תכנות מתקדם ושפת C++
מצגת 2

מיכלים

נושאים

- הוספות של C++11
- string
- מיכלים
- איטרטורים

ייצוג נתונים

C Data Type	Typical 32-bit	Typical 64-bit
`char`	1	1
`short`	2	2
`int`	4	4
`long`	4	8
`float`	4	4
`double`	8	8
pointer	4	8

הוספות של C++11 - auto

auto אומר לקומפיילר להסיק את סוג המשתנה לפי המאתחל:

```
auto b = true;          // bool
auto ch = 'x';          // char
auto i = 123;           // int
auto d = 1.2;           // double
auto p = &i;            // pointer to int
auto& ri = x;           // reference on x (char)
const auto& cri = x;    // constant reference on x
auto z = sqrt(y);       // whatever sqrt(y) returns
auto item = val1 + val2; // result of val1 + val2
```

הוספות של C++11 - decltype()

decltype() אומר לקומפיילר להסיק את סוג המשתנה מתוך ביטוי שאינו משמש לאתחול:

```
decltype(f()) sum = x;   // whatever f() returns
const int ci = 0;
decltype(ci) x = 0;      // const int
struct A { int i; double d; };
A* ap = new A();
decltype(ap->d) x;       // double
vector<int> ivec;
for (decltype(ivec.size()) ix = 0; ix != 10; ++ix)
    ivec.push_back(ix); // vector<int>::size_type
```

הוספות של C++11 - List Initialization

אתחול vector:

```
vector<int> v{1,2,3,4,5,6,7,8,9};
vector<string> articles = {"a", "an", "the"};
vector<int> v1(10); // v1 has ten elements with value 0
vector<int> v2{10}; // v2 has one element with value 10
vector<int> v3(10, 1); // v3 has ten elements with value 1
vector<int> v4{10, 1}; // v4 has two elements, 10 and 1
```

ארבע אפשרויות לאתחול משתנה:

```
int units_sold = 0;
int units_sold(0);
int units_sold{0};
int units_sold = {0};

int i1 = 7.2; // i1 becomes 7
int i2{7.2};  // error : narrowing conversion
```

range for - הוספות של C++11

תחביר:

```
for (declaration : expression)
     statement
```

expression represents a sequence

declaration variable to access the sequence

דוגמה:

```
vector<int> v{1,2,3,4,5,6,7,8,9};
for (auto i : v)            // i is a copy
     cout << i << " ";
cout << endl;
for (auto &i : v)         // i is a reference
    i *= i;               // square the element
```

C++11.cpp

הוספות של C++11 - class enum

```
enum class Color { red, blue , green };
enum class Traffic_light { green, yellow, red };
Color col = Color::red;
Traffic_light light = Traffic_light::red;
Color x = red;                    // error, which red?
Color y = Traffic_light::red; // error, not a Color
Color z = Color::red;             // OK
int i = Color::red;               // error, not an int
Color c = 2;                      // error, 2 is not a Color
// enum with no class are not scoped within their enum
enum color { red, green, blue };      // no class
enum stoplight {red, yellow, green}; // error, redefines
int col = green; // convert to their integer value
```

enum.cpp

string

string הוא מחרוזת של תווים בגודל משתנה.

הגדרה ואתחול של string:

```
string s1; // default initialization, empty string
string s2 = s1;      // copy of s1
string s2(s1);       // copy of s1
string s3 = "abc";   // copy of the string literal
string s3("abc");    // copy of the string literal
string s4(10, 'c'); // cccccccccc
```

פעולות על string:

```
s.size()             // number of characters in s
s[n]                 // reference to char at position n
s1 + s2              // concatenation of s1 and s2
```

size_type

```
string::size_type len = line.size();
// size() returns a string::size_type value
```

The string class defines **size_type** so we can use the library in **machine independent** manner

We use the scope operator to say that the name **size_type** is defined in the **string** class

It is an **unsigned** type big enough to hold the size of any string

string

מעבר על תווי מחרוזת עם **אינדקס**:

```
string s("some string");
// The subscript operator (the [ ] operator)
// takes a string::size_type
for (string::size_type index = 0;
     index != s.size() && !isspace(s[index]);
     ++index)
          s[index] = toupper(s[index]);

The output of this program is:
SOME string
```

string

מעבר על תווי מחרוזת עם **range for**:

```
string s("Hello World!!!");
for (auto &c : s)       // note: c is a reference
     c = toupper(c);    // so the assignment changes the char
cout << s << endl;
```

```
The output of this program is:
HELLO WORLD!!!
```

string

מעבר על תווי מחרוזת עם **range for**:

```
string s("Hello World!!!");
decltype(s.size()) cnt = 0;
for (auto c : s) // for every char in s
     if (ispunct(c))
          ++cnt;
     cout << cnt << " characters" << endl;
```

```
The output of this program is:
3 characters
```

string

קריאת מספר לא ידוע של מחרוזות:

```
string word;
// end-of-file or invalid input will put cin in error state
// error state is converted to boolean false
while (cin >> word)
      cout << word << endl;
```

enum.cpp

קריאת שורה שלמה:

```
string line;
// read up to and including the first newline
// store what it read not including the newline
while (getline(cin, line))
      cout << line << endl;
```

vector

vector הוא אוסף של אוביקטים מאותו סוג בגודל משתנה.

הגדרה ואתחול של vector:

```
A vector is a class template
We have to specify which objects the vector will hold
vector<int> ivec;                // initially empty
vector<Sales_item> Sales_vec;
vector<vector<string>> file;  // vector of vectors
```

פעולות על vector:

```
v.size()          // number of elements in v
v[n]              // reference to element at position n
v.push_back(t)    // add element to end of v
```

vector

הוספת אלמנטים למיכל vector:

```
vector<int> ivec;          // empty vector
for (decltype(ivec.size()) ix = 0; ix != 10; ++ix)
     ivec.push_back(ix); // adds element with value ix
```

טעויות נפוצות בשימוש ב- vector:

```
vector<int> ivec;
cout << ivec[0];

vector<int> ivec2(10);
cout << ivec2[10];

vector::size_type idx;
```

vector

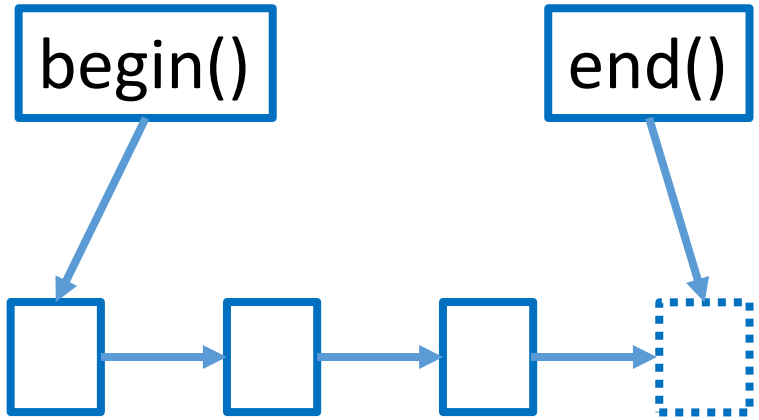
הוספת אלמנטים למיכל vector:

```
vector<int> ivec;          // empty vector
for (decltype(ivec.size()) ix = 0; ix != 10; ++ix)
     ivec.push_back(ix); // adds element with value ix
```

טעויות נפוצות בשימוש ב- vector:

```
vector<int> ivec;
cout << ivec[0];        // error: no elements
vector<int> ivec2(10);
cout << ivec2[10];      // error: elements 0 to 9
vector::size_type       // error
vector<int>::size_type // ok
```

איטרטורים



איטרטור הוא אובייקט שמצביע על איבר של הסדרה

האיטרטור begin מצביע על האיבר הראשון

האיטרטור end מצביע על המקום שאחרי האיבר האחרון

איטרטור צריך לספק את הפעולות הבאות:

השוואה בין שני איטרטורים, האם מצביעים לאותו איבר:

`iter1 == iter2 , iter1 != iter2`

התיחסות לערך של האיבר שהאיטרטור מצביע עליו:

`val = *iter , *iter = val`

`(*iter).member` במקום `iter->member`

קידום האיטרטור כך שיצביע לאיבר הבא:

`++iter`

- ישנם איטרטורים שמספקים פעולות נוספות

סוגי איטרטורים

לכל המיכלים יש איטרטורים, הם מוגדרים בספריה של כל מיכל:

```
vector<int>::iterator it; // it can read and write
string::iterator it2;     // it2 can read and write
vector<int>::const_iterator it3; // can read but not write
string::const_iterator it4;      // can read but not write
// The type returned by begin and end depends on whether the
// object is const
auto it1 = v.begin();
// To ask for the const_iterator type:
auto it3 = v.cbegin(); // it3 has type const_iterator
```

מעבר על string באמצעות איטרטור

```
string s("some string");
for (auto it = s.begin();
     it != s.end() && !isspace(*it); ++it)
          *it = toupper(*it);
```

```
The output of this program is:
SOME string
```

גישה לחלקי אובייקט באמצעות איטרטור

נניח שיש לנו וקטור של מחרוזות ואנו רוצים לבדוק אם מחרוזת היא ריקה:

```
(*iter).empty()   // dereferences iter and calls empty()
*iter.empty()     // error, no member named empty in iter
// To simplify, use the arrow operator

vector<string> file;
for (auto it = text.cbegin();
     it != text.cend() && !it->empty(); ++it)
          cout << *it << endl;
```

גישה לחלקי אובייקט באמצעות איטרטור

גישה לחלקי אובייקט והתקדמות בביטוי אחד:

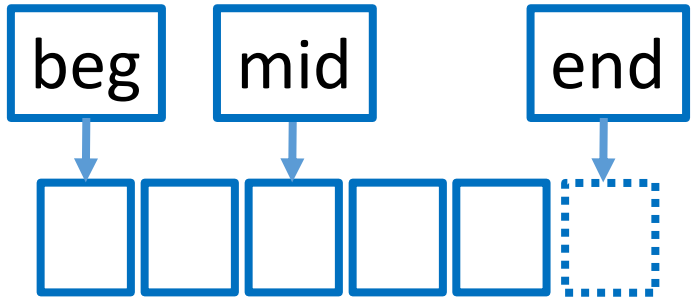
```
vector<int> ivec;
. . .
auto pbeg = v.begin();
while (pbeg != v.end() && *beg >= 0)
     cout << *pbeg++ << endl; // print and advance

// *(pbeg++): pbeg++ yields a copy of the previous value
```

אריתמטיקה של איטרטורים

הוספת מספר שלם לאיטרטור מקדמת אותו בכמה אלמנטים בהתאם למספר.

חיפוש בינרי עם איטרטורים (הטכסט ממוין):

```
vector<string> text = {"abc", "def", . . .};
auto beg = text.begin(), end = text.end();
auto mid = text.begin() + (end - beg) / 2;
while (mid != end && *mid != sought) {
    if (sought < *mid)
        end = mid;
    else
        beg = mid + 1;
    mid = beg + (end - beg) / 2;
} // mid will be equal to end or *mid will equal sought
```



מיכלים

string מיכל שמכיל תווים בלבד.

גישה אקראית מהירה, הוספה ומחיקה מהירה בסוף המחרוזת.

vector מערך בגודל משתנה.

גישה אקראית מהירה, הוספה ומחיקה מהירה בסוף הווקטור.

deque תור עם שני קצוות.

גישה אקראית מהירה, הוספה ומחיקה מהירה בתחילת ובסוף התור.

list רשימה מקושרת כפולה.

גישה סדרתית בלבד, הוספה ומחיקה מהירה בכול מקום ברשימה.

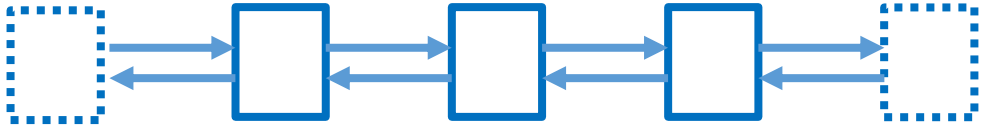
map אוסף של זוגות מפתח-ערך.

חיפוש מהיר של ערך לפי המפתח

set אוסף של מפתחות.

חיפוש מהיר האם מפתח נמצא באוסף

push_front



למיכל `list` ולמיכל `deque` אפשר להוסיף במהירות איבר גם לתחילת הרשימה עם `push_front`:

```
list<int> ilist;
for (size_t ix = 0; ix != 4; ++ix)
    ilist.push_front(ix);
```

insert

הוספת איבר או איברים בכל מקום במיכל.

הפרמטר הראשון הוא איטרטור שמצביע על המקום **שלפניו** יוכנסו האיברים:

```
list<string> slist;
// equivalent to calling slist.push_front("Hello!")
slist.insert(slist.begin(), "Hello ");
slist.insert(slist.end(), "world!");  // {"Hello", "world!"}
// insert the last two elements of v at the beginning of slist
slist.insert(slist.begin(), v.end() - 2, v.end());

vector<string> svec;
svec.insert(svec.begin(), "Hello!");
// No push_front on vector or string
// Can insert anywhere in a vector or string
// However, doing so can be an expensive operation
```

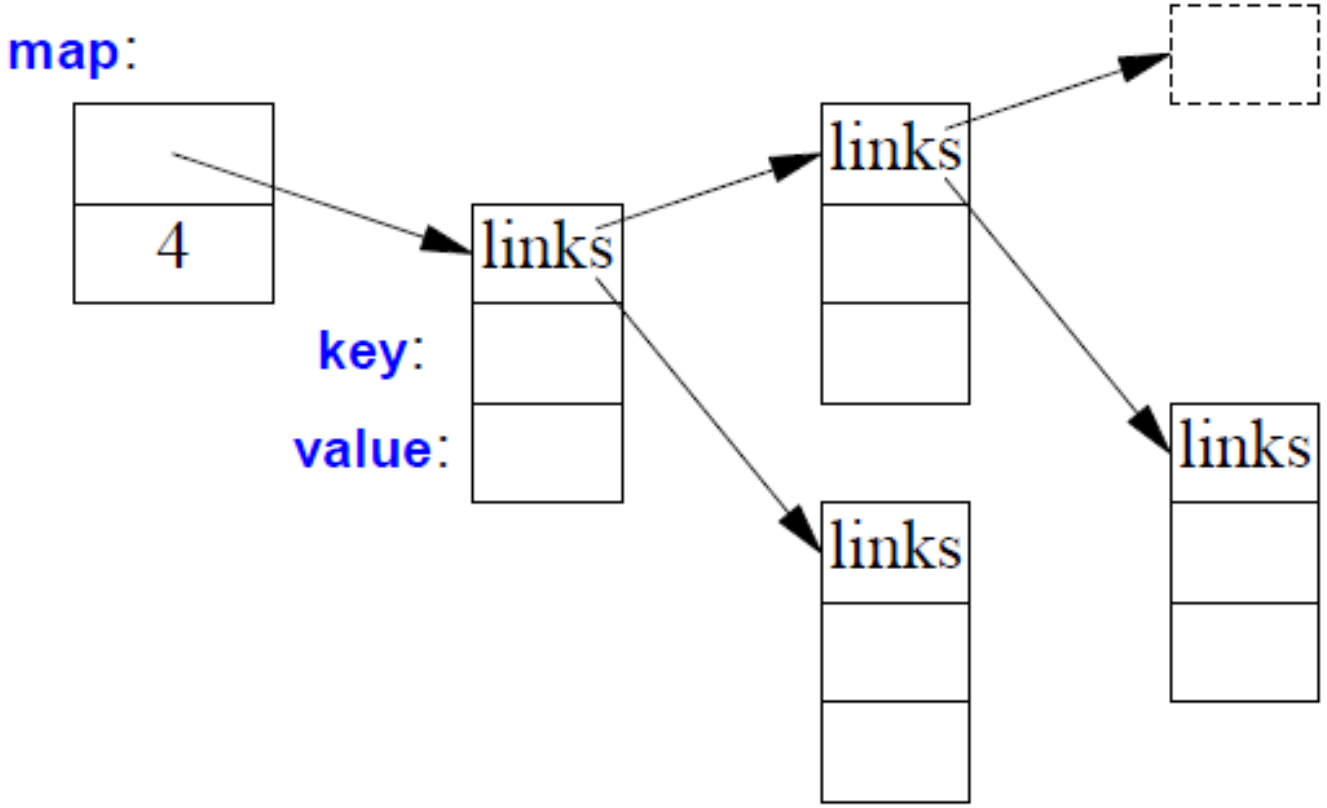
emplace

בנית אלמנט באמצעות בנאי והכנסתו למיכל.

```
// construct a Sales_data object at the end of c
// uses the three-argument Sales_data constructor
c.emplace_back("978-0590353403", 25, 15.99);
// equivalent to creating a temporary Sales_data object
// and passing it to push_back
c.push_back(Sales_data("978-0590353403", 25, 15.99));
// in front of iter
c.emplace(iter, "999-999999999", 25, 15.99);
// The arguments to emplace must match a constructor
```

map

- חיפוש ברשימה אינו יעיל, מחיר החיפוש הוא O(n)
- מפה היא עץ חיפוש מאוזן, ומחיר החיפוש הוא O(log(n))



map

ספר טלפונים:

```
map<string,int> phone_book {
    {"David Hume", 123456},
    {"Karl Popper", 234567},
    {"Bertrand Arthur William Russell", 345678}
};
int get_number(const string& s)
    { return phone_book[s]; }
// When indexed by a key, a map returns the value
// If a key isn't found, it is entered into the map with a
// default value
// To avoid entering invalid numbers into our phone book,
// we could use find() instead of []:
phone_book.find(s)
```

map

מספר הפעמים שמילים מופיעות בקלט:

```
// count the number of times each word occurs in the input
map<string, size_t> word_count;  // empty map
string word;
while (cin >> word)
     ++word_count[word];
for (const auto &w : word_count) // for each element in map
     cout << w.first << " occurs " << w.second
     << ((w.second > 1) ? " times" : " time") << endl;
The output of this program is:
Although occurs 1 time   // Elements in a map are of type pair
Before occurs 1 time     // A pair holds two data members:
. . .                    // first and second
```

set

מספר הפעמים שמילים מופיעות בקלט למעט מילים שכיחות:

```
map<string, size_t> word_count;
set<string> exclude = {"the", "and", "or", "an", "a"};
string word;
while (cin >> word)
// count only words that are not in exclude
     if (exclude.find(word) == exclude.end())
          ++word_count[word];
```

map

הדפסת מספר המילים באמצעות איטרטור:

```
auto map_it = word_count.cbegin();
while (map_it != word_count.cend()) {
     cout << map_it->first << " occurs "
     << map_it->second << " times" << endl;
     ++map_it;
} // from smallest key to largest key
```

פעולות על מיכלים

Construction

`C c;`	Default constructor, empty container (`array`; see p. 336)
`C c1(c2);`	Construct `c1` as a copy of `c2`
`C c(b, e);`	Copy elements from the range denoted by iterators `b` and `e`;

Add/Remove Elements

Note: the interface to these operations varies by container type

`c.insert(`*args*`)`	Copy element(s) as specified by *args* into `c`
`c.emplace(`*inits*`)`	Use *inits* to construct an element in `c`
`c.erase(`*args*`)`	Remove element(s) specified by *args*
`c.clear()`	Remove all elements from `c`; returns `void`

פעולות על מיכלים

Type Aliases

`iterator`	Type of the iterator for this container type
`const_iterator`	Iterator type that can read but not change its elements
`size_type`	Unsigned integral type big enough to hold the size of the largest possible container of this container type

Size

`c.size()`	Number of elements in `c`
`c.max_size()`	Maximum number of elements `c` can hold
`c.empty()`	`false` if `c` has any elements, `true` otherwise

max_size.cpp

Equality and Relational Operators

`==, !=`	Equality valid for all container types
`<, <=, >, >=`	Relationals (**not valid for unordered associative containers**)

Obtain Iterators

`c.begin(), c.end()`	Return iterator to the first, one past the last element in `c`
`c.cbegin(), c.cend()`	Return `const_iterator`