CSCI 2170
Pointers (1)

- Pointer variable contains the location/address of a memory cell

```
int x;
int *p;                         // p is a pointer to a memory cell of an int var

p = &x;                         // & -- address operator
x = 5;
```

- *p is the content of the memory to which p points to

```
cout << p << *p << endl;              // what will be the output?
*p = 10;
cout << p << *p << endl;

int *q;
q=p;                            // what is assigned to q?
cout << *q << endl;             //  what is the output?
```

- Arithmetic involving pointer

```
*p = *p + 4;                     // *p can be used just like a int type variable
cout << *p << *q << x << endl;
```

- int *p, q; is not the same as int *p, *q;

- typedef int * IntPtr;

(1) IntPtr p, q;

Exercises:

```
typedef     float * FloatPtr;
float     v1=2.5, v2=3.0;
FloatPtr    p, q;

p=&v1;
q = p;
v1 += 3;
v2 = *p;
q = &v2;
cout << v1 << v2 << *p << *q << endl;
```

- Dynamically allocate space using pointer

Static memory allocation – memory allocated during compile time

```
e.g.,     int   x;
          int   array[SIZE];
```

Dynamic memory allocation – memory allocated during run time (program execution)

```
IntPtr    p, q;
p = new int;              // the memory content can only be accessed with *p

*p = 5;
q = p;
q = new int;
*q = *p +2;
```

- Memory leak: memory not released upon termination of the program

When can this happen?

```
int main()
{
    IntPtr    p;
    *p = 5;

   return 0;
} // memory holding by p is not de-allocated and returned back to memory pool.
```

Solution:

```
int main()
{
    IntPtr    p;
    *p = 5;
    delete p;       // returns memory to the system for reuse
    p = NULL;    // p is set not pointing to anything.

   return 0;
}
```

!! Dangerous situation!!

Pointer points to de-allocated memory space

```
p = new int;
q = p;
delete p;
p = NULL;
```

what happens to q? what if you have this in the program?

```
cout << *q ;
```

// q is still pointing at the same memory cell, which might have been re-used/ reassigned to other variables in the program

solution—add → q = NULL;

Exercises:

What is the output of the following program?

```
int  main()
{
        ptyType   p, q;
        p = new int;
        *p = 2;
```

```
        q = new int;
        *q = 5;
        cout << *p << "  " << *q << endl;
        *p = *q + 10;
        delete p;
        p = q;
        cout << *p << *q << endl;
        *p = 8;
        cout << *p << *q << endl;
        delete q;
        p = NULL;
        q = NULL;

        return 0;
}
```

(2)
```
struct  Contact
    {
      string  name;
      string    phone;
    }
    typedef   Contact * ContactPtr;

    ContactPtr    p;
     p = new Contact;
     p-> name = "John Smith";                          // access member of struct using
                                                       // pointer
     p->phone = "(615)332-9823";     // or  (*p).phone = "(615)332-9823";

      Contact     friend1;
      friend1.name = "Mary";
      friend1.phone = "(615)983-0948";

      p = &friend1;
      cout << p->name << endl;
```

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

static vs dynamic array allocation

static : `int  array [SIZE];` ← fixed SIZE (constant)

dynamic:

```
        int    * arrayP = new   int [actualSize];
```
← actualSize may be change 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   array elements
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 for all customers (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  ** array;
array = new int * [numOfRows];
for (i=0; i<numOfRows; i++)
    array[i] = new int [numOfCols];
```

Releasing 2D array that is allocated dynamically

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

numOfRows, numOfCols can be changed dynamically, array is allocated dynamically
graphically what does this array looks like in the memory?