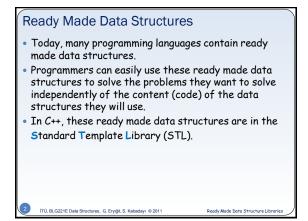
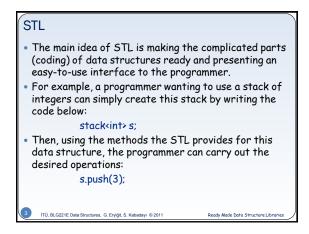
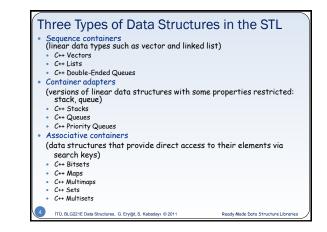
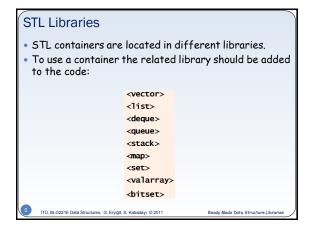
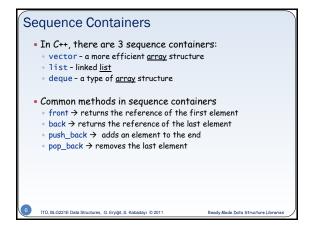
#### Data Structures Ready Made Data Structure Libraries

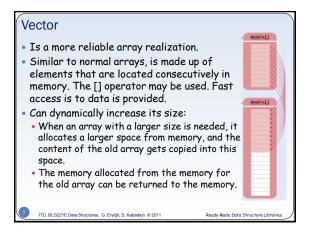


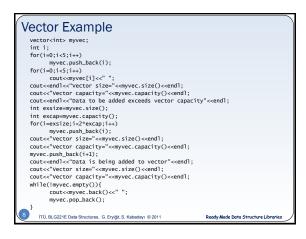




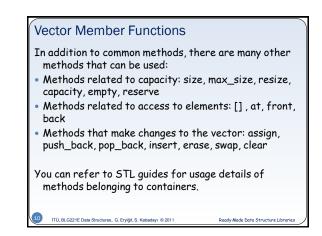






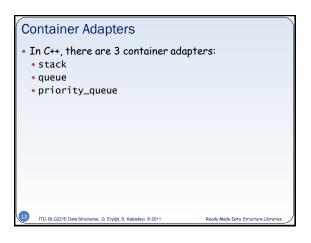


# Vector Example 0 1 2 3 4 Uektorun boyutu=5 Uektorun kapasitesi=8 Uektore kapasitesinden fazla veri ekleniyor Uektorun boyutu=16 Uektorun kapasitesi=16 Uektore bir veri ekleniyor Uektorun boyutu=17 Uektorun boyutu=17 Uektorun kapasitesi=32 17 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 Pre



# The STL container "list" presents an efficient implementation of adding to and removing from the linked list. If addition and removal operations are generally taking place at the end of the list, the "deque" container is more suitable. The "list" container is implemented as a doubly linked list. Therefore, we can traverse the list quickly in both directions. 110. BLGG21E Data Structures. G. EryGS, S. Kabaday, © 2011 Recoty Made Data Structures. Libraries

# • "double-ended queue" • The deque is a structure that has been designed to use the best of the vector and list structures together. • Like the vector, it provides fast access to elements using an index [] (this is implemented on the array.) • As in the list, operations such as fast addition and removal of elements at the front and the back have been implemented in an efficient manner (But, it is also possible to insert in between elements of the list.) • It is the default data structure for a queue structure. • In addition to the basic operations in a vector, contains the push\_front and pop\_front functions. • Note: push\_back exists in all containers, but push\_front exists in only list and deque.



```
Stack Member Functions

• empty → isempty()

• pop → its difference from our pop() method is that it does not have a return value, void pop(); removes the element at the top of the stack but does not provide access to the element.

• push → push(...)

• size → returns the number of elements in the stack

• top → provides access to the topmost element of the stack but does not remove the element from the stack.
```

```
Stack Example
#include <iostream>
#include <stack>

■ Select "H:\mydocume

using namespace std;
int main(){
 stack<int> stack:
 for(int i=0;i<5;i++)
                                Press any key to continue
      stack.push(i);
 while(!stack.empty()){
      cout<<stack.top()<<endl;</pre>
      stack.pop();
 return EXIT_SUCCESS;
   ÎTÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadayı © 2011
                                         Ready Made Data Structure Librarie
```

```
Queue Member Functions

• back → provides access to the last element of queue

• empty → isempty()

• front → provides access to the first element of queue

• pop → its difference from our pop() method is that it does not have a return value. void pop(); removes the first element of the queue but does not provide access to the element.

• push → push()

• size → returns the number of elements in the queue
```

```
Queue Example
#include <iostream>
#include <stack>
                                  "H:\mydocuments\de
#include <queue>
using namespace std;
int main(){
 queue<int> myq;
                                  Press any key to continue
 for(int i=0;i<5;i++)
      myq.push(i);
 while(!myq.empty()){
      cout<<myq.front()<<endl;</pre>
      myq.pop();
 return EXIT_SUCCESS;
    ÎTÜ. BLG221E Data Structures. G. Erviğit. S. Kabadavı © 2011
```

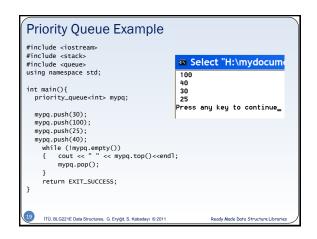
```
Priority Queue

By default, it is ordered from largest to smallest.

Largest gets processed first.

It is also possible to specify your own priority type, but more advanced programming knowledge is needed.

empty
size
top → provides access to the first element
push → add element based on priority
pop → removes the first element.
```



```
Container Adapters

The advantage of a container adapter is that the user can choose the underlying data structure.

For example, the underlying data structure for the stack structure is the deque by default. It has been implemented using the deque.

If we want to use the version of the stack realized on the linked list:

stack<int> stack; is replaced with

stack<int, list<int> stack;

If we want to do this using a vector, we write

stack<int, vector<int> stack;

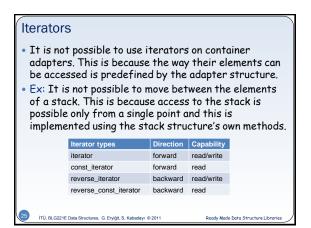
Ready Made Data Structure. Q. Enylgt. S. Kadaday: © 2011

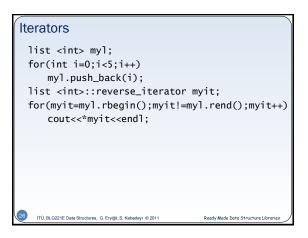
Ready Made Data Structure Libraries
```

#### Associative Containers Associative containers: • provide direct access to elements via keys • store search keys in order • There are 5 associative containers: • multiset – holds only keys, repeats are possible • set - holds only keys, repeats are not possible • bitset - set used for bit operations • multimap - holds keys and associated values, repeats are possible. • map - holds keys and associated values, repeats are not possible. Common functions: find, lower\_bound, upper\_bound, count ÎTÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadayı © 2011 Ready Made Data Structure Libraries

```
Example of Using Keys (map)
                                   Carsamba 3
 map<string, string> strMap;
                                   Cuma 5
 strMap["Monday"]
                     = "1";
                                   Cumartesi 6
                                   Pazar 7
 strMap["Tuesday"] = "2";
                                   Pazartesi 1
 strMap["Wednesday"] = "3";
                                   Persembe 4
 strMap["Thursday"] = "4";
                                   Sali 2
 strMap["Friday"] = "5";
                                   Press any key to continue
 strMap["Saturday"] = "6";
 strMap.insert(pair<string, string>("Sunday", "7"));
 while(!strMap.empty()){
      cout<<strMap.begin()->first<<" "</pre>
            <<strMap.begin()->second<<endl;
      strMap.erase(strMap.begin());
   ÎTÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadayı © 2011
                                           Ready Made Data Structure Librarie
```

```
    When we use the ready made libraries of C++ for data structures, we use structures called iterators to iterate over the elements of a data structure.
    Iterators resemble pointers.
    Just as we use pointers when traversing a linked list,
    Ex: Node *ptr=head;
    while(ptr!=NULL) ptr=ptr->next;
    when traversing the list structure of the STL, we use iterators. (We do not know what the node structure of the "list" is. We are only processing the data part.)
```





#### Example To measure the frequency of words in a file containing English plaintext, we will write a program that uses the "map" data type. Topic: STL and sequential access files As output, the program will display the 10 most frequently used words in the file.

```
Example: Word Frequency

• First, words read in order from the file should be put into a map structure. In this way, the frequency of each word will be calculated.

FILE *myfile= fopen( "english.txt", "r" ); if(!myfile){ ... } char word[100]; map <string,int> freq; while(!feof(myfile)){ fscanf(myfile, "%s", word); freq[word]++; } fclose(myfile);
```

```
Example: Word Frequency

The map data structure is ordered only by key.

It is not possible to order by value.

For that reason, the map should be assigned to another map such that not the words, but the value fields are the keys.

Since there will be more than one word with the same frequency value, the structure has to be a multimap:

multimap <int,string > freq_rev;
map<string,int>::iterator it;
for(it=freq.begin();it!=freq.end();it++)
freq_rev.insert(make_pair(it->second,it->first));
```

```
Example: Word Frequency

In the new multimap, values are ordered from smallest to largest.

To print the most frequent 10 words to the screen, the 10 records at the end of the structure have to be found.

multimap <int,string>::reverse_iterator myit; int count; for (myit=freq_rev.rbegin(),count=0;count<10;myit++,count++) cout<<(*myit).second<<" "<<(*myit).first<<endl;
```

