Pointers

Section 1D

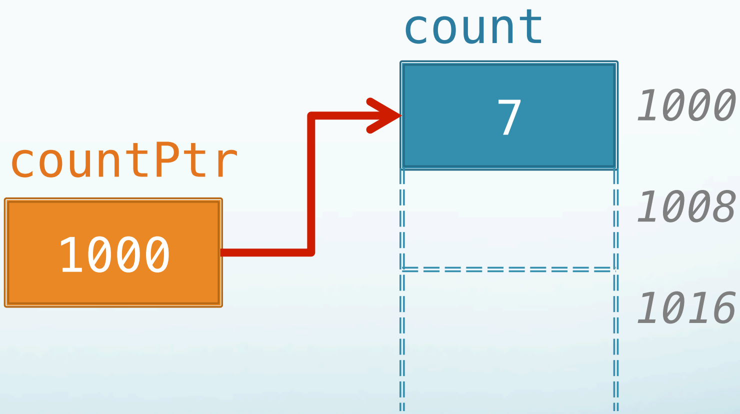
Declaring and Initializing Pointers

- A **pointer** is a variable that stores a memory address:

```
int count = 7;              // A variable of type integer.
int *countPtr = &count;     // A pointer to an integer var.
```

Pointer type

Address operator



count

7

1000

countPtr

1000

1008

1016

Basic Example

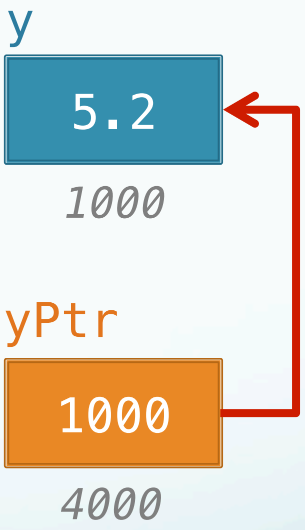
```
double y = 5.2;          // Variable that holds a double.
double *yPtr;            // Variable that holds a pointer to a double.

// What's the address of y?
cout << "The address of y is " << &y << endl;

// Assign address of y to yPtr.
yPtr = &y;

// What's the address that yPtr holds?
cout << "yPtr is " << yPtr << endl;

// What's the double that yPtr points to?
cout << "*yPtr is " << *yPtr << endl;
```

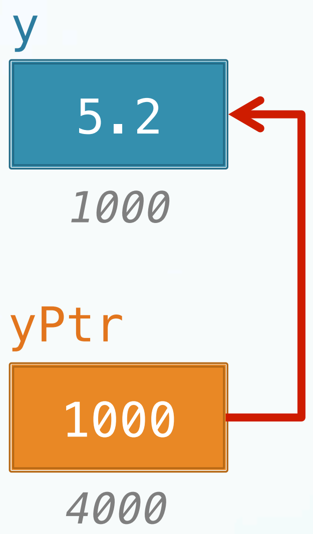


Dereferencing a pointer

Exercise 1

```
// What does the following print out?
cout << "*&y is " << *&y << endl;
cout << “&*y is " << &*y << endl;

// And...
cout << "&*yPtr is " << &*yPtr << endl;
cout << "*&yPtr is " << *&yPtr << endl;
```



y

5.2

1000

yPtr

1000

4000

Remember!
& and ***** are
complimentary
operators

Functions and Pointers

- Simulating pass-by-reference:

```
void cube( int* nPtr )
{
   *nPtr = (*nPtr) * (*nPtr) * (*nPtr);
}

int main()
{
   int number;

   cout << "Provide an integer number: ";
   cin >> number;

   cout << "The original number is: " << number << endl;
   cube( &number );    // We pass the address of number.
   cout << "The new value of number is: " << number << endl;
}
```

Exercise 2

- Find the error, if any, in the following statements:

a

```
int *number;
cout << *number;
```

b

```
double *dPtr;
int *iPtr;
iPtr = dPtr;
```

c

```
int *x, y;
x = y;
```

d

```
int *nPtr, result;
result = 3;
nPtr = &result;
result *= nPtr;
cout << result;
```

e

```
double x = 19.34;
double *xPtr = &x;
cout << “The address of x is “
     << *xPtr << endl;
```

Exercise 3

- Write a function that swaps the values between two integer variables. For example:
  - *Before function*: x = 1, y = 2.
  - *After function*: x = 2, y = 1.

Exercise 4

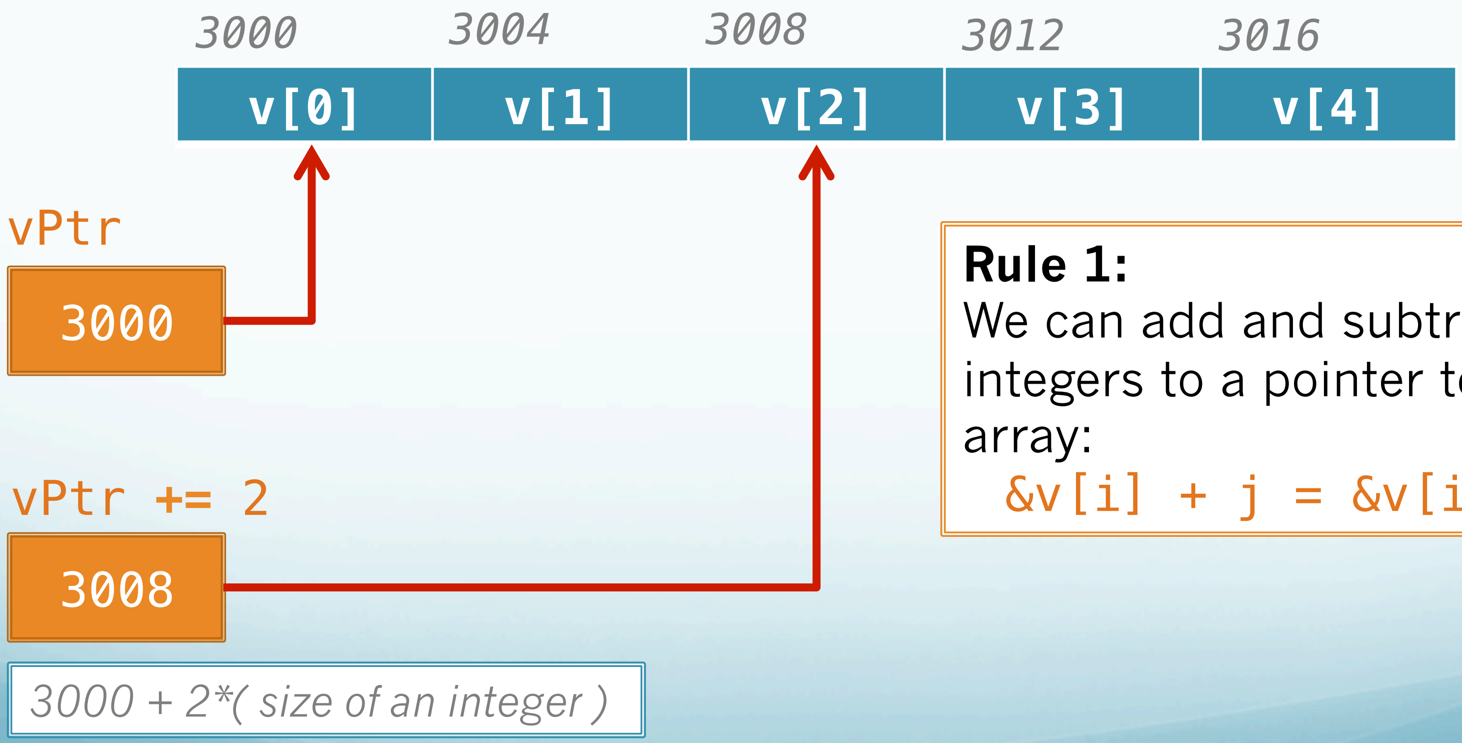
Write a function to find the **real roots** of a quadratic equation by using the closed form:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Your function must simulate pass-by-reference by using pointers to the outputs x_1 and x_2 and return **true** if they are real. Otherwise, return **false**, and leave x_1 and x_2 unchanged.

Pointers and Arrays

```
int v[5];                 // Array of 5 integers.
int *vPtr = &v[0];        // A pointer to the first element in v.
```



3000 3004 3008 3012 3016

v[0] v[1] v[2] v[3] v[4]

vPtr

3000

vPtr += 2

3008

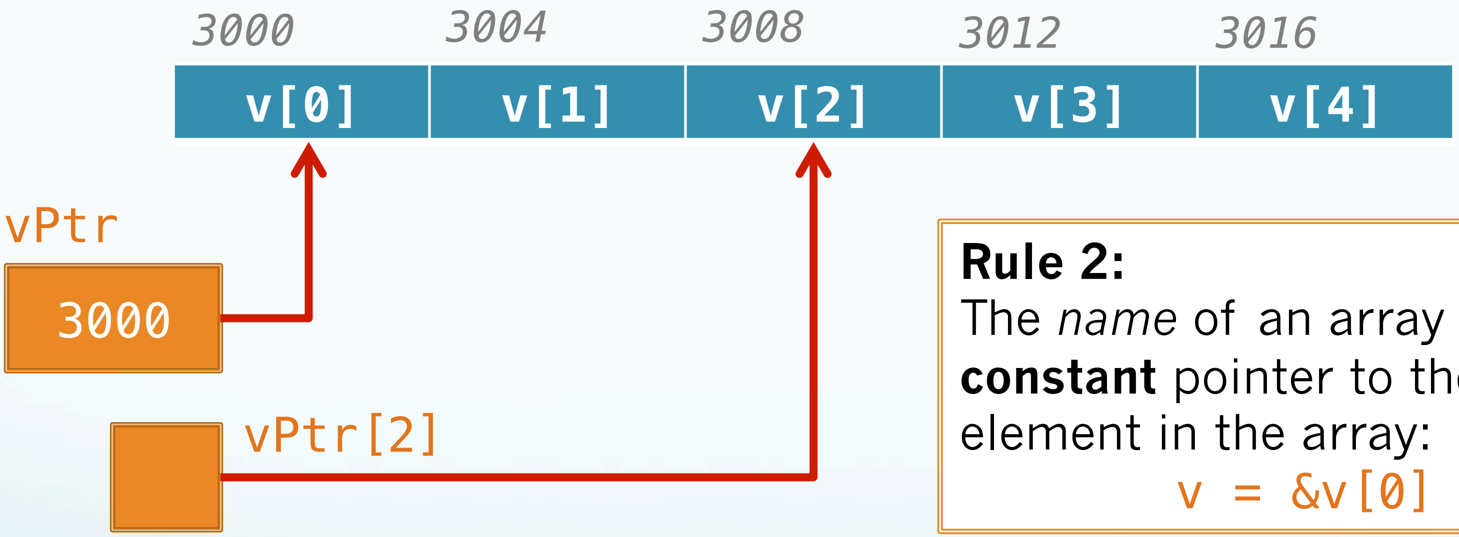
3000 + 2*(size of an integer)

Rule 1:
We can add and subtract integers to a pointer to an array:

`&v[i] + j = &v[i + j]`

Pointers and Arrays

```
int v[5];                // Array of 5 integers.
int *vPtr = v;           // A pointer to the first element in v.
```



Rule 2:
The *name* of an array is a **constant** pointer to the first element in the array:

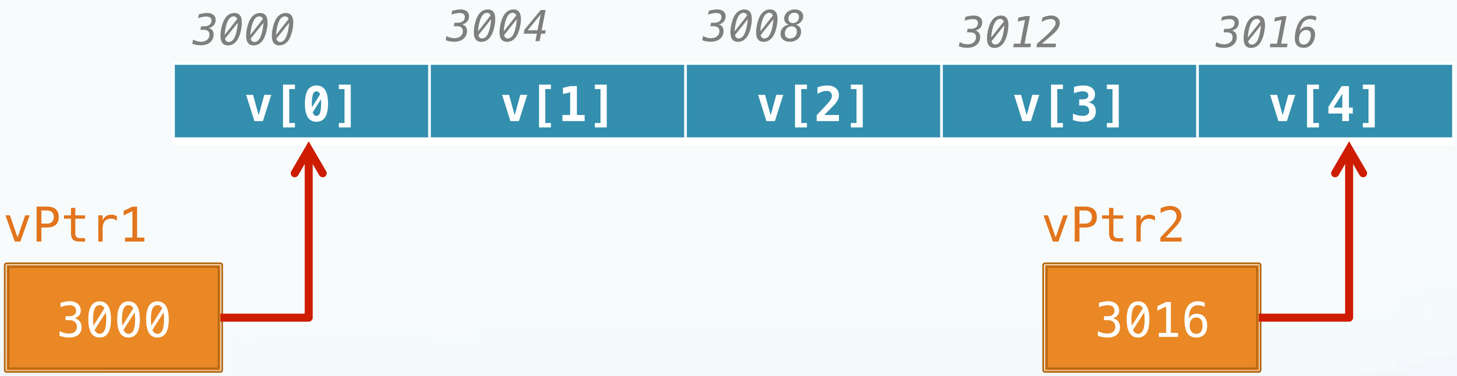
`v = &v[0]`

Rule 3:
We can use indexes with a pointers:

`vPtr[i] = *(vPtr + i)`

Pointers and Arrays

```
int v[5];              // Array of 5 integers.
int *vPtr1 = v;        // A pointer to the first element in v.
int *vPtr2 = &v[4];    // A pointer to the last element in v.
```



3000 3004 3008 3012 3016

v[0] v[1] v[2] v[3] v[4]

vPtr1

3000

vPtr2

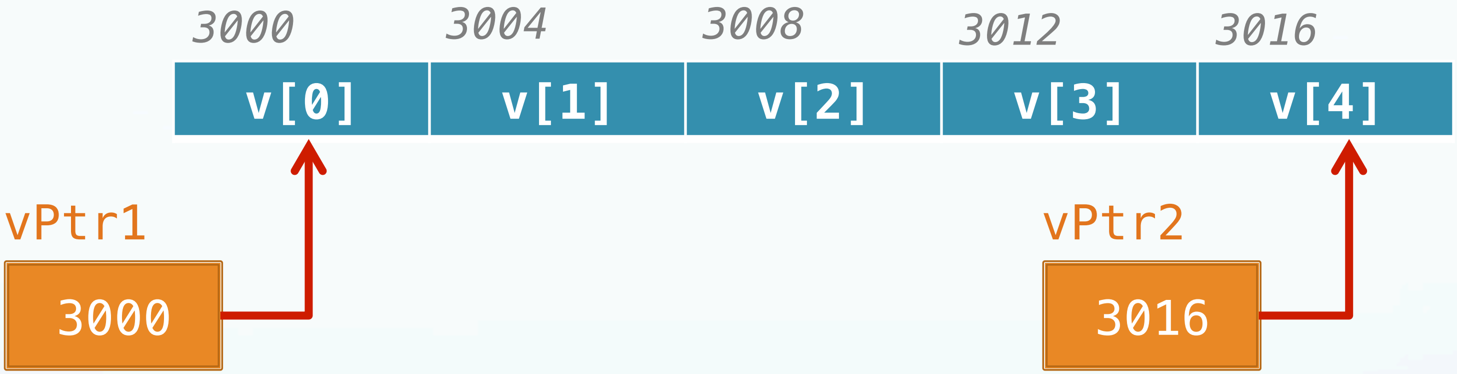
3016

Rule 4:
We can use relational operators to compare pointers:

```
vPtr1 == vPtr2
vPtr1 <= vPtr2
vPtr1 >= vPtr2
```

Pointers and Arrays

```
int v[5];             // Array of 5 integers.
int *vPtr1 = v;       // A pointer to the first element in v.
int *vPtr2 = &v[4];   // A pointer to the last element in v.
```



3000	3004	3008	3012	3016
v[0]	v[1]	v[2]	v[3]	v[4]

vPtr1
3000

vPtr2
3016

vPtr2 - vPtr1 = (3016 - 3000) / (size of an integer)

Rule 5:
We can subtract two pointers that are of the **same type** and **point into the same array**:

`vPtr2 – vPtr1 = &v[j] – &v[i] = j – i`

Exercise 5

- Write a program that converts letters in a C string to uppercase. Use pointers to traverse the array of characters.

```
int main()
{
        char s[] = "this is the string to transform";

        for( char *sPtr = s; *sPtr != '\0'; sPtr++ )
                *sPtr = toupper( *sPtr );

        cout << s << endl;

        return 0;
}
```

Exercise 6

- Make the snippet of code you wrote in Exercise 5 into a function that converts letters of a C string to uppercase. Your function must return a pointer to the just transformed string.

```
char* toUppercase( char *sPtr )
{
        // Remember where the string began.
        char *beginning = sPtr;

        for( ; *sPtr != '\0'; sPtr++ )
                *sPtr = toupper( *sPtr );

        return beginning;
}
```

Questions?

- You may find this material and solutions to the programming exercises at:

  http://cs.ucla.edu/~langel/cs31/session7