7 Object Pointers

Pointers to Objects

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

The 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, in returns a 0 pointer.

►When you use new with objects, it does not only allocates 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.

Pointers to Objects

The 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;
```

Don't 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.

Example

```
class String {
   int size;
   char *contents;
 public:
   String();
   String(const char *);
   String(const String &);
   const String& operator=(const String &);
   void print() const ;
   ~String();
};
```

```
int main()  {
    String *sptr = new String[3];
    String s1("String_1");
    String s2("String_2");
    *sptr = s1;
    *(sptr + 1) = s2;
    sptr->print();
    (sptr+1)->print();
    sptr[1].print();
    delete[] sptr;
    return 0;
}
```

Linked List of Objects

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


```
class Teacher {
    friend class Teacher_list;
    string name;
    int age, numOfStudents;
    Teacher * next;
  public:
    Teacher(const string &, int, int);
    void print() const;
    const string& getName() const {
        return name;    }
   ~Teacher()
};
```

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

Linked List of Objects

- In the previous example the Teacher class must have a pointer to the next object and the list class must be declared as a friend, to enable users of this class building linked lists.
- If this class is written by the same group then it is possible to put such a pointer in the class.
- But usually programmers use ready classes, written by other groups, for example classes from libraries.
- These classes may not have a next pointer.
- To build linked lists of such ready classes the programmer have to define a node class.
- Each object of the node class will hold the addresses of an element of the list.

Linked List of Objects

```
class Teacher_node{
    friend class Teacher_list;
    Teacher * element;                  // The element of the list
    Teacher_node * next;                // next node
    Teacher_node(const string &, int, int);     // constructor
    ~Teacher_node();                    // destructor
};
Teacher_node::Teacher_node(const string & n, int a, int nos){
    element = new Teacher(n, a, nos);
    next = 0;
}
Teacher_node::~Teacher_node(){
   delete element;
}
```

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. A pointer to Base can carry the address of an object of Derived.
- The opposite conversion, for pointer to Base to pointer to Derived, must be explicit.
- For 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.

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
```

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.

Pointers and Inheritance


```
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 = static_cast<Base*>(&d);      // ok: explicit conversion
bp->m2 = 5;      // ok
```

Heterogeneous Linked Lists

- Using the inheritance and pointers, heterogeneous linked lists can be created.
- A list specified in terms of pointers to a base class can hold objects of any class derived from this base class.
- We will discuss heterogeneous lists again, after we have learnt polymorphism.

Example: A list of teachers and principals