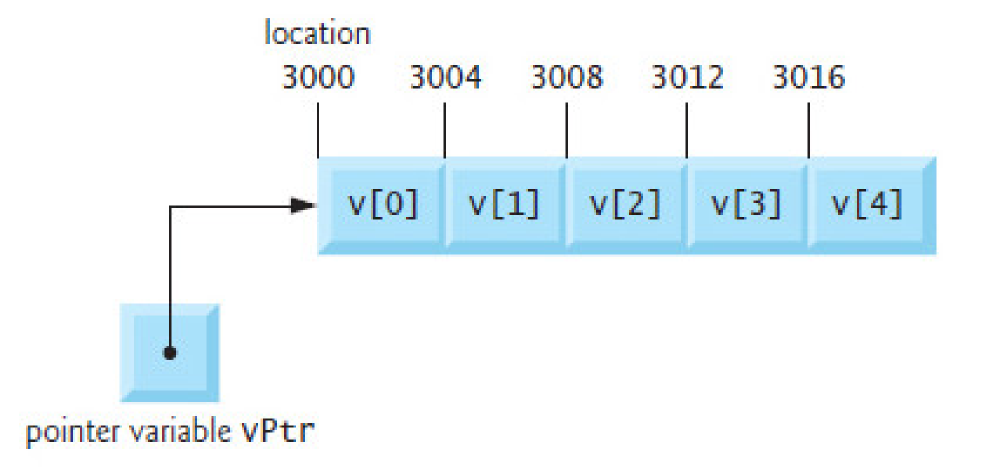
Pointer Expressions and Pointer Arithmetic

- Operators allowed for pointer arithmetic
  - Pointer incremented (++) or decremented (--)
  - Integer added to a pointer (+ or +=)
  - Integer subtracted from a pointer (- or -=)
  - One pointer can be subtracted from another

    Both pointers point to elements of the same array.
- Operations are based the variable type
  - address (+ or -) integer * bytes of memory

    Bytes of memory 4 for int, 8 for double



```
int v[5];
int *vPtr;
vptr = v;

vPtr += 3;
// Move 3 integer size in memory (forward)
vPtr -= 2;
// Move 2 integer size in memory (backward)
```

```
vPtr++;
++vPtr;
vPtr--;
--vPtr;

vPtr = v;
v2Ptr = v[2];

x = v2Ptr - vPtr;  // Point to the same array!!
// Assign x the number of array elements
```

- Pointer arithmetic is undefined unless on an array
- Cannot assume variables are stored contiguously
- ERROR : Running off either end of an array.

Pointer Void

- Pointer assigned to another pointer of the same type
- Pointer to void is the exception ie `void *`
- Generic Pointer can represent any pointer type
- All pointer types can be assigned a pointer to void, and a pointer to void can be assigned a pointer of any type.
- A pointer to void cannot be dereferenced.

Relationship between Pointers and Arrays

- Array and pointers are closely related
- Array names can be called constant pointers
    - Points to the first element of the array
    - Uses the index to locate the element values
- Pointers can be used operations involving indexing

```
int *pnum[30];
int b[5];
int *bPtr;
bPtr = b;      equivalent to       bPtr = &b[0];
```

Pointer/Offset Notation

```
*(bPtr + 3);   equivalent to   *(b + 3);   and b[3];
```

- The 3 is the offset to the pointer.
- () needed because * has a higher precedence that +

Pointer/Index Notation

```
bPtr = b;
bPtr[1];      equivalent to        b[1];

bPtr += 3;   // Valid pointer arithmetic
b +=3;       // Invalid array constant pointer
```

Array and Pointer Access to Array

```
for (size_t i = 0; i < A_SIZE; i++)
{ printf("b[%u] = %d\n", i, b[i]); }

for (size_t offset = 0; offset < A_SIZE; offset++) {
  printf("*(b + %u) = %d\n",
   offset, *(b + offset));
}

for (size_t i = 0; i < A_SIZE; i++)
{ printf("bPtr[%u] = %d\n", i, bPtr[i]); }

for (size_t offset = 0; offset < A_SIZE; offset++) {
  printf("*(bPtr + %u) = %d\n",
   offset, *(bPtr + offset));
}
```

```
char string1[SIZE];       // array string1
char *string2 = "Hello";  // pointer to a string

copy1(string1, string2);
void copy1 (char * const s1, char const * const s2) {
  for (size_t i = 0; (s1[i] = s2[i]) != '\0'; ++i)
  {  ;  }
}

char string3[SIZE];           // array string3
char string4[] = "Good Bye";  // array with string
copy2(string3,string4);

void copy2 (char *s1, char const *s2) {
  for (; (*s1 = *s2) != '\n'; ++s1, ++s2)
  {  ;  }
}
```

Arrays of Pointers

- A common use of an array of pointers is the form an array of strings, referred to simply as a string array.
- So each entry in an array of strings is actually a pointer to the first character of a string.

```
const char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spades"};
```

- The `suit` array is fixed in size, it provides access to character strings of any length.
- The suits could be placed in a two-dimensional array, in which each row would represent a suit and each column would represent a value from a suit name.
- Card Shuffling and Dealing Simulation

Questions