CSCI 2170 Pointers and Dynamic memory allocation

- **Dynamic memory allocation – allows for acquiring memory during program execution**
- **Pointer type variable contains the location/address of a memory cell**
- **Reference type needs to be initialized to point to a variable. Dereferencing is used afterwards.**

- Syntax for declaring a pointer variable
  data-type * variable-name;
  Example:
  `int *p;` // p is a pointer type variable, points to a memory location of an integer

- Explicitly access the address of a memory location using the **address operator &**
  `& value` // returns the address of the memory location for variable "value"

Here is an example to illustrate how to use the pointer variable and address operator:

```
int x;
x = 5;                  // direct accessing
int *p;
p = &x;                 // & -- address operator
```

- *p is the data at the memory location p points to, indirect accessing (accessing a variable in 2 steps by first using a pointer that gives the location of a variable

```
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;

- **Create an alias for a pointer type using typedef**

```
typedef int * IntPtr;
 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];

- **stack vs. heap memory allocation**

- **Dynamic memory allocation – memory allocated during run time (program execution)**
  - **new operator – allocate memory dynamically, it returns the address of the memory acquired dynamically**
  - **delete operator – release memory back to the heap**

```
IntPtr     p, q;
p = new int;              // the data at the memory location can only be accessed with *p
*p = 5;
delete p;

q = new int;
*q=10;
p=q;
*q = *p +2;
delete p;
```

- **Memory leak**: memory not released upon termination of the program. When does this happen?

- **Inaccessible object**

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

     q=new int;
     p = q;
     delete q;

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

- **Dangling pointer**: 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 location, which might have been re-used for 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;
```

- **Reference type variable**
  A **reference variable** is an alias, that is, another name for an already existing **variable**. Once a **reference** is initialized with a **variable**, either the **variable** name or the **reference** name may be used to refer to the **variable**.
  - Syntax for declaring a reference type variable
    data-type & variable-name;

```
int x;
int & r=x;       // r is a reference type variable, initialized to point to variable x
r = 10;          // deferencing, x is changed to 10
```

- **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];
```

- **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  ** 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?