BİL-142 Bilgisayar Programlama II (C/C++)

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Giriş
- **`const`** Nesneler ve **`const`** Üye Fonksiyonlar
- Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı
- **`friend`** Fonksiyonlar ve **`friend`** Class'lar
- **`this`** Pointer Kullanımı
- **`new`** ve **`delete`** ile Dinamik Hafıza Yönetimi
- **`static`** Class Üyeleri

Giriş

- **`const`** nesneler ve **`const`** fonksiyonlar değişiklik yapmayı engellemek için kullanılır.
- **`this`** pointer'ı bir sınıfın **`static`** olmayan üylerine erişimi sağlar.
- **`new`** ve **`delete`** operatörleri dinamik hafıza yönetimi sağlar.

3

Konular

- Giriş
- **`const`** Nesneler ve **`const`** Üye Fonksiyonlar
- Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı
- **`friend`** Fonksiyonlar ve **`friend`** Class'lar
- **`this`** Pointer Kullanımı
- **`new`** ve **`delete`** ile Dinamik Hafıza Yönetimi
- **`static`** Class Üyeleri

`const` Nesneler ve `const` Üye Fonksiyonlar

- Bazı nesneler değiştirilmemeye ihtiyaç duyar.
- **`const`** ile bir nesnenin değişmemesi garanti edilir.

```
const Time noon( 12, 0, 0 );
```

- Bir nesne sabit tanımlandıktan sonra değiştirilemez.
- C++ derleyicisi **`const`** bir elemanın bir fonksiyondan **`const`** tanımlanmadan çağırılmasına izin vermez.

5

`const` Nesneler ve `const` Üye Fonksiyonlar

```
1  // Fig. 10.1: Time.h
2  // Definition of class Time.
3  // Member functions defined in Time.cpp.
4  #ifndef TIME_H
5  #define TIME_H
6
7  class Time
8  {
9  public:
10    Time( int = 0, int = 0, int = 0 ); // default constructor
11
12    // set functions
13    void setTime( int, int, int ); // set time
14    void setHour( int ); // set hour
15    void setMinute( int ); // set minute
16    void setSecond( int ); // set second
17
18    // get functions (normally declared const)
19    int getHour() const; // return hour
20    int getMinute() const; // return minute
21    int getSecond() const; // return second
22
23    // print functions (normally declared const)
24    void printUniversal() const; // print universal time
25    void printStandard(); // print standard time (should be const)
26 private:
27    int hour; // 0 - 23 (24-hour clock format)
28    int minute; // 0 - 59
29    int second; // 0 - 59
30 }; // end class Time
31
32 #endif
```

6

const Nesneler ve const Üye Fonksiyonlar

```
1  // Fig. 10.2: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  using std::cout;
5
6  #include <iomanip>
7  using std::setfill;
8  using std::setw;
9
10 #include "Time.h" // include definition of class Time
11
12 // constructor function to initialize private data;
13 // calls member function setTime to set variables;
14 // default values are 0 (see class definition)
15 Time::Time( int hour, int minute, int second )
16 {
17    setTime( hour, minute, second );
18 } // end Time constructor
19
20 // set hour, minute and second values
21 void Time::setTime( int hour, int minute, int second )
22 {
23    setHour( hour );
24    setMinute( minute );
25    setSecond( second );
26 } // end function setTime
27
```

7

const Nesneler ve const Üye Fonksiyonlar

```
28 // set hour value
29 void Time::setHour( int h )
30 {
31    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
32 } // end function setHour
33
34 // set minute value
35 void Time::setMinute( int m )
36 {
37    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
38 } // end function setMinute
39
40 // set second value
41 void Time::setSecond( int s )
42 {
43    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
44 } // end function setSecond
45
46 // return hour value
47 int Time::getHour() const // get functions should be const
48 {
49    return hour;
50 } // end function getHour
51
52 // return minute value
53 int Time::getMinute() const
54 {
55    return minute;
56 } // end function getMinute
57
```

8

const Nesneler ve const Üye Fonksiyonlar

```
58  // return second value
59  int Time::getSecond() const
60  {
61     return second;
62  } // end function getSecond
63
64  // print Time in universal-time format (HH:MM:SS)
65  void Time::printUniversal() const
66  {
67     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
68        << setw( 2 ) << minute << ":" << setw( 2 ) << second;
69  } // end function printUniversal
70
71  // print Time in standard-time format (HH:MM:SS AM or PM)
72  void Time::printStandard() // note lack of const declaration
73  {
74     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
75        << ":" << setfill( '0' ) << setw( 2 ) << minute
76        << ":" << setw( 2 ) << second << ( hour < 12 ? " AM" : " PM" );
77  } // end function printStandard
78
```

9

const Nesneler ve const Üye Fonksiyonlar

```
1  // Fig. 10.3: fig10_03.cpp
2  // Attempting to access a const object with non-const member functions.
3  #include "Time.h" // include Time class definition
4
5  int main()
6  {
7     Time wakeUp( 6, 45, 0 ); // non-constant object
8     const Time noon( 12, 0, 0 ); // constant object
9
10                              // OBJECT      MEMBER FUNCTION
11    wakeUp.setHour( 18 );     // non-const   non-const
12
13    noon.setHour( 12 );       // const       non-const
14
15    wakeUp.getHour();         // non-const   const
16
17    noon.getMinute();         // const       const
18    noon.printUniversal();    // const       const
19
20    noon.printStandard();     // const       non-const
21
22    system("PAUSE");
23    return 0;
24 } // end main
25
```

```
error C2662: 'Time::setHour' : cannot convert 'this' pointer from 'const Time' to 'Time &'
```

```
error C2662: 'Time::printStandard' : cannot convert 'this' pointer from 'const Time' to 'Time &'
```

10

`const` Nesneler ve `const` Üye Fonksiyonlar

- Tüm data üyeleri member initializer syntax kullanılarak başlatılabilir.
- Const data üyeleri ve referans data üyeleri member initializer kullanılarak başlatılmak zorundadır.
- Member initializer constructor parametre listeesi ile sol parantez "{" arasında yer alır.
- Member initializer listesi parametre listesinden ":" ile ayrılır ve aralarına "," konulur.
- Her data üyesi adından sonra parantez "( )" içinde başlangıç değeri atanır.
- Const data üyelerine assignment "=" ile değer atama hata verir.

11

`const` Nesneler ve `const` Üye Fonksiyonlar

```
1  // Fig. 10.4: Increment.h
2  // Definition of class Increment.
3  #ifndef INCREMENT_H
4  #define INCREMENT_H
5
6  class Increment
7  {
8  public:
9     Increment( int c = 0, int i = 1 ); // default constructor
10
11    // function addIncrement definition
12    void addIncrement()
13    {
14       count += increment;
15    } // end function addIncrement
16
17    void print() const; // prints count and increment
18 private:
19    int count;
20    const int increment; // const data member
21 }; // end class Increment
22
23 #endif
```

12

const Nesneler ve const Üye Fonksiyonlar

```
// Fig. 10.5: Increment.cpp
// Member-function definitions for class Increment demonstrate using a
// member initializer to initialize a constant of a built-in data type.
#include <iostream>
using std::cout;
using std::endl;

#include "Increment.h" // include definition of class Increment

// constructor
Increment::Increment( int c, int i )
   : count( c ), // initializer for non-const member
     increment( i ) // required initializer for const member
{
   // empty body
} // end constructor Increment

// print count and increment values
void Increment::print() const
{
   cout << "count = " << count << ", increment = " << increment << endl;
} // end function print
```


const Nesneler ve const Üye Fonksiyonlar

```
// Fig. 10.6: fig10_06.cpp
// Program to test class Increment.
#include <iostream>
using std::cout;

#include "Increment.h" // include definition of class Increment

int main()
{
   Increment value( 10, 5 );

   cout << "Before incrementing: ";
   value.print();

   for ( int j = 1; j <= 3; j++ )
   {
      value.addIncrement();
      cout << "After increment " << j << ": ";
      value.print();
   } // end for

   system("PAUSE");
   return 0;
} // end main
```

```
c:\Documents and Settings\m.ali\My Documents\Visual Stu
Before incrementing: count = 10, increment = 5
After increment 1: count = 15, increment = 5
After increment 2: count = 20, increment = 5
After increment 3: count = 25, increment = 5
Press any key to continue . . .
```

`const` Nesneler ve `const` Üye Fonksiyonlar

- `const` üyelere = ile atama compile hatası verir.

```
1  // Fig. 10.8: Increment.cpp
2  // Attempting to initialize a constant of
3  // a built-in data type with an assignment.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "Increment.h" // include definition of class Increment
9
10 // constructor; constant member 'increment' is not initialized
11 Increment::Increment( int c, int i )
12 {
13    count = c; // allowed because count is not constant
14    increment = i; // ERROR: Cannot modify a const object
15 } // end constructor Increment
16
17 // print count and increment values
18 void Increment::print() const
19 {
20    cout << "count = " << count << ", increment = " << increment << endl;
21 } // end function print
```

- `const` üyelere = ile atama compile hatası verir.

```
error C2758: 'Increment::increment' : must be initialized in constructor base/member initializer list
```

15

Konular

- Giriş
- `const` Nesneler ve `const` Üye Fonksiyonlar
- Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı
- `friend` Fonksiyonlar ve `friend` Class'lar
- `this` Pointer Kullanımı
- `new` ve `delete` ile Dinamik Hafıza Yönetimi
- `static` Class Üyeleri

Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı

- Bir nesne başka bir class içinde üye olarak kullanılabilir. Bu şekildeki kullanıma **composition** (birleştirme) denir.
- Örneğin bir **`AlarmClock`** class'ı bir **`Time`** nesnesini kullanabilir.
- **`AlarmClok`** class'ı kendi özelliklerinin yanısıra **`Time`** nesnesinin tüm özlliklerinide kullanır.

17

Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı

```
1  // Fig. 10.10: Date.h
2  // Date class definition; Member functions defined in Date.cpp
3  #ifndef DATE_H
4  #define DATE_H
5
6  class Date
7  {
8  public:
9     Date( int = 1, int = 1, int = 1900 ); // default constructor
10    void print() const; // print date in month/day/year format
11    ~Date(); // provided to confirm destruction order
12 private:
13    int month; // 1-12 (January-December)
14    int day; // 1-31 based on month
15    int year; // any year
16
17    // utility function to check if day is proper for month and year
18    int checkDay( int ) const;
19 }; // end class Date
20
21 #endif
```

18

Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı

```
1  // Fig. 10.11: Date.cpp
2  // Member-function definitions for class Date.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Date.h" // include Date class definition
8
9  // constructor confirms proper value for month; calls
10 // utility function checkDay to confirm proper value for day
11 Date::Date( int mn, int dy, int yr )
12 {
13    if ( mn > 0 && mn <= 12 ) // validate the month
14       month = mn;
15    else
16    {
17       month = 1; // invalid month set to 1
18       cout << "Invalid month (" << mn << ") set to 1.\n";
19    } // end else
20
21    year = yr; // could validate yr
22    day = checkDay( dy ); // validate the day
23
24    // output Date object to show when its constructor is called
25    cout << "Date object constructor for date ";
26    print();
27    cout << endl;
28 } // end Date constructor
29
```


Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı

```
1  // Fig. 10.12: Employee.h
2  // Employee class definition.
3  // Member functions defined in Employee.cpp.
4  #ifndef EMPLOYEE_H
5  #define EMPLOYEE_H
6
7  #include "Date.h" // include Date class definition
8
9  class Employee
10 {
11 public:
12    Employee( const char * const, const char * const,
13       const Date &, const Date & );
14    void print() const;
15    ~Employee(); // provided to confirm destruction order
16 private:
17    char firstName[ 25 ];
18    char lastName[ 25 ];
19    const Date birthDate; // composition: member object
20    const Date hireDate; // composition: member object
21 }; // end class Employee
22
23 #endif
```

Date class'ı kullanılmıştır.

Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı

```
1  // Fig. 10.13: Employee.cpp
2  // Member-function definitions for class Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // strlen and strncpy prototypes
8  using std::strlen;
9  using std::strncpy;
10
11 #include "Employee.h" // Employee class definition
12 #include "Date.h" // Date class definition
13
14 // constructor uses member initializer list to pass initializer
15 // values to constructors of member objects birthDate and hireDate
16 // [Note: This invokes the so-called "default copy constructor" which the
17 // C++ compiler provides implicitly.]
```


Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı

```
18 Employee::Employee( const char * const first, const char * const last,
19    const Date &dateOfBirth, const Date &dateOfHire )
20    : birthDate( dateOfBirth ), // initialize birthDate
21      hireDate( dateOfHire ) // initialize hireDate
22 {
23    // copy first into firstName and be sure that it fits
24    int length = strlen( first );
25    length = ( length < 25 ? length : 24 );
26    strncpy( firstName, first, length );
27    firstName[ length ] = '\0';
28
29    // copy last into lastName and be sure that it fits
30    length = strlen( last );
31    length = ( length < 25 ? length : 24 );
32    strncpy( lastName, last, length );
33    lastName[ length ] = '\0';
34
35    // output Employee object to show when constructor is called
36    cout << "Employee object constructor: "
37       << firstName << ' ' << lastName << endl;
38 } // end Employee constructor
39
```

Member initializers `dateOfBirth`, `birthDate` constructor'ına `dataOfHire` ise `hireDate` constructor'ına gönderiliyor

Bu şekilde yazılabilir

```
Employee::Employee( const string &first, const string &last,
   const Date &dateOfBirth, const Date &dateOfHire )
   : firstName( first), // initialize firstName
     lastName( last ), // initialize lastName
     birthDate( dateOfBirth ), // initialize birthDate
     hireDate( dateOfHire ) // initialize hireDate
{
   // output Employee object to show when constructor is called
   cout << "Employee object constructor: "
      << firstName << ' ' << lastName << endl;
} // end Employee constructor
```

Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı

```
40  // print Employee object
41  void Employee::print() const
42  {
43     cout << lastName << ", " << firstName << "  Hired: ";
44     hireDate.print();
45     cout << "  Birthday: ";
46     birthDate.print();
47     cout << endl;
48  } // end function print
49
50  // output Employee object to show when its destructor is called
51  Employee::~Employee()
52  {
53     cout << "Employee object destructor: "
54        << lastName << ", " << firstName << endl;
55  } // end ~Employee destructor
56
```


Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı

```
1  // Fig. 10.14: fig10_14.cpp
2  // Demonstrating composition--an object with member objects.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Employee.h" // Employee class definition
8
9  int main()
10 {
11    Date birth( 7, 24, 1949 );
12    Date hire( 3, 12, 1988 );
13    Employee manager( "Bob", "Blue", birth, hire );
14
15    cout << endl;
16    manager.print();
17
18    cout << "\nTest Date constructor with invalid values:\n";
19    Date lastDayOff( 14, 35, 1994 ); // invalid month and day
20    cout << endl;
21    return 0;
22 } // end main
```

İki tane Date nesnesi oluşturuluyor. İkiside Employee constructor'ına gönderiliyor

```
c:\Documents and Settings\m.ali\My Documents\Visual Studi
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Blue

Blue, Bob  Hired: 3/12/1988  Birthday: 7/24/1949

Test Date constructor with invalid values:
Invalid month (14) set to 1.
Invalid day (35) set to 1.
Date object constructor for date 1/1/1994

Press any key to continue . . .
```

Konular

- Giriş
- **`const`** Nesneler ve **`const`** Üye Fonksiyonlar
- Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı
- **`friend`** Fonksiyonlar ve **`friend`** Class'lar
- **`this`** Pointer Kullanımı
- **`new`** ve **`delete`** ile Dinamik Hafıza Yönetimi
- **`static`** Class Üyeleri

`friend` Fonksiyonlar ve **`friend`** Class'lar

- Bir fonksiyon veya class başka bir class'ın scope'u dışında tanımlanır ve class içindeki non-public üyelerede erişebilir.
- **`ClassOne`** içinde aşağıdaki kod yazılırsa **`ClassTwo`** **`ClassOne`**' a friend olarak tanımlanır.

```
friend class ClassTwo;
```

- Friend fonksiyonlar class içinde tanımlanmasına rağmen üye fonksiyon değildir.
- Friend tanımlaması class içinde herhangi bir yerde yapılabilir (private, public, protected)

26

`friend` Fonksiyonlar ve `friend` Class'lar

```
1  // Fig. 10.15: fig10_15.cpp
2  // Friends can access private members of a class.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  // Count class definition
8  class Count
9  {
10    friend void setX( Count &, int ); // friend declaration
11 public:
12    // constructor
13    Count()
14       : x( 0 ) // initialize x to 0
15    {
16       // empty body
17    } // end constructor Count
18
19    // output x
20    void print() const
21    {
22       cout << x << endl;
23    } // end function print
24 private:
25    int x; // data member
26 }; // end class Count
27
```


`friend` Fonksiyonlar ve `friend` Class'lar

```
28 // function setX can modify private data of Count
29 // because setX is declared as a friend of Count (line 10)
30 void setX( Count &c, int val )
31 {
32    c.x = val; // allowed because setX is a friend of Count
33 } // end function setX
34
35 int main()
36 {
37    Count counter; // create Count object
38
39    cout << "counter.x after instantiation: ";
40    counter.print();
41
42    setX( counter, 8 ); // set x using a friend function
43    cout << "counter.x after call to setX friend function: ";
44    counter.print();
45    return 0;
46 } // end main
```

```
c:\Documents and Settings\m.ali\My Documents\Visual Stud
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
Press any key to continue . . .
```

`friend` Fonksiyonlar ve `friend` Class'lar

- Friend olmayan bir fonksiyonla private üye değiştirilemez.

```
1  // Fig. 10.16: fig10_16.cpp
2  // Non-friend/non-member functions cannot access private data of a class.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  // Count class definition (note that there is no friendship declaration)
8  class Count
9  {
10 public:
11    // constructor
12    Count()
13       : x( 0 ) // initialize x to 0
14    {
15       // empty body
16    } // end constructor Count
17
18    // output x
19    void print() const
20    {
21       cout << x << endl;
22    } // end function print
23 private:
24    int x; // data member
25 }; // end class Count
26
```


`friend` Fonksiyonlar ve `friend` Class'lar

- Friend olmayan bir fonksiyonla private üye değiştirilemez.

```
27 // function cannotSetX tries to modify private data of Count,
28 // but cannot because the function is not a friend of Count
29 void cannotSetX( Count &c, int val )
30 {
31    c.x = val; // ERROR: cannot access private member in Count
32 } // end function cannotSetX
33
34 int main()
35 {
36    Count counter; // create Count object
37
38    cannotSetX( counter, 3 ); // cannotSetX is not a friend
39    return 0;
40 } // end main
```

```
'Count::x' : cannot access private member declared in class 'Count'
```

Konular

- Giriş
- **`const`** Nesneler ve **`const`** Üye Fonksiyonlar
- Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı
- **`friend`** Fonksiyonlar ve **`friend`** Class'lar
- **`this`** Pointer Kullanımı
- **`new`** ve **`delete`** ile Dinamik Hafıza Yönetimi
- **`static`** Class Üyeleri

`this` Pointer Kullanımı

- Her nesne kendi data üyelerine this pointer'ı ile ulaşabilir.

```
1  // Fig. 10.17: fig10_17.cpp
2  // Using the this pointer to refer to object members.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  class Test
8  {
9  public:
10    Test( int = 0 ); // default constructor
11    void print() const;
12 private:
13    int x;
14 }; // end class Test
15
16 // constructor
17 Test::Test( int value )
18    : x( value ) // initialize x to value
19 {
20    // empty body
21 } // end constructor Test
22
```

32

`this` Pointer Kullanımı

```
23 // print x using implicit and explicit this pointers;
24 // the parentheses around *this are required
25 void Test::print() const
26 {
27    // implicitly use the this pointer to access the member x
28    cout << "        x = " << x;
29
30    // explicitly use the this pointer and the arrow operator
31    // to access the member x
32    cout << "\n  this->x = " << this->x;
33
34    // explicitly use the dereferenced this pointer and
35    // the dot operator to access the member x
36    cout << "\n(*this).x = " << ( *this ).x << endl;
37 } // end function print
38
39 int main()
40 {
41    Test testObject( 12 ); // instantiate and initialize testObject
42
43    testObject.print();
44    return 0;
45 } // end main
```

```
c:\Documents and Settings\m.ali\My Docu
        x = 12
  this->x = 12
(*this).x = 12
Press any key to continue . . .
```

33

Konular

- Giriş
- **`const`** Nesneler ve **`const`** Üye Fonksiyonlar
- Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı
- **`friend`** Fonksiyonlar ve **`friend`** Class'lar
- **`this`** Pointer Kullanımı
- **`new`** ve **`delete`** ile Dinamik Hafıza Yönetimi
- **`static`** Class Üyeleri

`new` ve `delete` ile Dinamik Hafıza Yönetimi

- C++ **`new`** ve **`delete`** operatörleriyle dinamik hafıza yönetimine olanak sağlar.
- **`new`** operatörü dinamik olarak hafızada yer ayırır.
- **`delete`** operatörü hafızada ayrılmış yeri boşaltır ve yeni kullanımlara hazır hale getirir.
- Aşağıda **`new`** operatörü ile **`timePtr`** için hafıza uygun boyutta bir alan ayrılır.

```
Time *timePtr;
timePtr = new Time;
```

- Aşağıda ayrılan yer boşaltılır.

```
delete timePtr;
```

35

`new` ve `delete` ile Dinamik Hafıza Yönetimi

- Aşağıda nesne, dizi ve temel veri tanımları verilmiştir.

```
double *ptr = new double( 3.14159 );

Time *timePtr = new Time( 12, 45, 0 );

int *gradesArray = new int[ 10 ];

delete [] gradesArray;
```

36

Konular

- Giriş
- **`const`** Nesneler ve **`const`** Üye Fonksiyonlar
- Birleştirme: Nesnelerin Class Üyesi Olarak Kullanımı
- **`friend`** Fonksiyonlar ve **`friend`** Class'lar
- **`this`** Pointer Kullanımı
- **`new`** ve **`delete`** ile Dinamik Hafıza Yönetimi
- **`static`** Class Üyeleri

`static` Class Üyeleri

- Bir sınıfın nesnesi kendisi için tüm üyelerin bir kopyasını oluşturur.
- **`static`** data üyeleri bir sınıfın tüm nesneleri için tek kopya olur ve aynıdır.
- **`static`** üyeler class scope'a sahiptir.
- **`static`** üyeler **`public`**, **`private`** veya **`protected`** tanımlanabilir.
- **`Static`** data üyeleri sadece bir kez initialize edilebilir.

38

static Class Üyeleri

```
1  // Fig. 10.21: Employee.h
2  // Employee class definition.
3  #ifndef EMPLOYEE_H
4  #define EMPLOYEE_H
5
6  class Employee
7  {
8  public:
9     Employee( const char * const, const char * const ); // constructor
10    ~Employee(); // destructor
11    const char *getFirstName() const; // return first name
12    const char *getLastName() const; // return last name
13
14    // static member function
15    static int getCount(); // return number of objects instantiated
16 private:
17    char *firstName;
18    char *lastName;
19
20    // static data
21    static int count; // number of objects instantiated
22 }; // end class Employee
23
24 #endif
```

public static function

private static data member

39

static Class Üyeleri

```
1  // Fig. 10.22: Employee.cpp
2  // Member-function definitions for class Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // strlen and strcpy prototypes
8  using std::strlen;
9  using std::strcpy;
10
11 #include "Employee.h" // Employee class definition
12
13 // define and initialize static data member at file scope
14 int Employee::count = 0;
15
16 // define static member function that returns number of
17 // Employee objects instantiated (declared static in Employee.h)
18 int Employee::getCount()
19 {
20    return count;
21 } // end static function getCount
22
```

40

static Class Üyeleri

```
23 // constructor dynamically allocates space for first and last name and
24 // uses strcpy to copy first and last names into the object
25 Employee::Employee( const char * const first, const char * const last )
26 {
27    firstName = new char[ strlen( first ) + 1 ];
28    strcpy( firstName, first );
29
30    lastName = new char[ strlen( last ) + 1 ];
31    strcpy( lastName, last );
32
33    count++; // increment static count of employees
34
35    cout << "Employee constructor for " << firstName
36       << ' ' << lastName << " called." << endl;
37 } // end Employee constructor
38
39 // destructor deallocates dynamically allocated memory
40 Employee::~Employee()
41 {
42    cout << "~Employee() called for " << firstName
43       << ' ' << lastName << endl;
44
45    delete [] firstName; // release memory
46    delete [] lastName; // release memory
47
48    count--; // decrement static count of employees
49 } // end ~Employee destructor
50
```

firstname ve lastname için hafızada yer ayrılır

41

static Class Üyeleri

```
51 // return first name of employee
52 const char *Employee::getFirstName() const
53 {
54    // const before return type prevents client from modifying
55    // private data; client should copy returned string before
56    // destructor deletes storage to prevent undefined pointer
57    return firstName;
58 } // end function getFirstName
59
60 // return last name of employee
61 const char *Employee::getLastName() const
62 {
63    // const before return type prevents client from modifying
64    // private data; client should copy returned string before
65    // destructor deletes storage to prevent undefined pointer
66    return lastName;
67 } // end function getLastName
68
```

42

static Class Üyeleri

```
1  // Fig. 10.23: fig10_23.cpp
2  // Driver to test class Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Employee.h" // Employee class definition
8
9  int main()
10 {
11    // use class name and binary scope resolution operator to
12    // access static number function getCount
13    cout << "Number of employees before instantiation of any objects is "
14       << Employee::getCount() << endl; // use class name
15
16    // use new to dynamically create two new Employees
17    // operator new also calls the object's constructor
18    Employee *e1Ptr = new Employee( "Susan", "Baker" );
19    Employee *e2Ptr = new Employee( "Robert", "Jones" );
20
21    // call getCount on first Employee object
22    cout << "Number of employees after objects are instantiated is "
23       << e1Ptr->getCount();
24
```

static üyeler için class adı kullanılır

iki nesne oluşturulup hafızaya yerleştirilir

static Class Üyeleri

```
25    cout << "\n\nEmployee 1: "
26       << e1Ptr->getFirstName() << " " << e1Ptr->getLastName()
27       << "\nEmployee 2: "
28       << e2Ptr->getFirstName() << " " << e2Ptr->getLastName() << "\n\n";
29
30    delete e1Ptr; // deallocate memory
31    e1Ptr = 0; // disconnect pointer from free-store space
32    delete e2Ptr; // deallocate memory
33    e2Ptr = 0; // disconnect pointer from free-store space
34
35    // no objects exist, so call static member function getCount again
36    // using the class name and the binary scope resolution operator
37    cout << "Number of employees after objects are deleted is "
38       << Employee::getCount() << endl;
39
40    system("PAUSE");
41    return 0;
42 } // end main
```

```
c:\Documents and Settings\m.ali\My Documents\Visual Studio 2008\Projects\
Number of employees before instantiation of any objects is 0
Employee constructor for Susan Baker called.
Employee constructor for Robert Jones called.
Number of employees after objects are instantiated is 2

Employee 1: Susan Baker
Employee 2: Robert Jones

~Employee() called for Susan Baker
~Employee() called for Robert Jones
Number of employees after objects are deleted is 0
Press any key to continue . . .
```