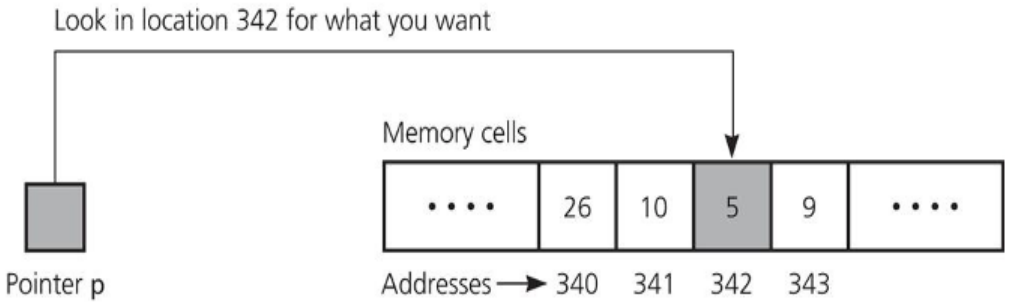
Review on Pointer and use of Dynamic memory

- What is a pointer? A pointer contains the location, or address in memory, of a memory cell

- Declarations

```
int     *ip;     //a pointer to integer
char    *cp;     //a pointer to char
float   *fp;     //a pointer to float
int     *p, q;   //an integer pointer & an integer
```

- DeReferencing, the address operator
  The expression *p denotes the memory cell to which p points
  The & address-of operator returns variable's address &x
  Use & to place the address of a variable into a pointer variable
  p = &x;



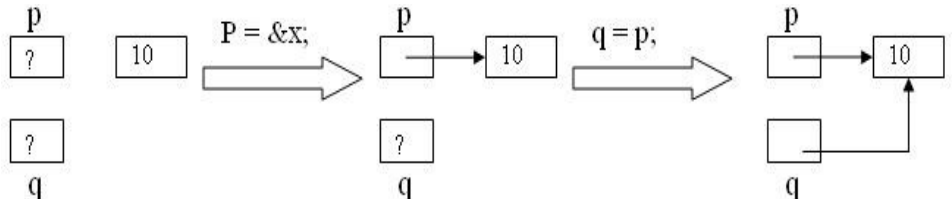

```
int main(int argc, char ** argv)
{
        int x = 10;
        int *p = & x;

        cout << "The address of pointer p is " << &p << endl;
        cout << "The address stored in pointer p is " << p << endl;
        cout << "The address of x is " << &x << endl;
        cout << "The value stored in x is " << x << endl;
        cout << "The value pointed by the pointer p is "<< *p << endl;
}
```

Understand the difference between *p, &p, and p.
Understand the difference between the following statements:

```
// x y are integer variables and p is an integer pointer
p = &x;
*p = x;
y = *p;
```

- Pointer Assignment

```
int  x = 10;
int  *p, *q;
p = &x;
q = p;
```



Cannot assign a pointer to another pointer of **DIFFERENT** type

```
double x = 10.5;
double *p = &x;
int       *q;
q = p;    // Wrong. Assignment between pointers of different types
```

- NULL pointer (A special constant pointer value NULL)
  - Points to nothing
  - Can be assigned to a pointer of any type

```
int *p = NULL;
float *q = NULL;
```

  - A pointer initially undefined, but not NULL
  - int *p; //What's the address stored in pointer p?
  - Can compare a pointer with NULL:

```
if ( p == NULL )
{
    //p does not point to anything
 }
```

Cannot obtain the value pointed by a NULL pointer

```
int *p = NULL;
int x = *p; // What happens? Program aborts due to segmentation fault.
```

- Pointer to Structure/Class

```
Class Student {
private:
        string name;
        MajorType major;
        int score;
public:
        Student(string n, MajorType mt);
        string GetName() const;
        void SetScore(int s); ...
};
```

```
Student  john( "John", ComputerScience );
Student  david( "david", ComputerEngineering );
Student  *anyone;
anyone = &john;
anyone = &david;
anyone = NULL;
```

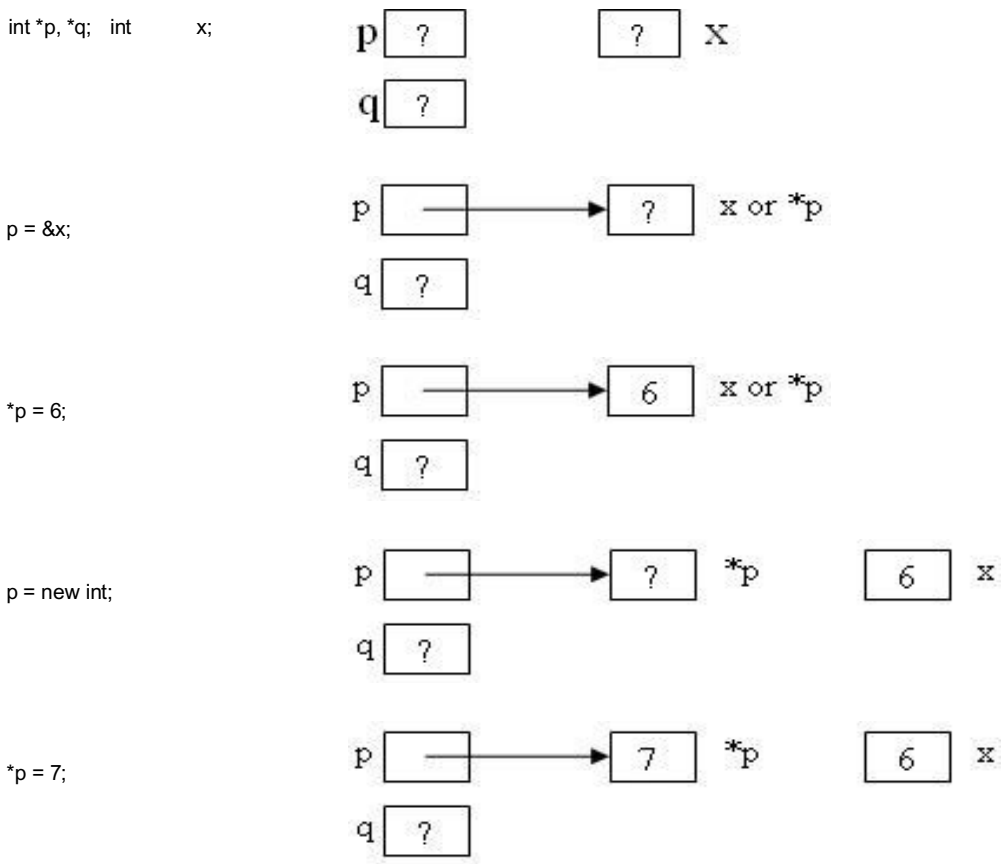
- How to access structure fields (class members) through a pointer?

```
anyone=&john;
anyone->SetScore(100);
(*anyone).SetScore(100);
anyone.SetScore(100);  //wrong
 john->SetScore(100);  // wrong
```

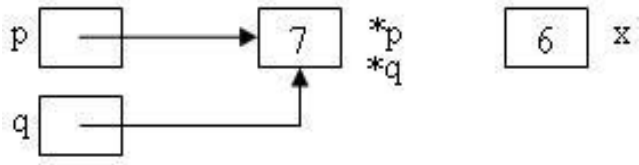
- Cannot access members through a NULL pointer

```
anyone = NULL;
anyone->SetScore(100); // What happens?
```

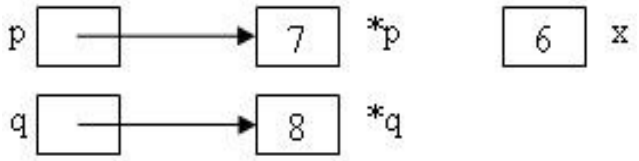
Example 1:



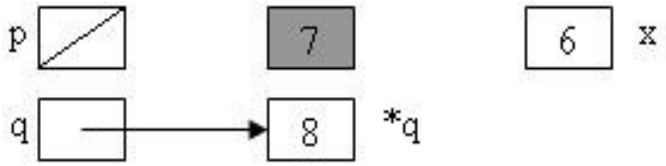
q = p;



q = new int;
*q = 8;



p = NULL;



delete q;
q = NULL;

- **Dynamically allocate array**
  - **1D array**

static allocation : int array [SIZE]; ← fixed SIZE (constant)
dynamic allocation :
int * arrayP = new int [actualSize]; ← actualSize may be changed during run time
int size;
cin >> size;
int *arrayP; ➔ int * arrayP = new int [size];
arrayP = new int [size];

Here, arrayP – holds the address of the first element of the array Access
array elements:

Equivalent pointer arithmetic notation

```
arrayP[0]     *arrayP
arrayP[1]     *(arrayP+1)      // advance to the address of the next element
arrayP[2]     *(arrayP+2)            // in the array
…                    …

for (int  i=0; i<size; i++)
        cin >> *(arrayP+1); same as        cin >> arrayP[i];
```

- **Increase memory size dynamically in the program**

```
int    * accounts = new int [initialSize];
for (int i=0; i<initialSize; i++) {
        cin >> accounts[i];
        …
}
```

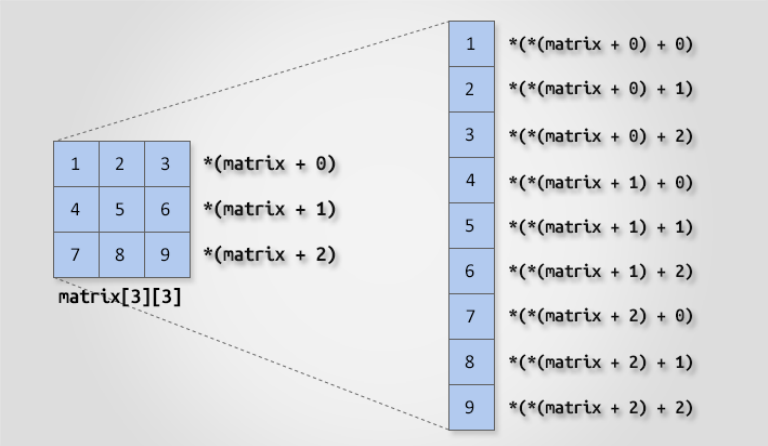
… < realize that "accounts" does not have enough space (during program execution>
… dynamically double the size of the accounts, making sure that the original account information is still kept in the new array

```
int  *oldAccounts = accounts;
int  *doubleAccounts=new int [initialSize*2];

// copy old accounts information to doubleAccounts
for (int   i=0; i<initialSize; i++)
        doubleAccounts[i] = oldAccounts[i];

delete [] oldAccounts;  // releasing memory space allocated
```

- **2D array**

Allocate 2D array dynamically

```
int  ** matrix;
matrix = new int * [numOfRows];
for (i=0; i<numOfRows; i++)
  matrix[i] = new int [numOfCols];
```



Releasing 2D array that is allocated dynamically

```
for (int i=0; i<numOfRows; i++)
      delete [] matrix[i];
delete [] matrix;
```

numOfRows, numOfCols can be changed dynamically, array is allocated dynamically