UNIT 4: POINTERS AND ARRAYS

Nguyen Thi Thu Huong
Department of Computer Science
School of of Information and Communication Technology
Hanoi University of Technology

POINTERS

- Powerful, but difficult to master
- Simulate call-by-reference
- Close relationship with arrays and strings

Nguyen Thi Thu Huong

POINTERS AND ADDRESSES

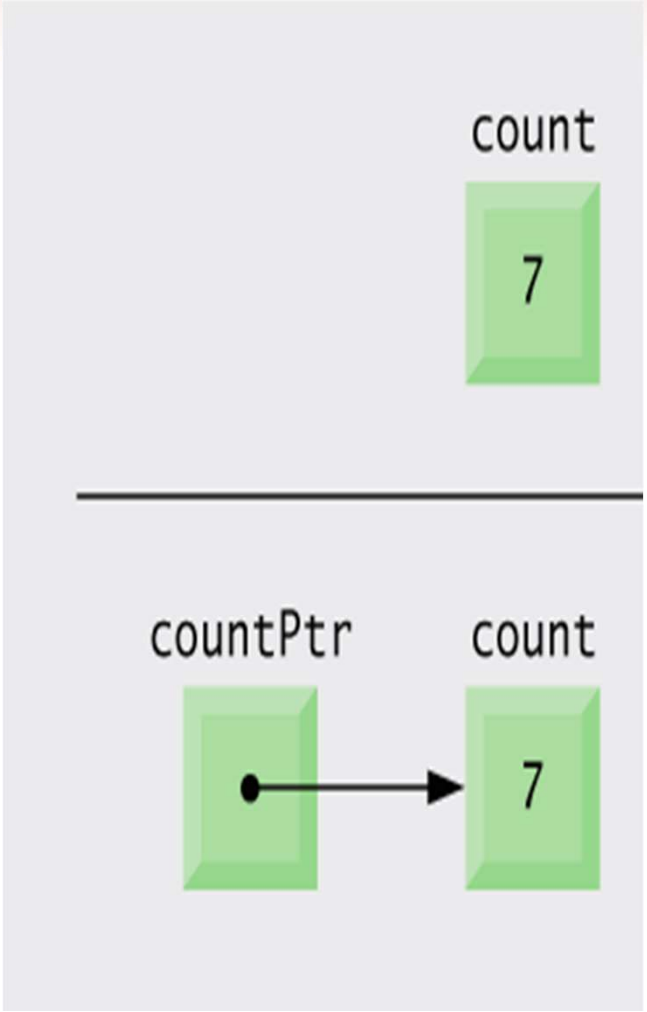
- One variable always has two properties:
  - The address of the variable
  - The value of the variable.
- Example:

```
int i, j;
i = 3;
j = i;
```

Variable	Address	Value
i	FFEC	3
j	FFEE	3

A pointer is a variable that contains the address of a variable

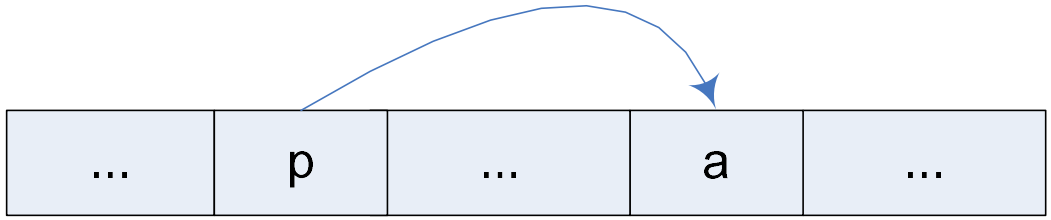
Directly and indirectly referencing a variable



POINTERS

- **A pointer is a special type of variable that can hold an address**
- **Pointer declaration**

  type * name ;

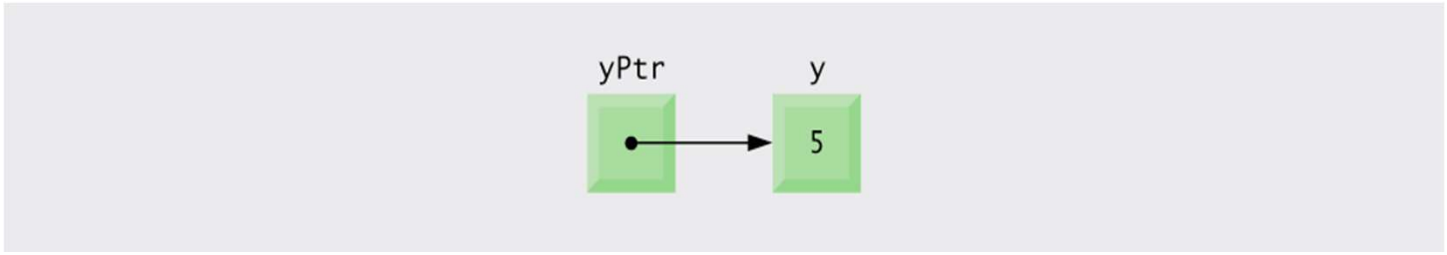


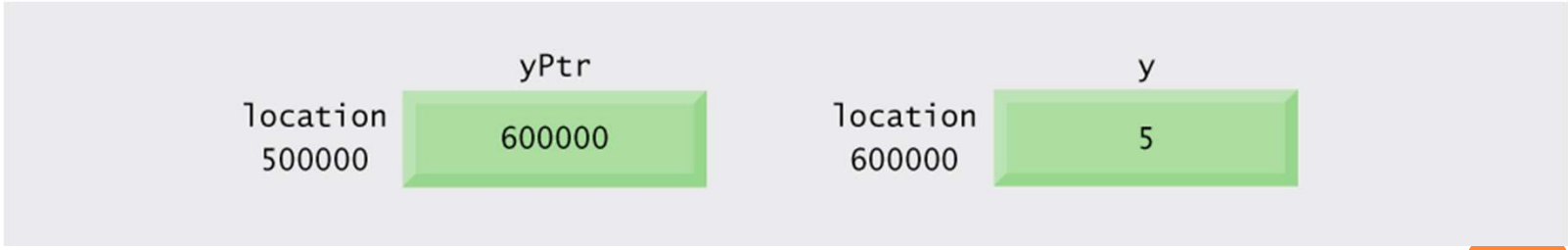
Pointer Operators

- & (address operator)
  - Returns address of operand

```
int y = 5;
int *yPtr;
yPtr = &y;     /* yPtr gets address of y */
yPtr "points to" y
```

Graphical representation of a pointer





POINTER OPERATORS

- * (indirection/dereferencing operator)
  - Returns a synonym/alias of what its operand points to
  - `*yptr` returns `y` (because `yptr` points to `y`)
  - * can be used for assignment
    - Returns alias to an object

```
*yptr = 7;  /* changes y to 7 */
```

  - Dereferenced pointer (operand of *) must be an lvalue (no constants)

Outline

```
1  /*
2     Using the & and * operators */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int a;        /* a is an integer */
8     int *aPtr;    /* aPtr is a pointer to an integer */
9
10    a = 7;
11    aPtr = &a;    /* aPtr set to address
12
13    printf( "The address of a is %x"
14            "\nThe value of aPtr is %x", &a, aPtr );
15
16    printf( "\n\nThe value of a is %d"
17            "\nThe value of *aPtr is %d", a, *aP
18
19    printf( "\n\nShowing that * and & are complements of "
20            "each other\n&*aPtr = %x"
21            "\n*&aPtr = %x\n", &*aPtr, *&aPtr );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

If **aPtr** points to **a**, then **&a** and **aPtr** have the same value.

a and ***aPtr** have the same value

&*aPtr and ***&aPtr** have the same value

Result of the program on previous slide

```
The address of a is 0012FF7C
The value of aPtr is 0012FF7C

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other.
&*aPtr = 0012FF7C
*&aPtr = 0012FF7C
```

Nguyen Thi Thu Huong

Lists and Arrays

- Problem solving often requires information be viewed as a list
  - List may be one-dimensional or multidimensional
- C provides a mechanisms: Arrays
  - Traditional and important because of legacy libraries
  - Restrictions on its use
- C++ provides another mechanism: Container classes
  - Common containers includes vector, queue, stack, map, …

Nguyen Thi Thu Huong

Array Terminology

- List (array) is composed of *elements*
- Elements in a list have a *common name*
  - The list as a whole is referenced through the common name
- In the scope ò the course list elements are of the same type — the *base* type
- Elements of a list are referenced by *subscripting* or *indexing* the common name

Nguyen Thi Thu Huong

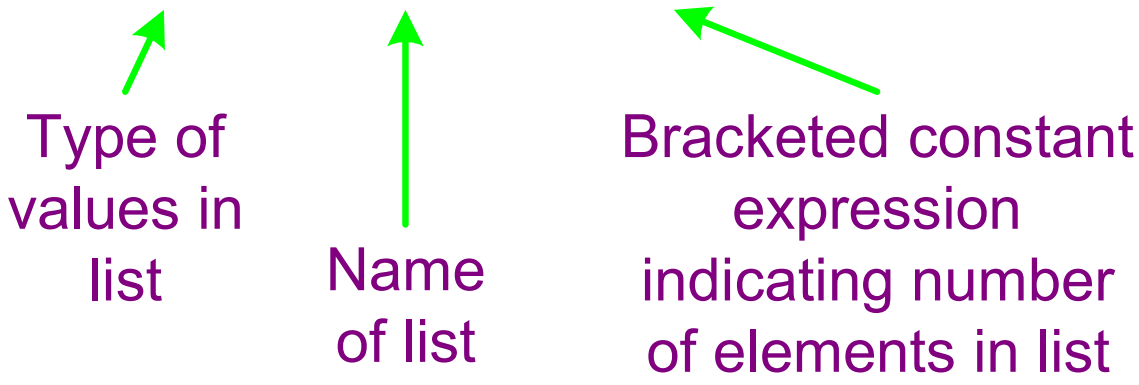
Restrictions

- Subscripts are denoted as expressions within brackets: [ ]
- Base type can be any fundamental, library-defined, or programmer - defined type
- The index type is integer and the index range must be 0 ... `n`-1 where `n` is a programmer-defined constant expression.

Nguyen Thi Thu Huong

Basic Array Declaration

```
BaseType Id [ SizeExp ] ;
```




```
double X [ 100 ] ;
```

Nguyen Thi Thu Huong

EXAMPLE OF ARRAY DECLARATIONS

- Suppose

```
const int N = 20;
const int M = 40;
const int MaxStringSize = 80;
const int MaxListSize = 1000;
```

- Then the following are all correct array declarations

```
int A[10];                    // array of 10 ints
char B[MaxStringSize];        // array of 80 chars
double C[M*N];                // array of 800 floats
int Values[MaxListSize];      // array of 1000 ints
Rational D[N-15];             // array of 5 Rationals
```

Nguyen Thi Thu Huong

SUBSCRIPTING

- Suppose

```
int A[10];     // array of 10 ints A[0], … A[9]
```

- To access individual element must apply a subscript to list name **A**
  - A subscript is a bracketed expression also known as the index
  - First element of list has index 0

```
A[0]
```

  - Second element of list has index 1, and so on

```
A[1]
```

  - Last element has an index one less than the size of the list

```
A[9]
```

  - Incorrect indexing is a common error

```
A[10]      // does not exist
```

Nguyen Thi Thu Huong

INPUT DATA INTO AN ARRAY

```
puts("Enter number of elements:");
scanf("%d",&n);
for(i =0;i<n;i++)
{printf("\na[%d]=",i);
scanf("%d",&a[i]);}
```

DISPLAYING A LIST

```
// List A of n elements has already
  been set
for (int i = 0; i < n; ++i)
{printf("%d  ", A[i]);
}
```

Nguyen Thi Thu Huong

SMALLEST VALUE

- Problem
  - Find the smallest value in a list of integers
- Input
  - A list of integers and a value indicating the number of integers
- Output
  - Smallest value in the list
- Note
  - List remains unchanged after finding the smallest value!

Nguyen Thi Thu Huong

Necessary Information

- Information to be maintained
  - Number of values in array
  - Array with values to be inspected for smallest value
  - Index of current element being considered
  - Smallest value so far

A More Detailed Design

- Solution
  - Initialize smallest value so far to first element
  - For each of the other elements in the array in turn
    - If it is smaller than the smallest value so far, update the value of the smallest value so far to be the current element
  - Print smallest value

Passing An Array. . .

```
int SmallestValueSoFar = A[0];
for (int i = 1; i < asize; ++i) {
   if (A[i] < SmallestValueSoFar ) {
     SmallestValueSoFar = A[i];
  }
}
```

Nguyen Thi Thu Huong

SEARCHING

- Problem
  - Determine whether a value key is one of the element values
- Does it matter if
  - Element values are not necessarily numbers
  - Element values are not necessarily unique
  - Elements may have key values and other fields

Nguyen Thi Thu Huong

Sequential List Searching

```
found=0;
for (int i = 0; i < m; ++i) {
  if (A[i] == Key) {
    {found=1; break};
  }
```

Nguyen Thi Thu Huong

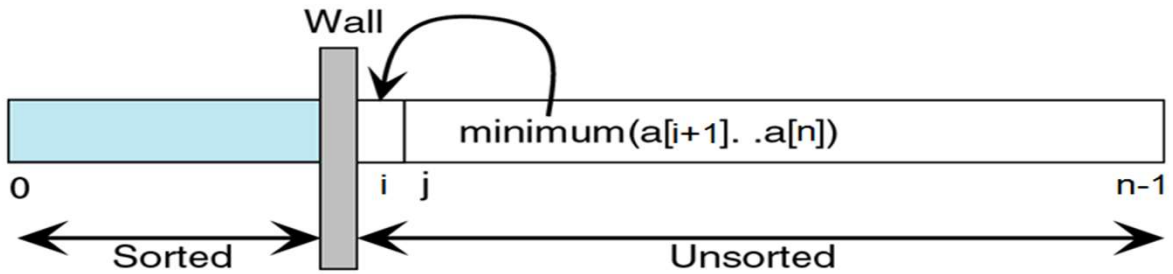
SORTING

- Problem
  - Arranging elements so that they are ordered according to some desired scheme
    - Standard is non-decreasing order
      - Why don't we say increasing order?
- Major tasks
  - Comparisons of elements
  - Updates or element movement

Nguyen Thi Thu Huong

Common Sorting Techniques

- Selection sort
  - On ith iteration place the ith smallest element in the ith list location



- Bubble sort
  - Iteratively pass through the list and examining adjacent pairs of elements and if necessary swap them to put them in order. Repeat the process until no swaps are necessary

Selection Sort

```
for ( i = 0; i < n-1; i++)
   for ( j = i + 1; j < n; j++)
     if (A[j] < A[i])
       {temp=A[i];
       A[i]=A[j];
       A[j]=temp;}
```

Nguyen Thi Thu Huong

BUBBLE SORT

```
for (i = n-1 ; i >=1; i--)
  {
    for (j = 0 ; j <i; j++)
    {
      if (A[j] > A[j+1])
      {
        temp      = A[j];
        A[j]   = A[j+1];
        A[j+1] = temp;
      }
    }
  }
```

Other Sorting Techniques

- Insertion sort
  - On ith iteration place the ith element with respect to the i-1 previous elements
    - In text
- Quick sort
  - Divide the list into sublists such that every element in the left sublist ≤ to every element in the right sublist. Repeat the Quick sort process on the sublists
    - In text