Pointers ☺

Week 10

Pointers

- Pointer is a variable just like other variables of c but only difference is unlike the other variable it stores the memory address of any other variables of c.
- This variable may be type of int, char, array, structure, function or any other pointers.

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;
```

  - Can declare pointers to any data type

Pointer Variable Initialization

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

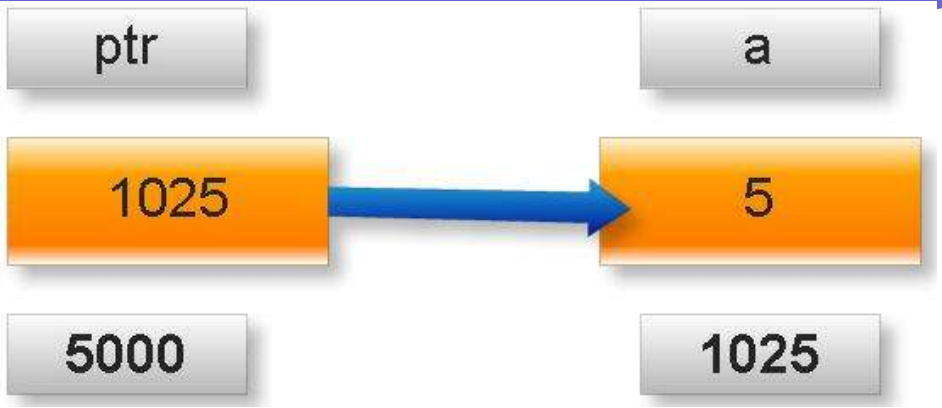
```
int y = 5;
int *yPtr;
yPtr = &y;      // yptr is the address of y
                // yPtr "points to" y
```

Pointers

int a=5;

int * ptr;

ptr=&a;



About variable a:

- 1. Name of variable : a
- 2. Value of variable which it keeps: 5
- 3. Address where it has stored in memory : 1025 (assume)

About variable ptr:

- 4. Name of variable : ptr
- 5. Value of variable which it keeps: 1025
- 6. Address where it has stored in memory : 5000 (assume)

Calling Functions by Reference

- Call by reference with pointer arguments
  - Pass address of argument using **&** operator
  - Allows you to change actual location in memory
  - 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 ); }
```

```
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**).

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

Outline

Function prototype

1 Initialize variables

2. Call function

3. Define function

Program Output

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

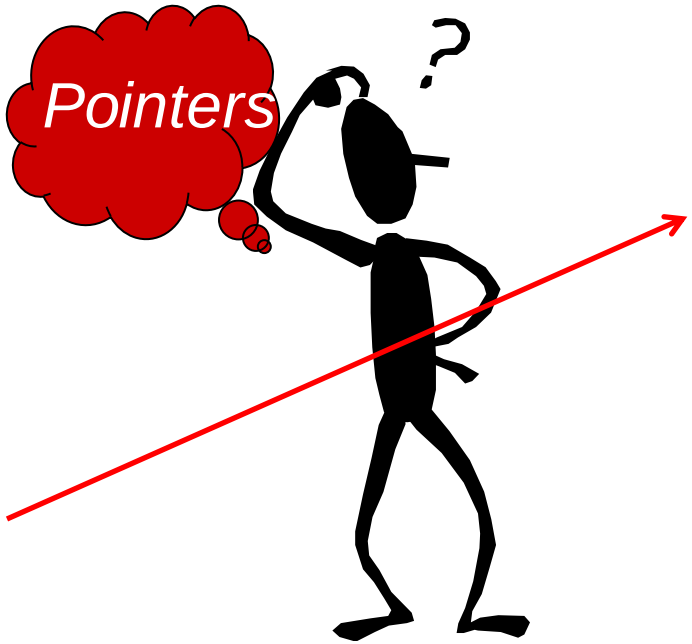
```
#include<stdio.h>
int main()
{
float *p=(float *)1000;
float *q=(float *)2000;
printf("Difference= %d",q-p);
return 0;
}
```

Output: Difference= 250

Pointers with arrays

```
#include <stdio.h>
int my_array[] = {1,23,17,4,-5,100};
int *ptr;
 int main(void)
 {    int i;
      ptr = &my_array[0]; /* point our pointer to the first element of the array */
      printf("\n\n");
      for (i = 0; i < 6; i++)
            {
            printf("my_array[%d] = %d ",i,my_array[i]);
            printf("ptr + %d = %d\n",i, *(ptr + i));
            }
      return 0;
 }
```

Pointers Power



Pointers to Functions

Pointers to Structure

Pointers to ……