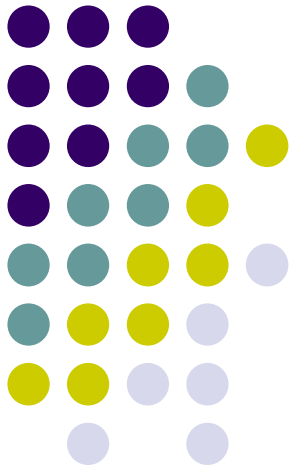
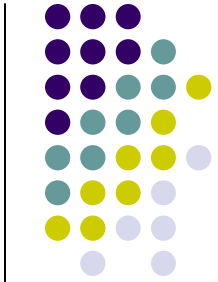
Types and Passing Arguments



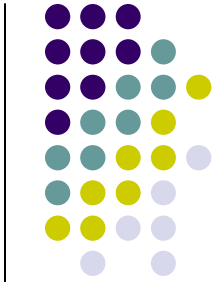
Value Types



If a variable is a value type, then that variable contains the actual data of that type.

Normally a value type contains a single piece of information, such as an integer, a float, a double, a bool, or a char.

Reference Types



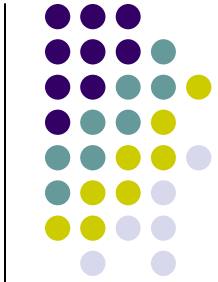
A reference type, on the other hand, contains the address of the location where the relevant data of that type is stored.

An example of a built-in reference type is string.

string MyStr = “Hello world”;

MyStr now contains an address, and that address is the location in memory where the literal “Hello World” is stored.

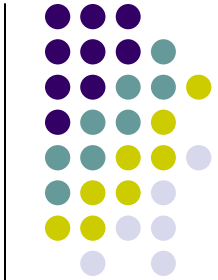
Pass-by-value



Consider the method definition:

```
static double MyAbs(double Y)
{
    if(Y>=0) return Y;
    else
        return –Y;
}
```

There is one parameter in the header. It’s name is Y.

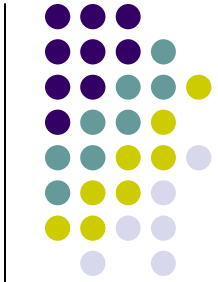


To call the function MyAbs, there needs to be one argument in the call.

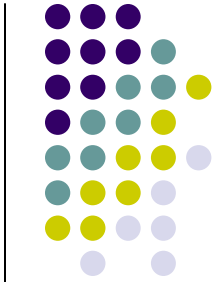
Z = MyAbs(Q);

Q is the argument in the call.

When this call is made, the *value* of Q is copied into the value of Y, the parameter in the function. So, in effect there are two variables, one in the calling program, and the other in MyAbs that contain the same information.



In the previous call, Q was copied to Y. In this case the method MyAbs does not alter Y; however, even if it did, Q would remain unchanged, since only a copy of its value was sent to MyAbs.

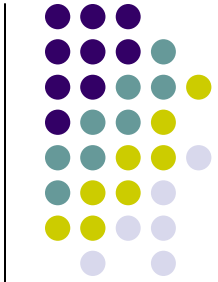


```
static void Mystery(string Z){

    do something with Z

}
```

In the Main method consider:

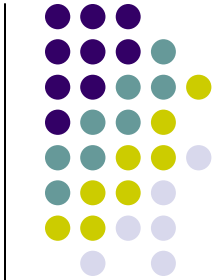


string Y;

Y = “The moon is made of green cheese”;

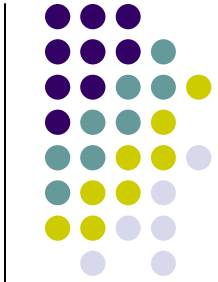
At this point Y contains an address, and that is the address where the string is stored. Note that the string is read only.

If the following call is made:



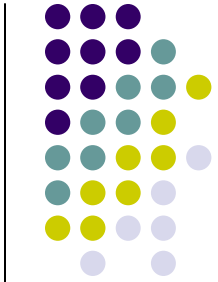
Mystery(Y);

Then the content of Y, namely an address, is copied and becomes the content of Z. Thus, Y in the calling program, and Z in the Mystery Method contain the same address. In this way, information about the string is sent to the method, without actually copying the string it in its entirety. Only its address was copied.



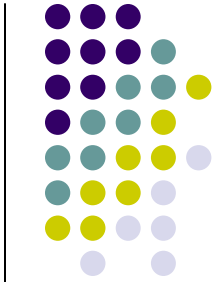
In the previous example, a reference type was passed. It's address was passed by value; hence, the string itself was passed by reference. The string still resides in only one place in memory. Its address is stored in both Z and Y.

Remark about pass-by-value



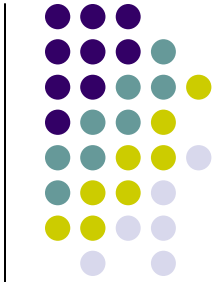
When a variable is passed by value to a method parameter, changes to that parameter inside the method, leave the corresponding variable in the calling program unchanged.

Jeopardy



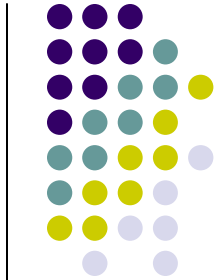
Answer:

You pass the variable by reference rather than value.



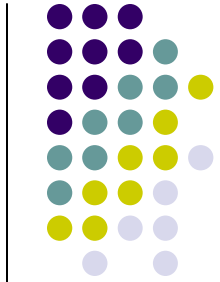
Questions:

1. What do you do if you want changes to a parameter in the method to result in changes to the corresponding argument?
2. What do you do if you want a method to have access to an argument's data, but you don't want to pass the entire set of data?



```
static void MySwap(ref int A, ref int B)
   {
        int Temp = A;
        A = B;
        B = Temp;
   }
```

By placing the key word ref in front of the variable declarations in the header, A and B will now “reference” the corresponding arguments in a call to this function.



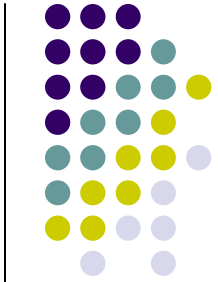
```
int C=5;
int D=6;

MySwap(ref C, ref D);
```

Note: C and D are value types, which are being “sent” as arguments to reference variables in the parameter list. Thus, in the call, we also use the keyword reference.

We will see a run of this in class and see what happens if the ref is removed from either or both of C and D in the call.

Built-in value and reference types



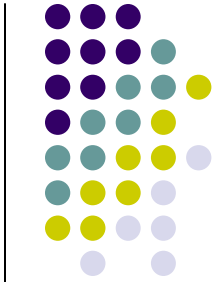
Value Types

- Integral: sbyte, byte char, short, ushort,int uint,long, and ulong.
- Floating: float and double.
- Others: decimal and bool

Reference Types

- string and object

Summary of types



Look at the table in Figure 6.7 on page 196.