Date:....................................

ITEM 13: Use objects to manage resources

```
void f()
  {    Investmen *pInv = createInvest();
          ...
          delete pINV;
  }
```

Regardless of how delete was skipped, we'd leak not only the memory containing investment object but also any resources held by that object.

We can avoid this by putting resources inside objects.

Eg → auto_ptr is a smart pointer whose destructor calls delete on what it points to.

```
void f()
  {
     auto_ptr <Investment> pInv(createInvest());

  }
```

→ Resources are acquired & immediately turned over to resource managing objects.

called as Resource Aquisition Is Initializatn (RAII)

Date:..................................

Problem with auto-ptr.

① auto-ptr pointing to an object should not be more than once, since it will be deleted more than once

② Unusual characteristic of auto-ptr. copying them (copy ctor/ ~~ass~~ cop copy assignment operator) sets them to null.

```
auto-ptr <Investment> pInv1 (create Invest());

① auto-ptr <Investment) pINV2 (pInv1);
        //pINV1 is null now in both cases
② pInv1 = pInv2;
```

I Reference counting Smart pointers.

Keeps track of how many objects pt to a particular resource & automatically deletes the resource when nobody is pointing to it any longer.

Date:..............................

Since STL containers require normal copy this technique is useful.

```
Void f()
  {
     tr1::shared ptr <Investment>
           pInv1(creatInvestment());

     tr1::shared-ptr <Investment> pInv2(pInv1);
   pInv1 = pInv2;
```

//pInv1 & pInv2 are destroyed & object they point to is automatically deleted.

BUT:

Both auto-ptr & tr1::shared-ptr use delete in their destructors, not delete[].

Dynamically allocated arrays is a bad idea.

```
   auto-ptr <string> aps( new string[10]);
tr1::shared-ptr <int> spi( new int[10]);
  //will lead to undefined behaviour.
```

Date:..................................

We can use boost::scoped_array and &boost::shared-array can be used if you still want to use auto_ptr shared ptr.