CS360L-(E) Week 7

Array and pointer

Array

C++ provides **array**, which stores a fixed-size sequential collection of elements of the same data type.

An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as a0, a1, ..., and a99, you declare one array variable such as a[100] and use a[0], a[1], and ..., a[99] to represent individual variables. A specific element in an array is accessed by an index.

Declaring an array

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows.

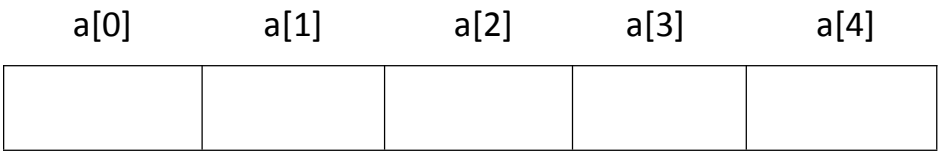
Syntax:

```
type    array_name[ array_size];
```

```
int a[10];    // create an array of 10 integer elements.
```

Initializing an array

To initialize an array with following structure:



Index: 0 1 2 3 4

1) You can initialize C++ array elements using a single statement as follows:

```
int a[5] = { 1, 2, 3, 4, 5};      //cannot assign more than 5 numbers
```

Or

```
int a[ ] = { 1, 2, 3, 4, 5};       //compiler will generate the size
```

2) Initialize an array element by element using the index as following:

```
int a[5];
a[0] = 1;       // array element index starts from 0 to (size - 1)
a[1] = 2;
a[2] = 3;
a[3] = 4;
a[4] = 5;
```

Or

Using loop to initialize array which stores the data of certain relations.

```
for ( int i = 0; i < 5; i++ )
{
    a[i] = i + 1;
}
```

Accessing an array

Using array index to access specific array elements:

```
for ( int i = 0; i < 5; i++ )
{
    cout << a[i] << endl;
}
```

Address(&) operator

As you know every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator which denotes an address in memory. All memory address are represented by long hexadecimal numbers.

```
int x = 1;
cout << x << " " << &x;
```

For a declared variable, the address is unique and different from others, which means we can use the address to specify the variable, therefore, we have pointer.

Pointer

A pointer is a variable whose value is the address of another variable. Like any other variable, you must declare a pointer before you can work with it.

Declaring an pointer

The general form of a pointer variable declaration is:

type *pointer_name;

By here, **type** is the pointer's base type; it must be a valid C++ type and **pointer_name** is the name of the pointer variable.

The * you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer.

```
int    *ip1, *ip2;        // pointers to an integer
double    *dp;        // pointer to a double
float    *fp;        // pointer to a float
```

```
char    *ch         // pointer to character
```

The type means which type of data the pointer points to, not the actually value type of pointer. Value of an initialized pointer is always the memory address.

Initializing an pointer

Same as other normal variable, we use = to initialize a pointer variable, that is:

```
int a, *ptr;
ptr = &a;
```

By initializing a pointer, we are pointing the pointer to an existing variable with the same data type.

Indirection(*) operator

Also take as deference operator.for example:

```
int y = 10;
int *yptr = &y;
```

From here we can use *yptr to deference, like:

```
cout << *yptr;          // will display value of y , 10
*yptr = 9;              // will overwrite y value to 9
```

Or

```
cin >> *yptr;
```

Pointer points to an array

Point a pointer to an array also works:

```
int v[5], *p;

p = v;
```

Or

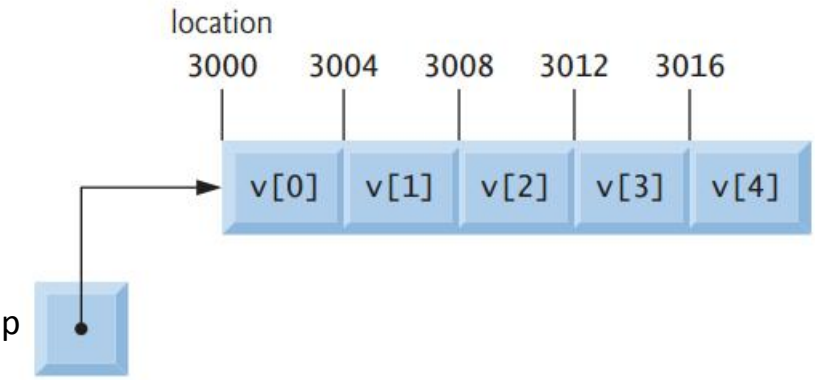
```
p = &v[0];
```

Upon statements points pointer variable p to the first element of array v;

Pointer arithmetic

In C++, there are 4 arithmetic operators that can be use with pointer:

Use the previous example, we can do:




```
++p;    // p now points to v[1]

--p;     // p points to v[0] again

p += 4    // p points to v[4] now

p -= 2    // p now points to v[2]
```

For example : value of p now is 3000, since p points to v[0], so we plus 1

to p, means:

3000 + 1 * 4 = 3004 and we assign 3004 to p

Therefore p points to v[1], and value of p will be 3004.

Relations between array and pointer

In C++, pointer and array are intimately related, point can be used to do any operation involving array subscripting.

```
int b[3];
int *bptr;
bptr = b;          // bptr points to b[0]
```

Array elements			
b[i]	bptr[i]	*(bptr + i)	*(b + i)
b[0]	bptr[0]	*bptr	*b
b[1]	bptr[1]	*(bptr + 1)	*(b + 1)
b[2]	bptr[2]	*(bptr + 2)	*(b + 2)
Array elements address			
&b[i]	&bptr[i]	bptr + i	
&b[0]	&bptr[0]	bptr	
&b[1]	&bptr[1]	bptr + 1	
&b[2]	&bptr[2]	bptr + 2	

Within the same line, are the different way to represent same thing.