

# CmpE 150 - Week 8

Section - 04

# 6th quiz today

- Start: 12:00
- End: 12:20
- Be careful! **Do not** write extra things other than required, otherwise automatic grading **does not** give your points!
- No cell phones, no personal computers!

# Pointers

- Pointers are variables whose values are *memory addresses*.
- Normally, a variable directly contains a specific value.
- A pointer, on the other hand, contains an *address* of a variable that contains a specific value.

# Important operators with pointers

- & operator: Gives the address of the operand
- \* operator: Can be applied to pointers. Used for accessing the variable inside the pointer.

MA	val	var
3212		
3216		
3220		
3224		
3228		
3232		
3236		
3240		

MA	val	var
3212	3	a
3216	2	b
3220		ptr
3224		x
3228		y
3232		
3236		
3240		

MA	val	var
3212	3	a
3216	2	b
3220	3212	ptr
3224		x
3228		y
3232		
3236		
3240		



MA	val	var
3212	3	a
3216	3	b
3220	3212	ptr
3224		x
3228		y
3232		
3236		
3240		





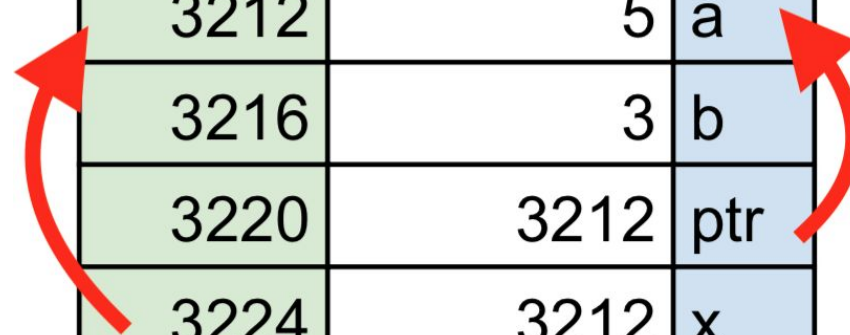
MA	val	var
3212	3	a
3216	3	b
3220	3212	ptr
3224		x
3228		y
3232		
3236		
3240		



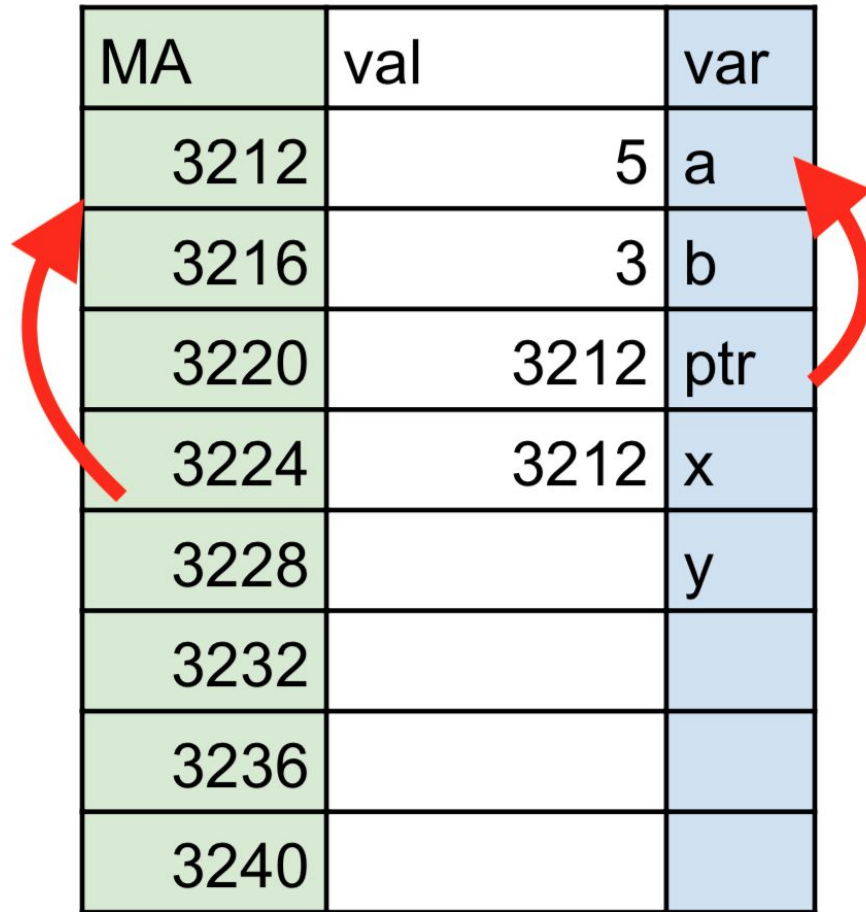
MA	val	var
3212	5	a
3216	3	b
3220	3212	ptr
3224		x
3228		y
3232		
3236		
3240		



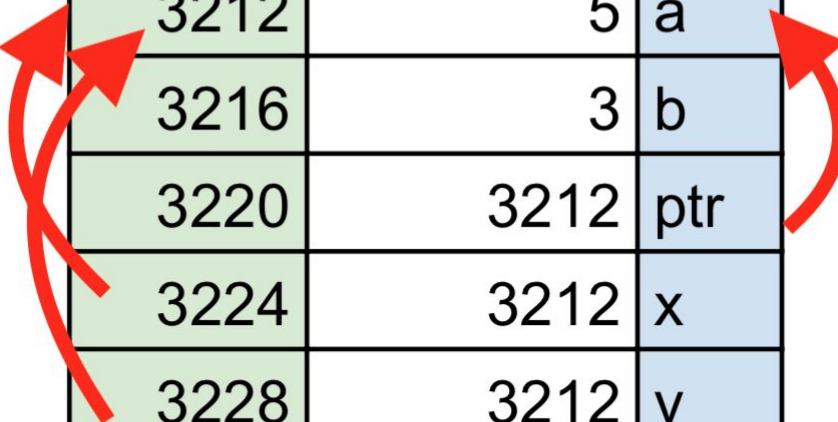
MA	val	var
3212	5	a
3216	3	b
3220	3212	ptr
3224	3212	x
3228		y
3232		
3236		
3240		



MA	val	var
3212	5	a
3216	3	b
3220	3212	ptr
3224	3212	x
3228		y
3232		
3236		
3240		



MA	val	var
3212	5	a
3216	3	b
3220	3212	ptr
3224	3212	x
3228	3212	y
3232		
3236		
3240		



MA	val	var
3212	3	a
3216	3	b
3220	3212	ptr
3224	3212	x
3228	3212	y
3232		
3236		
3240		

The diagram illustrates a memory structure with three columns: MA (Memory Address), val (Value), and var (Variable). The MA column contains addresses from 3212 to 3240 in increments of 4. The val column contains values, and the var column contains variable names. Red arrows indicate pointer relationships: one arrow points from the 'ptr' variable at address 3220 to the memory location 3212, and another arrow points from the 'ptr' variable to the 'a' variable at address 3212.

# Call by Reference

```
int main() {  
    int num;  
    scanf ("%d", &num) ;  
}
```

- We have already seen an example of call by reference.
- When you call a function, a frame for the function in the stack is created
- After the return statement within the function, this frame is erased

# Call by Value



Step 1: Before `main` calls `cubeByValue`:

```
int main( void )
```

```
{
```

```
    int number = 5;
```

```
    number = cubeByValue( number );
```

```
}
```

number

5

```
int cubeByValue( int n )
```

```
{
```

```
    return n * n * n;
```

```
}
```

n

undefined

Step 2: After `cubeByValue` receives the call:

```
int main( void )
```

```
{
```

```
    int number = 5;
```

```
    number = cubeByValue( number );
```

```
}
```

number

5

```
int cubeByValue( int n )
```

```
{
```

```
    return n * n * n;
```

```
}
```

n

5

Step 3: After `cubeByValue` cubes parameter `n` and before `cubeByValue` returns to `main`:

```
int main( void )
```

```
{
```

```
    int number = 5;
```

number

5

```
    number = cubeByValue( number );
```

```
}
```

```
int cubeByValue( int n )
```

```
{
```

```
    return n * n * n;
```

```
}
```

125

n

5

Step 4: After `cubeByValue` returns to `main` and before assigning the result to `number`:

```
int main( void )  
{  
    int number = 5;  
    number = cubeByValue( number );  
}
```

number

5

125

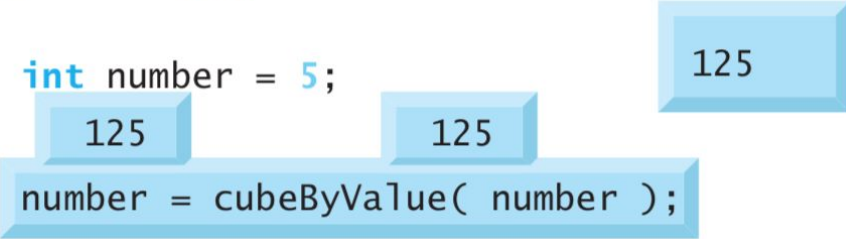
```
int cubeByValue( int n )  
{  
    return n * n * n;  
}
```

n

undefined

Step 5: After `main` completes the assignment to `number`:

```
int main( void )  
{  
    int number = 5;  
    number = cubeByValue( number );  
}
```



```
int cubeByValue( int n )  
{  
    return n * n * n;  
}
```

n

undefined

# Call by Reference

Step 1: Before main calls cubeByReference:

```
int main( void )
```

```
{
```

```
    int number = 5;
```

```
    cubeByReference( &number );
```

```
}
```

number

5

```
void cubeByReference( int *nPtr )
```

```
{
```

```
    *nPtr = *nPtr * *nPtr * *nPtr;
```

```
}
```

nPtr

undefined

Step 2: After `cubeByReference` receives the call and before `*nPtr` is cubed:

```
int main( void )
```

```
{
```

```
    int number = 5;
```

```
    cubeByReference( &number );
```

```
}
```

number

5

```
void cubeByReference( int *nPtr )
```

```
{
```

```
    *nPtr = *nPtr * *nPtr * *nPtr;
```

```
}
```

nPtr

*call establishes this pointer*



Step 3: After `*nPtr` is cubed and before program control returns to `main`:

```
int main( void )  
{  
    int number = 5;  
  
    cubeByReference( &number );  
}
```

number

125

```
void cubeByReference( int *nPtr )  
{  
    125  
    *nPtr = *nPtr * *nPtr * *nPtr;  
}
```

*called function modifies caller's variable*

nPtr