ECE 220: Computer Systems & Programming

Lecture 9: Pointers and Arrays

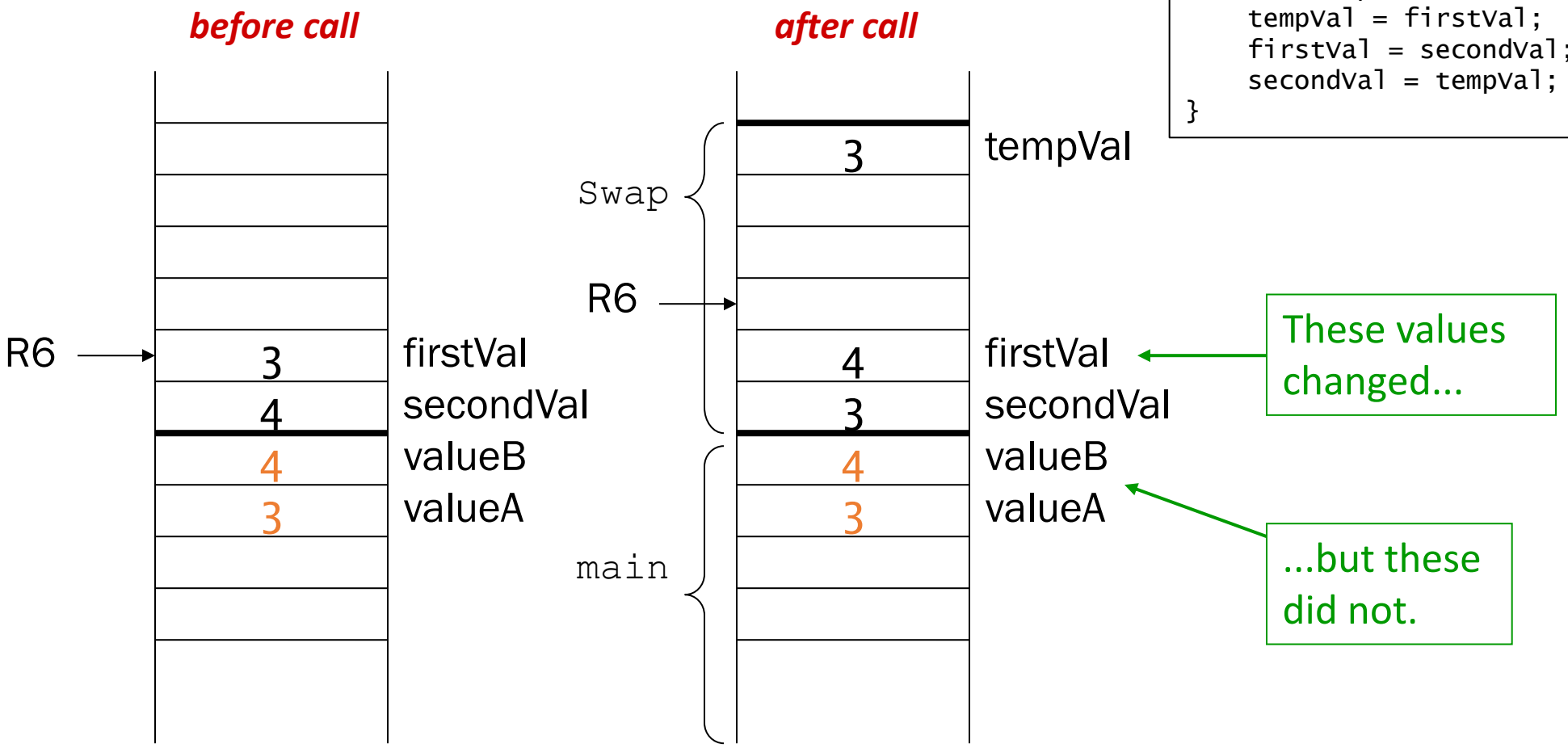
Swap Function

```
void Swap(int firstVal, int secondVal);
int main()
{
    int valueA = 3;
    int valueB = 4;

    Swap(valueA, valueB);
}

void Swap(int firstVal, int secondVal)
{
    int tempVal;

    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
}
```

Goal:
Swap `valueA` and `valueB` in main.

Swap Function – Activation Record

```
int main(){
    int valueA = 3;
    int valueB = 4;
    Swap(valueA, valueB);
}
void Swap(int firstVal, int secondVal){
    int tempVal;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
}
```



- Swap needs *addresses* of variables (valueA and valueB) outside its own activation record. Pointers solve this problem!

Solution:

```
void NewSwap(int *firstVal, int *secondVal);

int main()
{
    int valueA = 3;
    int valueB = 4;

    NewSwap(&valueA, &valueB);
}

void NewSwap(int *firstVal, int *secondVal)
{
    int tempVal;

    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
```

Pointers in C

Declaration

```
     int *ptr;   /* ptr is a pointer to an int */
or   int* ptr;
```

A pointer in C is always a pointer to a particular data type:
`int*, double*, char*, etc.`

'*' used in different purposes

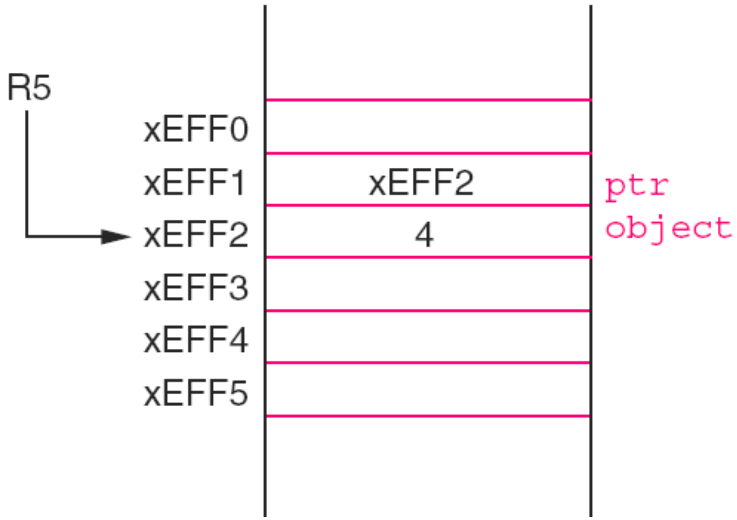
Operators

`*ptr` → dereference operator: returns the value pointed to by ptr. It also allows us to manipulate the value pointed by ptr

`&val` → address operator: returns the address of variable **val**

Pointers in LC3

```
int object;
int *ptr;

object = 4;
ptr = &object;
```

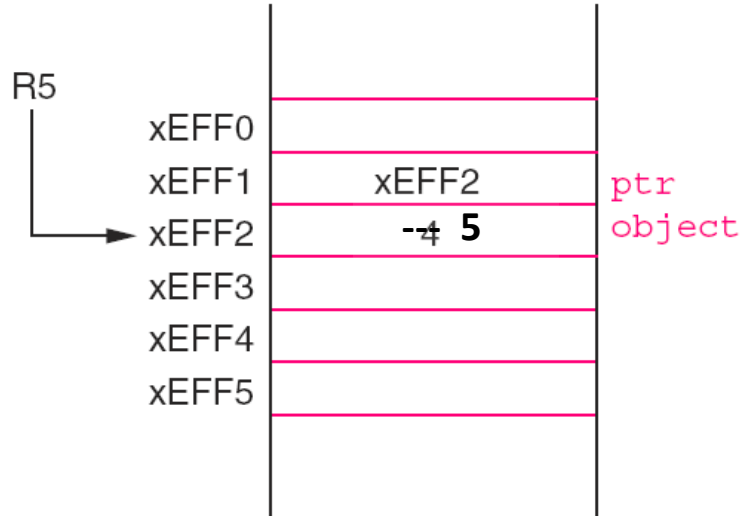


C to LC3

```
AND   R0, R0, #0    ;    Clear R0
ADD   R0, R0, #4    ;    R0 = 4
STR   R0, R5, #0    ;    Object = 4;

ADD   R0, R5, #0    ;    Generate memory address of object
STR   R0, R5, #-1   ;    Ptr = &object;
```

Pointers in LC3

```
*ptr = *ptr + 1; ??
```



C to LC3

```
LDR   R0, R5, #-1   ;   R0 contains the value of ptr
LDR   R1, R0, #0    ;   R1 <- *ptr
ADD   R1, R1, #1    ;   *ptr + 1
STR   R1, R0, #0    ;   *ptr = *ptr + 1;
```

Solve Swap Problem

```
void Swap(int firstVal, int secondVal);
int main()
{
    int valueA = 3;
    int valueB = 4;

    Swap(valueA, valueB);
}

void Swap(int firstVal, int secondVal)
{
    int tempVal;

    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
}
```

call by value

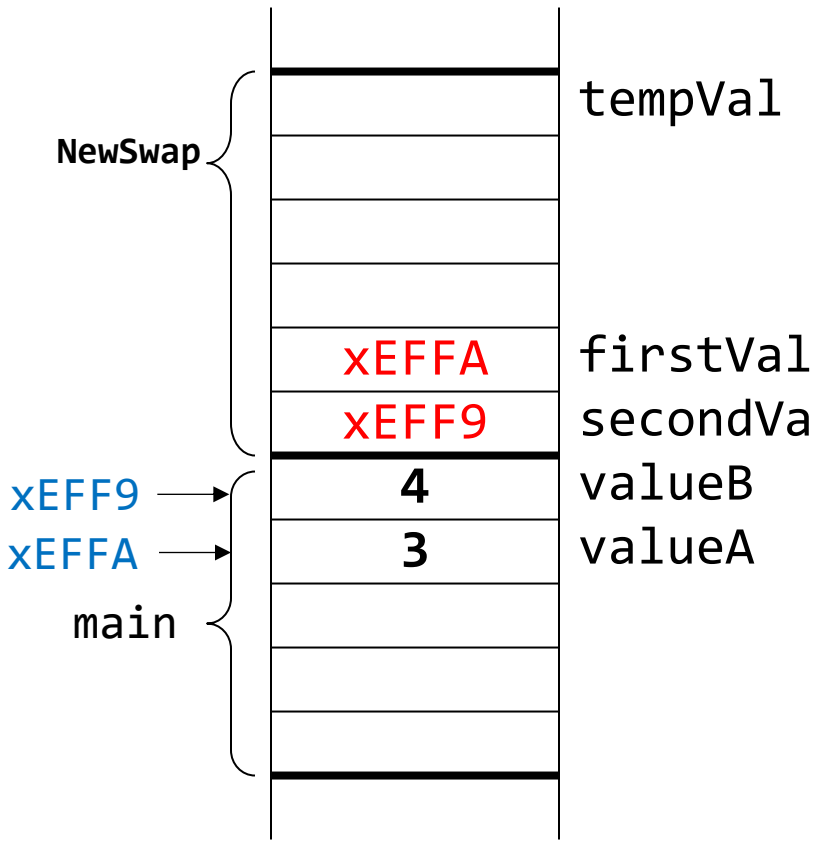
```
void NewSwap(int *firstVal, int *secondVal);
int main()
{
    int valueA = 3;
    int valueB = 4;

    NewSwap(&valueA, &valueB);
}

void NewSwap(int *firstVal, int *secondVal)
{
    int tempVal;

    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
```

call by reference

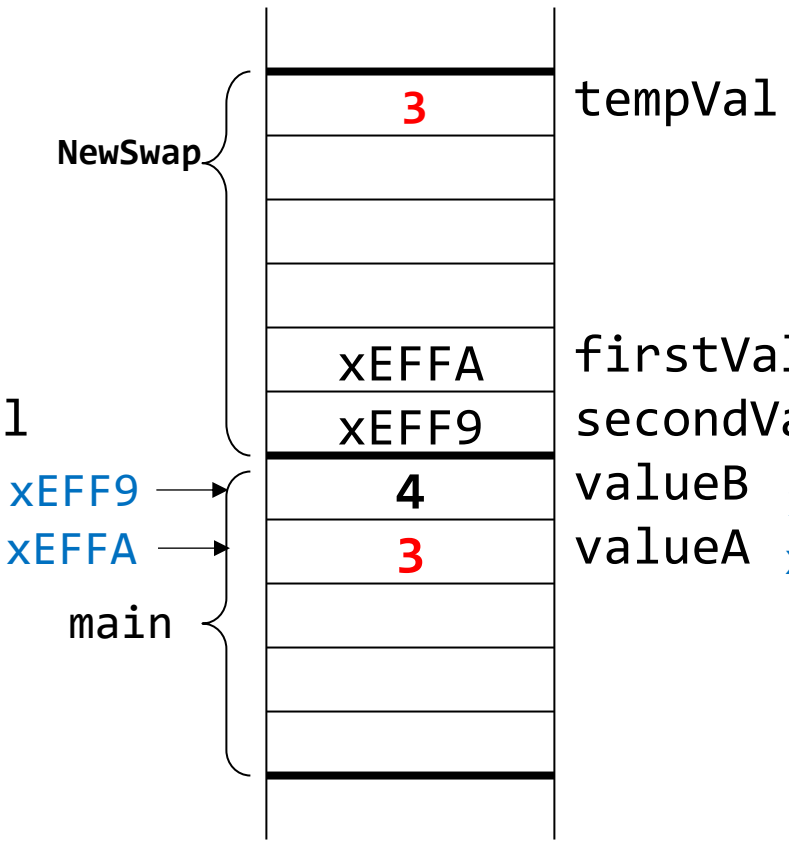
New Swap – Activation Record

```
void NewSwap(int *firstVal, int *secondVal)
{
    int tempVal;

    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
```

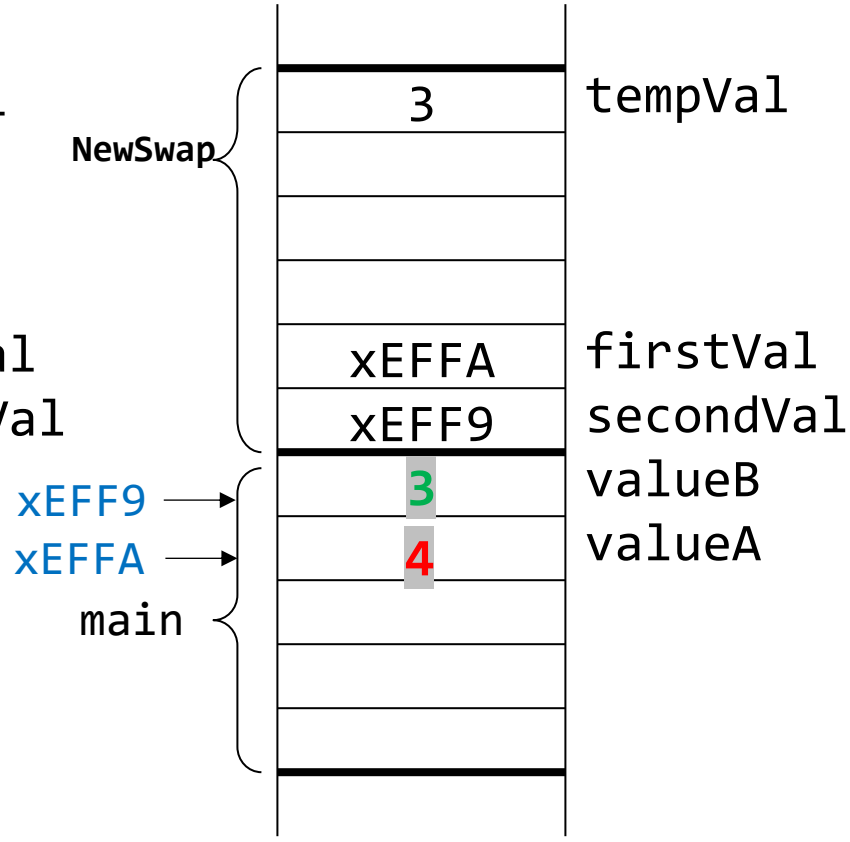
```
int main(){
…
   NewSwap(&valueA, &valueB);
```



tempVal = *firstVal;



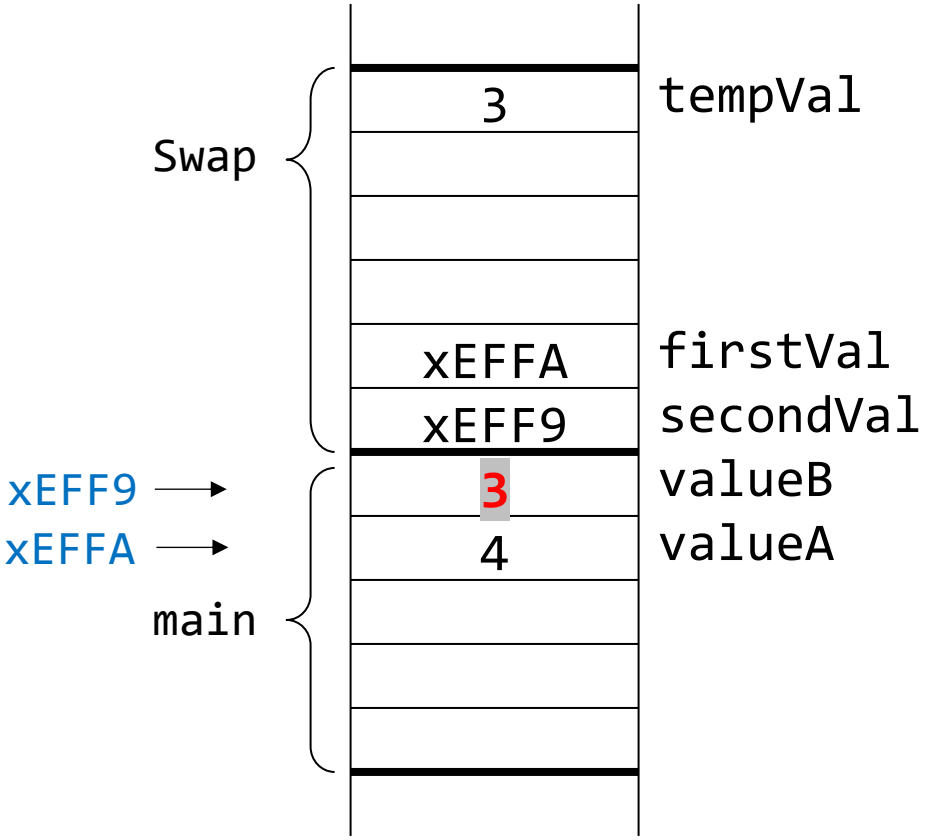
***secondVal = tempVal;**



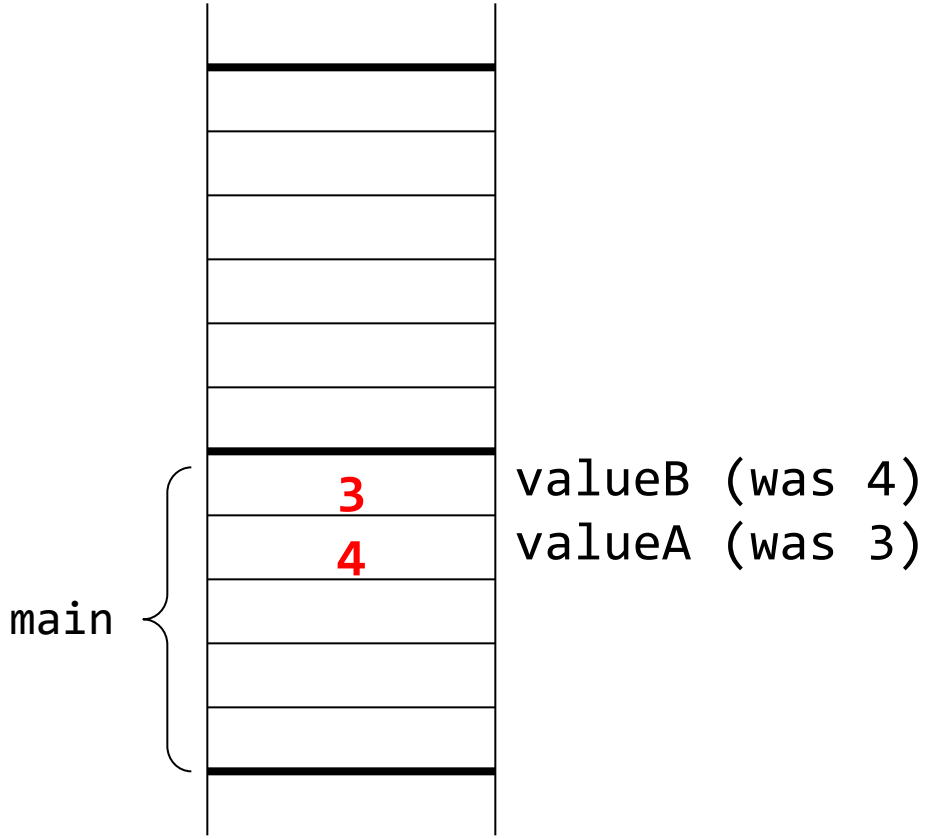
New Swap – Activation Record (continued)

```
void NewSwap(int *firstVal, int *secondVal)
{
    int tempVal;

    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
```

*secondVal = tempVal;



after call



Heads up: *??

```
int main()
{
    int valueA = 3;
    int valueB = 4;

    NewSwap(&valueA, &valueB);
}
                ①           ②
void NewSwap(int *firstVal, int *secondVal)
{
    int tempVal;
                ③
    tempVal = *firstVal;
④   *firstVal = *secondVal; ⑤
⑥   *secondVal = tempVal;
}
```

Which * is used for the dereference operator?

3,**4**,**5**,**6**

Which * is used for declaring a pointer variable?

1,2

```
int *firstVal = &valueA;
int *secondVal = &valueB;
```

```
int *firstVal;
firstVal = &valueA;

int *secondVal;
secondVal = &valueB;
```

More on Pointers

- Null pointer: a pointer that points to nothing

```
int *ptr;
int valueA;

ptr = &valueA;
printf("x%X\n", ptr);

ptr = NULL;
printf("x%X\n", ptr);
```

```
x1B2F5F3C
x0
```

- Demystifying '&' in scanf

```
int input;
scanf("%d", &input);
```

scanf needs to update the variable with data from the keyboard. Therefore, It needs the address of the variable, not its value.

Common Mistakes on Pointers

```
int val = 5;
char *ptr;
ptr = &val;
```

Type mismatch

```
int *ptr;
*ptr = 4;
```

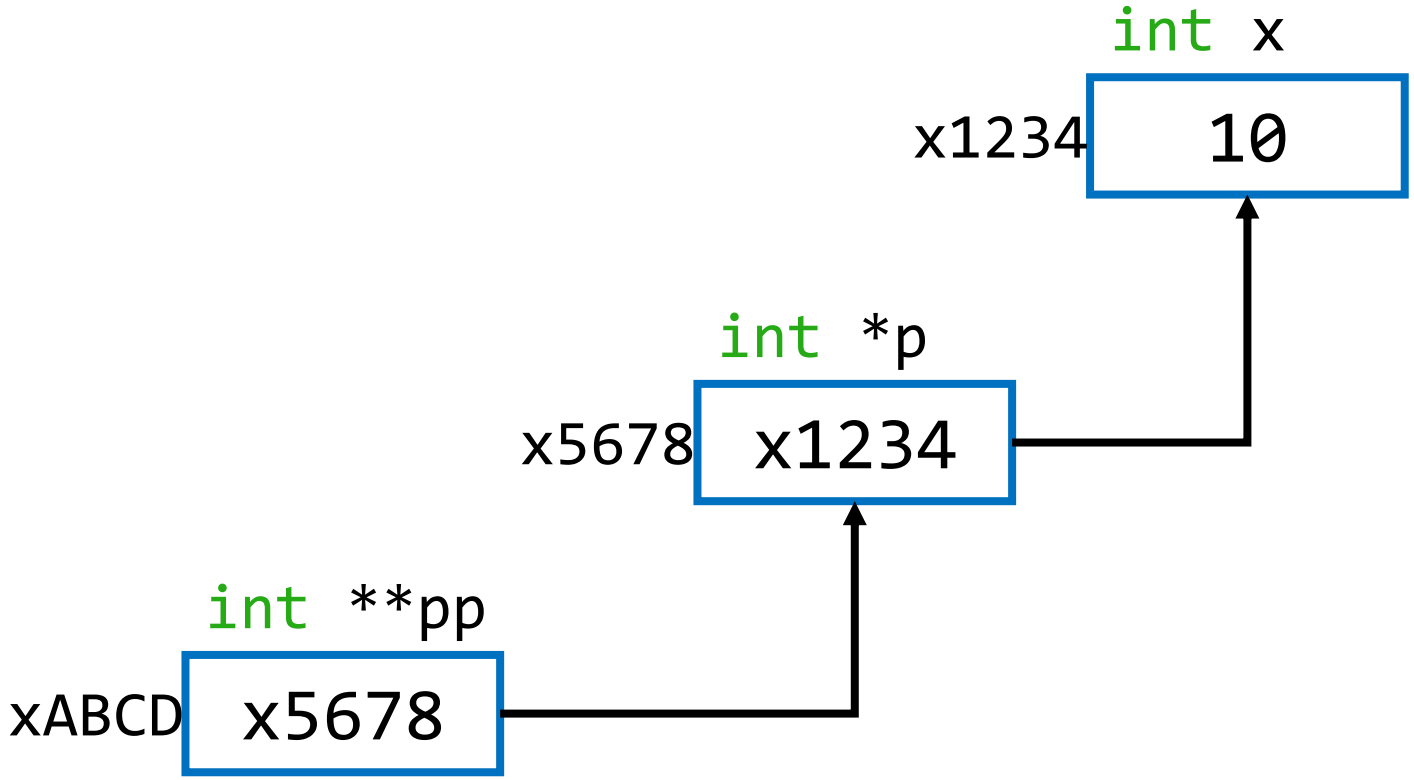
Dereferencing a pointer before initialized

```
int *ptr;
ptr = 4;
```

Pointing memory address “4”

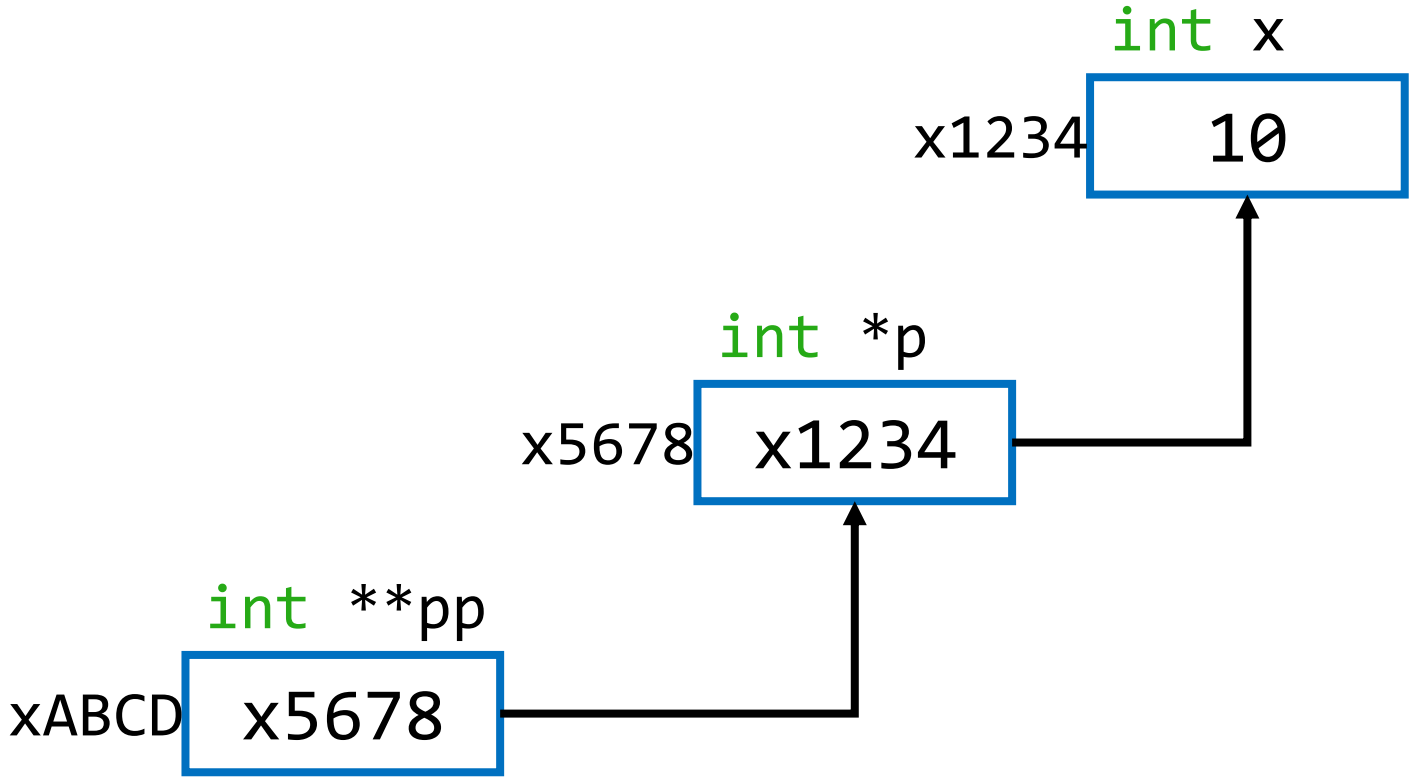
Double Pointer

```
int x = 10;
int *p;

p = &x;

int **pp;
pp = &p;


printf(”x%X\n", &pp);
printf(”x%X\n", pp);
printf(”x%X\n", *pp);
printf(”%d\n", **pp);
```



Double Pointer

```
int x = 10;
int *p;

p = &x;

int **pp;
pp = &p;
```




```
printf("x%X\n", &pp);
printf("x%X\n", pp);
printf("x%X\n", *pp);
printf("%d\n", **pp);
```

```
xABCD
x5678
x1234
10
```

Array

- How do we allocate a group of memory locations?
  - character string
  - table of numbers

```
int stdn0;
int stdn1;
int stdn2;
int stdn3;
.
.
.
```

- Problem1:
  What if 100 numbers?
- Problem2:
  How do you sort the numbers?

- Solution: Array

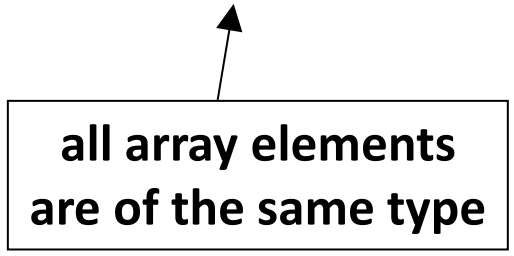
```
int stdn[100];
stdn[0]=90;
```

max/min index?

Array Syntax

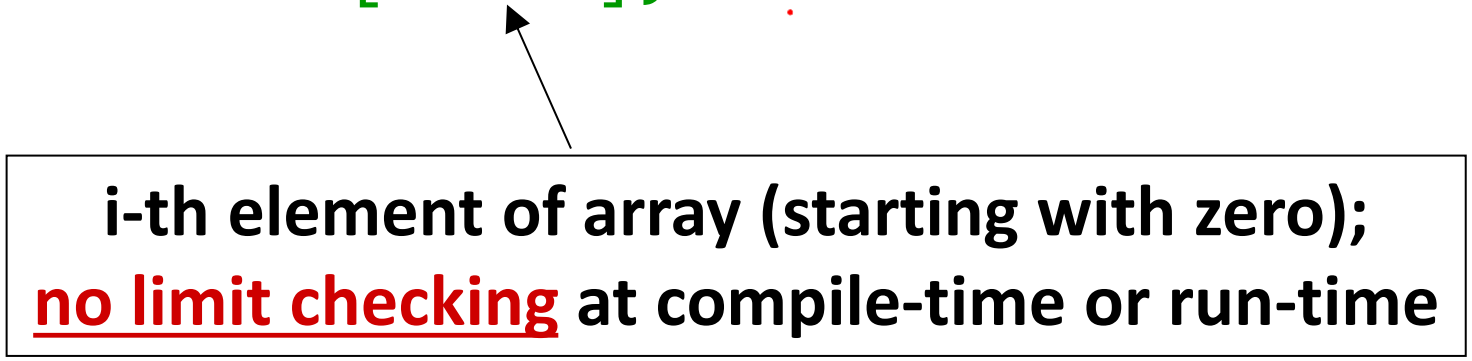
- Declaration

```
type  variable[num_elements];
```



- Array reference

```
variable[index];
```



Example

```
int grid[10] = {2,10,21,3,6,1,0,9,11,12};


grid[6] = grid[3]+1;


int i;
for(i=0;i<10;i++)
    printf("%d\n", grid[i]);
```

```
2
10
21
3
6
1
4
9
11
12
```

Example

```
int grid[10] = {2,10,21,3,6,1,4,9,11,12};

int x = 4;
grid[x+1] = grid[x]+2;


int i;
for(i=0;i<10;i++)
    printf("%d\n", grid[i]);
```

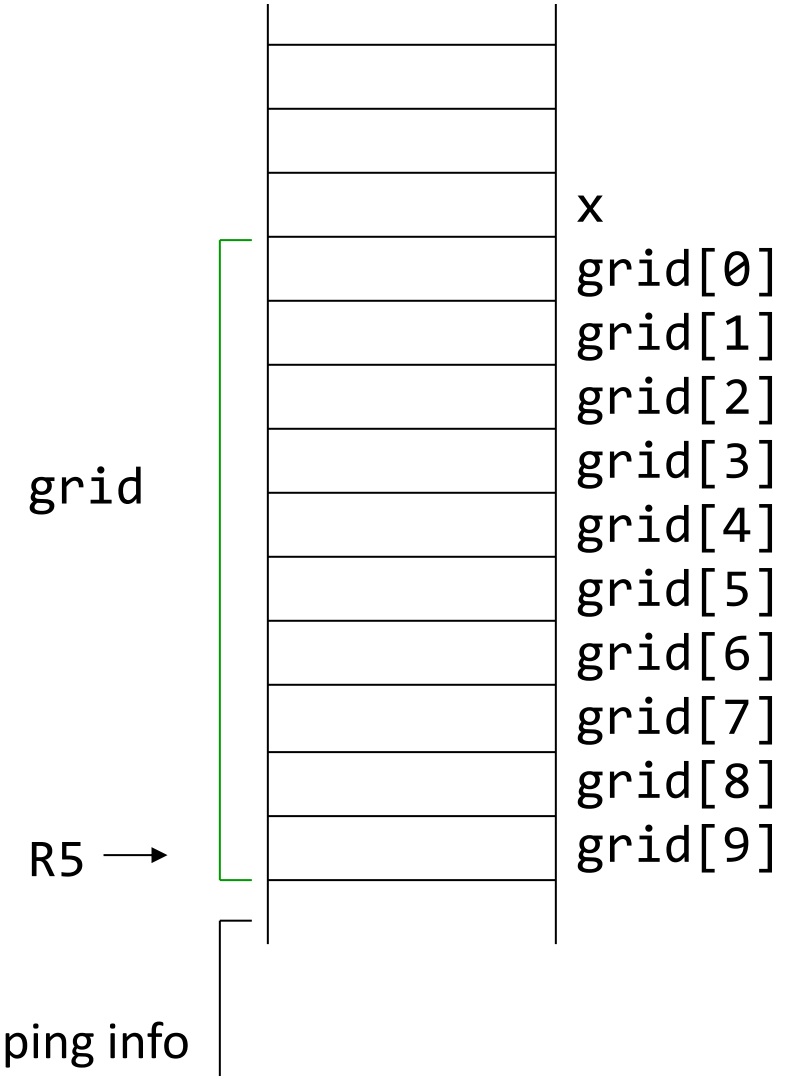
```
2
10
21
3
6
8
4
9
11
12
```

LC-3 for Array References

```
int grid[10] = {2,10,21,3,6,1,4,9,11,12};

int x = 4;
grid[x+1] = grid[x]+2;
```

grid[x+1] = grid[x] + 2

```
LDR R0, R5, #-10   ; Load the value of x
ADD R1, R5, #-9    ; Put the base address of grid into R1
ADD R1, R0, R1     ; Calculate address of grid[x]
LDR R2, R1, #0     ; R2 <-- grid[x]
ADD R2, R2, #2     ; R2 <-- grid[x] + 2
LDR R0, R5, #-10   ; Load the value of x
ADD R0, R0, #1     ; R0 <-- x + 1
ADD R1, R5, #-9    ; Put the base address of grid into R1
ADD R1, R0, R1     ; Calculate address of grid[x+1]
STR R2, R1, #0     ; grid[x+1] = grid[x] + 2;
```



Relationship between Pointers and Arrays

- An array name points to the first element in the array.

```
char word[10];
char *cptr;

cptr = word; // points to word[0]
```

- What is difference?

cptr is a variable, but the name "word" is not a variable.

```
cptr = cptr + 1;
word = word + 1; // compile error
```

Correspondence between Pointer and Array Notation

```
char word[10];
char *cptr;

cptr = word; // points to word[0]
```

- Each line below gives three equivalent expressions:

`cptr`	`word`	`&word[0]`
`(cptr + n)`	`word + n`	`&word[n]`
`*cptr`	`*word`	`word[0]`
`*(cptr + n)`	`*(word + n)`	`word[n]`