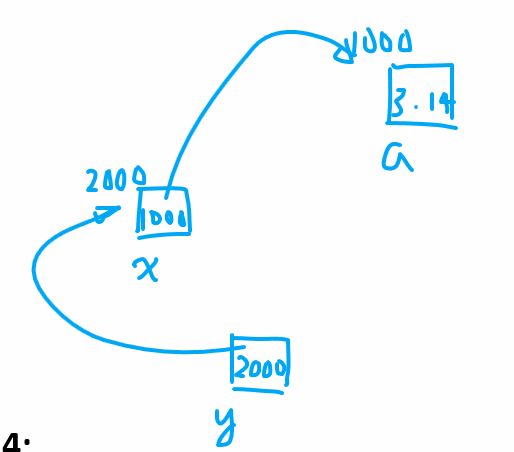
**March 10, 2021**

1. **Lab 7 extended for one week**
2. **Pointers (continued)**

**Fact 1: pointer variables are internally equivalent to integer variables in the aspect of memory usage. In other words, any pointer variable occupies 4 bytes of your memory.**

**Fact 2: We could have more than one star in declaration of a pointer variable.**

****

**Double \*x;**

**Double \*\*y;**

**Double \*\*\*z;**

**Double a = 3.14;**

**X = &a;**

**Y = &x; z = &y;**

**Fact 3: NULL is a special value for pointer variables. NULL is equal to 0, and 0 is not a valid memory address. Because of this feature, we use NULL to represent the e end of a linked list.**

**Fact 4: Dynamic arrays can be referred to by pointers. In this context, the name of point variables are the alias of array names.**

**Dynamic array:**

**Int x, y;**

**Cin >> x >> y; // the values of x and y are determined at run-time**

**Float a[x]; // not ok in c++**

**// one-dimensional dynamic array**

**Float \*a = new float[x]; // ok in c++ (dynamic memory allocation)**

**// counterpart of new: malloc( ) in c**

**For(int i=0; i<x; i++) a[i] = I; // a is a pointer name and an array name**

**delete []a; // it performs memory deallocation to avoid memory leak**

**// two-dimensional dynamic array**

**Float \*\*b = new float\*[x]; // each time we peel off one star**

**for(int i=0; i< x; i++)**

**b[i] = new float[y]; // b[i] --- float\*; b[i][j] – float**

**for(int i=0; i<x; i++)**

**for(int j=0; j<y; j++)**

**b[i][j] = i\*j;**

**…**

**For(int j=0; j<y; j++)**

**delete []b[j]; // remove the 2nd dimension first**

**delete []b; // delete the first dimension**

|  |
| --- |
|  |

**// 3-D Dynamic arrays**

**Float \*\*\*c = new float\*\*[x];**

**For(int i=0; i<x; i++)**

**C[i] = new float\*[y];**

**For(int j=0; j<y; j++)**

**C[i][j] = new float[z];**

**For(int i=0; i<x; i++)**

**For(int j=0; j<y; j++)**

**For(int k=0; k<z; k++)**

**C[i][j][k] = i\*j\*k;**

**// deallocation of c[i][j][k]**

**Void pointer: it is a special type of pointers that point to a generic data type.**

**Int \*p1; // p1 points to an integer variable**

**Float \*p2; // p2 pints to a float variable**

**Void \* p3; // p3 points to any data type**

**Void swap3(void \*p1, void \*p2)**

**{**

**}**

**Int comp\_int(void \*a, void \*b)**

**{**

**If****( \*(int\*) a > \*(int\*) b) // red-color: type casting of pointers**

**Return 1;**

**Else if ( \*(int\*) a < \*(int\*) b)**

**Return -1;**

**Else**

**Return 0;**

**}**

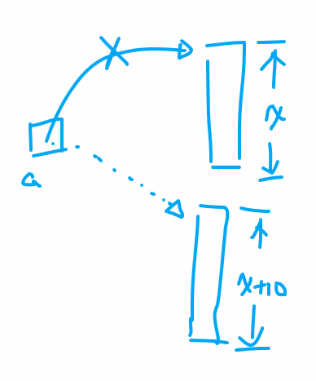
**Risks of using pointers:**

1. **Memory leak**

**Float \*a = new float[x];**

**// delete []a; for fixing the memory leak**

**a = new float[x+10];**

****

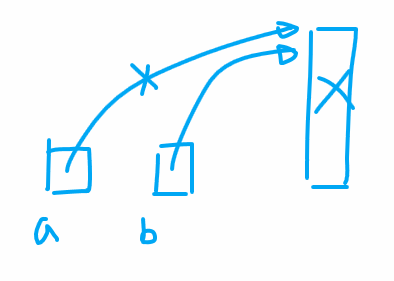
1. **Dangling pointers**

**Float \*a = new float[x];**

**Float \*b = a;**

**Delete []a;**

**Cout << b[0] << endl; // b becomes a dangling pointer**

****