Lecture 15: Pointer – Part II

Class page: https://github.com/tsung-wei-huang/cs1410-40

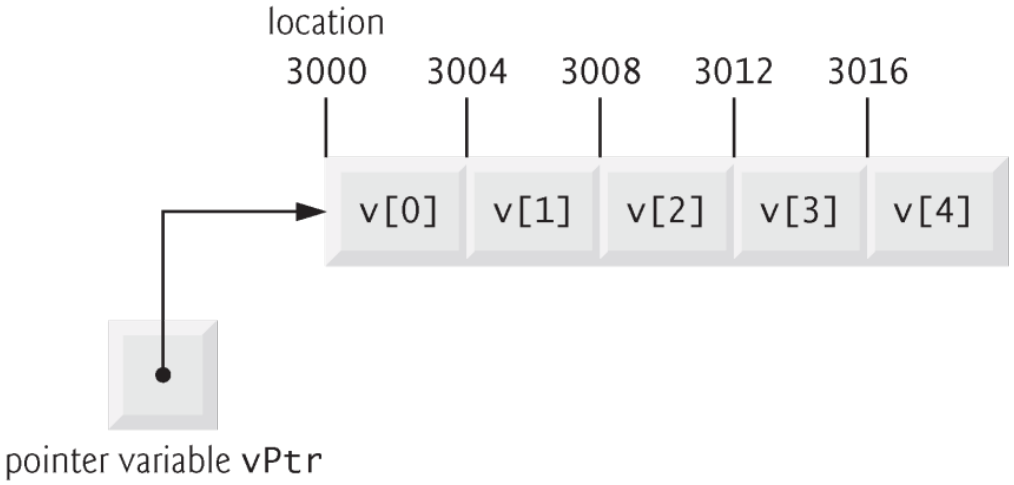
Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Pointer Expression and Arithmetic

- **Pointers are valid operands in arithmetic expressions, assignment expressions and comparison expressions.**
- **pointer arithmetic—certain arithmetic operations may be performed on pointers:**
  - increment (++)
  - decremented (--)
  - an integer may be added to a pointer (+ or +=)
  - an integer may be subtracted from a pointer (- or -=)
  - one pointer may be subtracted from another of the same type

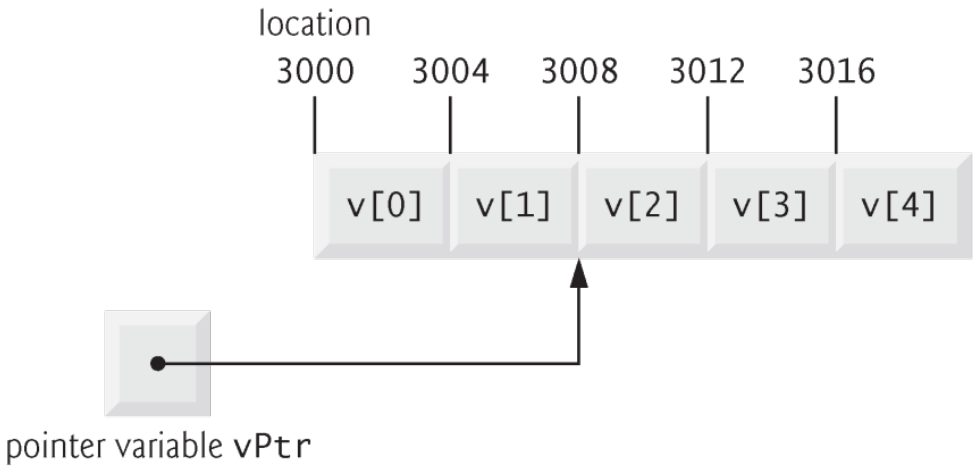
Pointer Expression and Arithmetic

- Assume that array `int v[5]` has been declared and that its first element is at memory location 3000.
- Assume that pointer `vPtr` has been initialized to point to `v[0]` (i.e., the value of `vPtr` is 3000).



Pointer Expression and Arithmetic

- **In conventional arithmetic, the addition 3000 + 2 yields the value 3002.**
  - Not true for pointer arithmetic
- **When an integer is added to, or subtracted from, a pointer, the pointer is incremented or decremented by that integer times the size of the object to which the pointer refers**
  - The number of bytes depends on the object's data type.



Pointer Expression and Arithmetic

- Pointer variables pointing to the same array may be subtracted from one another.
- For example, if `vPtr` contains the address 3000 and `v2Ptr` contains the address 3008, the statement
  - `x = v2Ptr - vPtr;`
- would assign to `x` the number of array elements from `vPtr` to `v2Ptr`—in this case, 2.
- Pointer arithmetic is meaningless unless performed on a pointer that points to an array.

sizeof Operator

- The unary operator sizeof determines the size of an array (or of any other data type, variable or constant) in bytes during program compilation.
- When applied to the name of an array, the sizeof operator returns the total number of bytes in the array as a value of type size_t.
- When applied to a pointer parameter in a function that receives an array as an argument, the sizeof operator returns the size of the pointer in bytes—not the size of the array.

Example

```
1
2   // Sizeof operator when used on an array name
3   // returns the number of bytes in the array.
4   #include <iostream>
5   using namespace std;
6
7   size_t getSize( double * ); // prototype
8
9   int main()
10  {
11     double array[ 20 ]; // 20 doubles; occupies 160 bytes on our system
12
13     cout << "The number of bytes in the array is " << sizeof( array );
14
15     cout << "\nThe number of bytes returned by getSize is "
16        << getSize( array ) << endl;
17  } // end main
18
19  // return size of ptr
20  size_t getSize( double *ptr )
21  {
22     return sizeof( ptr );
23  } // end function getSize
```

sizeof Operator

- The number of elements in an array also can be determined using the results of two `sizeof` operations.
- Consider the following array declaration:
  - `double realArray[ 22 ];`
- To determine the number of elements in the array, the following expression (which is evaluated at compile time) can be used:
  - `sizeof realArray / sizeof( realArray[ 0 ] )`
- The expression determines the number of bytes in array `realArray` and divides that value by the number of bytes used in memory to store the array's first element.

Bytes of Types

```
1
2   // Demonstrating the sizeof operator.
3   #include <iostream>
4   using namespace std;
5
6   int main()
7   {
8      char c; // variable of type char
9      short s; // variable of type short
10     int i; // variable of type int
11     long l; // variable of type long
12     float f; // variable of type float
13     double d; // variable of type double
14     long double ld; // variable of type long double
15     int array[ 20 ]; // array of int
16     int *ptr = array; // variable of type int *
17
18     cout << "sizeof c = " << sizeof c
19        << "\tsizeof(char) = " << sizeof( char )
20        << "\nsizeof s = " << sizeof s
21        << "\tsizeof(short) = " << sizeof( short )
22        << "\nsizeof i = " << sizeof i
```

Bytes of Types

```
23          << "\tsizeof(int) = " << sizeof( int )
24          << "\nsizeof l = " << sizeof l
25          << "\tsizeof(long) = " << sizeof( long )
26          << "\nsizeof f = " << sizeof f
27          << "\tsizeof(float) = " << sizeof( float )
28          << "\nsizeof d = " << sizeof d
29          << "\tsizeof(double) = " << sizeof( double )
30          << "\nsizeof ld = " << sizeof ld
31          << "\tsizeof(long double) = " << sizeof( long double )
32          << "\nsizeof array = " << sizeof array
33          << "\nsizeof ptr = " << sizeof ptr << endl;
34    } // end main
```

```
sizeof c = 1     sizeof(char) = 1
sizeof s = 2     sizeof(short) = 2
sizeof i = 4     sizeof(int) = 4
sizeof l = 4     sizeof(long) = 4
sizeof f = 4     sizeof(float) = 4
sizeof d = 8     sizeof(double) = 8
sizeof ld = 8    sizeof(long double) = 8
sizeof array = 80
sizeof ptr = 4
```

Relationship between Pointers and Array

- **The array name (without a subscript) is a constant pointer to the first element of the array.**
  - Although it's a pointer, it cannot be modified in arithmetic expressions.
- **Pointers can be used to do any operation involving array subscripting.**
- **Assume the following declarations:**

```
int b[ 5 ]; // create 5-element int array b
int *bPtr; // create int pointer bPtr
```

- **We can set `bPtr` to the address of the first element in array `b` with the statement**

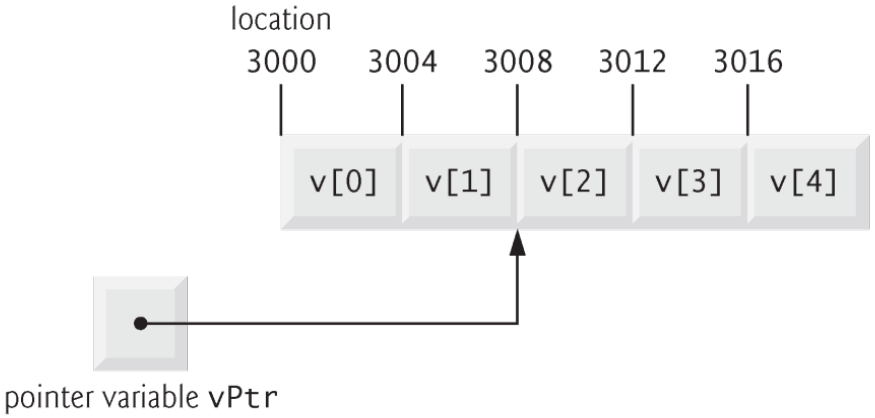
```
bPtr = b; // assign address of array b to bPtr
```

- **equivalent to**

```
bPtr = &b[ 0 ]; // also assigns address of
array b to bPtr
```

Relationship between Pointers and Array

- **Array element `b[ 2 ]` can alternatively be referenced with the pointer expression**
  - `*( bPtr + 2 )`
- **The `2` in the preceding expression is the offset to the pointer.**
- **This notation is referred to as pointer/offset notation.**
  - The parentheses are necessary, because the precedence of * is higher than that of +.



Relationship between Pointers and Array

- **Pointers can be subscripted exactly as arrays can.**
- **For example, the expression `bPtr[ 1 ]` refers to the array element `b[ 1 ]`; this expression uses pointer/ subscript notation.**
- **In this section, four notations are discussed for referring to array elements to accomplish the same task :**
  - Array subscript notation,
  - Pointer/offset notation with the array name as a pointer,
  - Pointer subscript notation, and
  - Pointer/offset notation with a pointer

Relationship between Pointers and Array

```
1  // Fig. 7.18: fig07_18.cpp
2  // Using subscripting and pointer notations with arrays.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int b[] = { 10, 20, 30, 40 }; // create 4-element array b
9     int *bPtr = b; // set bPtr to point to array b
10
11    // output array b using array subscript notation
12    cout << "Array b printed with:\n\nArray subscript notation\n";
13
14    for ( int i = 0; i < 4; i++ )
15       cout << "b[" << i << "] = " << b[ i ] << '\n';
16
17    // output array b using the array name and pointer/offset notation
18    cout << "\nPointer/offset notation where "
19       << "the pointer is the array name\n";
20
21    for ( int offset1 = 0; offset1 < 4; offset1++ )
22       cout << "*(b + " << offset1 << ") = " << *( b + offset1 ) << '\n';
```

Relationship between Pointers and Array

```
23
24      // output array b using bPtr and array subscript notation
25      cout << "\nPointer subscript notation\n";
26
27      for ( int j = 0; j < 4; j++ )
28         cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';
29
30      cout << "\nPointer/offset notation\n";
31
32      // output array b using bPtr and pointer/offset notation
33      for ( int offset2 = 0; offset2 < 4; offset2++ )
34         cout << "*(bPtr + " << offset2 << ") = "
35            << *( bPtr + offset2 ) << '\n';
36   } // end main
```

```
Array b printed with:

Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
```

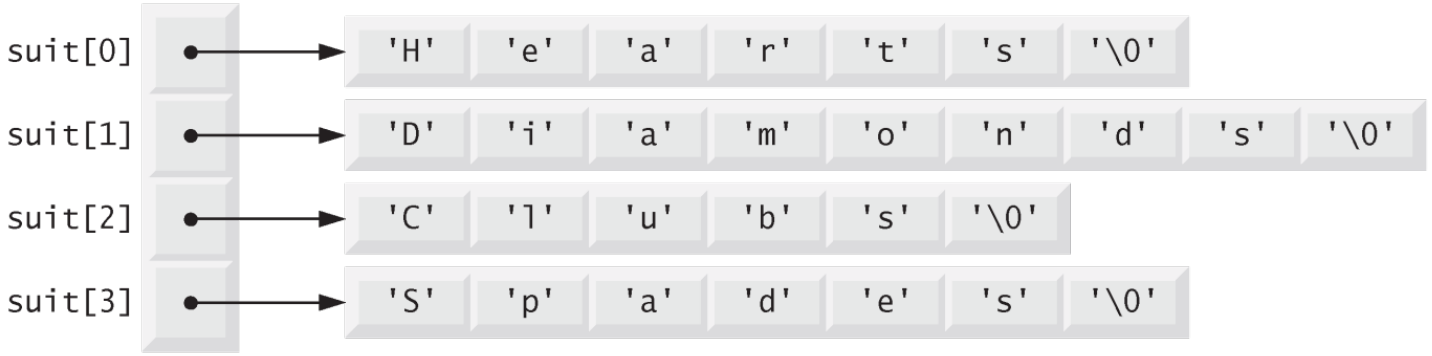
Relationship between Pointers and Array

```
Pointer/offset notation where the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

Pointer-based String

- **Character constant**
  - An integer value represented as a character in single quotes.
  - The value of a character constant is the integer value of the character in the machine's character set.
- **A string is a series of characters treated as a single unit.**
  - May include letters, digits and various special characters such as +, -, *, /and $.
- **String literals, or string constants, in C++ are written in double quotation marks**
- **A pointer-based string is an array of characters ending with a null character (\0).**
  - Ex: "happy" → 'h', 'a', 'p', 'p', 'y', '\0'

Pointer-based String



cout << suit[0]; → print out "Hearts"

cout << suit[2]; → print out "Clubs"

What will be printed by the following program?

```
for (int i=0; i<10; i++) {
        idx = rand()%4;
        cout << suit[idx];
}
```

Summary

- Pointer Arithmetic
- Query the size of types
- Relationship between Pointer and Array
- Pointer-based string