`struct` Copy Operation

In C, variables are copied in three situations:

- when used as the right side of an assignment operation
- when used as a parameter in a function call
- when used as the return value from a function

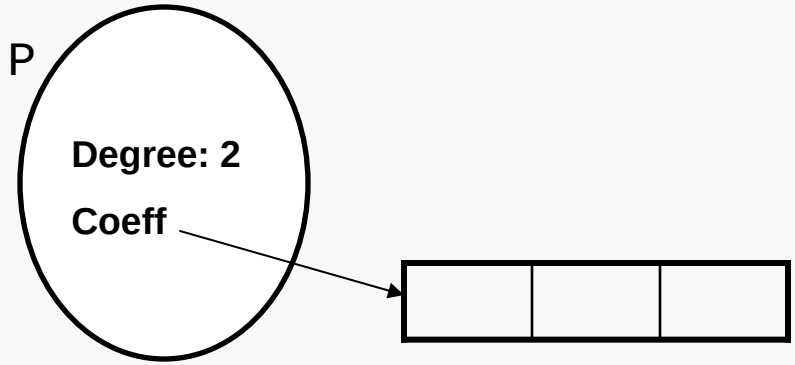
```
struct LocationType {
   int X, Y;
};
typedef struct LocationType Location;
. . .
Location A;
A.X = 1;
A.Y = 5;

Location B;
B = A;                  // members of A are copied into the
                        // corresponding members of B
```

In most cases, the default copy mechanism for `struct` types is adequate.

`struct` with Deep Content

A `struct` type may have "deep" content:

```
struct PolynomialType {
   unsigned int Degree;
   int*         Coeff;   // dynamically-allocated array
};

typedef struct PolynomialType Polynomial;
. . .
Polynomial P;

P.Degree = 2;
P.Coeff = malloc(3 * sizeof(int));
```



Note:

`Coeff` is a member of `P`, but…

… the array is NOT a member.

`struct` Variable Initialization


```
bool initPoly(Polynomial* const P, const uint8_t D, const int64_t* const C) {

   if ( P == NULL || C == NULL ) return false;

   P->Degree = D;
   P->Coeff = malloc((P->Degree + 1) * sizeof(int64_t));

   if ( P->Coeff == NULL ) {
      P->Degree = 0;
      return false;
   }

   for (uint8_t i = 0; i <= P->Degree; i++) {
      P->Coeff[i] = C[i];
   }

   return true;
}
```

Guard against NULL pointer errors

Allocate array to hold coefficients

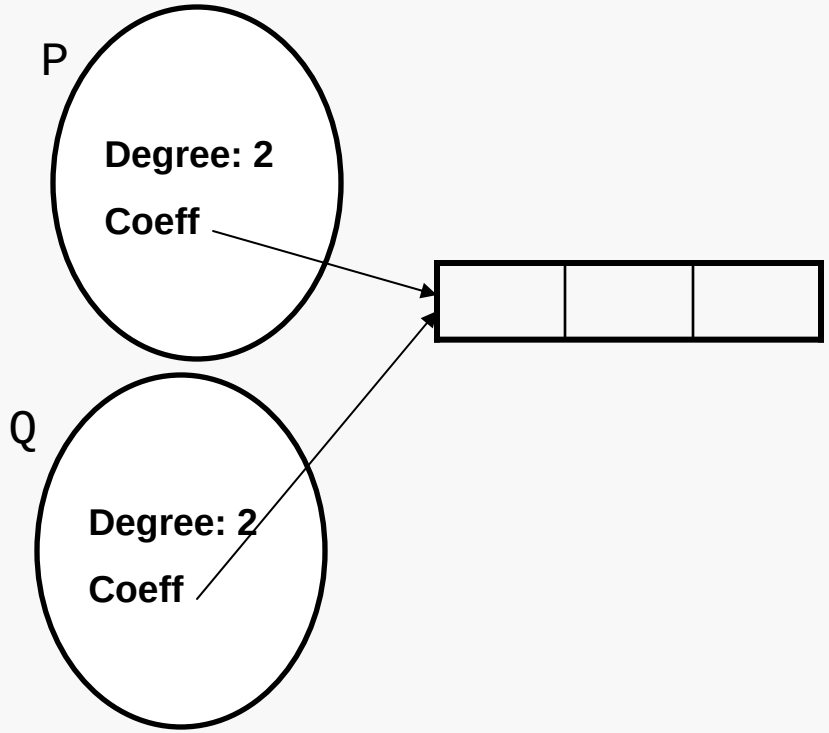
Check for allocation failure

Copy coefficients

Copying a `struct` with Deep Content

Copying a `struct` variable that has "deep" content may have unintended consequences:

```
Polynomial P, Q;

P.Degree = 2;
P.Coeff = malloc(3 * sizeof(int));

Q = P;
```

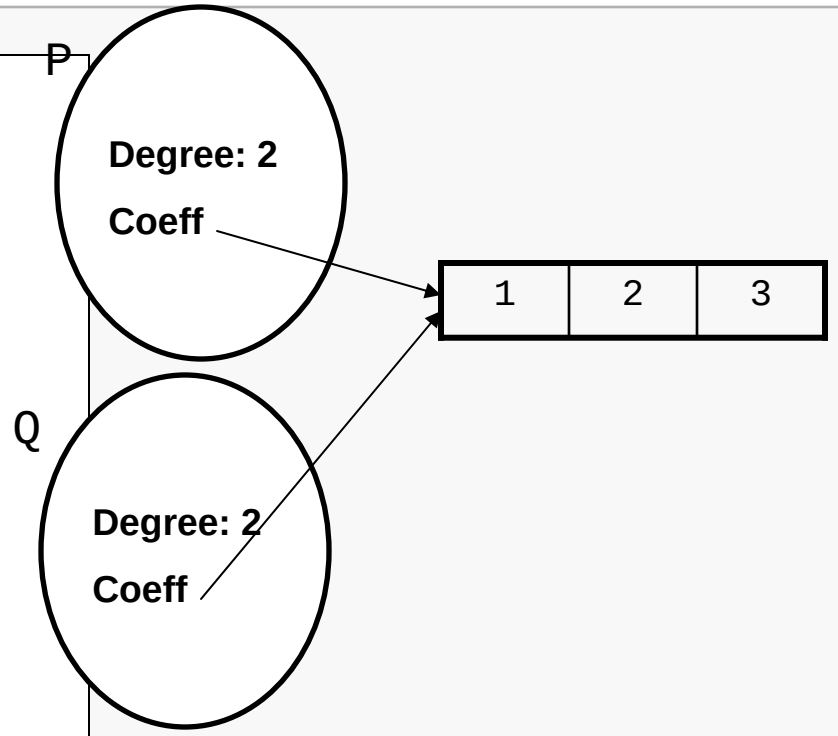


When the value of `P.Coeff` is copied into `Q.Coeff`, we get an effect of sharing that is rarely desirable…

… this is known as the *deep copy problem* (or the *shallow copy problem*).

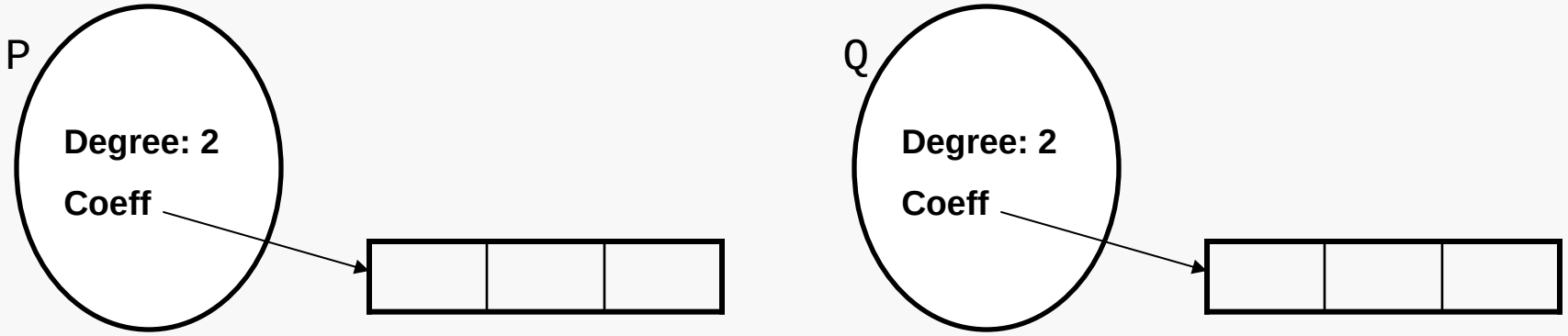
What's Wrong with a Shallow Copy?

```
Polynomial P, Q;

P.Degree = 2;
P.Coeff = malloc(3 * sizeof(int));

P->Coeff[0] = 1;
P->Coeff[1] = 2;
P->Coeff[2] = 3;

Q = P;

P->Coeff[2] = 5;  // "back-door"
                  //   change to Q

free(Q->Coeff);   // P loses its
                  //   coefficients
```



Making a Deep Copy

The usual semantics of assignment would lead you to expect we'd have gotten:



But this is NOT what happens by default.

To make this happen, you must explicitly (i.e., via code you write):

- copy the "shallow" content from `P` into `Q`
- allocate new memory for the "deep" content to be copied from `P` into `Q`
- copy the "deep" content (e.g., the coefficient values in the array)

Implementing a Deep Copy

In C we handle deep copying by implementing an appropriate function:

```
/**
 * Initializes *Target from *Source as described below.
 *
 * Pre:  Target != NULL,
 *       Source != NULL,
 *       Source->C[i] initialized for i = 0:Source->Degree
 * Post: Target->Degree == Source->Degree
 *       Target->Coeff != Source->Coeff
 *       Target->Coeff[i] == Source->Coeff[i] for i = 0:Source->Degree
 *
 * Returns: false if *Target cannot be properly initialized, true otherwise
 */
bool copyPoly(Polynomial* const Target, const Polynomial* const Source) {

   . . .
}
```

Implementing a Deep Copy

The basic steps are relatively straightforward:

```
bool copyPoly(Polynomial* const Target, const Polynomial* const Source) {

   if ( Source == NULL || Source->Coeff == NULL || Target == NULL )
      return false;

   free(Target->Coeff);


   Target->Coeff = malloc( (Source->Degree) * sizeof(int64_t) );

   if ( Target->Coeff == NULL ) {
      Target->Degree = 0;
      return false;
   }

   Target->Degree = Source->Degree;
   for (int i = 0; i <= Source->Degree; i++) {
      Target->Coeff[i] = Source->Coeff[i];
   }
   return true;
}
```

Guard against NULL pointer errors

Deallocate old array in Target, if any

Allocate new array for Target

Copy Source into Target

Could this be simplified by calling `initPoly()`? If so, how?