Lecture 6:

Relationship between Arrays & Pointers

Computer Programming 2

2nd Semester 2023-2024

Recall

- A pointer *points* to another variable
- A pointer contains the *address* of another variable
- The **`*`** symbol is used to declare a pointer
  - e.g. **`int* xPtr;`**
- The **`&`** operator gives you the address of a variable
- The **`*`** operator *dereferences* a pointer - gives you the value the pointer points to

Recall

- A pointer is declared to point to a specific data type
- Any data type can have a pointer pointing to it
- Like any other variable, the type of a pointer is fixed
  - So a variable that is declared to be a **`char*`** will *always* point to variables of type **`char`**

Relationship Between Pointers and Arrays

- Arrays and pointers closely related
  - Array name like a constant pointer
  - Pointers can do array subscripting operations
- Define an array **`b[5]`** and a pointer `bPtr`
  - To set them equal to one another use:

```
bPtr = b;
```

  - The array name (**`b`**) is actually the address of first element of the array **`b[5]`**

```
bPtr = &b[0]
```

  - Explicitly assigns **`bPtr`** to address of first element of **`b`**

Relationship Between Pointers and Arrays

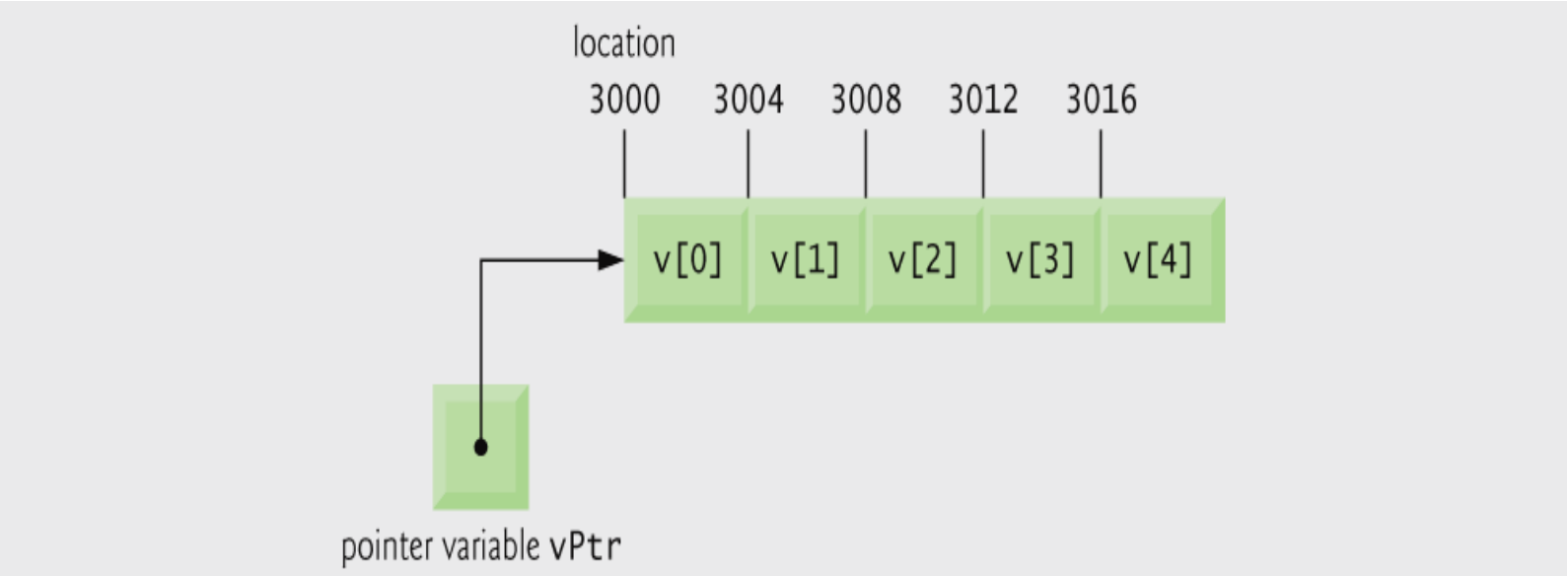
- Element **`b[3]`**
  - Can be accessed by **`*(bPtr+3`**)
    - where **`3`** is called the offset.
    - called pointer/offset notation
  - Can be accessed by **`bPtr[3]`**
    - Called pointer/subscript notation
    - **`bPtr[3]`** same as **`b[3]`**
  - Can be accessed by performing pointer arithmetic on the array itself
    - **`*(b+3)`**

Pointer Expressions and Pointer Arithmetic

- **Arithmetic operations can be performed on pointers**
  - Increment/decrement pointer (++ or −−)
  - Add an integer to a pointer(+ or += , − or −=)
  - Pointers may be subtracted from each other
  - Operations meaningless unless performed on an array

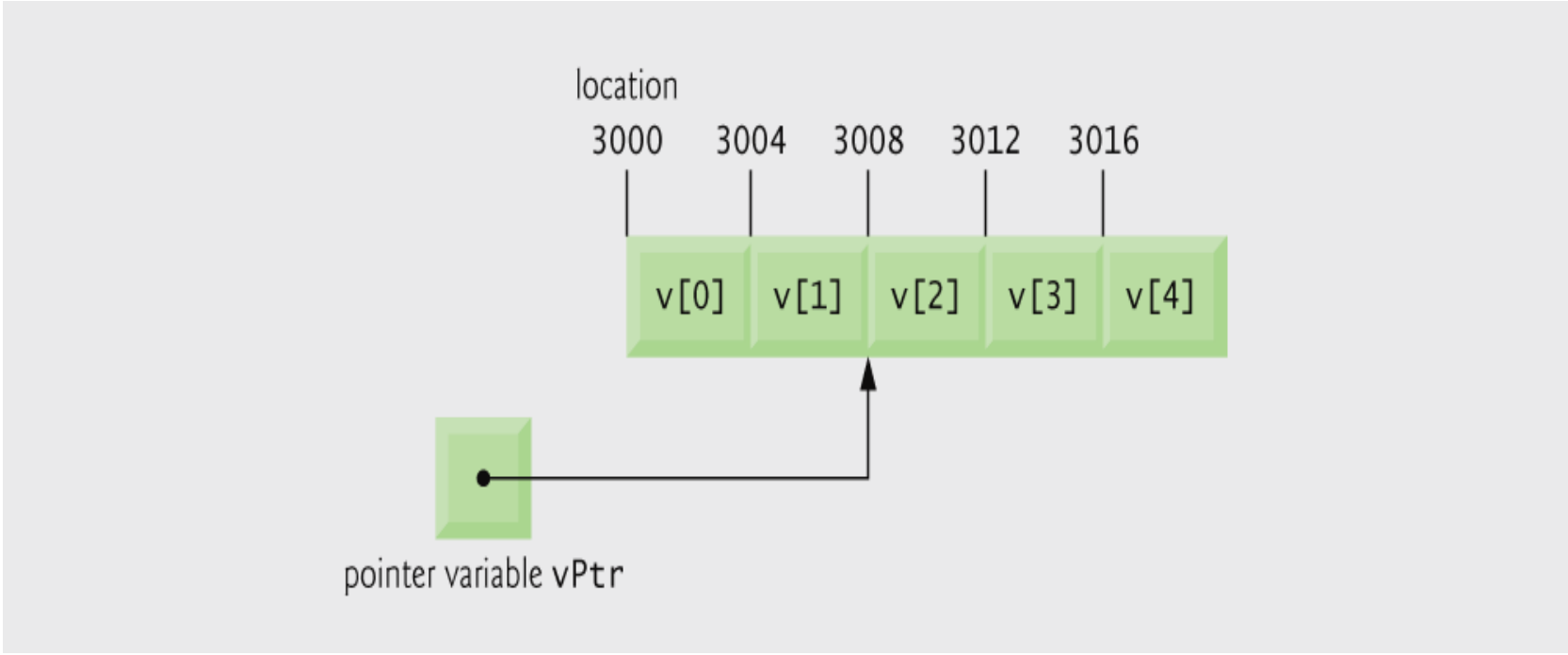
Pointer Expressions and Pointer Arithmetic

- **5 element int array on machine with 4 byte ints**
  - **`vPtr`** points to first element **`v[0]`**
    - at location 3000 (`vPtr = 3000`)



Pointer Expressions and Pointer Arithmetic

- **5 element int array on machine with 4 byte ints**
  - **`vPtr`** points to first element **`v[0]`** at location 3000 (`vPtr=3000`)
  - **`vPtr+=2;`** sets **`vPtr`** to 3008
    - **`vPtr`** points to v[2] (incremented by 2), but the machine has 4 byte ints, so it points to address 3008



Pointer Expressions and Pointer Arithmetic

- Subtracting pointers
  - Returns number of elements from one to the other. If

```
vPtr2 => v[ 2 ];
vPtr => v[ 0 ];
```

  - **`vPtr2-vPtr` would produce 2**

- Pointer comparison (<, == , >)
  - See which pointer points to the higher numbered array element
  - Also, see if a pointer points to `0`

Common Programming Error

Using pointer arithmetic on a pointer that does not refer to an element in an array.

Common Programming Error

Subtracting or comparing two pointers that do not refer to elements in the same array.

Common Programming Error

Running off either end of an array when using pointer arithmetic.

Pointer Expressions and Pointer Arithmetic

- **Pointers of the same type can be assigned to each other**
  - **If not the same type, a cast operator must be used**
  - **Exception: pointer to `void` (type `void *`)**
    - **Generic pointer, represents any type**
    - **No casting needed to convert a pointer to `void` pointer**
    - **`void` pointers cannot be dereferenced**

Common Programming Error

Assigning a pointer of one type to a pointer of another type if neither is of type `void *` is a syntax error.

Common Programming Error

Dereferencing a `void *` pointer is a syntax error.

Common Programming Error

Attempting to modify an array name with pointer arithmetic is a syntax error.

```c
/* relationship.c
      Using subscripting and pointer notations with arrays */
#include <stdio.h>

int main( void ){
   int b[] = { 10, 20, 30, 40 };
   int *bPtr = b;
   int i, offset;
/* output array b using array subs
   printf( "Array b printed with:\nA    subscript notation\n" );
   for ( i = 0; i < 4; i++ )
      printf( "b[%d] = %d\n", i, b[i] );

/* output array b using array name and point          n */
   printf( "\nPointer/offset notation where\
                  "the pointer is the array nam          );
   for ( offset = 0; offset < 4; offset++ )
      printf( "*(b+%d) = %d\n", offset, *(b+offset) );

/* output array b using bPtr and array          */
   printf( "\nPointer subscript notation\n          );
   for ( i = 0; i < 4; i++ )
      printf( "bPtr[%d] = %d\n", i, bPtr[i] );

/* output array b using bPtr and pointer/offset n
   printf( "\nPointer/offset notation\n" );
   for ( offset = 0; offset < 4; offset++ )
      printf( "*(bPtr+%d) = %d\n", offset, *(bPtr+offset) );

   return 0;
}
```

Array subscript notation

Pointer offset notation

Pointer subscript notation

Pointer offset notation

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

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
```

Output of the program **relationship.c**

```c
/* relationship2.c
   Copying a string using array notation and pointer notation. */
#include <stdio.h>

void copy1( char * const s1, const char * const s2 );
void copy2( char *s1, const char *s2 );

int main( void ){
   char string1[10];
   char *string2 = "Hello";
   char string3[10];
   char string4[] = "Good Bye";

   copy1( string1, string2 );
   printf( "string1 = %s\n", string1 );

   copy2( string3, string4 );
   printf( "string3 = %s\n", string3 );

   return 0;
}
```

```
/* copy s2 to s1 using array notation */
void copy1( char * const s1, const char * const s2 )
{
    int i;

    for ( i = 0; ( s1[i] = s2[i] ) != '\0'; i++ )
        ;
}

/* copy s2 to s1 using pointer notation */
void copy2( char *s1, const
{
    for ( ; (*s1 = *s2) != '\0'; s1++, s2++ )
        ;
}
```

Do nothing in the body

Condition of `for` loop actually performs an action

Condition of `for` loop actually performs an action

Output:

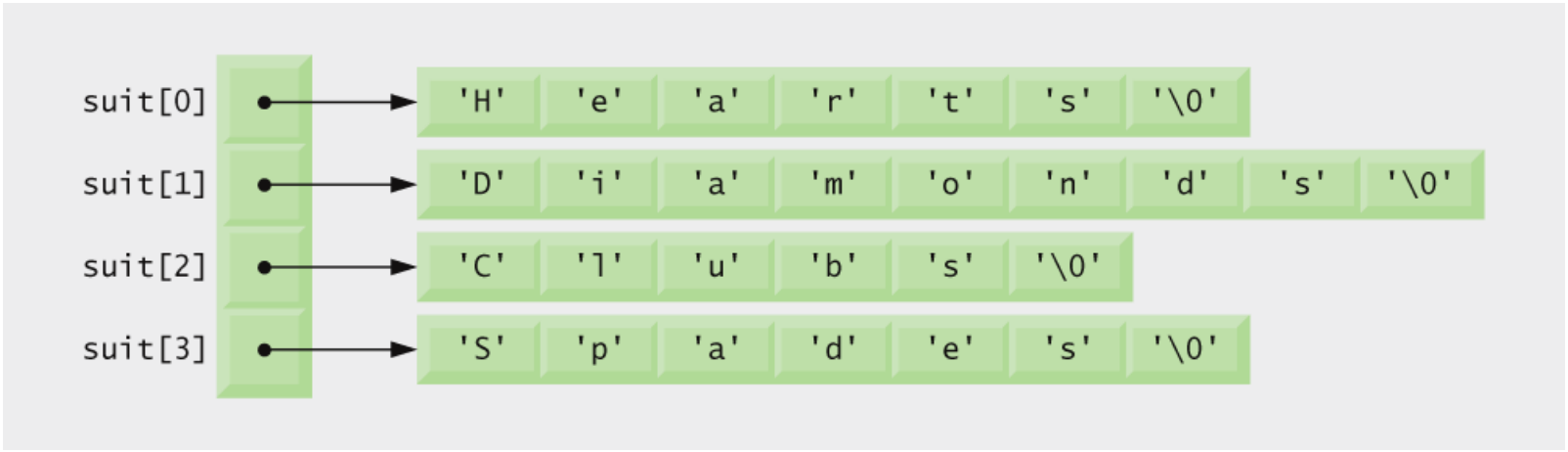
string1 = Hello
string3 = Good Bye

Arrays of Pointers

- Arrays can contain pointers
- For example: an array of strings

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

  - Strings are pointers to the first character
  - **`char *`** – each element of **`suit`** is a pointer to a `char`
  - The strings are not actually stored in the array **`suit`**, only pointers to the strings are stored



Passing Array of Pointers as Parameters to Function **`main()`**

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int i=0;
    for (i=0; i<argc; i++) {
        printf("argv[%d]=%s\n", i, argv[i]);
    }
    return 0;
}
```

Array of pointers/array of strings as parameter

- **argc** refers to the number of command line arguments passed in, which includes the actual name of the program, as invoked by the user.
- **argv** contains the actual arguments, starting with index 1. Index 0 is the program name.

So, if the program is run like this:

./program hello world

Then:

argc would be 3.

argv[0] would be "./program".

argv[1] would be "hello".

argv[2] would be "world".

Thank You!

Any Questions?