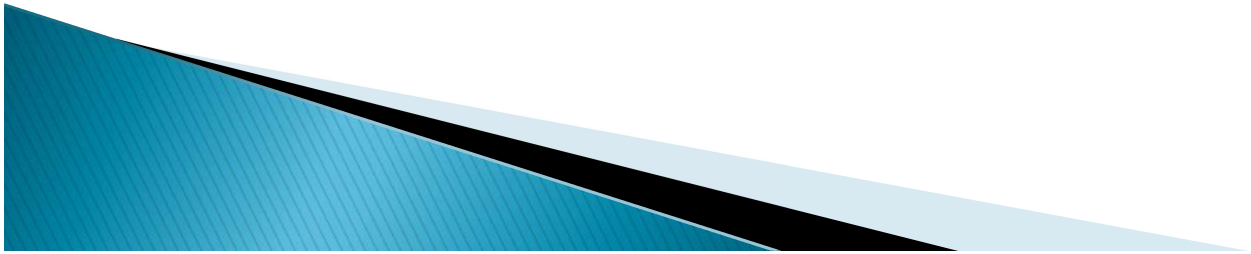
Pointers

- A pointer is a variable just like other variable.
- The only difference from other variables is that it stores the memory address other variables.
- This variable may be of type int, char, array, structure, function or any other.



Pointer Variable

- Normal variables
  - Contain a specific value (direct reference)
- Pointer variables
  - Contain memory addresses as their values

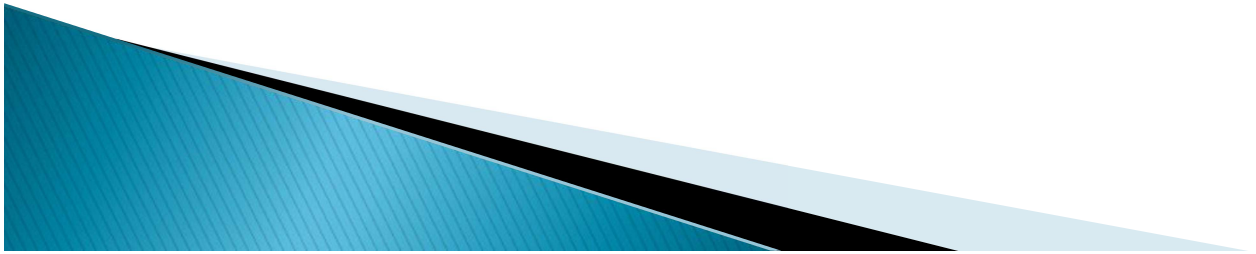
Pointer Variable Declarations

- Pointer declarations
  - '*' used with pointer variables

```
int  *myPtr;
float *Ptr;
Char *strPtr;
```

  - Multiple pointers require using a * before each variable declaration

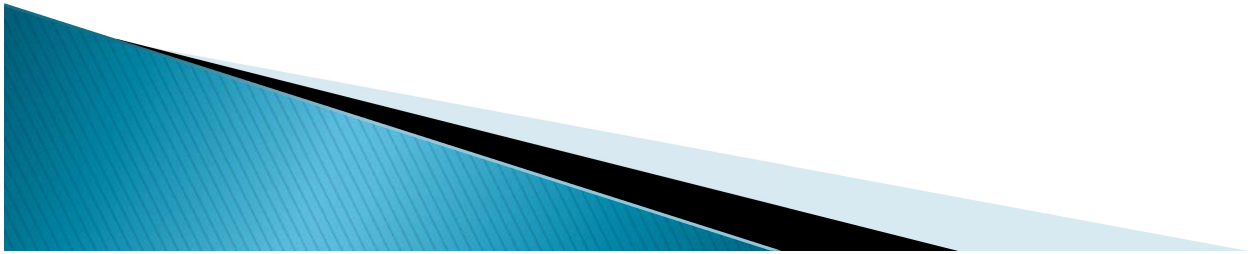
```
int *myPtr1, *myPtr2;
```



Pointer Variable Initialization

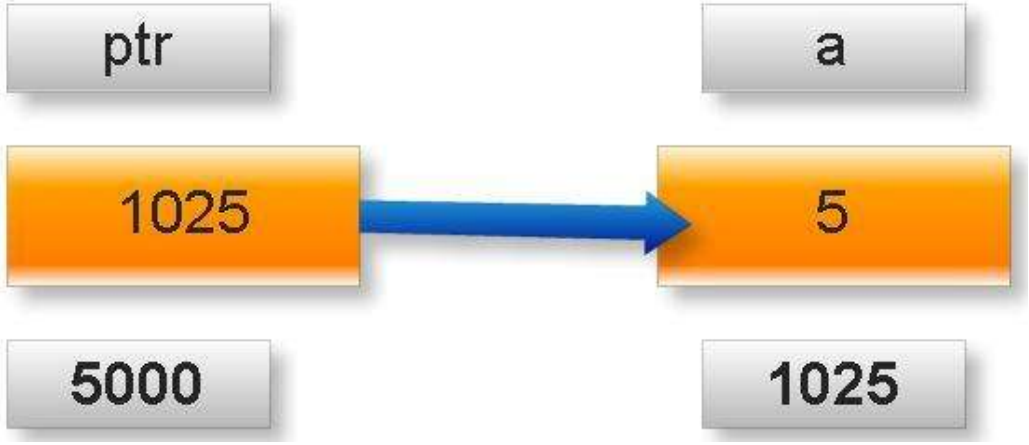
- Initialize pointers to:
  - **0 or NULL,**
    - points to nothing (**NULL** preferred)
  - an address
    - points to a certain place/address in the memory

```
int y = 5;
int *yPtr;
yPtr = &y;     // yptr is the address of y
               // yPtr "points to" y
```



Pointers

```
int a=5;
int * ptr;
ptr=&a;
```



About variable a:

- 1. Name of variable : a
- 2. Value of variable: 5
- 3. Address: 1025 (assume)

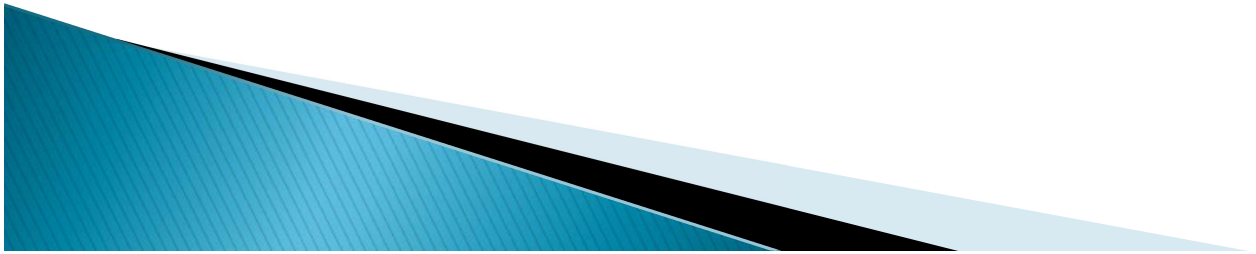
About variable ptr:

- 4. Name of variable: ptr
- 5. Value of variable: 1025
- 6. Address: 5000 (assume)

Calling Functions by Reference

- Call by reference with pointer arguments
  - Pass address of argument using **&** operator
  - Arrays are not passed with **&** because the array name is already a pointer
- * operator
  - Used as nickname for variable inside of function

```
void double( int *number )
 {     *number = 2 * ( *number ); }
```



Outline

Function prototype

1 Initialize variables

2. Call function

3. Define function

```
1  /* Example (2)
2     Cube a variable using c
3     with a pointer argument
4
5  #include <stdio.h>
6
7  void cubeByReference( int * );
8
9  int main()
10 {
11    int number = 5;
12
13    printf( "The original value of number is %d", number );
14    cubeByReference( &number );
15    printf( "\nThe new value of number is %d\n", number );
16
17    return 0;
18 }
19
20 void cubeByReference( int *nPtr )
21 {
22    *nPtr = *nPtr * *nPtr * *nPtr;  /* cube number in main */
23 }
```

Notice that the function prototype takes a pointer to an integer (**int ***).

Notice how the address of **number** is given - **cubeByReference** expects a pointer (an address of a variable).

Inside **cubeByReference**, ***nPtr** is used (***nPtr** is **number**).

Program Output

```
The original value of number is 5
The new value of number is 125
```

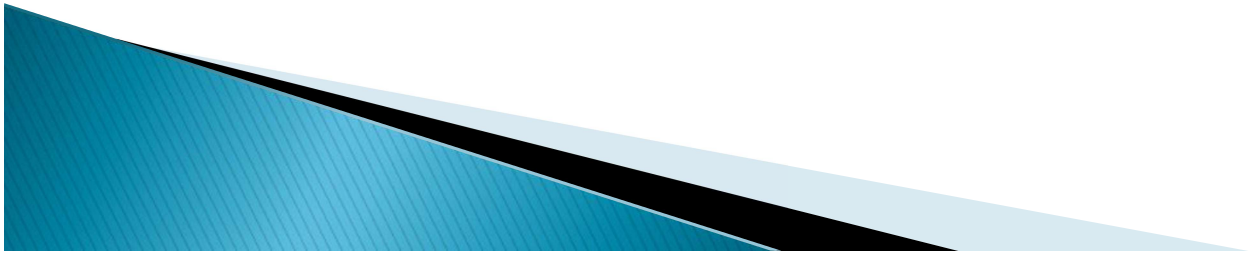
Arithmetic operation with pointer

```
#include<stdio.h>
int main()
{
int *ptr=( int *)1000;
ptr=ptr+1;
printf(" %d",ptr);
return 0;
}
```

Output: 1004

```
#include<stdio.h>
int main()
{
double *p=(double *)1000;
p=p+3;
printf(" %d",p);
return 0;
}
```

Output: 1024



Arithmetic operation with pointer

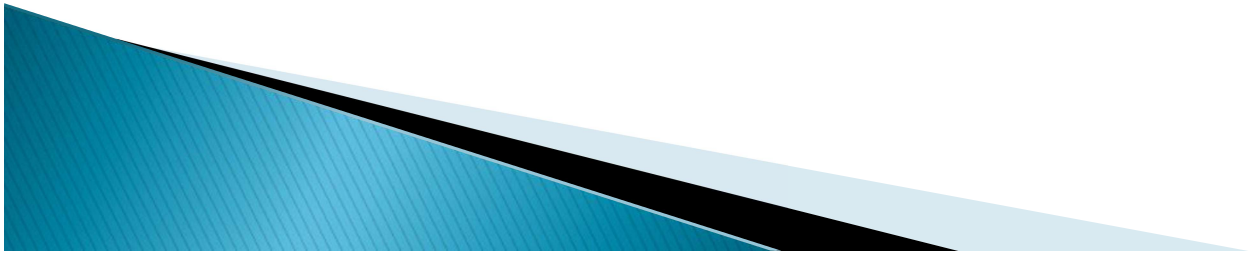
```
#include<stdio.h>
int main()
{
int *p=(int *)1000;
int *temp;
temp=p;
p=p+2;
printf("%d %d\n",temp,p);
printf("difference= %d",p-
temp);
return 0;
}
```

Output: 1000 1008

Difference= 2

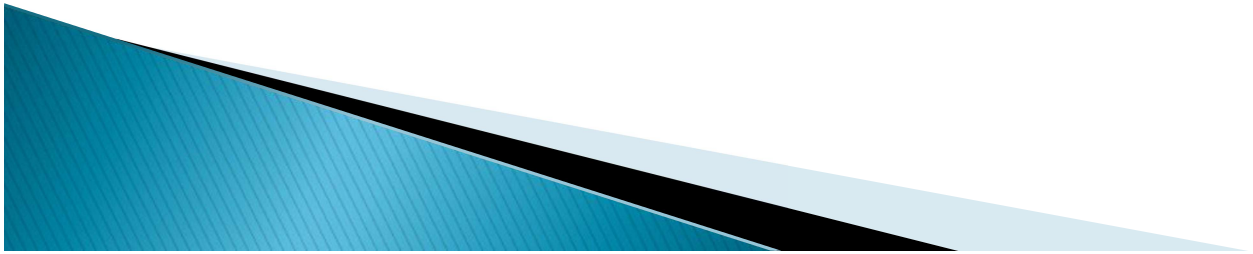
```
#include<stdio.h>
int main()
{
float *p=(float *)1000;
float *q=(float *)2000;
printf("Difference= %d",q-p);
return 0;
}
```

Output: Difference= 250



Pointer Expressions and Arithmetic

- Arithmetic operations can be performed on pointers
  - Increment/decrement pointer (++ or --)
  - Add an integer to a pointer
    (+ or += , - or -=)
  - Pointers may be subtracted from each other
  - Operations meaningless unless performed on an array



Pointer Expressions and Arithmetic

operation	Description
p++, p--	Increment (decrement) p to point to the next element, it is equivalent to p+=1 (p -=1)
p+i (p-i)	Point to i-th element beyond (in front of) p but value of p is fixed
p[i]	Equivalent to p + i
p + n (integer)	n must be an integer, its meaning is offset
p - q	Offset between pointer p and pointer q
p+q, p*q, p/q, p%q	invalid
Relational operator of two pointers p, q	valid, including p > q, p < q, p == q, p >= q, p <= q, p != q

Pointer Expressions and Arithmetic

- Pointer comparison (<, == , >)
  - See which pointer points to the higher numbered array element (index)
  - Also, see if a pointer points to 0

Long Form	Short Form
if (ptr == NULL)	if (!ptr)
if (ptr != NULL)	if (ptr)