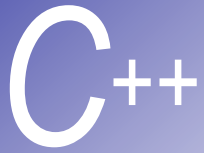


C++

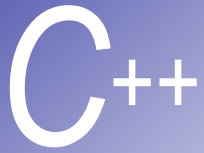
Ders 10

Veri Yapıları



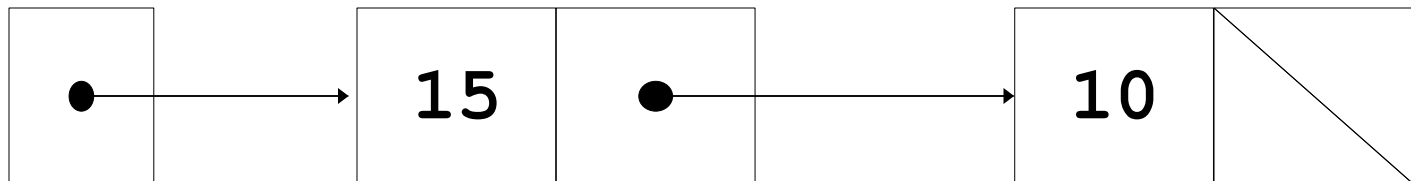
Giriş

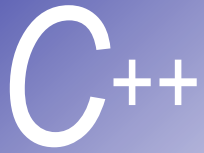
- *Dinamik veri yapıları* – Veri yapısını programın çalışması esnasında büyütme veya küçültme
- *Bağlı listeler* – Her hangi bir yerde ekleme ve çıkartma yapmak
- *Stack'lar* – Stack başından ekleme ve çıkarma yapmak (LIFO)
- *Kuyruk'lar* – Kuyruk sonundan ekleme ve kuyruk başından çıkarma yapmak (FIFO)
- *Binary ağaçlar* – yüksek hızla arama, veri sınıflandırması ve aynı verileri yok etme



Kendi Türünden Referanslı Sınıflar

- *Kendi türünden referanslı sınıf*
 - Kendi sınıfının türünden referans gösteren pointer içeren sınıflar
 - Kendi aralarında liste kuyruk stack ve ağaç yapısı olarak listelenebilir
 - **NULL** pointer (0) ile sonlandırılır
- İki tane kendi türünden referanslı sınıf bir birine link edilmiştir

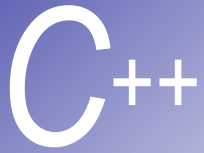




Kendi Türünden Referanslı Sınıflar

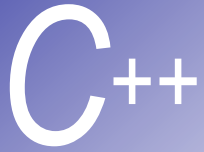
```
class Node {  
    public:  
        Node( int );  
        void setData( int );  
        int getData() const;  
        void setNextPtr( Node * );  
        const Node *getNextPtr() const;  
    private:  
        int data;  
        Node *nextPtr;  
};
```

- **nextPtr** – pointerler **Node** türünden bir nesneyi gösterir
- Bir **Node** bir diğer **Node**'a bağlanmıştır



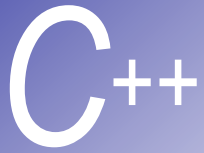
Dinamik Bellek Tahsisatı

- Dinamik bellek tahsisatı
 - Çalışma sırasında bellek tahsisatı ve serbest bırakımı yapılır
 - **new**
 - Bir argüman alır ve alan tahsisi yaptığı pointer' in adresi ile geri döner
 - **Node *newPtr = new Node(10);**
 - **sizeof(Node)** byte kadar yer ayrılır
 - **Node** constructor' ı çalışarak ayrılan alanı **newPtr** içinde tutar
 - Eğer bellek mevcut değilse **bad_alloc** hatası oluşur
 - **Node** destructor çağrılarak **new** ile tahsis edilen alan serbest bırakılır
 - **delete newPtr;**
 - **newPtr** silinmez, bellekte gösterdiği alan serbest bırakılır



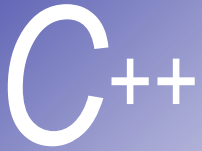
Bağlı Listeler

- Bağlı listeler
 - Kendi türünden referanslı sınıf nesneleri *links* pointerine bağlanmak için *nodes*' ı çağırırlar
 - Listenin ilk sırasındaki nesneye ulaşmak için pointerler kullanılır
 - Listenin son nesnesinin kend, türünden pointeri, listenin sonuna geldiği anlaşılması için NULL karakteri gösterir
- Şu durumlarda dizi yerine bağlı listeleri kullan:
 - Data sayısı bilinmediği zaman
 - Listenin kısaltılması gerektiğinde



Bağlı Listeler (II)

- Bağlı listelerin çeşitleri:
 - *Yalnız listelenmiş listeler*
 - İlk nodu gösteren bir pointer ile başlar
 - NULL pointer ile biter
 - Sadece bir yönde ulaşılabilir
 - *Dairesel, yalnız bağlanmış listeler*
 - Son noddaki pointer ilk nodu gösterir
 - *Çift bağlanmış listeler*
 - iki “başlangıç pointeri” - ilk eleman ve son eleman
 - Her bir node bir sonraki ve bir önceki nodları gösteren iki pointere sahiptir
 - İki yöndede dolaşılmaya izin verir
 - *Dairesel ve çift bağlı listeler*
 - Son nodun ileriye göstermesi gereken pointeri bir ilk nodu gösterir, ilk nodun önceki nodu göstermesi gereken pointeri son nodu gösterir



```

1 // Fig. 15.3: listnd.h
2 // ListNode template definition
3 #ifndef LISTND_H
4 #define LISTND_H
5
6 template< class NODETYPE > class List; // forward declaration
7
8 template<class NODETYPE>
9 class ListNode {
10     friend class List< NODETYPE >; // make List a friend
11 public:
12     ListNode( const NODETYPE & ); // constructor
13     NODETYPE getData() const; // return data in the node
14 private:
15     NODETYPE data; // data
16     ListNode< NODETYPE > *nextPtr; // next node in the list
17 };
18
19 // Constructor
20 template<class NODETYPE>
21 ListNode< NODETYPE >::ListNode( const NODETYPE &info )
22     : data( info ), nextPtr( 0 ) { }
23
24 // Return a copy of the data in the node
25 template< class NODETYPE >
26 NODETYPE ListNode< NODETYPE >::getData() const { return data; }
27
28 #endif

```

Nodlar datanın yanında bile
pointer içerir

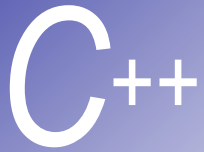
C++

```

29 // Fig. 15.3: list.h
30 // Template List class definition
31 #ifndef LIST_H
32 #define LIST_H
33
34 #include <iostream>
35 #include <cassert>
36 #include "listnd.h"
37
38 using std::cout;
39
40 template< class NODETYPE >
41 class List {
42 public:
43     List():           // constructor
44     ~List():          // destructor
45     void insertAtFront( const NODETYPE & );
46     void insertAtBack( const NODETYPE & );
47     bool removeFromFront( NODETYPE & );
48     bool removeFromBack( NODETYPE & );
49     bool isEmpty() const;
50     void print() const;
51 private:
52     ListNode< NODETYPE > *firstPtr; // pointer to first node
53     ListNode< NODETYPE > *lastPtr;  // pointer to last node
54
55     // Utility function to allocate a new node
56     ListNode< NODETYPE > *getNewNode( const NODETYPE & );
57 };
58
59 // Default constructor
60 template< class NODETYPE >
61 List< NODETYPE >::List() : firstPtr( 0 ), lastPtr( 0 ) { }

```

List iki pointere sahiptir ve yeni nodlar yaratabilir



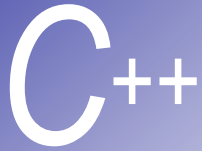
```

62
63 // Destructor
64 template< class NODETYPE >
65 List< NODETYPE >::~~List()
66 {
67     if ( !isEmpty() ) {      // List is not empty
68         cout << "Destroying nodes ...\n";
69
70         ListNode< NODETYPE > *currentPtr = firstPtr, *tempPtr;
71
72         while ( currentPtr != 0 ) { // delete remaining nodes
73             tempPtr = currentPtr;
74             cout << tempPtr->data << '\n';
75             currentPtr = currentPtr->nextPtr;
76             delete tempPtr;
77         }
78     }
79
80     cout << "All nodes destroyed\n\n";
81 }
82
83 // Insert a node at the front of the list
84 template< class NODETYPE >
85 void List< NODETYPE >::insertAtFront( const NODETYPE &value )
86 {
87     ListNode< NODETYPE > *newPtr = getNewNode( value );
88
89     if ( isEmpty() ) // List is empty
90         firstPtr = lastPtr = newPtr;
91     else {          // List is not empty
92         newPtr->nextPtr = firstPtr;
93         firstPtr = newPtr;
94     }
95 }

```

Destructor bağlı listeyi ve nodu siler

Sets the new node to point to what **firstPtr** points to, then sets **firstPtr** to the new node



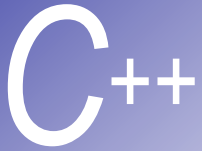
```

96
97 // Insert a node at the back of the list
98 template< class NODETYPE >
99 void List< NODETYPE >::insertAtBack( const NODETYPE &value )
100 {
101     ListNode< NODETYPE > *newPtr = getNewNode( value );
102
103     if ( isEmpty() ) // List is empty
104         firstPtr = lastPtr = newPtr;
105     else {           // List is not empty
106         lastPtr->nextPtr = newPtr;
107         lastPtr = newPtr;
108     }
109 }
110
111 // Delete a node from the front of the list
112 template< class NODETYPE >
113 bool List< NODETYPE >::removeFromFront( NODETYPE &value )
114 {
115     if ( isEmpty() ) // List is empty
116         return false; // delete unsuccessful
117     else {
118         ListNode< NODETYPE > *tempPtr = firstPtr;
119
120         if ( firstPtr == lastPtr )
121             firstPtr = lastPtr = 0;
122         else
123             firstPtr = firstPtr->nextPtr;
124
125         value = tempPtr->data; // data being removed
126         delete tempPtr;
127         return true; // delete successful
128     }
129 }

```

Son nod yeni nodu gösterir, sonra **lastPtr** yeni nodu gösterir

firstPtr'i ikinci noda taşı ve ilk nodu sil..



```

130
131 // Delete a node from the back of the list
132 template< class NODETYPE >
133 bool List< NODETYPE >::removeFromBack( NODETYPE &value )
134 {
135     if ( isEmpty() )
136         return false:    // delete unsuccessful
137     else {
138         ListNode< NODETYPE > *tempPtr = lastPtr;
139
140         if ( firstPtr == lastPtr )
141             firstPtr = lastPtr = 0;
142         else {
143             ListNode< NODETYPE > *currentPtr = firstPtr;
144
145             while ( currentPtr->nextPtr != lastPtr )
146                 currentPtr = currentPtr->nextPtr;
147
148             lastPtr = currentPtr;
149             currentPtr->nextPtr = 0;
150         }
151
152         value = tempPtr->data;
153         delete tempPtr;
154         return true:    // delete successful
155     }
156 }
157
158 // Is the List empty?
159 template< class NODETYPE >
160 bool List< NODETYPE >::isEmpty() const
161 { return firstPtr == 0; }
162
163 // return a pointer to a newly allocated node

```

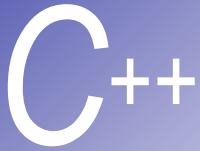
Change **lastPtr** to the second to last node and **delete** the last node

C++

```
164 template< class NODETYPE >
165 ListNode< NODETYPE > *List< NODETYPE >::getNewNode(
166                                     const NODETYPE &value )
167 {
168     ListNode< NODETYPE > *ptr =
169         new ListNode< NODETYPE >( value );
170     assert( ptr != 0 );
171     return ptr;
172 }
173
174 // Display the contents of the List
175 template< class NODETYPE >
176 void List< NODETYPE >::print() const
177 {
178     if ( isEmpty() ) {
179         cout << "The list is empty\n\n";
180         return;
181     }
182
183     ListNode< NODETYPE > *currentPtr = firstPtr;
184
185     cout << "The list is: ";
186
187     while ( currentPtr != 0 ) {
188         cout << currentPtr->data << ' ';
189         currentPtr = currentPtr->nextPtr;
190     }
191
192     cout << "\n\n";
193 }
194
195 #endif
```

Yeni nod yarat ve onun pointeri ile dön.

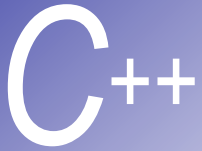
Liste boyunca yürü ve nodun değerini yazdır.



```

196// Fig. 15.3: fig15_03.cpp
197// List class test
198#include <iostream>
199#include "list.h"
200
201using std::cin;
202using std::endl;
203
204// Function to test an integer List
205template< class T >
206void testList( List< T > &listObject, const char *type )
207{
208    cout << "Testing a List of " << type << " values\n";
209
210    instructions();
211    int choice;
212    T value;
213
214    do {
215        cout << "? ";
216        cin >> choice;
217
218        switch ( choice ) {
219            case 1:
220                cout << "Enter " << type << ": ";
221                cin >> value;
222                listObject.insertAtFront( value );
223                listObject.print();
224                break;
225            case 2:
226                cout << "Enter " << type << ": ";
227                cin >> value;

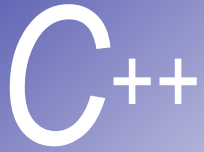
```



```
228         listObject.insertAtBack( value );
229         listObject.print();
230         break;
231     case 3:
232         if ( listObject.removeFromFront( value ) )
233             cout << value << " removed from list\n";
234
235         listObject.print();
236         break;
237     case 4:
238         if ( listObject.removeFromBack( value ) )
239             cout << value << " removed from list\n";
240
241         listObject.print();
242         break;
243     }
244 } while ( choice != 5 );
245
246 cout << "End list test\n\n";
247 }
248
249 void instructions()
250 {
251     cout << "Enter one of the following:\n"
252         << " 1 to insert at beginning of list\n"
253         << " 2 to insert at end of list\n"
254         << " 3 to delete from beginning of list\n"
255         << " 4 to delete from end of list\n"
256         << " 5 to end list processing\n";
257 }
258
```

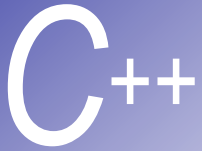


Uygun switch ifadesini seç



```
259 int main()
260 {
261     List< int > integerList;
262     testList( integerList, "integer" ); // test integerList
263
264     List< double > doubleList;
265     testList( doubleList, "double" );    // test doubleList
266
267     return 0;
268 }
```

İnteger ve double liste yaratmak temlateleri kullan



Testing a List of integer values

Enter one of the following:

- 1 to insert at beginning of list
- 2 to insert at end of list
- 3 to delete from beginning of list
- 4 to delete from end of list
- 5 to end list processing

? 1

Enter integer: 1

The list is: 1

? 1

Enter integer: 2

The list is: 2 1

? 2

Enter integer: 3

The list is: 2 1 3

? 2

Enter integer: 4

The list is: 2 1 3 4

? 3

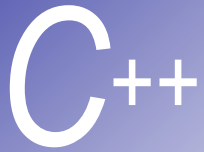
2 removed from list

The list is: 1 3 4

? 3

1 removed from list

The list is: 3 4



```
? 4
4 removed from list
The list is: 3
```

```
? 4
3 removed from list
The list is empty
```

```
? 5
End list test
```

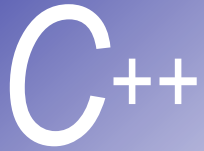
```
Testing a List of double values
Enter one of the following:
  1 to insert at beginning of list
  2 to insert at end of list
  3 to delete from beginning of list
  4 to delete from end of list
  5 to end list processing
```

```
? 1
Enter double: 1.1
The list is: 1.1
```

```
? 1
Enter double: 2.2
The list is: 2.2 1.1
```

```
? 2
Enter double: 3.3
The list is: 2.2 1.1 3.3
```

```
? 2
Enter double: 4.4
The list is: 2.2 1.1 3.3 4.4
```



```
? 3
2.2 removed from list
The list is: 1.1 3.3 4.4
```

```
? 3
1.1 removed from list
The list is: 3.3 4.4
```

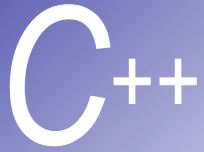
```
? 4
4.4 removed from list
The list is: 3.3
```

```
? 4
3.3 removed from list
The list is empty
```

```
? 5
End list test
```

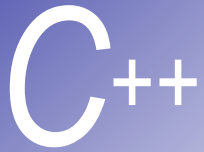
```
All nodes destroyed
```

```
All nodes destroyed
```



Stack' lar

- stack – yeni node sadece başa eklenir yada mevcut node sadece baş kısmından çıkarılır
 - Tabak sırasına benzer
 - Son giren ilk çıkar (last-in, first-out) (LIFO)
 - Stack' ın alt kısmı bağlı listedeki üye NULL gösterir
 - Bağlı listelerin sınırlandırılmışıdır
- push
 - Stack' ın başına yeni node ekler
- pop
 - Baştan node siler
 - popped değeri saklar
 - **pop** başarılı olmuşsa **true** ile döner

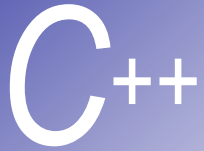


```

1  // Fig. 15.9: stack.h
2  // Stack class template definition
3  // Derived from class List
4  #ifndef STACK_H
5  #define STACK_H
6
7  #include "list.h"
8
9  template< class STACKTYPE >
10 class Stack : private List< STACKTYPE > {
11 public:
12     void push( const STACKTYPE &d ) { insertAtFront( d ); }
13     bool pop( STACKTYPE &d ) { return removeFromFront( d ); }
14     bool isEmpty() const { return isEmpty(); }
15     void printStack() const { print(); }
16 };
17
18 #endif
19 // Fig. 15.9: fig15_09.cpp
20 // Driver to test the template Stack class
21 #include <iostream>
22 #include "stack.h"
23
24 using std::endl;
25
26 int main()
27 {
28     Stack< int > intStack;
29     int popInteger, i;
30     cout << "processing an integer Stack" << endl;
31
32     for ( i = 0; i < 4; i++ ) {
33         intStack.push( i );

```

Notice the functions **Stack** has:
insertAtFront (push) and
removeFromFront (pop)



```

34     intStack.printStack();
35 }
36
37 while ( !intStack.isStackEmpty() ) {
38     intStack.pop( popInteger );
39     cout << popInteger << " popped from stack" << endl;
40     intStack.printStack();
41 }
42
43 Stack< double > doubleStack;
44 double val = 1.1, popdouble;
45 cout << "processing a double Stack" << endl;
46
47 for ( i = 0; i < 4; i++ ) {
48     doubleStack.push( val );
49     doubleStack.printStack();
50     val += 1.1;
51 }
52
53 while ( !doubleStack.isStackEmpty() ) {
54     doubleStack.pop( popdouble );
55     cout << popdouble << " popped from stack" << endl;
56     doubleStack.printStack();
57 }
58 return 0;
59 }

```

The list is: 0

The list is: 1 0

3 popped from stack

The list is: 2 1 0

2 popped from stack

The list is: 1 0

processing a double Stack

1 popped from stack

4.4 popped from stack

The list is: 3.3 2.2 1.1

The 3.3 popped from stack

The list is: 2.2 1.1

The 2.2 popped from stack

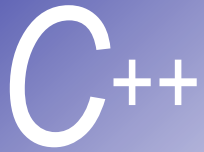
The list is: 1.1

1.1 popped from stack

All nodes destroyed

list is empty

All nodes destroyed



```
processing an integer Stack
```

```
The list is: 0
```

```
The list is: 1 0
```

```
The list is: 2 1 0
```

```
The list is: 3 2 1 0
```

```
3 popped from stack
```

```
The list is: 2 1 0
```

```
2 popped from stack
```

```
The list is: 1 0
```

```
1 popped from stack
```

```
The list is: 0
```

```
0 popped from stack
```

```
The list is empty
```

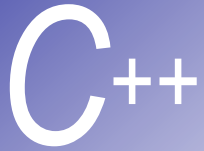
```
processing a double Stack
```

```
The list is: 1.1
```

```
The list is: 2.2 1.1
```

```
The list is: 3.3 2.2 1.1
```

```
The list is: 4.4 3.3 2.2 1.1
```



```
4.4 popped from stack  
The list is: 3.3 2.2 1.1
```

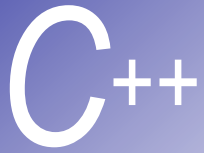
```
3.3 popped from stack  
The list is: 2.2 1.1
```

```
2.2 popped from stack  
The list is: 1.1
```

```
1.1 popped from stack  
The list is empty
```

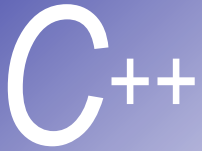
```
All nodes destroyed
```

```
All nodes destroyed
```

Kuyruklar

- Kuyruk – Alışveriş merkezindeki kontrol noktasına benzer
 - İlk gelen, ilk çıkar(*first-in, first-out*) (*FIFO*)
 - Node' lar sadece baş kısmından alınır
 - Node' lar sadece son kısma yerleştirilir
- Ekleme ve çıkarma işlemi *enqueue* ve *dequeue* olarak bilinir
- Hesaplamada kullanışlıdır
 - Yazıcı iş kuyruğu, ağdaki veri paketleri, ve dosya sunucu talepleri bu sınıfa girer

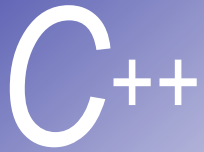


```

1  // Fig. 15.12: queue.h
2  // Queue class template definition
3  // Derived from class List
4  #ifndef OUEUE H
5  #define OUEUE H
6
7  #include "list.h"
8
9  template< class OUEUETYPE >
10 class Oueue: private List< OUEUETYPE > {
11 public:
12     void enqueue( const OUEUETYPE &d ) { insertAtBack( d ); }
13     bool dequeue( OUEUETYPE &d )
14         { return removeFromFront( d ); }
15     bool isQueueEmpty() const { return isEmpty(); }
16     void printQueue() const { print(); }
17 };
18
19 #endif
20 // Fig. 15.12: fig15_12.cpp
21 // Driver to test the template Oueue class
22 #include <iostream>
23 #include "queue.h"
24
25 using std::endl;
26
27 int main()
28 {
29     Oueue< int > intQueue;
30     int dequeueInteger, i;
31     cout << "processing an integer Oueue" << endl;
32
33     for ( i = 0; i < 4; i++ ) {

```

queue yalnızca limitli bağlı listeler operasyonuna sahip (insertAtBack ve removeFromFront)



```

34     intQueue.enqueue( i );
35     intQueue.printQueue();
36 }
37
38 while ( !intQueue.isEmpty() ) {
39     intQueue.dequeue( dequeuedInteger );
40     cout << dequeuedInteger << " dequeued" <<
41     intQueue.printQueue();
42 }
43
44 Queue< double > doubleQueue;
45 double val = 1.1, dequeuedouble;
46
47 cout << "processing a double Queue" << endl;
48
49 for ( i = 0; i < 4; i++ ) {
50     doubleQueue.enqueue( val );
51     doubleQueue.printQueue();
52     val += 1.1;
53 }
54
55 while ( !doubleQueue.isEmpty() ) {
56     doubleQueue.dequeue( dequeuedouble );
57     cout << dequeuedouble << " dequeued" << endl;
58     doubleQueue.printQueue();
59 }
60
61 return 0;
62 }

```

The list is: 0

0 dequeued

The list is: 1 2 3

1 dequeued

The list is: 2 3

2 dequeued

The list is: 1.1

processing a

The list is: 1.1 2.2

1.1 dequeued

The list is: 2.2 3.3 4.4

2.2 dequeued

The list is: 3.3 4.4

3.3 dequeued

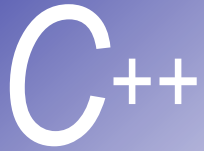
The list is: 4.4

4.4 dequeued

The list is empty

All nodes destroyed

All nodes destroyed



processing an integer Queue

The list is: 0

The list is: 0 1

The list is: 0 1 2

The list is: 0 1 2 3

0 dequeued

The list is: 1 2 3

1 dequeued

The list is: 2 3

2 dequeued

The list is: 3

3 dequeued

The list is empty

processing a double Queue

The list is: 1.1

The list is: 1.1 2.2

The list is: 1.1 2.2 3.3

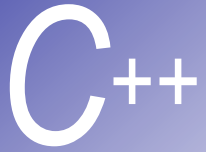
The list is: 1.1 2.2 3.3 4.4

1.1 dequeued

The list is: 2.2 3.3 4.4

2.2 dequeued

The list is: 3.3 4.4



```
3.3 dequeued
```

```
The list is: 4.4
```

```
4.4 dequeued
```

```
The list is empty
```

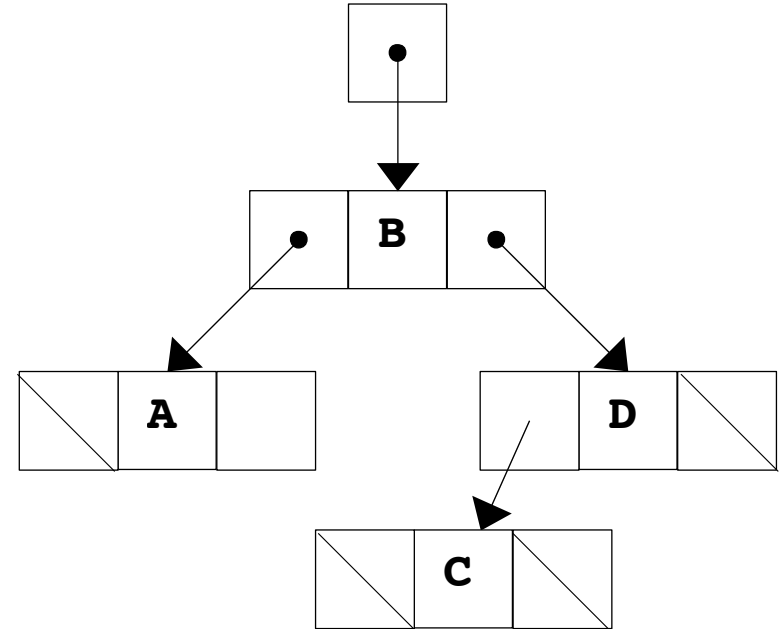
```
All nodes destroyed
```

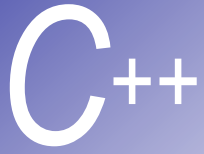
```
All nodes destroyed
```

C++

Ağaç Yapısı

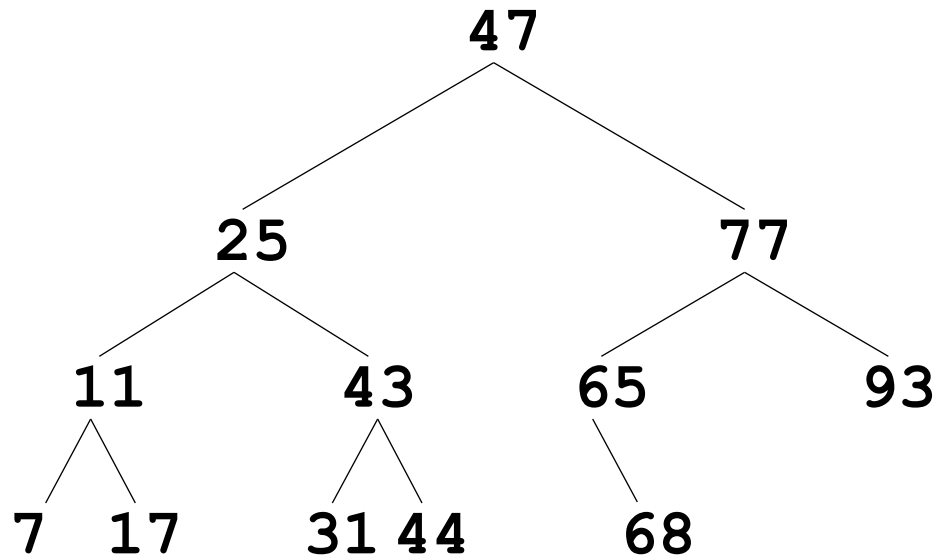
- Ağaç yapıları iki yada daha fazla bağ içerir
 - Daha önce bahsettiğimiz veri yapıları türünü içeren bir yapıdır
- Binary Ağaç
 - Tüm node' lar iki bağ içerir
 - Hiçbiri, biri, veya her ikisi **NULL** içerebilir
 - Ağaç yapısında ilk node kök node' dur
 - Kök node' daki her bir bağa çocuk denir
 - Çocuğu' u olmayan node' a yaprak node denir

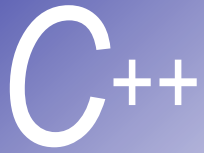




Ağaç Yapısı(II)

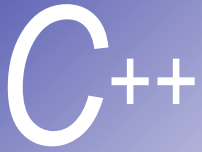
- Binary arama ağacı
 - Soldaki çocuğun değeri kök değerinden daha azdır
 - Sağdaki çocuğun değeri kök değerinden yüksektir
 - *çifte yoketme* kolaylığı
 - Hızlı arama – dengeli bir ağaç için, maksimum $\log n$ mukayese





Ağaç Yapısı(III)

- Tree traversals:
 - inorder traversal of a binary search tree prints the node values in ascending order
 1. Traverse the left subtree with an inorder traversal.
 2. Process the value in the node (i.e., print the node value).
 3. Traverse the right subtree with an inorder traversal.
 - preorder traversal:
 1. Process the value in the node.
 2. Traverse the left subtree with a preorder traversal.
 3. Traverse the right subtree with a preorder traversal.
 - postorder traversal:
 1. Traverse the left subtree with a postorder traversal.
 2. Traverse the right subtree with a postorder traversal.
 3. Process the value in the node.

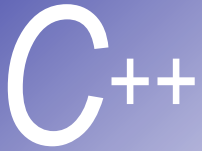


```

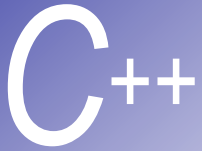
1  // Fig. 15.16: treenode.h
2  // Definition of class TreeNode
3  #ifndef TREENODE H
4  #define TREENODE H
5
6  template< class NODETYPE > class Tree:  // forward declaration
7
8  template< class NODETYPE >
9  class TreeNode {
10     friend class Tree< NODETYPE >;
11 public:
12     TreeNode( const NODETYPE &d )
13         : leftPtr( 0 ), data( d ), rightPtr( 0 ) {
14         NODETYPE getData() const { return data; }
15 private:
16     TreeNode< NODETYPE > *leftPtr:  // pointer to left subtree
17     NODETYPE data:
18     TreeNode< NODETYPE > *rightPtr: // pointer to right subtree
19 };
20
21 #endif
22 // Fig. 15.16: tree.h
23 // Definition of template class Tree
24 #ifndef TREE H
25 #define TREE H
26
27 #include <iostream>
28 #include <cassert>
29 #include "treenode.h"
30
31 using std::endl;
32
33 template< class NODETYPE >

```

Ağaç her bir nod içim iki pointer içerir



```
34 class Tree {
35 public:
36     Tree();
37     void insertNode( const NODETYPE & );
38     void preOrderTraversal() const;
39     void inOrderTraversal() const;
40     void postOrderTraversal() const;
41 private:
42     TreeNode< NODETYPE > *rootPtr;
43
44     // utility functions
45     void insertNodeHelper(
46         TreeNode< NODETYPE > **, const NODETYPE & );
47     void preOrderHelper( TreeNode< NODETYPE > * ) const;
48     void inOrderHelper( TreeNode< NODETYPE > * ) const;
49     void postOrderHelper( TreeNode< NODETYPE > * ) const;
50 };
51
52 template< class NODETYPE >
53 Tree< NODETYPE >::Tree() { rootPtr = 0; }
54
55 template< class NODETYPE >
56 void Tree< NODETYPE >::insertNode( const NODETYPE &value )
57     { insertNodeHelper( &rootPtr, value ); }
58
59 // This function receives a pointer to a pointer so the
60 // pointer can be modified.
61 template< class NODETYPE >
62 void Tree< NODETYPE >::insertNodeHelper(
63     TreeNode< NODETYPE > **ptr, const NODETYPE &value )
64 {
```

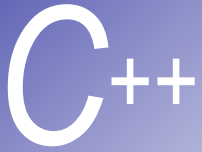


```

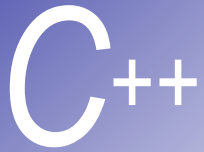
65     if ( *ptr == 0 ) {                                     // tree is empty
66         *ptr = new TreeNode< NODETYPE >( value );
67         assert( *ptr != 0 );
68     }
69     else                                                    // tree is not empty
70         if ( value < ( *ptr )->data )
71             insertNodeHelper( &( ( *ptr )->leftPtr ), value );
72         else
73             if ( value > ( *ptr )->data )
74                 insertNodeHelper( &( ( *ptr )->rightPtr ), value );
75             else
76                 cout << value << " dup" << endl;
77 }
78
79 template< class NODETYPE >
80 void Tree< NODETYPE >::preOrderTraversal() const
81 { preOrderHelper( rootPtr ); }
82
83 template< class NODETYPE >
84 void Tree< NODETYPE >::preOrderHelper(
85     TreeNode< NODETYPE > *ptr ) const
86 {
87     if ( ptr != 0 ) {
88         cout << ptr->data << ' ';
89         preOrderHelper( ptr->leftPtr );
90         preOrderHelper( ptr->rightPtr );
91     }
92 }
93
94 template< class NODETYPE >
95 void Tree< NODETYPE >::inOrderTraversal() const
96 { inOrderHelper( rootPtr ); }
97

```

Traversal kendi recursive tanımlanmışlardır



```
98 template< class NODETYPE >
99 void Tree< NODETYPE >::inOrderHelper(
100                                     TreeNode< NODETYPE > *ptr ) const
101{
102     if ( ptr != 0 ) {
103         inOrderHelper( ptr->leftPtr );
104         cout << ptr->data << ' ';
105         inOrderHelper( ptr->rightPtr );
106     }
107}
108
109template< class NODETYPE >
110void Tree< NODETYPE >::postOrderTraversal() const
111    { postOrderHelper( rootPtr ); }
112
113template< class NODETYPE >
114void Tree< NODETYPE >::postOrderHelper(
115                                     TreeNode< NODETYPE > *ptr ) const
116{
117     if ( ptr != 0 ) {
118         postOrderHelper( ptr->leftPtr );
119         postOrderHelper( ptr->rightPtr );
120         cout << ptr->data << ' ';
121     }
122}
123
124#endif
```



```

125// Fig. 15.16: fig15_16.cpp
126// Driver to test class Tree
127#include <iostream>
128#include <iomanip>
129#include "tree.h"
130
131using std::cout;
132using std::cin;
133using std::setiosflags;
134using std::ios;
135using std::setprecision;
136
137int main()
138{
139    Tree< int > intTree;
140    int intVal, i;
141
142    cout << "Enter 10 integer values:\n";
143    for ( i = 0; i < 10; i++ ) {
144        cin >> intVal;
145        intTree.insertNode( intVal );
146    }
147
148    cout << "\nPreorder traversal\n";
149    intTree.preOrderTraversal();
150
151    cout << "\nInorder traversal\n";
152    intTree.inOrderTraversal();
153
154    cout << "\nPostorder traversal\n";
155    intTree.postOrderTraversal();
156

```

Preorder traversal

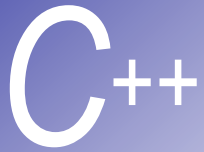
50 25 12 6 13 33 75 67 68 88

Inorder traversal

6 12 13 25 33 50 67 68 75 88

Postorder traversal

6 13 12 33 25 68 67 88 75 50



```

157  Tree< double > doubleTree;
158  double doubleVal;
159
160  cout << "\n\nEnter 10 double values:\n"
161        << setiosflags( ios::fixed | ios::showpoint )
162        << setprecision( 1 );
163  for ( i = 0; i < 10; i++ ) {
164      cin >> doubleVal;
165      doubleTree.insertNode( doubleVal );
166  }
167
168  cout << "\nPreorder traversal\n";
169  doubleTree.preOrderTraversal();
170
171  cout << "\nInorder traversal\n";
172  doubleTree.inOrderTraversal();
173
174  cout << "\nPostorder traversal\n";
175  doubleTree.postOrderTraversal();
176
177  return 0;
178}

```

create a **double** tree

Enter 10 double values:

39.2 16.5 82.7 3.3 65.2 90.8 1.1 4.4
89.5 92.5

Preorder traversal

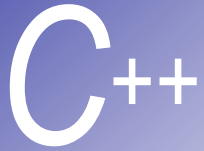
39.2 16.5 3.3 1.1 4.4 82.7 65.2
90.8 89.5 92.5

Inorder traversal

1.1 3.3 4.4 16.5 39.2 65.2 82.7
89.5 90.8 92.5

Postorder traversal

1.1 4.4 3.3 16.5 65.2 89.5 92.5
90.8 82.7 39.2

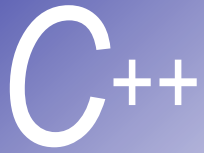


```
Enter 10 integer values:  
50 25 75 12 33 67 88 6 13 68
```

```
Preorder traversal  
50 25 12 6 13 33 75 67 68 88  
Inorder traversal  
6 12 13 25 33 50 67 68 75 88  
Postorder traversal  
6 13 12 33 25 68 67 88 75 50
```

```
Enter 10 double values:  
39.2 16.5 82.7 3.3 65.2 90.8 1.1 4.4 89.5 92.5
```

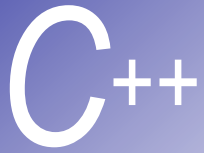
```
Preorder traversal  
39.2 16.5 3.3 1.1 4.4 82.7 65.2 90.8 89.5 92.5  
Inorder traversal  
1.1 3.3 4.4 16.5 39.2 65.2 82.7 89.5 90.8 92.5  
Postorder traversal  
1.1 4.4 3.3 16.5 65.2 89.5 92.5 90.8 82.7 39.2
```



Struct Tanımlama

```
struct Card {  
    char *face;  
    char *suit;  
};
```

- **struct** anahtar kelimesi **Card** yapısının tanımlamasına giriştir
- **Card** yapının ismidir ve bu türden tanımlama yapılmakta kullanılacaktır
Card yapısı iki tip değişken içeriyor, **char * - face** ve **suit**
- Bir yapının içerdiği elamanlar bir çok türden olabilir
 - Kendi tütünden olamaz
 - Fakat kendi türünden pointer içerebilir
- Yapı tanımlamasında bellek tahsisatı yapılmaz
 - Bu yapıyıda içerisine alacak yeni bir yapı tanımlaması yapılabilir.



Struct Tanımlama (II)

- Tanımlama

- Diğer değişkenler gibi tanımlanır:

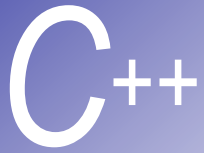
```
Card oneCard, deck[ 52 ], *cPtr;
```

- Yapı tanımlamasından sonra virgül kullanılarak o türden değişken tanımlanabilir

```
struct Card {  
    char *face;  
    char *suit;  
} oneCard, deck[ 52 ], *cPtr;
```

- Geçerli Operasyonlar

- Aynı yapı türünden değişkenleri bir birine atanabilir
- Bir yapı türünden değişkenin adresi (&) ile alınır
- Yapının bir değişkenine ulaşılabilir
- **sizeof** ile yapının büyüklüğüne belirlenebilir



Struct Türünden Değişkene Başlangıç Değeri Atama

- Initializer listesi

- Örnek:

```
Card oneCard = { "Three", "Hearts" };
```

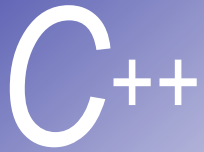
- Atama ifadesi

- Örnek:

```
Card threeHearts = oneCard;
```

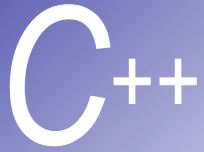
- veya:

```
Card threeHearts;  
threeHearts.face = "Three";  
threeHearts.suit = "Hearts";
```



Struct' ları Fonksiyonlarla Birlikte Kullanma

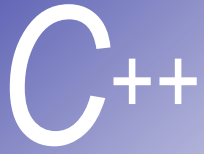
- Bir yapıyı fonksiyona geçme
 - Tüm yapı guruba geçilebilir
 - Veya yapının ilgili üyeleri fonksiyona geçilebilir
 - Her ikiside “call by value” dür
- Yapıyı “call-by-reference” olarak geçme
 - Yapının adresi geçilir
 - Referans geçilebilir
- Dizi ” call-by-value” olarak geçilebilir
 - Dizi ile yapıyı bir tanımla
 - Yapıyı parametre olarak geç



Typedef

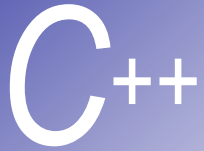
- **typedef**
 - Yapılara takma ad ta verilebilir
 - **typedef** kullanarak daha kısa bir takma isim verilir.
 - Örnek:

```
typedef Card *CardPtr;
```
- **Card *** yerine yeni data türünü **CardPtr** olarak tanımla
- **typedef** yeni tür oluşturmaz sadece takma isim oluşturur



Örnek: High-Performance Card-shuffling and Dealing Simulation

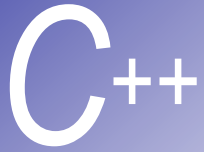
- Pseudocode:
 - `Card` yapısından bir dizi oluştur
 - Kartları `deck`' e koy
 - `Deck`' i karıştır
 - Kartları dağıt



```


1  // Fig. 16.2: fig16_02.cpp
2  // Card shuffling and dealing program using structures
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::ios;
9
10 #include <iomanip>
11
12 using std::setiosflags;
13 using std::setw;
14
15 #include <cstdlib>
16 #include <ctime>
17
18 struct Card {
19     char *face;
20     char *suit;
21 };
22
23 void fillDeck( Card * const, char *[], char *[] );
24 void shuffle( Card * const );
25 void deal( Card * const );
26
27 int main()
28 {
29     Card deck[ 52 ];
30     char *face[] = { "Ace", "Deuce", "Three", "Four",
31                     "Five", "Six", "Seven", "Eight",
32                     "Nine", "Ten", "Jack", "Queen",
33                     "King" };

```



```
34     char *suit[] = { "Hearts", "Diamonds",
35                       "Clubs", "Spades" };
36
37     srand( time( 0 ) );           // randomize
38     fillDeck( deck, face, suit );
39     shuffle( deck );
40     deal( deck );
41     return 0;
42 }
43
44 void fillDeck( Card * const wDeck, char *wFace[],
45               char *wSuit[] )
46 {
47     for ( int i = 0; i < 52; i++ ) {
48         wDeck[ i ].face = wFace[ i % 13 ];
49         wDeck[ i ].suit = wSuit[ i / 13 ];
50     }
51 }
```

52 kartı deck'e koy. **face** ve **suit** karar veriliyor

An arrow originates from the text box and points to the line of code: `wDeck[i].face = wFace[i % 13];`

C++

52

53 void shuffle(Card * const wDeck)

54 {

55 for (int i = 0; i < 52; i++) {

56 int j = rand() % 52;

57 Card temp = wDeck[i];

58 wDeck[i] = wDeck[j];

59 wDeck[j] = temp;

60 }

61 }

62

63 void deal(Card * const wDeck)

64 {

65 for (int i = 0; i < 52; i++)

66 cout << setiosflags(ios::right)

67 << setw(5) << wDeck[i].face << " of "

68 << setiosflags(ios::left)

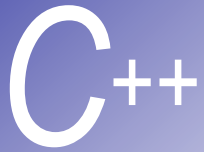
69 << setw(8) << wDeck[i].suit

70 << ((i + 1) % 2 ? '\t' : '\n');

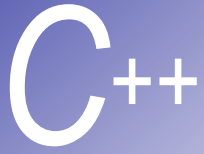
71 }

0 ile 51 arasında bir rasgele ir sayı seç. İ yi o elemanla değiştir

Dizi boyunca ilerve ekrana yazdır

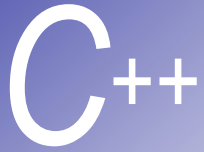


Eight of Diamonds	Ace of Hearts
Eight of Clubs	Five of Spades
Seven of Hearts	Deuce of Diamonds
Ace of Clubs	Ten of Diamonds
Deuce of Spades	Six of Diamonds
Seven of Spades	Deuce of Clubs
Jack of Clubs	Ten of Spades
King of Hearts	Jack of Diamonds
Three of Hearts	Three of Diamonds
Three of Clubs	Nine of Clubs
Ten of Hearts	Deuce of Hearts
Ten of Clubs	Seven of Diamonds
Six of Clubs	Queen of Spades
Six of Hearts	Three of Spades
Nine of Diamonds	Ace of Diamonds
Jack of Spades	Five of Clubs
King of Diamonds	Seven of Clubs
Nine of Spades	Four of Hearts
Six of Spades	Eight of Spades
Queen of Diamonds	Five of Diamonds
Ace of Spades	Nine of Hearts
King of Clubs	Five of Hearts
King of Spades	Four of Diamonds
Queen of Hearts	Eight of Hearts
Four of Spades	Jack of Hearts
Four of Clubs	Queen of Clubs

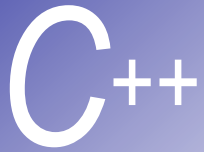


Bitwise Operatörleri

Operant	İsim	Açıklama
&	bitwise AND (Ve)	Her iki operantın 1 olması durumunda ifadenin sonucu 1 olur
	bitwise OR	Operantlardan en biri 1 olduğunda ifadenin sonucu 1 olur
^	bitwise exclusive OR	The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent.
~	One's complement	Tüm 0 bitlere 1 veya tüm 1 bitlere sıfır yerleştirir



```
1 // Fig. 16.5: fig16_05.cpp
2 // Printing an unsigned integer in bits
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7
8 #include <iomanip>
9
10 using std::setw;
11 using std::endl;
12
13 void displayBits( unsigned );
14
15 int main()
16 {
17     unsigned x;
18
19     cout << "Enter an unsigned integer: ";
20     cin >> x;
```



```

21     displayBits( x );
22     return 0;
23 }
24
25 void displayBits( unsigned value )
26 {
27     const int SHIFT = 8 * sizeof( unsigned ) - 1;
28     const unsigned MASK = 1 << SHIFT;
29
30     cout << setw( 7 ) << value << " = ";
31
32     for ( unsigned c = 1; c <= SHIFT + 1; c++ ) {
33         cout << ( value & MASK ? '1' : '0' );
34         value <<= 1;
35
36         if ( c % 8 == 0 )
37             cout << ' ';
38     }
39
40     cout << endl;
41 }

```

MASK bir bitle yaratılıyor

i.e. (10000000
00000000)

MASK ve **value** aralarında
ANDleniyor.

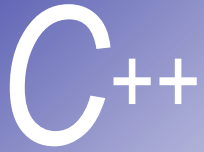
MASK yalnızca bir bit içeriyor.
AND ile doğru dönmesi **value**
ninde aynı bite sahip olduğunu
gösterir.

value nin değeri testten sonra
değiştirilir.

```

Enter an unsigned integer: 65000
65000 = 00000000 00000000 11111101 11101000

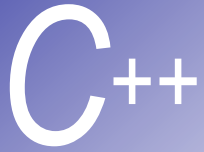
```



Bit Alanları

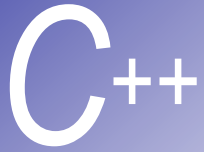
- bit alanı
 - Bir yapı elamanının bit olarak belirtilmesi
 - Daha verimli bellek kullanımı için
 - **int** veya **unsigned** olarak tanımlanmalıdır
- Örnek:

```
Struct BitCard {  
    unsigned face : 4;  
    unsigned suit : 2;  
    unsigned color : 1;  
};
```
- Bit alanların bildirimi
 - **unsigned** ve **int** olan ve : ile tanımlamaya bağlanan elamanlar bit genişliği kadar temsil edilir



Karakter İşleme Kütüphanesi

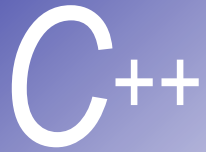
- Bir çok data karakter olarak girilir
 - Harf, rakam, özel semboller
- Character Handling kütüphanesi
 - Karakter datasını test eden fonksiyonlar
 - Fonksiyonlar argüman olarak bir karakter alır
 - Karakterler bir `int` sayı tarafından temsil edilir
 - Karakterler çoğu zaman `int` olarak işlenir
 - `EOF` 'in değeri genellikle -1 dir



Karakter İşleme Kütüphanesi

■ In <cctype>

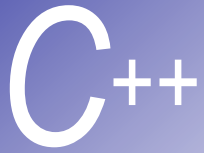
Prototype	Description
<code>int isdigit(int c)</code>	Returns true if c is a digit and false otherwise.
<code>int isalpha(int c)</code>	Returns true if c is a letter and false otherwise.
<code>int isalnum(int c)</code>	Returns true if c is a digit or a letter and false otherwise.
<code>int isxdigit(int c)</code>	Returns true if c is a hexadecimal digit character and false otherwise.
<code>int islower(int c)</code>	Returns true if c is a lowercase letter and false otherwise.
<code>int isupper(int c)</code>	Returns true if c is an uppercase letter; false otherwise.
<code>int tolower(int c)</code>	If c is an uppercase letter, tolower returns c as a lowercase letter. Otherwise, tolower returns the argument unchanged.
<code>int toupper(int c)</code>	If c is a lowercase letter, toupper returns c as an uppercase letter. Otherwise, toupper returns the argument unchanged.
<code>int isspace(int c)</code>	Returns true if c is a white-space character—newline (<code>'\n'</code>), space (<code>' '</code>), form feed (<code>'\f'</code>), carriage return (<code>'\r'</code>), horizontal tab (<code>'\t'</code>), or vertical tab (<code>'\v'</code>)—and false otherwise
<code>int iscntrl(int c)</code>	Returns true if c is a control character and false otherwise.
<code>int ispunct(int c)</code>	Returns true if c is a printing character other than a space, a digit, or a letter and false otherwise.
<code>int isprint(int c)</code>	Returns true value if c is a printing character including space (<code>' '</code>) and false otherwise.
<code>int isgraph(int c)</code>	Returns true if c is a printing character other than space (<code>' '</code>) and false otherwise.



String Dönüştürme Fonksiyonları

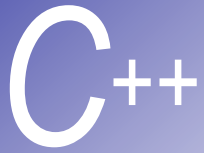
- Dönüştürme Fonksiyonları
 - In `<cstdlib>` (general utilities library)
 - Karakter gurubunu sayıya çevirirler

Prototype	Description
<code>double atof(const char *nPtr)</code>	Converts the string <code>nPtr</code> to <code>double</code> .
<code>int atoi(const char *nPtr)</code>	Converts the string <code>nPtr</code> to <code>int</code> .
<code>long atol(const char *nPtr)</code>	Converts the string <code>nPtr</code> to long <code>int</code> .
<code>double strtod(const char *nPtr, char **endPtr)</code>	Converts the string <code>nPtr</code> to <code>double</code> .
<code>long strtol(const char *nPtr, char **endPtr, int base)</code>	Converts the string <code>nPtr</code> to long.
<code>unsigned long strtoul(const char *nPtr, char **endPtr, int base)</code>	Converts the string <code>nPtr</code> to <code>unsigned long</code> .



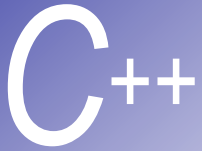
Karakter İşleme Kütüphanesinin Arama Fonksiyonları

- Yükle **<cstdlib>**
- String' leri ara:
 - Karakter olarak
 - Diğer string türleri olarak
- **size_t**
 - Bazı arama fonksiyonları tarafından geri döndürülür
 - Geri dönüş değerinin **sizeof**' u kadar standart olarak tanımlanmıştır
 - **57**



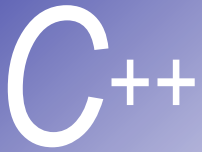
Karakter İşleme Kütüphanesinin Arama Fonksiyonları (II)

Prototype	Description
<code>char *strchr(const char *s, int c)</code>	Locates the first occurrence of character c in string s . If c is found, a pointer to c in s is returned. Otherwise, a NULL pointer is returned.
<code>char *strrchr(const char *s, int c)</code>	Locates the last occurrence of c in string s . If c is found, a pointer to c in string s is returned. Otherwise, a NULL pointer is returned.
<code>size_t strspn(const char *s1, const char *s2)</code>	Determines and returns the length of the initial segment of string s1 consisting only of characters contained in string s2 .
<code>char *strpbrk(const char *s1, const char *s2)</code>	Locates the first occurrence in string s1 of any character in string s2 . If a character from string s2 is found, a pointer to the character in string s1 is returned. Otherwise, a NULL pointer is returned.
<code>size_t strcspn(const char *s1, const char *s2)</code>	Determines and returns the length of the initial segment of string s1 consisting of characters not contained in string s2 .
<code>char *strstr(const char *s1, const char *s2)</code>	Locates the first occurrence in string s1 of string s2 . If the string is found, a pointer to the string in s1 is returned. Otherwise, a NULL pointer is returned.



```
1  // Fig. 16.31: fig16_31.cpp
2  // Using strchr
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <cstring>
9
10 int main()
11 {
12     const char *string1 = "A zoo has many animals "
13                           "including zebras";
14     int c = 'z';
15
16     cout << "The remainder of string1 beginning with the\n"
17           << "last occurrence of character '"
18           << static_cast< char >( c )
19           << "' is: \"" << strchr( string1, c ) << "\"' << endl;
20     return 0;
21 }
```

The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"



```

1  // Fig. 16.32: fig16 32.cpp
2  // Using strspn
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <cstring>
9
10 int main()
11 {
12     const char *string1 = "The value is 3.14159";
13     const char *string2 = "aehils Tuv";
14
15     cout << "string1 = " << string1
16         << "\nstring2 = " << string2
17         << "\n\nThe length of the initial segment of string1\n"
18         << "containing only characters from string2 = "
19         << strspn( string1, string2 ) << endl;
20     return 0;
21 }

```

```

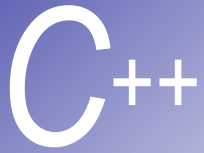
string1 = The value is 3.14159
string2 = aehils Tuv

```

```

The length of the initial segment of string1
containing only characters from string2 = 13

```



Karakter İşleme Kütüphanesinin Bellek Fonskisyonları

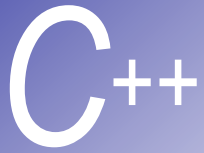
- Bellek Fonskisyonları
 - **<cstdlib>** de yer alır
- İşleme , karşılaştırma ve bellekte blok arama
- Bellekteki bütün veri blokları işlenebilir
 - Bloklara birer karakter dizileri gibi davranılır



Karakter İşleme Kütüphanesinin Bellek Fonskiyonları

"Object" bir blok dataya karşılık gelir

Prototype	Description
<code>void *memcpy(void *s1, const void *s2, size_t n)</code>	Copies n characters from the object pointed to by s2 into the object pointed to by s1 . A pointer to the resulting object is returned.
<code>void *memmove(void *s1, const void *s2, size_t n)</code>	Copies n characters from the object pointed to by s2 into the object pointed to by s1 . The copy is performed as if the characters are first copied from the object pointed to by s2 into a temporary array, and then copied from the temporary array into the object pointed to by s1 . A pointer to the resulting object is returned.
<code>int memcmp(const void *s1, const void *s2, size_t n)</code>	Compares the first n characters of the objects pointed to by s1 and s2 . The function returns 0 , less than 0 , or greater than 0 if s1 is equal to, less than or greater than s2 , respectively.
<code>void *memchr(const void *s, int c, size_t n)</code>	Locates the first occurrence of c (converted to unsigned char) in the first n characters of the object pointed to by s . If c is found, a pointer to c in the object is returned. Otherwise, 0 is returned.
<code>void *memset(void *s, int c, size_t n)</code>	Copies c (converted to unsigned char) into the first n characters of the object pointed to by s . A pointer to the result is returned.



Karakter İşleme Kütüphanesinin Diğer bir Fonksiyonu

- `strerror`
 - In `<cstdlib>`
 - Hata numarasını al ve mesaj string' i yarat

```
#include <iostream>
#include <cstring>
```

```
int main()
{
    cout << strerror( 2 ) << endl;
    return 0;
}
```

```
No such file or directory
```