam
[0]	0x0012ff10 {name=0x0012ff10 "gogogo" }
[1]	0x0012fefc {name=0x0012fefc "momo" }

מימושי ה- STL יעילים ועוברים ב- move c'tor כשאפשר

המחלקה

```
class Person
{
    char* name;
public:
    Person(const char* name)
    {
        cout << "In Person::Person name=" << name << endl;
        this->name = strdup(name);
    }
    Person(const Person& other)
    {
        cout << "In Person(copy) " << other.name << "\n";
        this->name = strdup(other.name);
    }
    Person(Person&& other)
    {
        cout << "In Person(move) " << other.name << "\n";
        this->name = other.name;
        other.name = nullptr;
    }
    ~Person()
    {
        cout << "In Peron::~Person ";
        if (name != nullptr)
            cout << name << "\n";
        else
            cout << " that was moved\n";
        delete[]name;
    }
};
```

מימושי ה- STL יעילים ועוברים ב- move c'tor כשאפשר ה- main

```
int main()
{
    vector<Person> persons;

    Person p1("gogo"), p2("momo");
    cout << "1 ----------------\n";

    persons.push_back(p1);
    cout << "2 ----------------\n";

    persons.push_back(p2);
    cout << "3 ----------------\n";

    persons.push_back(Person("yoyo"));
    cout << "4 ----------------\n";
}
```

```
In Person::Person name=gogo
In Person::Person name=momo
1 ----------------
In Person(copy) gogo
2 ----------------
In Person(move) gogo
In Peron::~Person  that was moved
In Person(copy) momo
3 ----------------
In Person::Person name=yoyo
In Person(move) gogo
In Person(move) momo
In Peron::~Person  that was moved
In Peron::~Person  that was moved
In Person(move) yoyo
In Peron::~Person  that was moved
4 ----------------
In Peron::~Person momo
In Peron::~Person gogo
In Peron::~Person gogo
In Peron::~Person momo
In Peron::~Person yoyo
```

STL יתרונות

- שימוש בקוד מוכן לפעולות שכיחות
- ניהול זכרון
- מימושי האוספים השונים בעלי ממשק כמעט זהה
  - מקל על למידתם
  - מקל על החלפת שימוש אחד באחר

- בקישור הבא נתן למצוא את כל מה שה- STL מכיל:
  http://www.cplusplus.com/reference/

Sequences Container סיכום

- לכולם יש את המתודות הבאות (ועוד):

שם המתודה	פעולה
`void push_back(const T& val)`	מקבלת ערך ומוסיפה **העתק** שלו לסוף האוסף
`bool empty() const`	מחזירה אמת אם באוסף אין איברים, שקר אחרת
`T front() const`	מחזירה **העתק** של האיבר הראשון באוסף. עף אם האוסף ריק.
`void clear()`	מרוקנת את כל איברי האוסף. עוברת ב- d'tor של כל איבר. יש לשים לב שאם האיברים הם מצביעים, אינה משחררת אותם.
`void pop_back()`	מורידה מהאוסף את האיבר האחרון, ומשחררת אותו
`int size() const`	מחזירה את מספר האיברים באוסף (גודל לוגי)
`operator=`	מימוש אופרטור השמה
`operator==`	פונקציית friend הבודקת האם איברי האוסף זהים (גם בסדר)

Sequences Container סיכום (2)

- ל- list יש גם את המתודות הבאות:

שם המתודה	פעולה
`void push_front(const T& val)`	מקבלת ערך ומוסיפה **העתק** שלו לתחילת האוסף
`void pop_front()`	מורידה מהאוסף את האיבר הראשון, ומשחררת אותו

- ל- vector יש גם את המתודות:

שם המתודה	פעולה
`int capacity() const`	מחזירה את הגודל הפיזי של המערך
`void reserve(int n)`	במידה ו- capacity < n מגדילה את capacity בהתאם. משמש לצרכי יעילות.

איטרטורים

הבעיה בריצה על vector כאילו הוא מערך

- כאשר רצים על וקטור עם הלולאה כאילו היא מערך רגיל, הקוד לא עובר קומפילציה כאשר האוסף משתנה להיות מטיפוס אחר
- נרצה דרך לעבור על לולאה שתתמוך בכל סוגי האוספים

```
int main()
{
    list<int> numbers;

    numbers.push_back(1);
    numbers.push_back(2);
    numbers.push_back(3);

    for (int i = 0; i < numbers.size(); i++)
        cout << numbers[i] << " ";
    cout << endl;
}
```

```
1 2 3
```

מעבר על אוסף באמצעות איטרטור

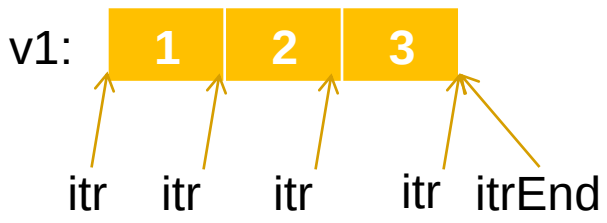
- איטרטור הוא אובייקט שמחזיק התייחסות לאיבר מסויים באוסף
  - לכן אנחנו מתייחסים לאיטרטור כמעין מצביע
- בכל מחלקה המממשת מבנה-נתונים ב- STL ממומשת מחלקה פנימית iterator

```
int main()
{
      list<int>  v1;

      v1.push_back(1);
      v1.push_back(2);
      v1.push_back(3);

      list<int>::iterator  itr    = v1.begin();
      list<int>::iterator  itrEnd = v1.end();
      for ( ; itr != itrEnd  ;   ++itr )
            cout << *itr << " ";          // → 1  2  3
      cout << endl;
}
```

שימוש באופרטור !=

שימוש באופרטור ++

שימוש באופרטור *



איטרטורים

```
int main()
{
     vector<int> v1;

     v1.push_back(1);
     v1.push_back(2);
     v1.push_back(3);

     vector<int>::iterator  itr    = v1.begin();
     vector<int>::iterator  itrEnd = v1.end();
     for ( ; itr != itrEnd ; ++itr)
            cout << *itr << " ";
     cout << endl;
}
```

- מחלקת האיטרטור מספקת מימשק למעבר על כל איברי האוסף באופן סדרתי
- בכל מחלקה המממשת מבנה נתונים יש את השיטות:
  - `begin()` – המחזירה איטרטור לאיבר הראשון באוסף
  - `end()` – המחזירה איטרטור לאיבר אחרי האחרון באוסף
- היתרון:
  - החלפת מבנה-נתונים אחד באחר לא תגרור שינוי בשימוש

מחלקות iterator שיטות

```
int main()
{
    vector<int> v1;

    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);

    vector<int>::iterator  itr    = v1.begin();
    vector<int>::iterator  itrEnd = v1.end();
    for ( ; itr != itrEnd ; ++itr)
            cout << *itr << " ";
    cout << endl;
}
```

בגלל המימוש הפנימי של אופרטור ++, עדיף להשתמש גרסת ה- prefix!

- אופרטורי ++
  - מקדמים את האיטרטור להכיל הצבעה לאיבר הבא באוסף
- אופרטור *
  - מחזיר את התוכן של האיבר אליו האיטרטור מכיל הצבעה
- אופרטור ==
  - בודק ששני איטרטורים מכילים הצבעה לאותו איבר
- אופרטור =!
  - בודק ששני איטרטורים אינם מכילים הצבעה לאותו איבר

הדפסת אוספים דוגמאות למימוש

```
void printVector(vector<int>& collection)
{
    vector<int>::iterator itr    = collection.begin();
    vector<int>::iterator itrEnd = collection.end();

    if (itr == itrEnd)
    {
        cout << "Collection is empty!\n";
        return;
    }

    for ( ; itr != itrEnd ; ++itr)
        cout << *itr << " ";
    cout << endl;
}
```

בכל מחלקה המממשת מבנה-נתונים ב- STL ממומשות 2 מחלקות פנימיות: iterator ו- const_iterator

- כאשר מבנה הנתונים עליו רצים הוא const נעבוד עם const_iterator המגן על תוכן האיבר

```
void printVector(const vector<int>& collection)
{
    vector<int>::const_iterator itr    = collection.begin();
    vector<int>::const_iterator itrEnd = collection.end();

    if (itr == itrEnd)
    {
        cout << "Collection is empty!\n";
        return;
    }

    for ( ; itr != itrEnd ; ++itr)
        cout << *itr << " ";
    cout << endl;
}
```

ואין סיבה להגביל מימוש זה לוקטור בלבד

הדפסת אוספים מימוש גנרי לכל סוג אוסף

```
/*
T should have the following:
const_iterator, methods 'begin', 'end',
and the type inside should have operator<<
 */

    printCollection          T
void printVector(const vector<int>& collection)
{
            T:: const_iterator itr    = collection.begin();
            T:: const_iterator itrEnd = collection.end();

    if (itr == itrEnd)
    {
        cout << "Collection is empty!\n";
        return;
    }

    for ( ; itr != itrEnd ; ++itr)
        cout << *itr << " ";
    cout << endl;
}
```

הפונקציה מקבלת אוסף כלשהו

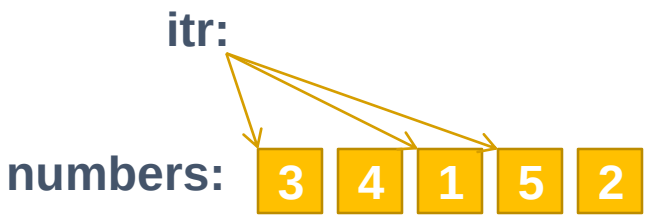
הפונקציה מהשקף הקודם לא עוברת קומפילציה ב-Visual Studio 2019

```
/*
T should have the following:
const_iterator, methods 'begin' and 'end, operator<<
 */
template <class T>
void printCollection(const T& collection)
{
    typename T::const_iterator itr    = collection.begin();
    typename T::const_iterator itrEnd = collection.end();

    if (itr == itrEnd)
    {
        cout << "Collection is empty!\n";
        return;
    }
    for ( ; itr != itrEnd ; ++itr)
        cout << *itr << " ";
    cout << endl;
}
```

C2760	syntax error: unexpected token 'identifier', expected ';'
C7510	'const_iterator': use of dependent type name must be prefixed with 'typename'

הפעולות erase ו- insert דוגמת שימוש

```
int main()
{
     list<int> numbers;

     numbers.push_back(1);
     numbers.push_back(2);
     numbers.push_front(3);

     list<int>::iterator itr = numbers.begin();
     ++itr;
     numbers.insert(itr, 4);

     numbers.insert(--numbers.end(), 5);  // 3 4 1 5 2

     ++itr;
     numbers.erase(itr); // 3 4 1 2

     numbers.erase(itr);   // crashes! The iterator points to no-where..
}
```



Sequences Container מתודות נוספות

- לכל מבני הנתונים יש את המתודות הבאות (ועוד):

שם המתודה	פעולה
`iterator insert(iterator pos, const T& val)`	הוספת הערך val לפני האיבר במיקום pos
`void insert(iterator pos, int n, const T& val)`	הוספת n ערכים val לפני האיבר במיקום pos
`iterator erase(iterator pos)`	מסירה את האיבר במיקום pos ומשחררת אותו
`iterator erase(iterator first, iterator last)`	מסירה את כל האיברים בטווח בין first ל- last (לא כולל last)

- לכל מבני הנתונים יש פעולות המחזירות איטרטורים:

שם המתודה	פעולה
`iterator begin()`	מחזיר איטרטור לאיבר הראשון באוסף
`iterator end()`	מחזיר איטרטור לאיבר אחרי האחרון באוסף

איטרטור דוגמת מימוש

iterator example.cpp

Object Function

מהו Object Function

- בשפת C יכולנו להגדיר פונקציה שאחד הפרמטרים שלה הוא מצביע לפונקציה
  - כלומר, אחד הפרמטרים הוא שם של פונקציה בעלת חתימה ספציפית

- C++ היא שפה מכוונת עצמים, ולכן הדגש הוא על אובייקטים:
  - במקום לשלוח שם של פונקציה, נשלח אובייקט שיתאר את הפעולה שאנחנו רוצים לבצע
  - במחלקה זו נעמיס את האופרטור()

דוגמה Object Function

```
template<class T>
class Print
{
public:
    void operator()(const T& val) const
    {
        cout << val << " ";
    }
};

class Inc
{
public:
    template<class T>
    void operator()(T& val) const
    {
        val++;
    }
};
```

הגדרת ה- template יכולה להיות או על כל המחלקה או רק על המתודה

למחלקות אלו קוראים Object Function משום שתפקידו של האובייקט הוא לייצג פעולה, ולכן הפונקציה היחידה שתמומש במחלקה זו היא העמסת האופרטור ()

Object Function דוגמת שימוש

```
template<class T, class S>
void applyAll(T begin, T end, const S& func)
{
    for ( ; begin != end ; ++begin)
        func(*begin);
}
```

הפונקציה מקבלת אובייקט המייצג פעולה

הפונקציה מקבלת 2 אובייקטים מטיפוס T שמועמסים להם האופרטורים הבאים: =!, ++ ו- *.

הפעלת אופרטור () של האובייקט func על כל איבר בטווח

הטיפוס T למעשה מייצג מחלקה שהיא איטרטור נעדיף לכתוב במקום T ← Iterator

הטיפוס S מייצג מחלקה שהיא Object Function ←נעדיף לקרוא לטיפוס במקום S ← Function

```
int main()
{
    vector<int> numbers;

    for (int i = 1; i < 10; i += 2)
        numbers.push_back(i);

    Print<int> p;
    applyAll(numbers.begin(), numbers.end(), p);
    cout << endl;
    applyAll(numbers.begin(), numbers.end(), Print<int>());
    cout << endl;
}
```

```
1 3 5 7 9
```

```
1 3 5 7 9
```

אובייקט זמני

תזכורת:

```
template<class T>
class Print
{
public:
    void operator()(const T& val) const
    {
        cout << val << " ";
    }
};
```

Object Function דוגמת שימוש (2)

```
template<class Iterator, class Function>
void applyAll(Iterator begin, Iterator end, const Function& f)
{
    for ( ; begin != end ; ++begin)
        f(*begin);
}
```

תזכורת:

```
template<class T>
class Print
{
    ...
};

class Inc
{
public:
    template<class T>
    void operator()(T& val) const
    {…}
};
```

תזכורת: הפעלה של פונקציה שהיא template אינה מחייבת ציון טיפוס כי הקומפיילר יודע לגזור את הטיפוס מסוג הפרמטר שנשלח

תזכורת: כאשר מייצרים משתנה מטיפוס מחלקה שהיא template יש לציין ביצירת האובייקט את סוג ה- T של המחלקה.

```
int main()
{
    vector<int> numbers;

    for (int i = 1; i < 10; i += 2)
        numbers.push_back(i);

    applyAll(numbers.begin(), numbers.end(), Print<int>());
    cout << endl;
    applyAll(numbers.begin(), numbers.end(), Inc());
    applyAll(numbers.begin(), numbers.end(), Print<int>());
    cout << endl;
}
```

```
1 3 5 7 9
2 4 6 8 10
```

נשים לה להבדל בשימוש בין Print המציינת את הטיפוס, לבין השימוש ב- Inc שאינה מציינת את הטיפוס

מיון דוגמת מימוש

```
template<class T>
void mySwap(T& num1, T& num2)
{
    T temp = num1;
    num1 = num2;
    num2 = temp;
}

template<class T, class Comparator>
void sort(T arr[], int size, const Comparator& compare)
{
    for (int i=size-1 ; i > 0 ; i--)
    {
        for (int j=0 ; j < i ; j++)
        {
            if (compare(arr[j], arr[j+1]) > 0)
                mySwap(arr[j], arr[j+1]);
        }
    }
}
```

← במחלקה Comparator ישנו מימוש לאופרטור () שמקבל שני פרמטרים שמחזיר int

מיון שימוש למיון מספרים

```
class CompareByOperator
{
public:
    template<class T>
    int operator()(const T& t1, const T& t2) const
    {
        if      (t1 < t2) return -1;
        else if (t1 > t2) return 1;
        else              return 0;
    }
};


int main()
{
    int arr[] = {3,2,7,1,9,4};
    sort(arr, sizeof(arr) / sizeof(arr[0]), CompareByOperator());
    sort(arr, sizeof(arr)/sizeof(arr[0]), compareNumbers);
}
```

```
template<class T, class Comparator>
void sort(T arr[], int size, const Comparator& compare)
{
    for (int i=size-1 ; i > 0 ; i--)
    {
        for (int j=0 ; j < i ; j++)
        {
            if (compare(arr[j], arr[j+1]) > 0)
                mySwap(arr[j], arr[j+1]);
        }
    }
}
```

```
int compareNumbers(int n1, int n2)
{
    if      (n1 < n2) return -1;
    else if (n1 > n2) return 1;
    else              return 0;
}
```

Name	Value	Type
arr	0x00cff7a8 {3, 2, 7, 1, 9, 4}	int[6]

arr	0x003afd04 {1, 2, 3, 4, 7, 9}

ניתן לשלוח במקום אובייקט עם העמסת () שם של פונקציה עם חתימה זהה

מיון שימוש למיון סטודנטים (1)

```
class Student
{
    char name[10];
    int id;
    float grade;

public:
    Student(const char* name, int id, float grade) : id(id), grade(grade)
    {
        strcpy(this->name, name);
    }

    int   getId()    const { return id; }
    float getGrade() const { return grade; }
};

class CompareStudentsByGrade
{
public:
    int operator()(const Student& v1, const Student& v2) const
    {
        if      (v1.getGrade() < v2.getGrade()) return -1;
        else if (v1.getGrade() > v2.getGrade()) return 1;
        else                                    return 0;
    }
};
```

מיון שימוש למיון סטודנטים (2)

```
class CompareStudentsById
{
public:
    int operator()(const Student& v1, const Student& v2) const
    {
        if      (v1.getId() < v2.getId()) return -1;
        else if (v1.getId() > v2.getId()) return 1;
        else                              return 0;
    }
};

int main()
{
    Student arr[3] = {
                        { "gogo", 333, 87 },
                        { "momo", 111, 91 },
                        { "yoyo", 222, 78 }
                      };
    sort(arr, 3, CompareStudentsById());
    sort(arr, 3, CompareStudentsByGrade());
}
```

Name	Value
arr	0x00dafce4 {{name=0x00dafce4 "gogo" id=333 grade=87
[0]	{name=0x00dafce4 "gogo" id=333 grade=87.0000000 }
[1]	{name=0x00dafcf8 "momo" id=111 grade=91.0000000 }
[2]	{name=0x00dafd0c "yoyo" id=222 grade=78.0000000 }

Name	Value
arr	0x00dafce4 {{name=0x00dafce4 "momo" id=111 grade=9
[0]	{name=0x00dafce4 "momo" id=111 grade=91.0000000 }
[1]	{name=0x00dafcf8 "yoyo" id=222 grade=78.0000000 }
[2]	{name=0x00dafd0c "gogo" id=333 grade=87.0000000 }

Name	Value
arr	0x00dafce4 {{name=0x00dafce4 "yoyo" id=222 grade=78
[0]	{name=0x00dafce4 "yoyo" id=222 grade=78.0000000 }
[1]	{name=0x00dafcf8 "gogo" id=333 grade=87.0000000 }
[2]	{name=0x00dafd0c "momo" id=111 grade=91.0000000 }

Associative Container

Associative Containers

- אלו מבני נתונים אשר ניתן לגשת אליהם עפ"י מפתח, ולא בהכרח עפ"י אינדקס (מספר)

- מבני נתונים אלו ממוינים עפ"י המפתח, לכן צריך לספק עבורם מתודת השוואה
  - במידה ולא נספק, מבנה הנתונים ישתמש ב- default, אם קיים, אחרת ייתן שגיאת קומפילציה

set

- set הוא מבנה נתונים לשמירת מפתחות בלבד
  - כל מפתח הוא ייחודי
  - ניסיון הוספה של מפתח קיים לא יבוצע
  - איברי ה- set ממוינים ומוחזקים בעץ בינארי

multiset

- כמו set אך כל מפתח יכול להיות קיים יותר מפעם אחת
- כאשר נוריד איבר מהקבוצה, במידה וקיימים כמה העתקים, ירד רק מופע אחד שלו

דוגמה set

```
int main()
{
    set<int> intSet;

    intSet.insert(4);
    intSet.insert(2);
    intSet.insert(1);
    intSet.insert(3);
    intSet.insert(1);
    printCollection(intSet);

    cout << "Is set empty? " << intSet.empty() << endl;
    cout << "Value 1 appears " << intSet.count(1) << " times\n";
    cout << "Value 8 appears " << intSet.count(8) << " times\n";
    intSet.erase(2);
    printCollection(intSet);
    cout << "There are " << intSet.size() << " values in the set\n";
    cout << "There can be max " << intSet.max_size() << " elements\n";
    set<int>::iterator itr = intSet.find(4);
    if (itr != intSet.end())
        cout << "4 is found\n";

    intSet.clear();
    printCollection(intSet);
}
```

```
1 2 3 4
Is set empty? 0
Value 1 appears 1 times
Value 8 appears 0 times
1 3 4
There are 3 values in the set
There can be max 214748364 elements
4 is found
Collection is empty!
```

set דוגמה (2)

ההגדרה של set היא אוסף ממוין,
אך בפועל אנחנו רואים שלא...

```
int main()
{
    const char* words[] = { "shalom", "kita", "alef", "shalom", "kita", "beit" };
    set<const char*> wordsSet;

    int numOfWords = sizeof(words) / sizeof(words[0]);

    for (int i = 0; i < numOfWords; i++)
        cout << words[i] << " \t" << (void*)words[i]<< endl;
    cout << endl;

    for (int i = 0; i < numOfWords; i++)
        wordsSet.insert(words[i]);

    printCollection(wordsSet);

    wordsSet.clear();
    printCollection(wordsSet);
}
```

```
shalom   00C61E80
kita     00C61E88
Alef     00C61E90
shalom   00C61E80
kita     00C61E88
beit     00C61E98

shalom kita alef beit
Collection is empty!
```

מחרוזות זהות נמצאות
באותה הכתובת

מיון set

```
shalom  00C61E80
kita    00C61E88
Alef    00C61E90
shalom  00C61E80
kita    00C61E88
beit    00C61E98

shalom kita alef beit
Collection is empty!
```

- בדוגמה הקודמת ראינו שאיברי ה- set אינם ממוינים כצפוי
  - set ממיין את איבריו בעזרת האופרטור <
  - מאחר ואיברי ה- set בדוגמה הם מטיפוס מצביע, המיון נעשה לפי כתובתם, ולא לפי תוכנם
- במידה ולא ממומש אופרטור < עבור T, נקבל שגיאת קומפילציה
- ניתן להגדיר set ולספק כארגומנט אובייקט שייצג את פונקציית ההשוואה
  - כאמור, אובייקט כזה נקרא object function וממש עבורו את האופרטור ()

מיון set דוגמה

```
class ltstr
{
    public:
    bool operator()(const char* s1, const char* s2) const
    {
        return strcmp(s1, s2) < 0;
    }
};

int main()
{
    const char* words[] = { "shalom", "kita", "alef", "shalom", "kita", "beit" };
    set<const char*, ltstr> wordsSet;

    int numOfWords = sizeof(words) / sizeof(words[0]);

    for (int i = 0; i < numOfWords; i++)
        cout << words[i] << " \t" << (void*)words[i]  << endl;
    cout << endl;

    for (int i = 0; i < numOfWords; i++)
        wordsSet.insert(words[i]);

    printCollection(wordsSet);

    wordsSet.clear();
    printCollection(wordsSet);
}
```

טיפוס המעמיס את האופרטור () ויודע להשוות בין שני משתנים מטיפוס מחרוזת לקסיקוגרפית

יצירת האוסף תכלול כפרמטר את הטיפוס שיודע לבצע את ההשוואה

```
shalom   006D1E80
kita     006D1E88
alef     006D1E90
shalom   006D1E80
kita     006D1E88
beit     006D1E98

alef beit kita shalom
Collection is empty!
```

ממוינות לקסיקוגרפית

map

- מבנה נתונים המכיל זוגות של key ו- value
  - לכל מפתח יש ערך יחיד
  - בעת הכנסה של מפתח קיים:
    - הערך הנוכחי יידרס
    - יש לבדוק האם צריך לשחרר את הזיכרון של הערך הנוכחי

multimap

- כמו map אך כל מפתח יכול להיות קיים יותר מפעם אחת
- כאשר נוריד מפתח, ירד ה- value הראשון

דוגמת שימוש map

```
class ltstr
{
public:
      bool operator()(const char* s1, const char* s2) const
                                        {return strcmp(s1, s2) < 0;}
};

int main()
{
    map<char*, char*, ltstr> phones;

    phones["gogo"] = "050-5566778";
    phones["yoyo"] = "052-8529632";
    phones["momo"] = "054-8866553";
    phones["gogo"] = "050-5555555"; // instead the previous value
    phones["koko"] = "050-7534218";

    printCollection(phones);
}
```

```
gogo    050-5555555
koko    050-7534218
momo    054-8866553
yoyo    052-8529632
```

הדפסתו map

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

template <class K, class V, class C>
void printCollection(const map<K, V, C>& collection)
{
    map<K, V, C>::const_iterator itr    = collection.begin();
    map<K, V, C>::const_iterator itrEnd = collection.end();

    if (itr == itrEnd)
    {
        cout << "Collection is empty!\n";
        return;
    }


    for ( ; itr != itrEnd ; ++itr)
        cout << itr->first << " \t" << itr->second << endl;
    cout << endl;
}
```

```
for (const auto& pair : collection)
    cout << pair.first << " \t" << pair.second << endl;
```

map דוגמה מתוך STL Reference

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

int main()
{
    map<const char*, int> months;

    months["january"] = 31;     months["february"] = 28;
    months["march"] = 31;       months["april"] = 30;
    months["may"] = 31;         months["june"] = 30;
    months["july"] = 31;        months["august"] = 31;
    months["september"] = 30;   months["october"] = 31;
    months["november"] = 30;    months["december"] = 31;

    cout << "june -> " << months["june"] << endl;
    map<const char*, int>::iterator cur  = months.find("june");
    map<const char*, int>::iterator prev = cur;
    map<const char*, int>::iterator next = cur;
    ++next;
    --prev;
    cout << "Previous is " << (*prev).first << endl;
    cout << "Next is " << (*next).first << endl;
}
```

האיברים מסודרים לפי כתובתם

```
june -> 30
Previous is may
Next is july
```

דוגמת שימוש multimap

```
int main()
{
    multimap<char*, char*, ltstr> phones;

    phones.insert(pair<char*, char*>("gogo", "050-5566778"));
    phones.insert(pair<char*, char*>("yoyo", "052-8529632"));
    phones.insert(pair<char*, char*>("momo", "054-8866553"));
    phones.insert(pair<char*, char*>("gogo", "050-5555555"));
    phones.insert(pair<char*, char*>("gogo", "054-8888888"));
    phones.insert(pair<char*, char*>("koko", "050-7534218"));

    printMultiMapCollection(phones);
    cout << "\ngogo has " << phones.count("gogo") << " numbers\n";
    cout << "total recors in phones: " << phones.size() << "\n\n";


    multimap<char*, char*, ltstr>::iterator found = phones.find("gogo");
    phones.erase(found); // deletes only the first value...
    printMultiMapCollection(phones);

    cout << "gogo has " << phones.count("gogo") << " numbers\n";
    cout << "total recors in phones: " << phones.size() << endl;
}
```

```
gogo    050-5566778
gogo    050-5555555
gogo    054-8888888
koko    050-7534218
momo    054-8866553
yoyo    052-8529632

gogo has 3 numbers
total recors in phones: 6

gogo    050-5555555
gogo    054-8888888
koko    050-7534218
momo    054-8866553
yoyo    052-8529632

gogo has 2 numbers
total recors in phones: 5
```

הדפסתו multimap

```
template <class K, class V, class C>
void printMultiMapCollection(const multimap<K, V, C>& collection)
{
    multimap<K, V, C>::const_iterator itr    = collection.begin();
    multimap<K, V, C>::const_iterator itrEnd = collection.end();

    if (itr == itrEnd)
    {
        cout << "Collection is empty!\n";
        return;
    }

    for ( ; itr != itrEnd ; ++itr)
        cout << itr->first << " \t" << itr->second << endl;
    cout << endl;
}
```

אפשר כמובן לשדרג את הלולאה כך שאם ה- key זהים בשני ערכים עוקבים, ה- values יודפסו באותה שורה

map הפתעות יעילות

```
class A
{
private:
    static int counter;
    int id;
public:
    A() : id(++counter)                    { cout << "A::A id=" << id << "\n"; }
    A(const A& other) : id(other.id*10) { cout << "A::copy id=" << id << "\n"; }
    ~A()                                   { cout << "A::~A id=" << id << "\n"; }

    operator int() const { return id; }
};
int A::counter = 0;

int main()
{
    map<A, int> theMap;

    for (int i=0 ; i < 2 ; i++)
        theMap.insert(pair<A, int>(A(), i));
    cout << "----------------\n";


    for (const std::pair<A, int>& item : theMap)
        cout << typeid(item).name()
                << " [" << item.first << " " << item.second << "]\n";}
    cout << "----------------\n";
}
```

נולד האובייקט הזמני

העתקת האובייקט לתוך ה- pair

העתקת ה- pair לתוך ה- map

הריגת ה- pair

הריגת האובייקט הזמני

```
A::A id=1
A::copy id=10
A::copy id=100
A::~A id=10
A::~A id=1
A::A id=2
A::copy id=20
A::copy id=200
A::~A id=20
A::~A id=2
--------------
```

מוזר מאוד מדוע נוצר פה העתק של ה- pair, הרי לקחנו אותו by ref!

```
A::copy id=1000
struct std::pair<class A,int> [1000 0]
A::~A id=1000
A::copy id=2000
struct std::pair<class A,int> [2000 1]
A::~A id=2000
```

```
----------------
A::~A id=200
A::~A id=100
```

מדוע עברנו ב- copy c'tor??

```
int main()
{
    map<A, int> theMap;

    for (int i=0 ; i < 2 ; i++)
        theMap.insert(pair<A, int>(A(), i));
    cout << "----------------\n";

    cout << "*** " << typeid(*theMap.begin()).name() << endl;
    for (const std::pair<A, int>& item : theMap)
    {
        cout << typeid(item).name()
        << " [" << item.first << " " << item.second << "]\n";
    }
    cout << "----------------\n";
}
```

```
*** struct std::pair<class A const ,int>
A::copy id=1000
struct std::pair<class A,int> [1000 0]
A::~A id=1000
A::copy id=2000
struct std::pair<class A,int> [2000 1]
A::~A id=2000
```

נשים לב להבדל בטיפוס בין ההדפסות!

בגלל ההבדל בטיפוס, למעשה נוצר קסטינג, מה שגרר את יצירת האובייקט הזמני!

שימו לב: קבלת pair מ- map מביאה את המפתח כ- const

ולאחר תיקון הטיפוס ב- pair

```
*** struct std::pair<class A const ,int>
struct std::pair<class A const ,int> [100 0]
struct std::pair<class A const ,int> [200 1]
```

```
int main()
{
    map<A, int> theMap;

    for (int i=0 ; i < 2 ; i++)
        theMap.insert(pair<A, int>(A(), i));
    cout << "----------------\n";

    cout << "*** " << typeid(*theMap.begin()).name() << endl;
    for (const std::pair<const A, int>& item : theMap)
    {
        cout << typeid(item).name()
        << " [" << item.first << " " << item.second << "]\n";
    }
    cout << "----------------\n";
}
```

עכשיו אין הבדל בטיפוס ולכן אין קסטינג,
ולכן הקוד משמעותית יותר יעיל!

auto היה פותר את הבעיה!

- בדוגמה הקודמת הטעות של המתכנת הייתה בתמימות, וגם מתכנתים מנוסים נופלים בבור הזה!
- שימוש ב- auto היה פותר את הבעיה!

```
int main()
{
    map<A, int> theMap;

    for (int i=0 ; i < 2 ; i++)
        theMap.insert(pair<A, int>(A(), i));
    cout << "----------------\n";

    cout << "*** " << typeid(*theMap.begin()).name() << endl;
    for (auto& item : theMap)
    {
        cout << typeid(item).name()
        << " [" << item.first << " " << item.second << "]\n";
    }
    cout << "----------------\n";
}
```

```
*** struct std::pair<class A const ,int>
struct std::pair<class A const ,int> [100 0]
struct std::pair<class A const ,int> [200 1]
```

בטיפוסים מורכבים של ה- STL, כדאי תמיד לעבוד עם auto, שכן הוא יודע באופן מדוייק מה הטיפוס שעליו להיות

מאוד חשוב לא לשכוח את ה- &, אחרת יתבצע העתק לטובת ההשמה

אלגוריתמים

אלגוריתמים שנסקור:

- min_element, max_element
- sort
- reverse
- swap, itr_swap
- find, find_if
- count, count_if
- for_each
- transform
- copy

יש להוסיף:
#include <algorithm>

אלו פונקציות גלובליות, שהסינטקס של רובן הוא ששני הפרמטרים הראשונים שלהן הם טווח איברים עליהם תבוצע הפעולה (לרוב, איטרטור begin ואיטרטור end)

min_element, max_element

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

#include <algorithm>
#include <vector>

int main()
{
    vector<int> numbers;

    numbers.push_back(4);
    numbers.push_back(1);
    numbers.push_back(7);
    numbers.push_back(4);
    numbers.push_back(2);
    numbers.push_back(5);

    vector<int>::iterator max = max_element(numbers.begin(), numbers.end());
    cout << "The max is " << *max << endl;

    vector<int>::iterator min = min_element(numbers.begin(), numbers.end());
    cout << "The min is " << *min << endl;
}
```

הפונקציות מקבלות איטרטור begin ו- end ומחזירות איטרטור לאיבר המתאים

דורשות של- T יועמס האופרטור <

```
The max is 7
The min is 1
```

sort

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

#include <algorithm>
#include <vector>

int main()
{
    vector<int> numbers;

    numbers.push_back(4);
    numbers.push_back(1);
    numbers.push_back(7);
    numbers.push_back(4);
    numbers.push_back(2);
    numbers.push_back(5);
                                    // 4 1 7 4 2 5
    sort(numbers.begin(), numbers.end());
 }                                  // 1 2 4 4 5 7
```

הפונקציה מקבלת איטרטור begin ו- end

דורשת של- T יועמס האופרטור <

reverse

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

#include <algorithm>
#include <vector>

int main()
{
    vector<int> numbers;

    numbers.push_back(4);
    numbers.push_back(1);
    numbers.push_back(7);
    numbers.push_back(4);
    numbers.push_back(2);
    numbers.push_back(5);
                                        // 4 1 7 4 2 5
    reverse(numbers.begin(), numbers.end());
 }                                      // 5 2 4 7 1 4
```

הפונקציה מקבלת איטרטור begin ו- end

הפונקציה מקבלת שני משתנים מאותו סוג ומשתמשת באופרטור= שלהם

swap

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

#include <algorithm>

int main()
{
     int x=2, y=3;
                    // x=2, y=3
     swap(x, y);
                    // x=3, y=2


     char str1[10]="hello", str2[10]="world";
     swap(str1, str2);   // doesn't compile, can't change addresses
 }
```

הפונקציה מקבלת שני משתנים מאותו סוג ומשתמשת באופרטור= שלהם

itr_swap

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

#include <algorithm>
#include <vector>

int main()
{
    vector<int> numbers;

    numbers.push_back(4);
    numbers.push_back(1);
    numbers.push_back(7);
    numbers.push_back(4);
    numbers.push_back(2);
    numbers.push_back(5);
                                        // 4 1 7 4 2 5
    itr_swap(numbers.begin(),  ++(numbers.begin());
 }                                      // 1 4 7 4 2 5
```

הפונקציה מקבלת 2 איטרטורים ומחליפה את תוכנם

הפונקציה משתמשת באופרטור= של T

find

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

#include <algorithm>
#include <vector>

int main()
{
      vector<int> numbers;

      numbers.push_back(4);
      numbers.push_back(1);
      numbers.push_back(7);
      numbers.push_back(4);
      numbers.push_back(2);
      numbers.push_back(5);

      vector<int>::iterator found = find(numbers.begin(), numbers.end(), 8);
      if (found == numbers.end())
           cout << "value doesn't exist\n";
      else
           cout << "value exists\n";
 }
```

הפונקציה מקבלת: איטרטור begin , איטרטור end ואיבר לחיפוש
הפונקציה מחזירה: איטרטור לאיבר הראשון באוסף שזהה לאיבר

דורשת של- T יועמס האופרטור ==

find_if

```
bool isOdd(int num) {return num%2==1;}

int main()
{
    vector<int> numbers;

    numbers.push_back(4);
    numbers.push_back(1);
    numbers.push_back(7);
    numbers.push_back(4);
    numbers.push_back(2);
    numbers.push_back(5);

    vector<int>::iterator found = find_if(numbers.begin(), numbers.end(), isOdd);

    if (found == numbers.end())
        cout << "No value is odd\n";
    else
        cout << "value exists: " << *found << "\n"; // value exists: 1
}
```

במקום פונקציה, אפשר לשלוח גם Object Function

הפונקציה מקבלת: איטרטור begin , איטרטור end ושם של פונקציה המקבלת T ומחזירה bool
הפונקציה מחזירה: איטרטור לאיבר הראשון באוסף שהפעלת הפונקציה עליו מחזירה true

דורשת של- T יועמס האופרטור ==

count

```
int main()
{
    vector<int> numbers;

    numbers.push_back(4);
    numbers.push_back(1);
    numbers.push_back(7);
    numbers.push_back(4);
    numbers.push_back(2);
    numbers.push_back(5);

    int num4 = count(numbers.begin(), numbers.end(), 4);
    cout << "4 appears " << num4 << "times\n"; // 4 appears 2 times
}
```

הפונקציה מקבלת: איטרטור begin , איטרטור end ואיבר לחיפוש
הפונקציה מחזירה: את מספר המופעים שלו באוסף

דורשת של- T יועמס האופרטור ==

count_if

```
bool isOdd(int num) {return num%2==1;}

int main()
{
     vector<int> numbers;

     numbers.push_back(4);
     numbers.push_back(1);
     numbers.push_back(7);
     numbers.push_back(4);
     numbers.push_back(2);
     numbers.push_back(5);

     int numOfOdd = count_if(numbers.begin(), numbers.end(), isOdd);
     cout << "There are " << numOfOdd << " odd values\n";
                                        // There are 3 odd values
}
```

הפונקציה מקבלת: איטרטור begin , איטרטור end ושם של פונקציה המקבלת T ומחזירה bool
הפונקציה מחזירה: את מספר האיברים באוסף שהפעלת הפונקציה עליהם מחזירה true

דורשת של- T יועמס האופרטור ==

for_each

```
void print(int num) {cout << num << " ";}

int main()
{
        vector<int> numbers;

        numbers.push_back(4);
        numbers.push_back(2);
        numbers.push_back(7);
        numbers.push_back(4);
        numbers.push_back(2);
        numbers.push_back(5);

        for_each(numbers.begin(), numbers.end(), print);     // 4 2 7 4 2 5
}
```

הפונקציה מקבלת: איטרטור begin , איטרטור end ושם של פונקציה המקבלת T ושאינה מחזירה דבר.
for_each מפעילה את הפונקציה שהתקבלה על כל אחד מאיברי הטווח.

transform

```
int square(int num) {return num*num;}

int main()
{
    std::list<int>   intList;
    std::vector<int> intArr;

    for (int i=1; i<=5; ++i)
         intList.push_back(i);  // intList: 1 2 3 4 5
                                // int arr:

    std::transform (intList.begin(),  intList.end(),
                    std::back_inserter(intArr),
                    square); // intList: 1 2 3 4 5
                             // int arr: 1 4 9 16 25
}
```

שולחת כל איבר בטווח לפונקציה המבוקשת,
ואת התוצאה מוסיפה לאוסף השני

טווח איברים עליו נעבוד

האוסף אליו נוסיף את
התוצאה ומיקום ההוספה

הפונקציה לחישוב

copy

```
int main()
{
     vector<int> numbers;
     list<int> frontIntList, backIntList;

     numbers.push_back(4);
     numbers.push_back(2);
     numbers.push_back(7);
     numbers.push_back(4);
     numbers.push_back(2);
     numbers.push_back(5);

     copy(numbers.begin(), numbers.end(), back_inserter(backIntList));    // backIntList: 4 2 7 4 2 5


     copy(numbers.begin(), numbers.end(), front_inserter(frontIntList)); // frontIntList: 5 2 4 7 2 4
}
```

הפונקציה מקבלת טווח איטרטורים וסדר הוספה לאוסף אחר (front/back) ומוסיפה את כל האיברים בטווח לאוסף האחר

שימוש ב- copy לצורך הדפסה

```
int main()
{
     vector<int> numbers;

     numbers.push_back(4);
     numbers.push_back(2);
     numbers.push_back(7);
     numbers.push_back(4);
     numbers.push_back(2);
     numbers.push_back(5);

     copy(numbers.begin(), numbers.end(), ostream_iterator<int>(cout, "_"));
     cout << endl;

     copy(numbers.begin(), numbers.end(), ostream_iterator<int>(cout, " # "));
     cout << endl;
}
```

ostream_inserter כותב איברים מטיפוס T למסך, עם מפריד '_' בין האיברים

```
4_1_7_4_2_5_
```

```
4 # 1 # 7 # 4 # 2 # 5 #
```

המחלקה string

```
class Person
{
    char* name;

public:
    Person(const char* str) : name(nullptr) { setName(str); }
    Person(const Person& other) : name(nullptr) { *this = other; }
    ~Person() { delete[]name; }

    const Person& operator=(const Person& other) {...}

    void setName(const char* newName)
    {
        delete[]name;
        name = new char[strlen(newName) + 1];
        strcpy(name, newName);
    }

    operator const char* () { return name; }
};
```

המחלקה string

```
class Person
{
    std::string name;

public:
    Person( const std::string& str  nullptr) { setName(str); }
    Person(const Person& other) : name(nullptr) { *this = other; }
    ~Person() { delete[]name; }

    const Person& operator=(const Person& other) {...}

    void setName( const std::string& newName
    {

        name = newName;

    }

    operator const char* () { return name.c_str();
};
```

Lambda Expressions
תכנות פונקציונלי / פונקציות אנונימיות

תכנות פונקציונלי

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

int main()
{
    [](){cout << "First Print\n"; }();

    auto f1 = []() {cout << "Hello World!\n"; };
    f1();

    auto f2 = [](double num1, double num2) {return num1 + num2; };
    cout << f2(3.2, 4) << endl;

    auto f3 = [](double num1, double num2) -> int{return num1 + num2; };
    cout << f3(3.2, 4) << endl;
}
```

– capture list
הסבר בהמשך

רשימת הפרמטרים
שהפונקציה מקבלת

גוף הפונקציה

הפעלת
הפונקציה

הפעלת פונקציה אנונימית, לפונקציה אין שם

הגדרת פונקציה ושמירתה במשתנה, ללא הפעלתה

הפעלת הפונקציה

הגדרת פונקציה שמקבלת פרמטרים
ומחזירה ערך, ושמירתה במשתנה

הפעלת הפונקציה

אפשר גם להצהיר על
טיפוס הערך המוחזר

```
First Print
Hello World!
7.2
7
```

תכנות פונקציונלי capture list

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


int main()
{
    int x = 10, y = 20;

    [x, y]() {
        cout << x << " " <<  y << endl;
    }();
}
```

קישור המשתנים שב- main לפונקציה האנונימית,
כדי שהיא תוכל להכיר אותם ולהשתמש בהם

```
10 20
```

תכנות פונקציונלי capture list

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


int main()
{
    int x = 10, y = 20;

    [&x, &y]() {
        x++;
        y++;
        cout << x << " " <<  y << endl;
    }();

    cout << x << " " << y << endl;
}
```

```
11 21
11 21
```

הפונקציה האנונימית יכולה גם לשנות את הערכים שבפונקציה שהגדירה אותה

תכנות פונקציונלי שימוש ב- STL

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

#include <vector>
#include <algorithm>

int main()
{
    vector<int> numbers = { 1,4,2,3,6 };
    for_each(numbers.begin(), numbers.end(), [](int val) {cout << val << " "; });
    cout << endl;

    auto print = [](int val) {cout << val << " "; };
    for_each(numbers.begin(), numbers.end(), print);
    cout << endl;
    for_each(numbers.begin(), numbers.end(), [](int& val) {val = val*val; });
    for_each(numbers.begin(), numbers.end(), print);
    cout << endl;

    sort(numbers.begin(), numbers.end(), [](int a, int b) {return a > b; });
    for_each(numbers.begin(), numbers.end(), print);
    cout << endl;

    sort(numbers.begin(), numbers.end(), [](int a, int b) {return a < b; });
    for_each(numbers.begin(), numbers.end(), print);
    cout << endl;
}
```

במקום Object Function ניתן לשלוח לאלגוריתם ב- STL פונקציה אנונימית (lambda expressions)

sort למשל יודעת לקבל כפרמטר Object Function / Lambda Expression על איך למיין את האיברים

```
1 4 2 3 6
1 4 2 3 6
1 16 4 9 36
36 16 9 4 1
1 4 9 16 36
```

אז מתי נשתמש ב- Lambda Expression?

- נשתמש כאשר נרצה שימוש קריא ומהיר לפונקציה בלי ללכת לחפש במקום אחר מה היא עושה
- נוח כאשר משתמשים בה פעם אחת בלבד

ביחידה זו למדנו:

- סקירת ה- STL
- סוגים של conatiner'ים
- איטרטור
- Object Functions
- אלגוריתמים
- המחלקה string
- תכנות פונקציונלי / Lambda Expression