CmpE 150 - Week 8

Section - 02

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		

```
int a = 3, b = 2, *ptr, *x, *y;
```

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

```
int a = 3, b = 2, *ptr, *x, *y;
ptr = &a;
```

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

```
int a = 3, b = 2, *ptr, *x, *y;
ptr = &a;
b = *ptr;
```

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

```
int a = 3, b = 2, *ptr, *x, *y;
ptr = &a;
b = *ptr;
*ptr = a;
```

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

```
int a = 3, b = 2, *ptr, *x, *y;
ptr = &a;
b = *ptr;
*ptr = a;
*ptr = 5;
```

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

```
int a = 3, b = 2, *ptr, *x, *y;
ptr = &a;
b = *ptr;
*ptr = a;
*ptr = 5;
x = ptr;
```

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

```
int a = 3, b = 2, *ptr, *x, *y;
ptr = &a;
b = *ptr;
*ptr = a;
*ptr = 5;
x = ptr;
x = &*ptr;
```

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

```
int a = 3, b = 2, *ptr, *x, *y;
ptr = &a;
b = *ptr;
*ptr = a;
*ptr = 5;
x = ptr;
x = &*ptr;
y = *&ptr;
```

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

```
int a = 3, b = 2, *ptr, *x, *y;
ptr = &a;
b = *ptr;
*ptr = a;
*ptr = 5;
x = ptr;
x = &*ptr;
y = *&ptr;
a = *&b;
```

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

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

Src: Deitel C How to Program

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

Src: Deitel C How to Program

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

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

   number = cubeByValue( number );
}
```

number: 5

```
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 );
}
```

125

number

5

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

n

undefined

Src: Deitel C How to Program

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

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

125 125

number

125

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

n

undefined

Src: Deitel C How to Program

Call by Reference

Step 1: Before `main` calls `cubeByReference`:

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

   cubeByReference( &number );
}
```

number

5

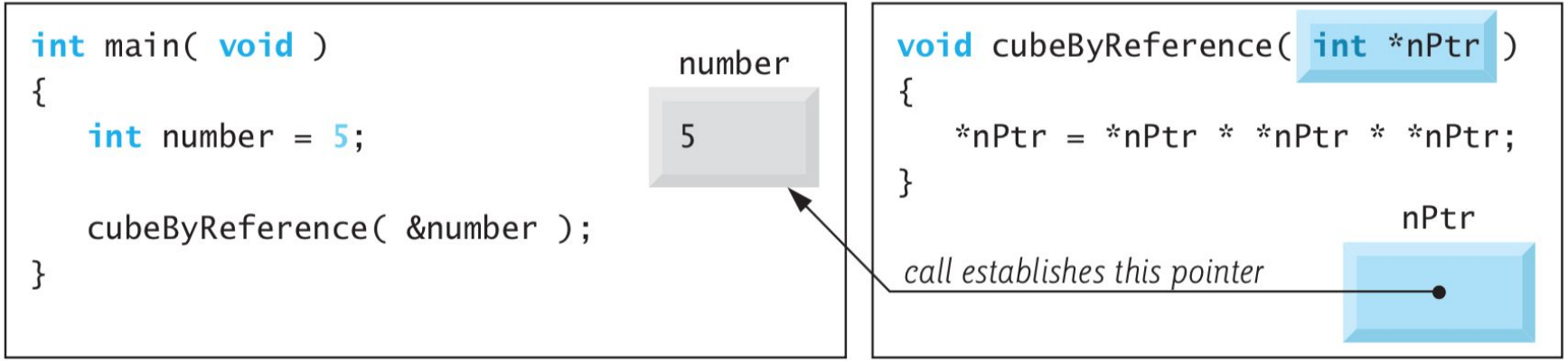
```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

nPtr

undefined

Src: Deitel C How to Program

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



Src: Deitel C How to Program

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 )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

125

called function modifies caller's variable

nPtr

Src: Deitel C How to Program