

Pointers to Objects

Feza BUZLUCA
Istanbul Technical University
Computer Engineering Department
<http://faculty.itu.edu.tr/buzluca>
<http://www.buzluca.info>



This work is licensed under a Creative Commons Attribution 3.0 License.
<http://creativecommons.org/licenses/by-nc-nd/3.0/>

7.1

1

Overview

- Pointers to Objects
- Memory Management with new and delete
- Building Linked Lists of Objects
- Pointers and Inheritance
- Heterogeneous Inheritance



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

7.2

2

Pointers to Objects

- Objects are stored in memory, so pointers can point to objects just as they can to variables of basic types



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

7.3

3

Memory Management with new and delete

- new operator
- delete operator



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

7.4

4

new Operator

- The new operator allocates memory of a specific size from the operating system and returns a pointer to its starting point
 - If it is unable to find space, it returns a 0 pointer
- When you use new with objects, it does not only allocate memory for the object, it also **creates** the object in the sense of invoking the object's constructor
 - This guarantees that the object is correctly initialized, which is vital for avoiding programming errors



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

7.5

5

delete Operator

- To ensure safe and efficient use of memory, the new operator is matched by a corresponding delete operator that releases the memory back to the operating system
- If you create an array with new Type[];, you need the brackets when you delete it:

```
int *ptr = new int[10];  
:  
delete[ ] ptr;
```

Do not forget the brackets when deleting arrays of objects. Using them ensures that all the members of the array are deleted and that the destructor is called for each one. If you forget the brackets, only the first element of the array will be deleted.



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

7.6

6

Example

```
class String {
    int size;
    char *contents;
public:
    String(); // default constructor
    String( const char * ); // constructor
    String( const String & ); // copy constructor
    const String& operator=( const String & ); // assignment operator
    void print() const ;
    ~String(); // destructor
};

int main()
{
    String *sptr = new String[3]; // create 3 objects
    String s1( "String_1" ); // a String object
    String s2( "String_2" ); // another String object
    *sptr = s1; // assignment to the first element of the array
    *( sptr + 1 ) = s2; // assignment to the second element of the array
    sptr->print(); // print the first element
    ( sptr + 1 )->print(); // print the second element
    sptr[1].print(); // print the second element
    delete[] sptr; // delete objects sptr points to
    return 0;}
```

See Example e71.cpp



1999-2016 Feza BUZLUCA
http://faculty.itu.edu.tr/buzluca/

7.7

7

Linked List of Objects

- A class may contain a pointer to objects of its own type
- This pointer can be used to build a chain of objects, a linked list

```
class Teacher {
    friend class TeacherList;
    string name;
    int age, numOfStudents;
    Teacher *next; // pointer to next Teacher object
public:
    Teacher( const string &, int, int ); // constructor
    void print() const;
    const string& getName() const { return name; }
    ~Teacher(); // destructor
};

// linked list for teachers
class TeacherList {
    Teacher *head;
public:
    TeacherList(){ head=0; }
    bool append(const string &,int,int);
    bool remove(const string &);
    void print() const;
    ~TeacherList();
};
```

See Example e72.cpp



1999-2016 Feza BUZLUCA
http://faculty.itu.edu.tr/buzluca/

7.8

8

Building Linked Lists of Ready-Made Classes: Problem and Solutions

- In the previous example,
 - Teacher class must have a pointer to the next object
 - TeacherList class must be declared as a friend
- If this class is written by the same group, then it is possible to put such a pointer in the class
- However, programmers usually use ready-made classes, written by other groups (e.g., classes from libraries)
 - These classes may not have a next pointer
- To build linked lists of such ready-made classes, there are two techniques
 - Using the is-a relationship
 - Using the has-a relationship



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.9

9

Using the Is-A Relationship

- Programmer can derive a new class (TeacherForList) with a next pointer

```
class TeacherForList : public Teacher {    // TeacherForList is a Teacher with a pointer
    friend class TeacherList;
    TeacherForList *next;                  // next Teacher
    TeacherForList(const string &, int, int); // constructor
};

TeacherForList :: TeacherForList (const string &n, int a, int nos)
                        :Teacher (n, a, nos)
{
    next = 0;
}
```



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.10

10

Using the Has-A Relationship

- Another way to build linked lists of such ready-made classes is to define a **node class** that **has a** teacher object
- The node objects are connected to build the list
- Each object of the node class will store the address of an element (Teacher)

```
class TeacherNode {                                // TeacherNode has a Teacher
    friend class TeacherList;
    Teacher *element;                               // element of the list
    TeacherNode *next;                             // next node
    TeacherNode(const string &, int, int);          // constructor
    ~TeacherNode();                                 // destructor
};

TeacherNode::TeacherNode(const string & n, int a, int nos){
    element = new Teacher(n, a, nos);
    next = 0;
}

TeacherNode::~~TeacherNode(){
    delete element;
}
```

See Example e73.cpp



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.11

11

Pointers and Inheritance

- If a class Derived has a public base class Base, then a pointer to Derived can be assigned to a variable of type pointer to Base without use of explicit type conversion
- In other words, a pointer to Base can carry the address of an object of Derived
- A pointer to Base can point to objects of Derived
- **Example:**
 - A pointer to Teacher can point to objects of Teacher and to objects of Principal
 - A principal **is a** teacher, but a teacher is not always a principal
- The opposite conversion (converting pointer to Base to pointer to Derived) must be explicit



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.12

12

Pointers and Inheritance

```
class Base {  
.....  
};  
  
class Derived : public Base {  
.....  
};  
  
Derived d;  
Base *bp = &d;                // implicit conversion  
Derived *dp = bp;              // ERROR! Base is not Derived  
dp = static_cast<Derived *>(bp); // explicit conversion
```



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.13

13

Accessing Members of the Derived Class Via a Pointer To the Base Class

- When a pointer to Base class points to objects of the Derived class, only the members inherited from the Base can be accessed via this pointer
- In other words, members just defined in the Derived class cannot be accessed via a pointer to the Base class



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.14

14

Accessing Members of the Derived Class Via a Pointer To the Base Class

- For example, a pointer to Teacher can store the address of an object of the Principal type
- Using this pointer (Teacher type), it is possible to access only teacher properties of the principal (i.e., only the members that the Principal inherits from the Teacher class)
- Using a pointer to the derived type (Principal), it is possible to access, as expected, all (public) members of the Principal (both inherited from the Teacher and defined in the Principal)
 - See the example in the next slide
- We will investigate some additional issues about pointers with inheritance (such as accessing overridden functions) in the next lecture (Polymorphism)



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.15

15

Example

```
class Teacher {                                // base class
protected:
    string name;
public:
    void teachClass ();                        // Teacher's behavior (responsibility)
};
class Principal : public Teacher {             // derived class
    string schoolName;
public:
    void directSchool();                      // Principal's behavior (responsibility)
};
//----- Test Code -----
Principal objPrincipal;                       // object of Principal type
Teacher *ptrTeacher = &objPrincipal;          // pointer to Teacher points to Principal
ptrTeacher->teachClass();                      // OK. Teaching is a Teacher behavior
ptrTeacher->directSchool();                   // ERROR! Directing a school is not a
                                              // Teacher behavior. It is not inherited
                                              // from the Teacher
Principal *ptrPrincipal = &objPrincipal;       // Pointer to Principal points to Principal
ptrPrincipal ->teachClass();                   // OK. Principal is a Teacher
ptrPrincipal ->directSchool();                 // OK. Principals behavior
```



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.16

16

Pointers and Inheritance

- If the class Base is a **private** base of Derived, then the implicit conversion of a Derived* to Base* would not be done
 - Because, in this case, a public member of Base can be accessed through a pointer to Base, but not through a pointer to Derived
- Example:**

```
class Base {
    int m1;
public:
    int m2; // m2 is a public member of Base
};

class Derived : private Base { // m2 is not a public member of Derived
    :
};

Derived d;
d.m2 = 5; // ERROR! m2 is private member of Derived
Base *bp = &d; // ERROR! private base
bp = reinterpret_cast<Base*>(&d); // ok: explicit conversion AVOID!
bp->m2 = 5; // ok, but dangerous AVOID!
```



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

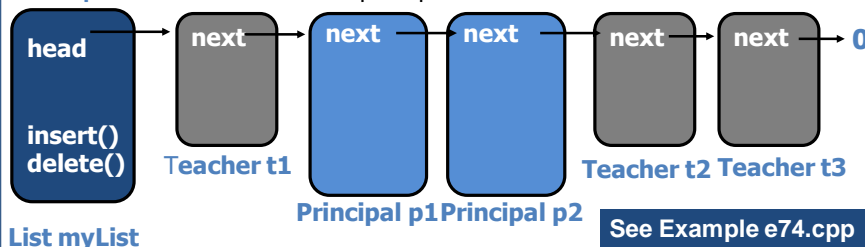
6.17

17

Heterogeneous Linked Lists

- Using inheritance and pointers, we can create **heterogeneous** linked lists
- A list specified in terms of pointers to a base class can store objects of any class derived from this base class
- We will discuss heterogeneous lists again, after we have learned **polymorphism**

Example: A list of teachers and principals



- There is a **list** class in the C++ Standard Template Library (STL)
- You do not need to write classes to build linked lists
- You can use the list class of the library



1999-2016 Feza BUZLUCA
<http://faculty.itu.edu.tr/buzluca/>

6.18

18