Date:

Item 26 → Postpone variable definitions as long as possible.

→ There is a cost associated with unused variables, so we should avoid them.

→ But we may think that we never create unused variable.

// Below function defines variable "encrypted" very soon.

```
string encryptPassword (const std string & password)
{
    using namespace std;
    string encrypted; // encrypted is unused if exception is thrown!
    if (password.length() < MinimumLength)
        throw error("Password short");
    :
    return encrypted;
}
```

another way to use.

```
{ if (password.length() < MinLength)
        throw error("    ");
  string encrypted.
  :
  return encrypted;
}
```

Date:.....

→ Above code is also not right, bcoz encrypted is defined without any initialization arguments That means its default ctor will be used and then we will assign it.

→ Item 4 explains why default constructing an object and then assigning to it is less efficient.

→ void encrypt (string &s);
String ---.
{ String encrypted;
encrypted = password;
encrypt (encrypted);
return encrypted;
}

// Best way to initialize encrypted.

{
String encrypted (password); // define & // initialized via copy ctor.
encrypt (encrypted);
return encrypted;
}

Date:.....

If a variable is used only inside a loop is it better to define it outside the loop or make an assignment to it on each loop's iteration.

Approach A: define outside loop

```
Widget w;
for(int i=0; i<n; i++)
{
    w = some value of i;
    ...
}
```

Approach B: define inside loop

```
for(int i=0; i<n; i++)
{
    Widget w(some value of i);
    ...
}
```

Approach A: 1 ctor + 1 dtor + n assignments

Approach B: n ctor + n dtor

→ Where an assignment costs less than ctor-dtor pair, Approach A is more efficient.

→ Also approach A make name w visible in a larger scope than Approach B, that is contrary to program comprehensibility & maintainability.

Date:

- when assignment is less expensive than ctor-dtor pair & we are dealing with performance sensitive part of code, default to using approach B.