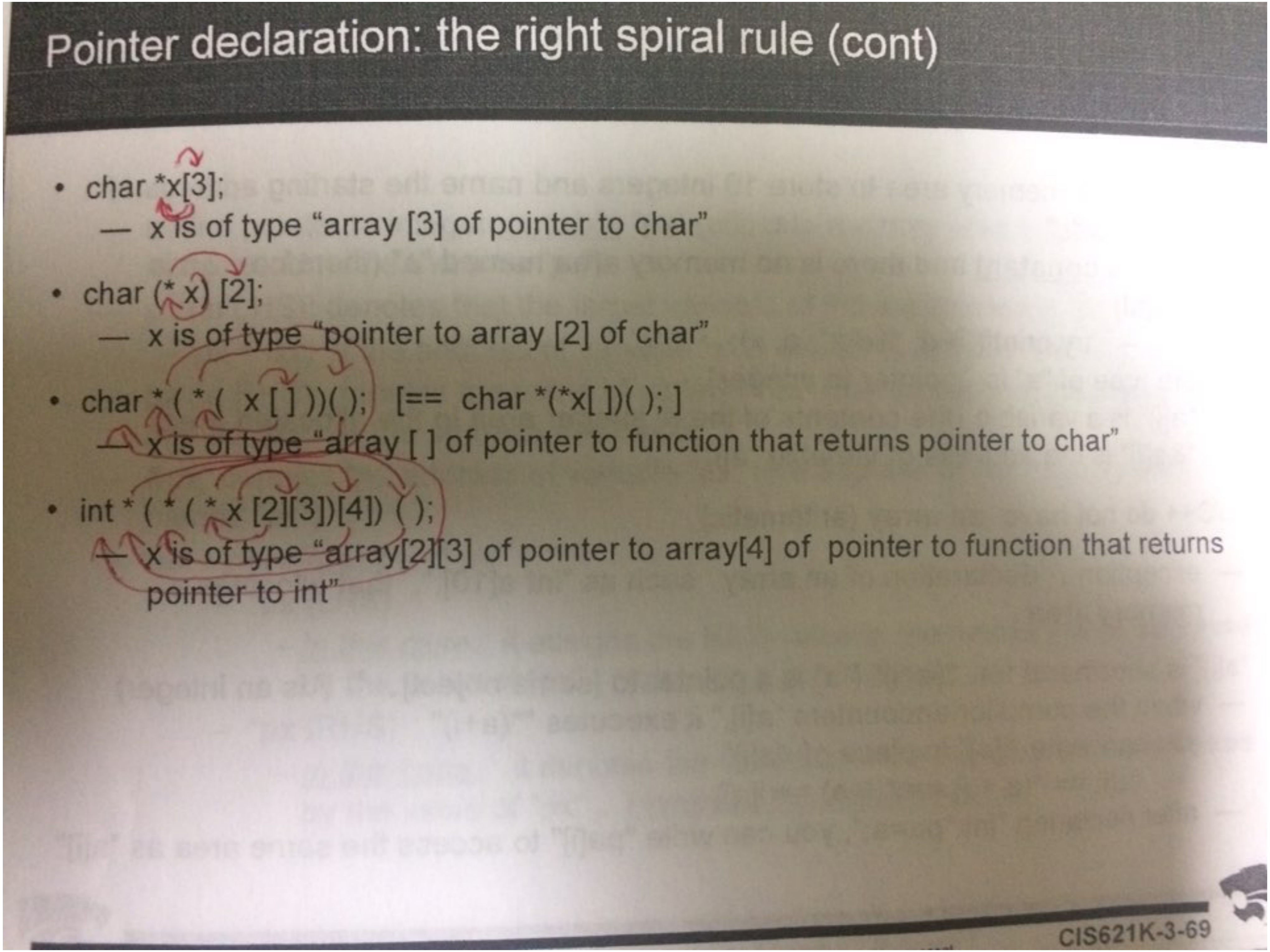


Pointer declaration: the right spiral rule (cont)

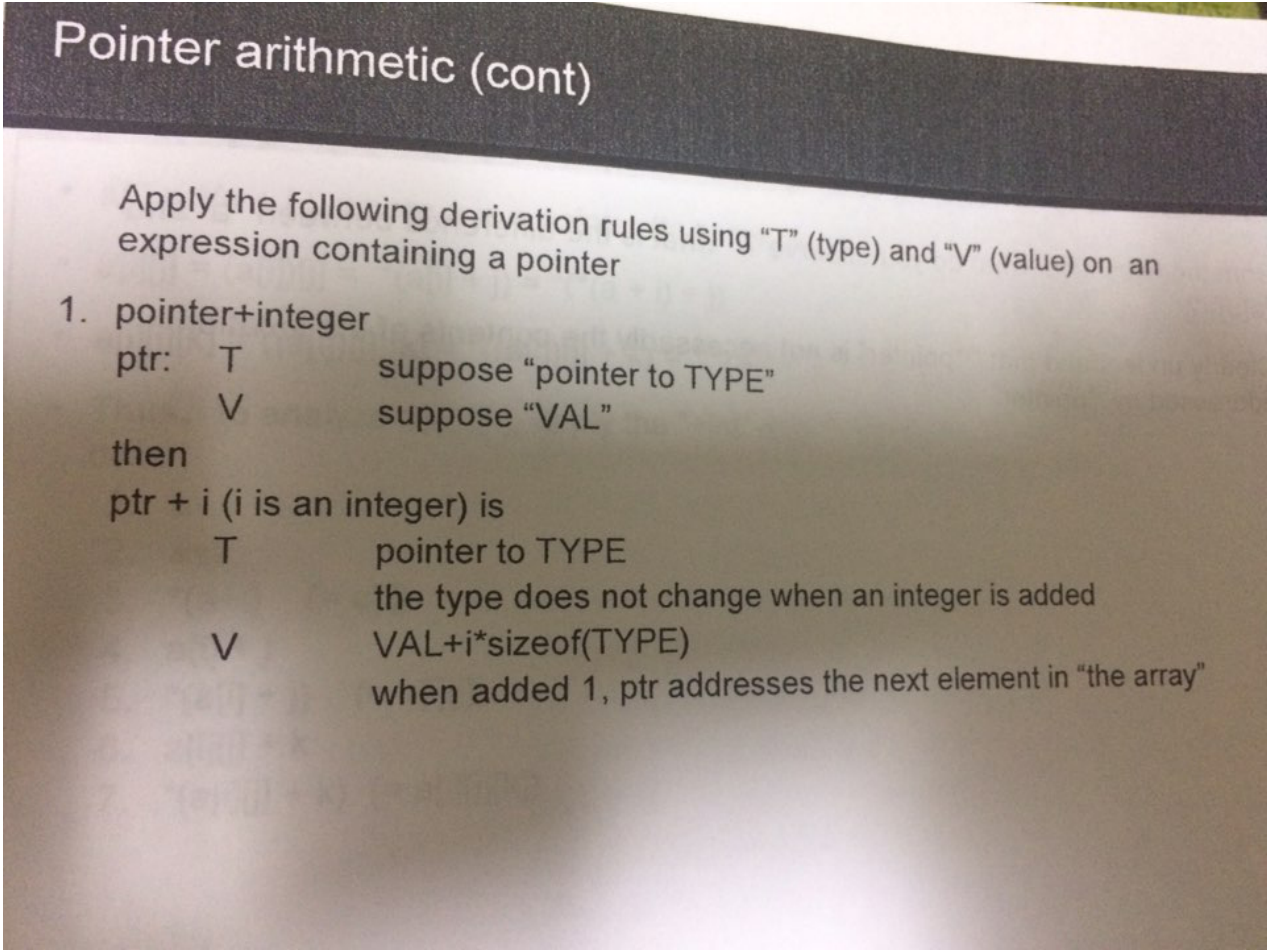
- char *x[3];
  - — x is of type “array [3] of pointer to char”
- char (* x) [2];
  - — x is of type “pointer to array [2] of char”
- char * (* (x [ ]))( ); [== char *(*x[ ])( );]
  - — x is of type “array [ ] of pointer to function that returns pointer to char”
- int * (* (* x [2][3])[4]) ( );
  - — x is of type “array[2][3] of pointer to array[4] of pointer to function that returns pointer to int”

Pointer arithmetic

- Consider the following type declaration

```
TYPE *ptr;      // "ptr" is a pointer to an instance of TYPE ("ptr" is a variable)
TYPE a[10];     //  "a" is a pointer to an instance of TYPE  ("a" a constant)
```

- Only two types of operations may be applied to pointer variables/constants
  - pointer + integer (similarly, pointer – pointer)
  - *pointer   // dereferencing
- Type "array[i][j][k]...[z]" is equivalent to "pointer to array[j][k]....[z]"
  - since there is no "array arithmetic in C", when the type under analysis is [i][j][k]…[z], it must be converted to "pointer to array[j][k]…[z]" <u>(RULE1)</u>



Pointer arithmetic (cont)

Apply the following derivation rules using "T" (type) and "V" (value) on an expression containing a pointer

1. pointer+integer

ptr: T suppose "pointer to TYPE"

V suppose "VAL"

then

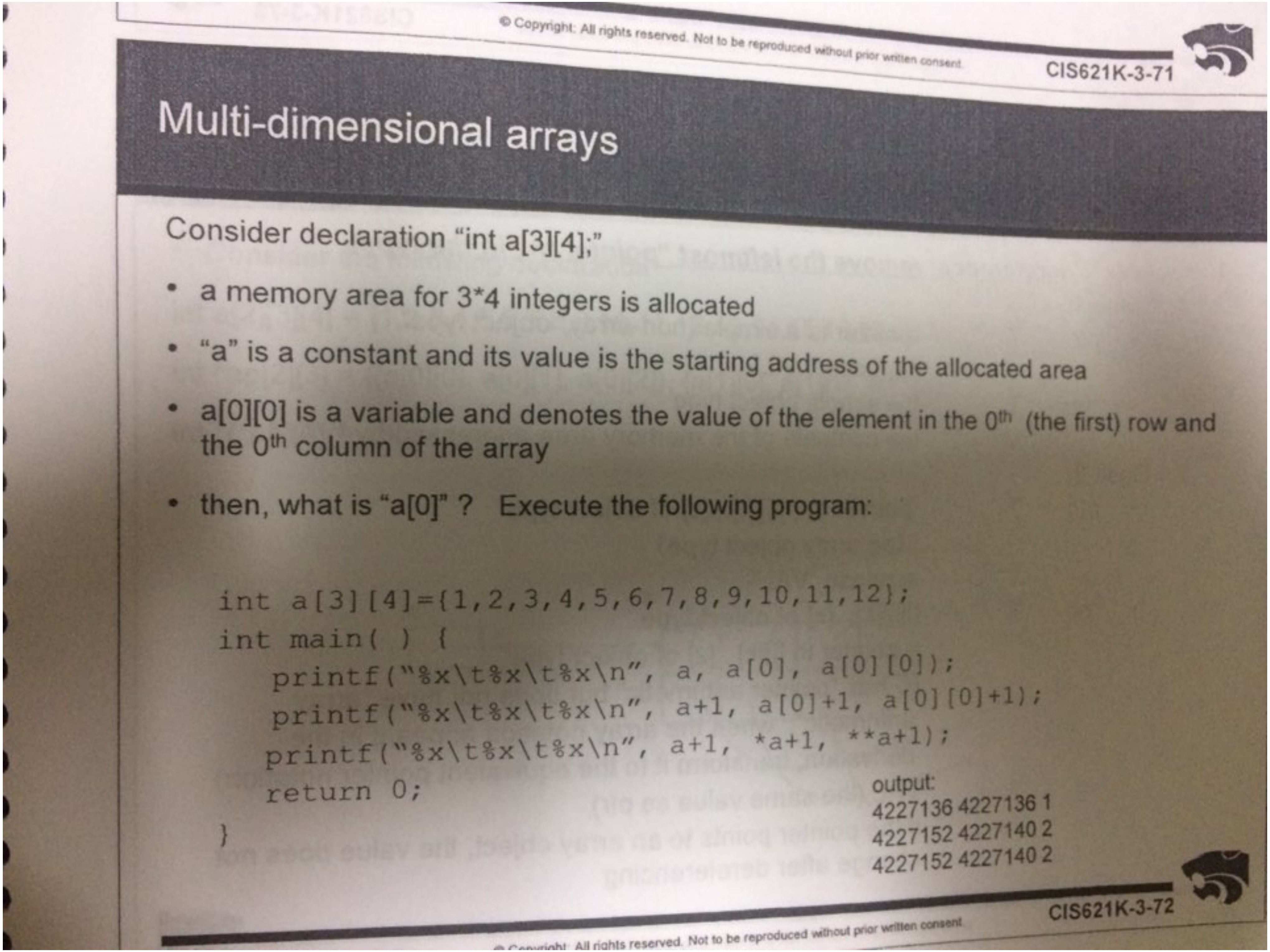
ptr + i (i is an integer) is

T pointer to TYPE

the type does not change when an integer is added

V VAL+i*sizeof(TYPE)

when added 1, ptr addresses the next element in "the array"



Multi-dimensional arrays

Consider declaration “int a[3][4];”

- a memory area for 3*4 integers is allocated
- “a” is a constant and its value is the starting address of the allocated area
- a[0][0] is a variable and denotes the value of the element in the 0th (the first) row and the 0th column of the array
- then, what is “a[0]” ? Execute the following program:

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
int main( ) {
   printf(“%x\t%x\t%x\n”, a, a[0], a[0][0]);
   printf(“%x\t%x\t%x\n”, a+1, a[0]+1, a[0][0]+1);
   printf(“%x\t%x\t%x\n”, a+1, *a+1, **a+1);
   return 0;
}
```

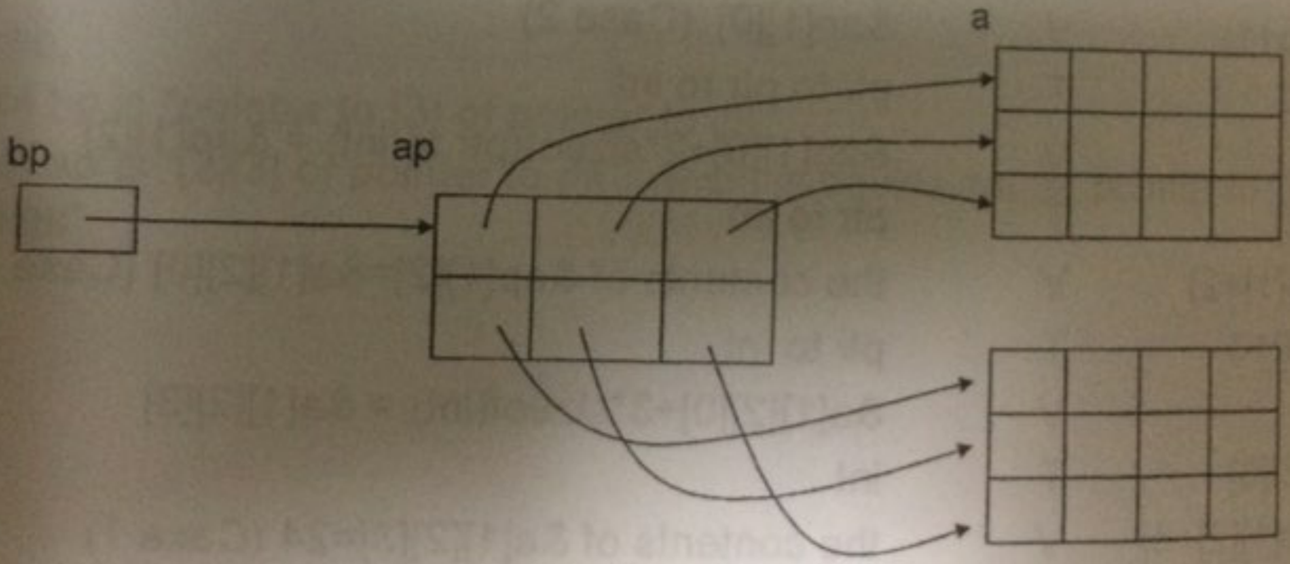
output:

```
4227136 4227136 1
4227152 4227140 2
4227152 4227140 2
```

Exercise on pointer arithmetic

- Consider the following declaration

```
int a[2][3][4] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24};
int *ap[2][3] = {{a[0][0], a[0][1], a[0][2]}, {a[1][0], a[1][1], a[1][2]}};
int *(*bp)[3] = ap;
```



Exercise on pointer arithmetic (cont)

- Derivation of a[1][2][3]

a:	T	[2][3][4] of int=ptr to [3][4] of int
	V	&a[0][0][0] (since "a" is an array name, it denotes the starting address of the array area)
a+1:	T	ptr to [3][4] of int
	V	&a[0][0][0]+1*sizeof([3][4] of int)=&a[1][0][0]
*(a+1)=a[1]	T	[3][4] of int = ptr to [4] of int
	V	&a[1][0][0] (Case 2)
a[1]+2	T	ptr to [4] of int
	V	&a[1][0][0]+2*sizeof([4] of int)=&a[1][2][0]
*(a[1]+2)=a[1][2]	T	[4] of int = ptr to int
	V	&a[1][2][0] (Case 2)
a[1][2]+3	T	ptr to int
	V	&a[1][2][0]+3*sizeof(int)=&a[1][2][3]
a[1][2][3]	T	int
	V	the contents of &a[1][2][3] =24 (Case 1)

Exercise on pointer arithmetic (cont)

- derivation of ap[1][2][3]

ap	T	[2][3] of ptr to int = ptr to [3] of ptr to int
	V	&ap[0][0]
ap+1	T	ptr to [3] of ptr to int
	V	&ap[0][0]+1*sizeof([3] of ptr to int) = &ap[1][0]
ap[1]	T	[3] of ptr to int = ptr to ptr to int
= *(ap+1)	V	&ap[1][0] (Case 2)
ap[1]+2	T	ptr to ptr to int
	V	&ap[1][0]+2*sizeof(ptr to int) = &ap[1][2]
ap[1][2]	T	ptr to int
= *(ap[1]+2)	V	the contents of &ap[1][2]=&a[1][2][0] (Case 1)
ap[1][2]+3	T	ptr to int
	V	&a[1][2][0]+3*sizeof(int) = &a[1][2][3]
ap[1][2][3]	T	int
= *(ap[1][2]+3)	V	the contents of &a[1][2][3]=24 (Case 1)

pointer arithmetic (cont)

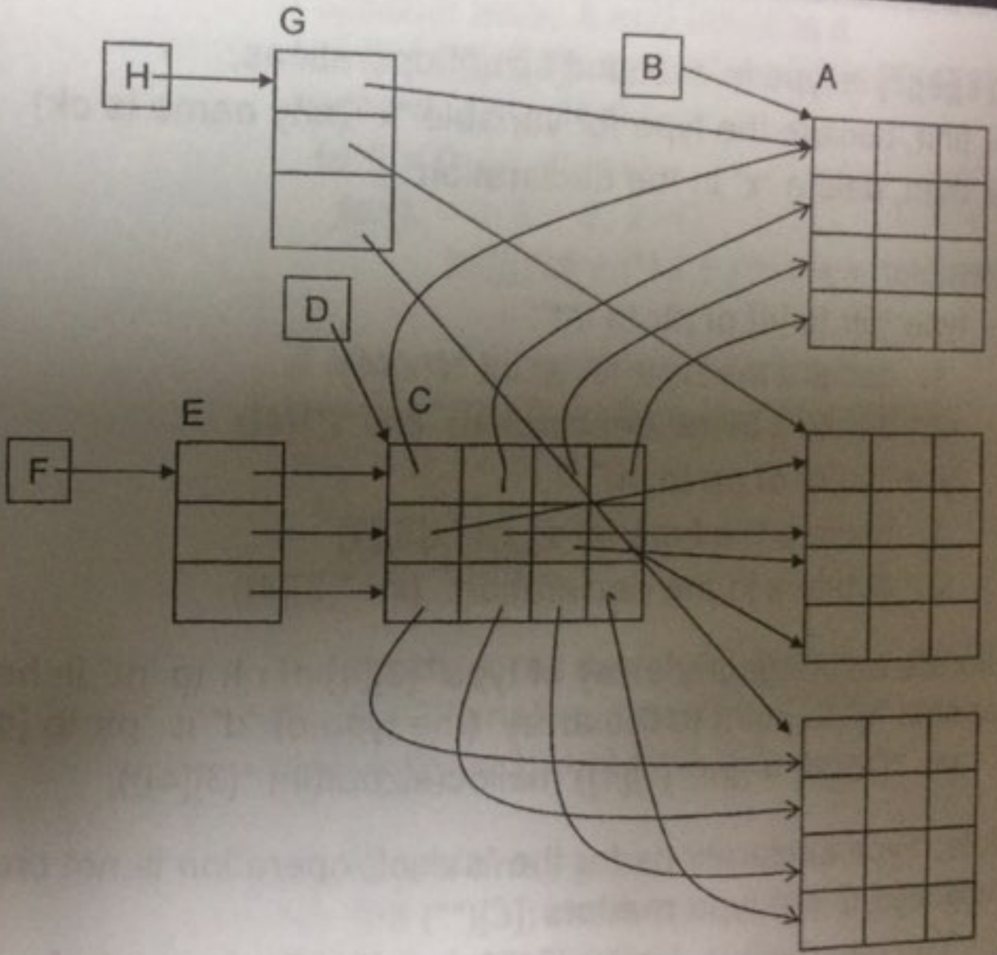
- Derivation of bp[1][2][3]

bp	T	ptr to [3] of ptr to int
	V	&ap[0][0] (since bp is a simple variable, its value is the contents of the variable)
bp+1	T	ptr to [3] of ptr to int
	V	&ap[0][0]+1*sizeof([3] of ptr to int)=&a[1][0]
bp[1]	T	[3] of ptr to int = ptr to ptr to int
= *(bp+1)	V	&ap[1][0] (Case 2)
bp[1]+2	T	ptr to ptr to int
	V	&ap[1][0]+2*sizeof(ptr to int) = &ap[1][2]
bp[1][2]	T	ptr to int
= *(bp[1]+2)	V	the contents of &ap[1][2] =&a[1][2][0] (Case 1)
bp[1][2]+3	T	ptr to int
	V	&a[1][2][0]+3*sizeof(int) = &a[1][2][3]
bp[1][2][3]	T	int
= *(bp[1][2]+3)	V	the contents of &a[1][2][3] =24 (Case 1)

Exercise on pointer declaration

Declare the pointer variables so that the following relation holds

A[i][j][k] ==
B[i][j][k] ==
C[i][j][k] ==
D[i][j][k] ==
E[i][j][k] ==
F[i][j][k] ==
G[i][j][k] ==
H[i][j][k]



Allocate purple colored arrays in heap and declare pointer variables correctly so that the following relation holds

B[i][j][k] ==
D[i][j][k] ==
F[i][j][k] ==
H[i][j][k]