Lecture 7

Pointers to Objects

Topics

- Pointers to Objects
- Linked List of Objects
- Pointers and Inheritance

Example : Pointer to an object

- Pointers can point to class objects, similar to variables of basic data types.

```
#include <iostream>
using namespace std;

class Customer {
public:
  int       ID;
  string   name;
  Customer () {};                 // Default constructor
  Customer (int id, string n)  // Parametered constructor
     : ID (id), name (n) {};
};
```

```
int main() {
  Customer  * ptr;                          // Declaration of pointer
  ptr = new  Customer (111, "ABCD");  // Dynamic allocation

  cout << ptr->ID << "  "
       << ptr->name  << endl;
}
```


The new Operator

- Dynamically (at run-time) allocates memory of a specific byte size and returns a pointer to its memory address.
- If it is unable to find memory space, it returns a NULL pointer.
- When the **new** operator is used with objects, it also invokes the object's **constructor**.

- Suppose the String class is a user-written class.
  (Not the built-in std::string class.)

```
String *  sp;          // Define a pointer to a String class object.
sp  =  new  String;    // Dynamically allocate a String object.
```

The delete Operator

- To ensure efficient use of computer memory, every new operator should have a corresponding delete operator that releases the memory.

```
String *  p =  new  String;  // Allocate one string
p = "Test";
......
delete  p;   // Remove the string
```

- To delete an array entirely, the brackets [ ] should be written.

```
// Allocate an array of 10 strings.
// Pointer is pointing to array.

String *  sp;
sp  =  new  String [10];
......
delete [ ]  sp;
```

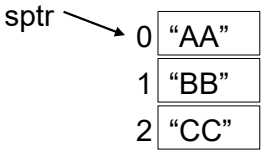

Example : String class

```
class  String
{
  int     size;
  char  *contents;

 public:
  String ();                    // Default constructor
  String (const  char *);       // Parametered constructor
  String (const  String &);     // Copy constructor

  // Overloaded assignment operator
  const  String&   operator=  (const  String  &);

  void print() ;

  ~String();      // Destructor
};
```

Example : Using a Pointer as an Array of String objects

```
int main () {
   // Define String objects
   String  s1 ("AA");
   String  s2 ("BB");
   String  s3 ("CC");

   // Dynamically allocate array of String objects
   String  * sptr;
   sptr   =   new  String [3];

   // Copy objects to array elements
   sptr [0]  =  s1;
   sptr [1]  =  s2;
   sptr [2]  =  s3;

   // Call print function of each element
   for (int  i=0;  i<3;  i++)
       sptr [i] . print();

   // Delete objects pointed by sptr
   delete []  sptr;
}
```

sptr is name of pointer, and also name of array.



Dynamically constructing String class objects

Method1 : Use constructors in assignments.

```
sptr   =   new  String [3];
sptr [0]   =  String ("AA");
sptr [1]   =  String ("BB");
sptr [2]   =  String ("CC");
```

Method2 : Use constructors in block initializer.

```
sptr   =   new  String [3]   {  String ("AA"),  String ("BB"),  String ("CC")  };
```

Method3 : Use short notation constructors in block initializer.

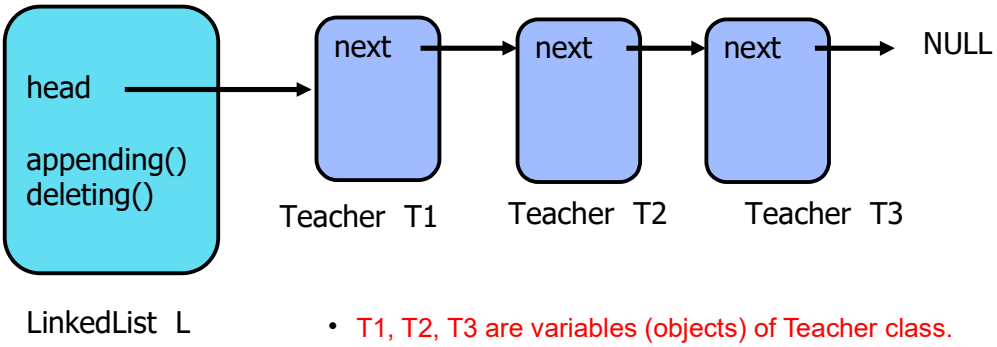
```
sptr   =   new  String [3]    {"AA",  "BB",  "CC"  };
```

Topics

- Pointers to Objects
- Linked List of Objects
- Pointers and Inheritance

Example: Linked List of teachers

- The linked list data structure below contains objects of Teacher class.
- Each node in linked list contains a pointer to next Teacher.
- Also a friend class (named LinkedList) of Teacher is defined.
- The LinkedList class contains the **head pointer.**
- Head pointer points to the first Teacher in the linked list.
- Adding or deleting nodes are done by member functions of LinkedList class.



- T1, T2, T3 are variables (objects) of Teacher class.
- L is a variable of LinkedList class.

Example : Teacher class and LinkedList class

- Teacher class contains a pointer to next Teacher.
- The next pointer is used to build a chain of objects, a linked list.

```
class   Teacher
{
   friend  class  LinkedList;

   string  name;
   int      numOfStudents;
   Teacher *  next;
    // Pointer to next object of teacher

 public:
   Teacher (string, int); // Constructor
   void print();
  ~Teacher()    // Destructor
};
```

```
// Linked List for teachers
class   LinkedList
{
  Teacher *   head;

 public:
  LinkedList ()   // Constructor
    { head = NULL; }

  bool  appending (Teacher);
  bool  deleting (Teacher);
  void  printall ();
  ~LinkedList ();   // Destructor
};
```


The **printall function** of LinkedList class iteratively travels from a node to next node (by looping), and calls the print function of Teacher class.

```
// Prints all elements of list on screen
void  LinkedList  ::  printall ()
{
  Teacher  * tempPtr;   // Temporary pointer

  if (head == NULL)
  {
     cout << "The list is empty \n";
     return;
  }

  // Travel through linked list
  tempPtr = head;
  while ( tempPtr != NULL)
  {
      tempPtr -> print(); //Function in Teacher
      tempPtr  =  tempPtr -> next;
   }
}
```

```
int main()
{
  Teacher   T1  ("AA", 100);
  Teacher   T2  ("BB", 150);
  Teacher   T3  ("CC", 80);

  LinkedList   L;
  L . appending (T1);
  L . appending (T2);
  L . appending (T3);

  L . printall();
}
```

Alternative function call :
Use the Teacher constructor.

```
L . appending (Teacher ("AA", 100) );
```



```
// Destructor
// Deletes all elements of the linked list one-by-one

LinkedList  ::  ~LinkedList ()
{
  Teacher  *  tempPtr;  // Temporary pointer

  // Travel through linked list
  while ( head != NULL )   // Check if the list is not empty
  {
     tempPtr = head;
     head = head -> next;  // Go to next element
     delete  tempPtr;
  }
}
```

Extending the existing classes

- In previous example, the Teacher class must have a pointer to the "next" object, and the LinkedList class must be declared as a friend.
- Usually programmers use **ready-made classes**, written by other programmers, such as classes from libraries. And those classes may not have a **next** pointer.
- To build linked lists of such ready-made classes (without a next pointer), there are two techniques.
  - Inheritance (is-a)
  - Composition (has-a)
- The example class below has **no next pointer** as a member variable.

```
class   Teacher
{
   string   name;
   int      numOfStudents;
 public:
   Teacher (string, int);
   void  print();
  ~Teacher()
};
```


Example1: Inheritance from Teacher

- Programmer can derive a new class (**TeacherForList**) from Teacher class.
- TeacherForList class contains a **next pointer,** to build the linked list.

```
// TeacherForList  is-a  Teacher
class TeacherForList : public Teacher
{
   friend  class  LinkedList;

   // Pointer to next TeacherForList
   TeacherForList * next;

   TeacherForList (string, int); // Constructor
};
```

```
// Constructor
TeacherForList :: TeacherForList (string  n,  int  nos)
                                  : Teacher (n, nos)
{
    next = NULL;
}
```

```
int main()
{
  TeacherForList   T1 ("AA", 100);
  TeacherForList   T2 ("BB", 150);
  TeacherForList   T3 ("CC", 80);
  LinkedList   L;
  L . appending (T1);
  L . appending (T2);
  L . appending (T3);
  L . printall();
}
```


Example2: Composition from Teacher

- Another way to a build linked list of ready classes is to define a **node class.**
- Each object of the node class **has-a** pointer to a Teacher object (element).
- Node also contains a **next pointer,** to build the linked list.

```
// TeacherNode  has-a  Teacher
class  TeacherNode
{
   friend  class  LinkedList;

   Teacher * element;   // The element of the linked list  (Composition)

   TeacherNode * next;  // Pointer to next node

   TeacherNode (string, int);   // Constructor
   ~TeacherNode ();             // Destructor
};
```

```
// TeacherNode  Constructor
TeacherNode  ::  TeacherNode  (string  n, int  nos)
{
   element  =  new  Teacher (n, nos);   // Teacher constructor
   next  =  NULL;
}
```

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



```
int main()
{
  TeacherNode   T1  ("AA", 100);
  TeacherNode   T2  ("BB", 150);
  TeacherNode   T3  ("CC", 80);

  LinkedList   L;
  L . appending (T1);
  L . appending (T2);
  L . appending (T3);
  L . printall();
}
```

Topics

- Pointers to Objects
- Linked List of Objects
- Pointers and Inheritance

Pointers and Inheritance

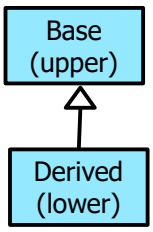
- If a Derived class has a public Base class, then a pointer to Derived can be assigned to a pointer to Base, without use of explicit type conversion (up-casting).
- In other words, a **pointer to base class** can carry the address of a **derived object**.
- For example, a pointer to Teacher can point to objects of Teacher and also to objects of Principal.
- A principal **is-a** teacher, but a teacher is not always a principal.
- The opposite conversion, from (pointer-to-Base) to (pointer-to-Derived), must be explicit (down-casting).

Example : Pointer up-casting and down-casting

- **Up-casting:** Pointer conversion from derived class to base class.
- **Down-casting:** Pointer conversion from base class to derived class.

```
class  Base
{ };
```

```
class  Derived  :  public  Base
{ };
```




```
int main()
{
  Derived   d;
  Base   * bp;
  bp  =  &d;   // Implicit conversion from derived to base
               //  (up-casting)
  Derived  * dp;
  dp  =   bp;    // Compiler error.
                 // Base can not be directly assigned to Derived.
  dp   =   static_cast < Derived * >   ( bp );
 // OK. Explicit conversion from base to derived  (down-casting)
}
```

Accessing members of Derived class, via a pointer to Base class

- When a pointer to Base class points to objects of Derived class, only the members inherited from Base can be accessed.
- The extra members (defined in Derived class) can not be accessed via a pointer to Base class.
- **Example:** A pointer to Teacher can hold the address of a Principal object.
- Using that pointer (Teacher type) it is possible to access only teacher properties of principal, i.e. only the members that the Principal inherits from Teacher class.
- Using a pointer to derived type (Principal) it is possible to access all (public) members of Principal (both inherited from Teacher and also defined in Principal).

Example : Using pointer to base class

```
// Base class
class  Teacher
{
  protected:
    string    name;
    int       numOfStudents;
  public:
    void  teachClass ();    // Teacher's  function
};
```

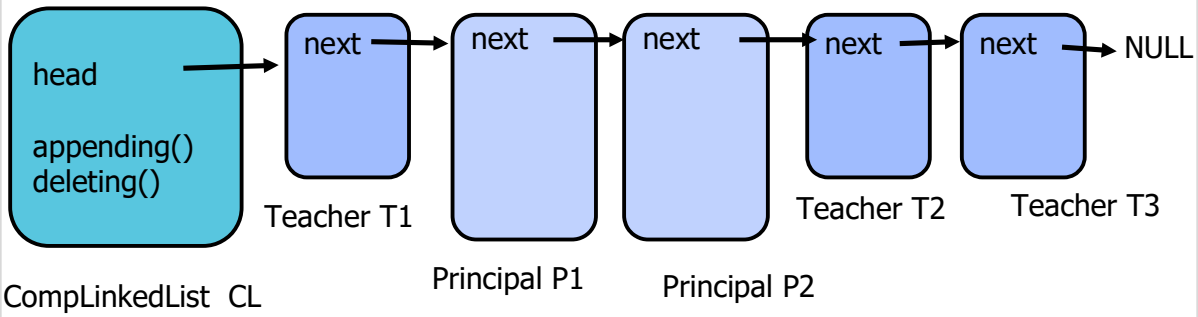
```
// Derived class
class  Principal  :  public  Teacher
{
     string    school_name;
   public:
      void  directSchool ();    // Principal's function
};
```

```
int main()
{
  Principal   objPrincipal;       // Object of Principal type

  Teacher   * ptrTeacher;      // Pointer to base class

  ptrTeacher   =   & objPrincipal; // Pointer to Teacher(base) points to Principal (derived)

  ptrTeacher -> teachClass ();     // OK. Teaching is a teacher-function.

  ptrTeacher -> directSchool ();  // Compiler error.
                                  // Directing a school is not a teacher-function.
  //-------------------------------------------------------------------------------------
  Principal   * ptrPrincipal;              // Pointer to derived class

  ptrPrincipal  =  & objPrincipal;       // Pointer to Principal

  ptrPrincipal -> teachClass ();         // OK. Principal is a Teacher also

  ptrPrincipal -> directSchool ();       // OK. Principal's function
}
```


Compound Linked List

- A linked list specified in terms of pointers to a base class can hold objects of any class derived from this base class.
- Using inheritance and pointers, a compound linked list can be built.
- The nodes in a compound linked list can be Teacher and Principal objects.
- Compound linked lists can be used in **polymorphism**.

Example: A compound linked list of teachers and principals.



- T1, T2, T3 are variables (objects) of Teacher class.
- P1, P2 are variables (objects) of Principal class.
- CL is a variable of CompLinkedList class.

```
// Compound Linked List
// for teachers and principals

class  CompLinkedList
{
  TeacherNode * head; // Base pointer

 public:
  CompLinkedList ()   // Constructor
    { head = NULL; }

  bool  appending (TeacherNode *);
  bool  deleting (TeacherNode *);
  void  printall ();
  ~CompLinkedList ();  // Destructor
};
```



```
int main()
{
  TeacherNode   T1  ("AA", 100);
  TeacherNode   T2  ("BB", 150);
  TeacherNode   T3  ("CC", 80);
  PrincipalNode   P1  ("DD", 300, "QQQ");
  PrincipalNode   P2  ("EE",  200, "VVV");

  CompLinkedList   CL;  // Compound Linked List

  // Pass addresses to the append function
  CL . appending ( & T1 );  // Teacher1
  CL . appending ( & P1 );  // Principal1
  CL . appending ( & P2 );  // Principal2
  CL . appending ( & T2 );  // Teacher2
  CL . appending ( & T3 );  // Teacher3

  CL . printall();
}
```