Lecture Notes for Lecture 7 of CS 5001 (Foundations of CS) for the Fall, 2018 session at the Northeastern University Silicon Valley Campus.

Pointers and Pointer Function Parameters

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 6 Review

- A *pointer* is the type returned by the *address-of operator* (’&’) that represents the memory address of a variable
- A variable of type pointer can be declared and assigned the result of applying the ’&’ operator to a variable.
- Applying the *value-at* operator (‘*’) to a pointer returns the value of the location that it points to for the declared type.
- A *void* pointer points to a value of an unknown type; applying ‘*’ operator requires assigning or casting it to known type.
- *NULL* can be assigned to any pointer variable; a null pointer points to no value, and the ‘*’ operator cannot be applied.
- Pointers can be used to implement *call by reference* that can change variables outside functions using pointer parameters.

Pointers and Arrays

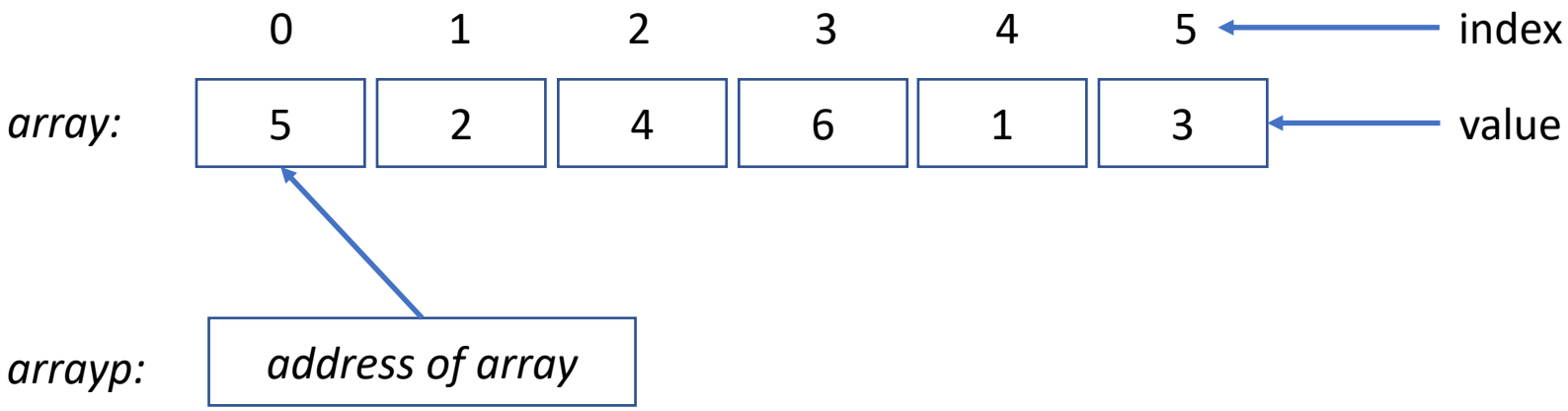
- Pointers can be used with arrays to access array values. A pointer to an array points to the first array location.

```
int array[] = { 5, 2, 4, 6, 1, 3 };

int *arrayp = array;      // equivalent to &array[0]
printf("*arrayp: %d\n", *arrayp);
```

- Output:

```
*arrayp: 5
```



Pointers and Arrays

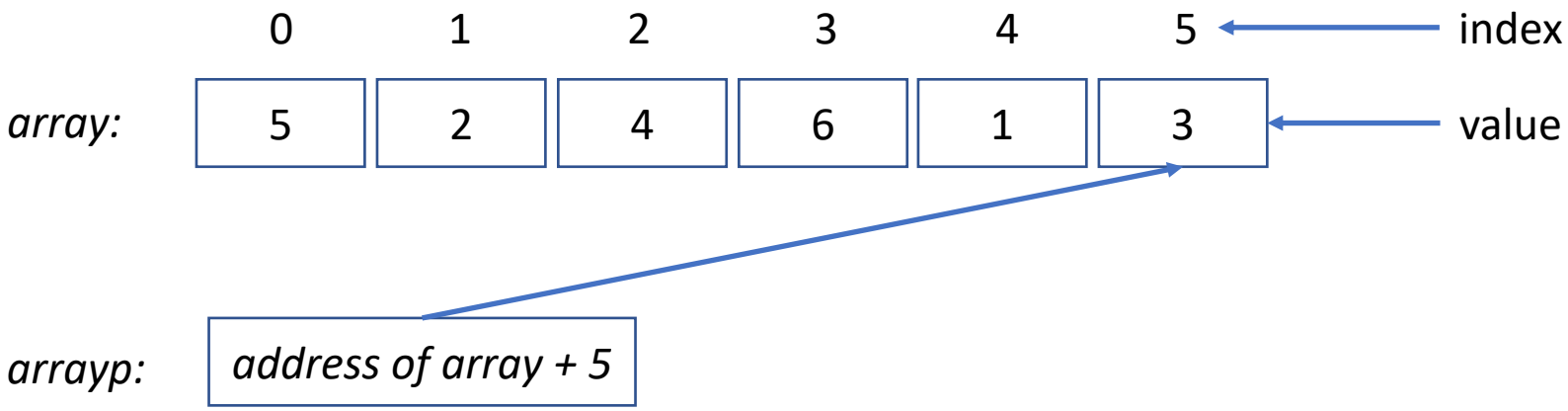
- Pointing to the end of the array requires adding (size-1); automatically adjusts for # bytes in element data type.

```
int array[] = { 5, 2, 4, 6, 1, 3 };

int *arrayp = array+5;  // equivalent to &array[5]
printf("*arrayp: %d\n", *arrayp);
```

- Output:

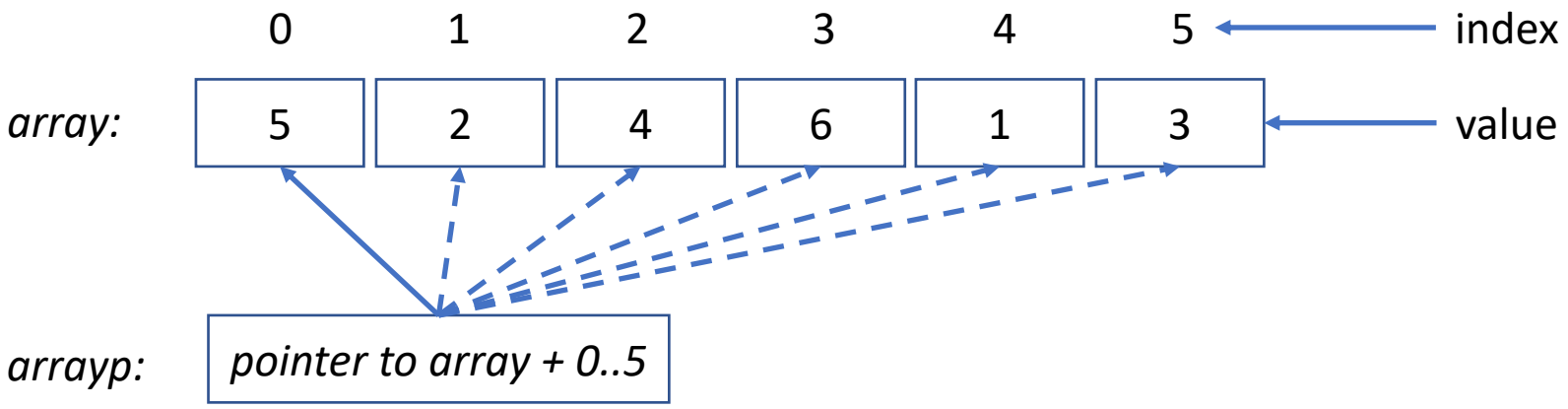
```
*arrayp: 3
```



Pointers and Arrays

- Naïve use of pointer requires address arithmetic for each step through array

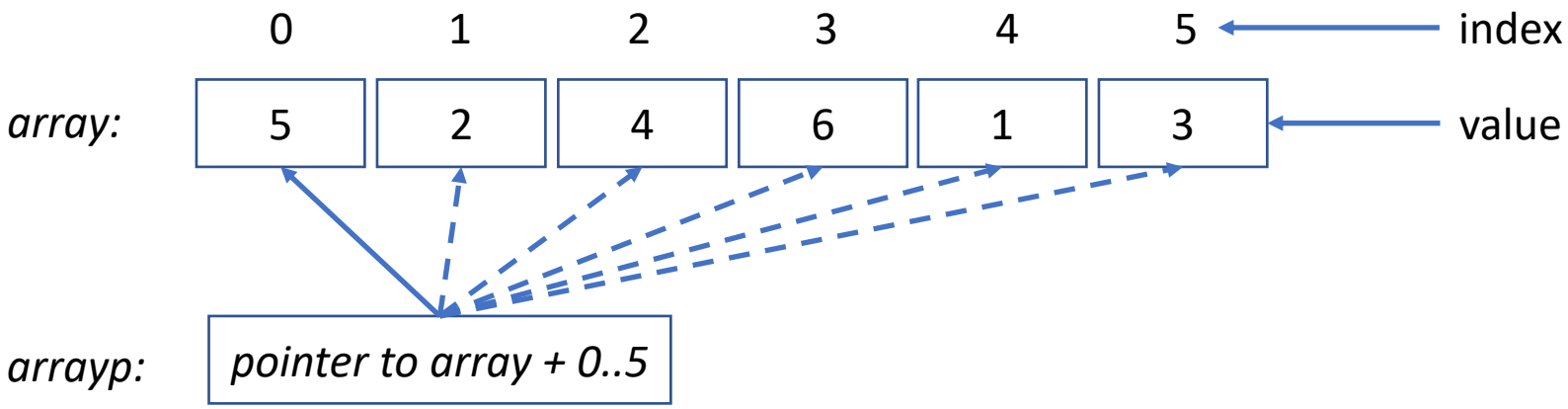
```
int array[] = { 5, 2, 4, 6, 1, 3 };
for (int i = 0; i < 6; i++) {
    int *arrayp  = array+i; // advances pointer (4 *i) bytes
    printf("*arrayp: %d\n", *arrayp);
}
```



Pointers and Arrays

- Incrementing pointer to access array sequentially is much more efficient:

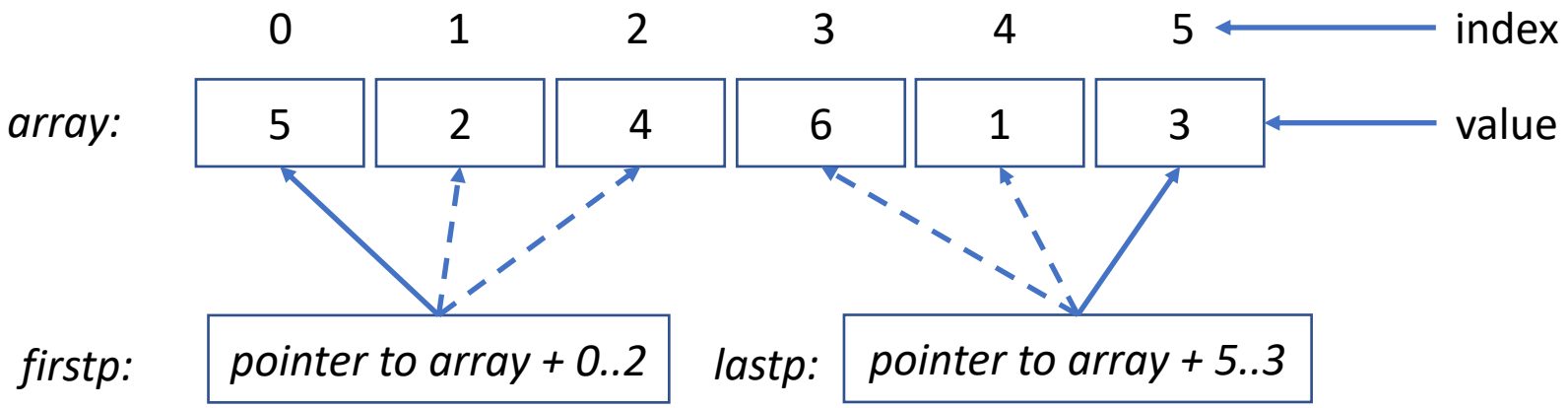
```
int array[] = { 5, 2, 4, 6, 1, 3 };
int *arrayp  = array;
for (int i = 0; i < 6; i++) {
    printf("*arrayp: %d\n", *arrayp++);  // advances 4 bytes
}
```



Pointers and Arrays

- Pointers can be compared

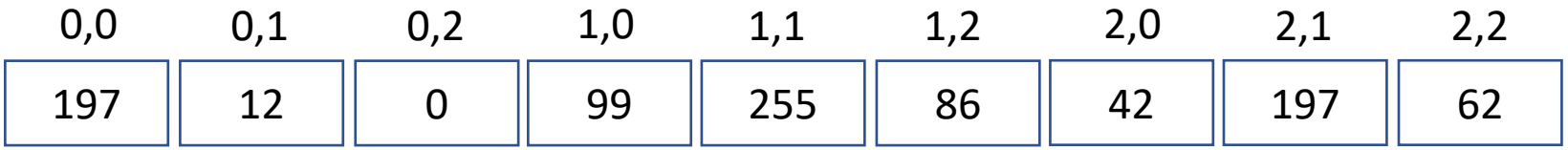
```
int array[] = { 5, 2, 4, 6, 1, 3 };
// reverse the array
for (int *firstp = array, *lastp = array+5; firstp < lastp; firstp++, lastp--) {
    int tmp = *firstp;
    *firstp = *lastp;
    *lastp = tmp;
}
```

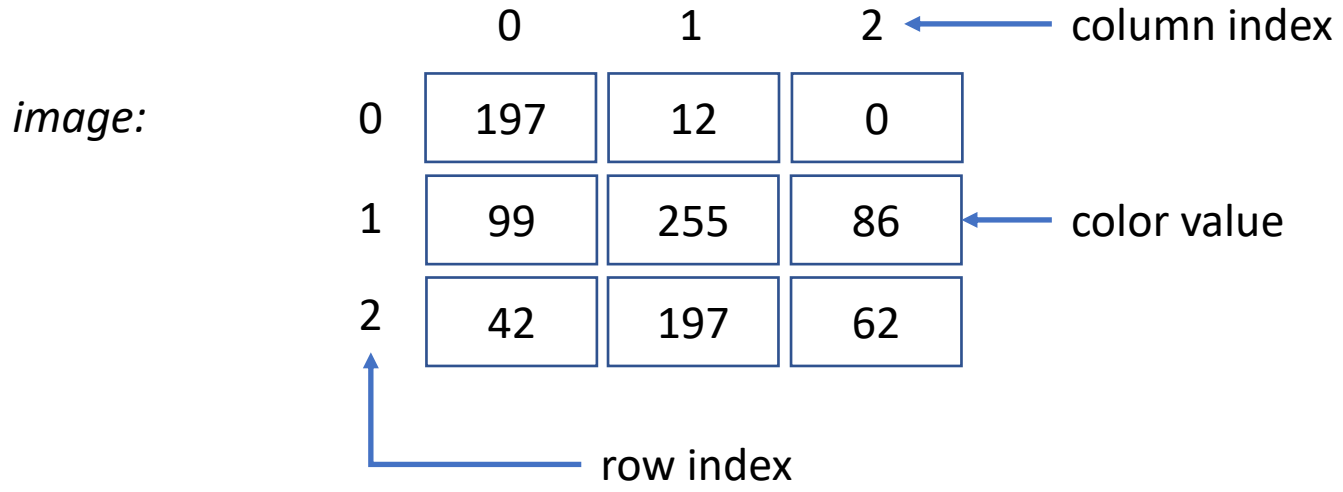


Pointers and Arrays

2D Arrays and Pointers

- Recall that actual storage in memory is in “row-major” order”.





Pointers and Arrays

2D Arrays and Pointers

- Pointing to the first element of a 2D array is not as simple as using the name of the array. Consider this statement:

  unsigned char *imagep = image;

- The compiler reports the following warning:

  warning: incompatible pointer types initializing 'unsigned char *' with an expression of type 'unsigned char [3][3]

- This is because a 2D array is actually an array of 1D arrays, so the type of *image* is unsigned char[], not unsigned char*.

- Use one of these to get a pointer to the first 2D array element

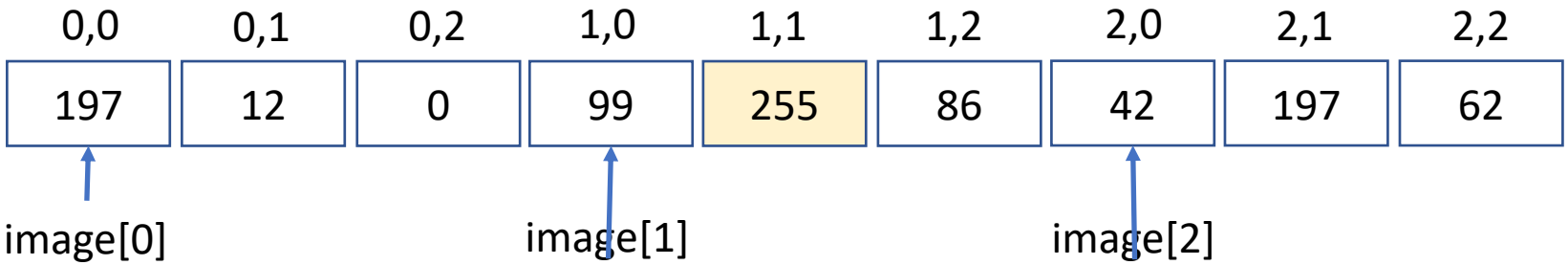
  unsigned char *imagep = &image[0][0]; // address of first element

  unsigned char *imagep = image[0]; // address of first element of row 0

Pointers and Arrays

2D Arrays and Pointers

- Accessing row r, column c in a 2D array with nCols columns requires the use of pointer arithmetic:



image[3][3] is an array of 3 3-element arrays, so image+1 is the the address image[1].

```
const unsigned nCols = 3;
unsigned r = 1, c = 1;
unsigned char *imagep = &image[0][0];  // could use image[0]
unsigned char ch = *(imagep+ nCols*r + c);
```

Pointers and Arrays

2D Arrays and Pointers

- Naïve use of address arithmetic leads to inefficiencies when accessing array sequentially.

```
unsigned char image[3][3] =
   { {197, 12, 0}, {99, 255, 86}, {42, 197, 62} };

const unsigned nCols = 3;
unsigned char* imagep = &image[0][0];
for (unsigned r = 0; r < 3; r++) {
   for (unsigned c = 0; c < 3; c++) {
      // address array using random-access formula
      unsigned char ch = *(imagep + nCols*r + c);
      printf(”image[%d][%d] = %d\n", r, c, ch);
   }
}
```

Output:

```
Image[0][0] = 197
Image[0][1] = 12
Image[0][2] = 0
Image[1][0] = 99
Image[1][1] = 255
Image[1][2] = 86
Image[2][0] = 42
Image[2][1] = 197
Image[2][2] = 62
```

Pointers and Arrays

2D Arrays and Pointers

- Intelligent use of pointer to step through array in row major order is much more efficient.

```
unsigned char image[3][3] =
   { {197, 12, 0}, {99, 255, 86}, {42, 197, 62} };

unsigned char*imagep = &image[0][0];
for (int r = 0; r < 3; r++) {
   for (int c = 0; c < 3; c++) {
      // address array using sequential pointer
      unsigned char val = *imagep++;
      printf("array[%d][%d] = %d\n", r, c, val);
   }
}
```

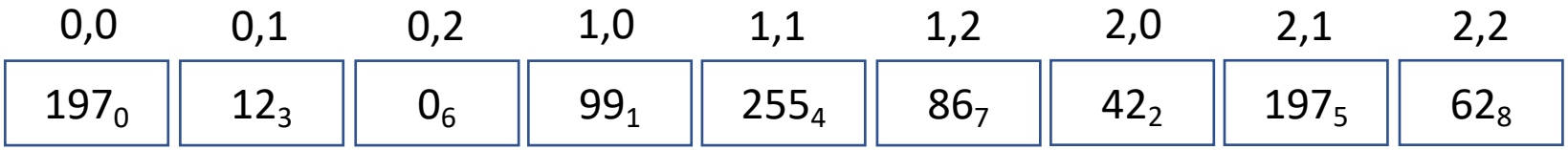
Output:

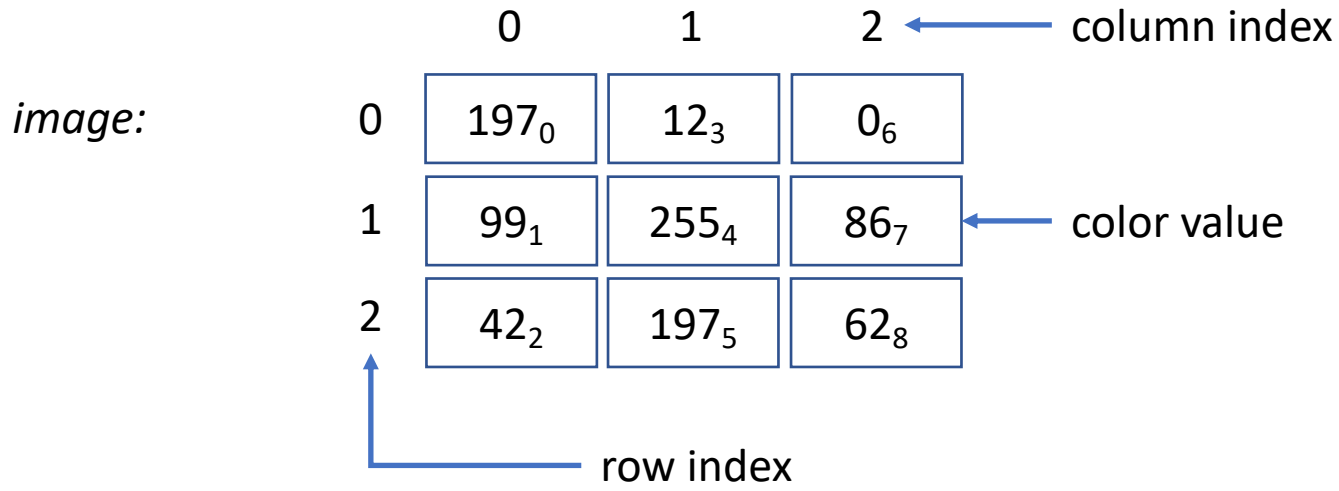
```
Image[0][0] = 197
Image[0][1] = 12
Image[0][2] = 0
Image[1][0] = 99
Image[1][1] = 255
Image[1][2] = 86
Image[2][0] = 42
Image[2][1] = 197
Image[2][2] = 62
```

Pointers and Arrays

2D Arrays and Pointers

- How would we step through the 2D array in *column-major* order?.





Pointers and Arrays

2D Arrays and Pointers

- Intelligent use of pointer to step through array in row major order is much more efficient.

```
unsigned char image[3][3] =
   { {197, 12, 0}, {99, 255, 86}, {42, 197, 62} };

unsigned char *pc = &image[0][0];
for (int c = 0; c < 3; c++, pc++) {  // point to next col
   unsigned char *pr = pc;
   for (int c = 0; c < 3; c++, pr+=3) { // point to next row
      unsigned char ch = *pr;
      printf(”image[%d][%d] = %d\n", r, c, ch);
   }
}
```

Output:

```
image[0][0] = 197
image[1][0] = 99
image[2][0] = 42
image[0][1] = 12
image[1][1] = 255
image[2][1] = 197
image[0][2] = 0
image[1][2] = 86
image[2][2] = 62
```

Pointers and Arrays

Passing Arrays to Functions As Pointers

- In formal array, parameters are processed as pointers.

```
size_t strlen(const char str[]) {
    int count = 0;
    for (int I = 0; str[i] != '\0'; i++) {
        count++;
    }
    return count
}
```

```
size_t strlen(const char* str) {
    int count = 0;
    for (int I = 0; *(str+i) != '\0'; i++) {
        count++;
    }
    return count
}
```

Pointers and Arrays

Passing Arrays to Functions As Pointers

- Since string is processed sequentially, more efficient to use pointer instead of array index to avoid address arithmetic.

```
size_t strlen(const char *str) {
    size_t count = 0;
    for (  ; *str != '\0'; str++) {
        count++;
    }
    return count
}
```

Pointers and Arrays

C String Functions Using Char*

- Many C string library functions use char* rather than array parameters and return char* results

char *strcpy(char *s1, const char *s2); Copies string s2 into string s1, return s1 pointer
char *stpcpy(char *s1, const char *s2); Copies string s2 into string s1, return pointer to ’\0’ character in s1.
char *strcat(char *s1, const char *s2); Concatenates string s2 onto the end of string s1; return s1 pointer
size_t **strlen(const char *s1);** Returns the length of string s1.
int strcmp(const char *s1, const char *s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

Pointers and Arrays

C String Functions Using Char*

- Many C string library functions use char* rather than array parameters and return char* results

char* strchr(const char*s1, char ch); Returns a pointer to the first occurrence of character ch in string s1.
char* strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1
size_t strspn(const char *str1, const char *str2) Calculates the length of the initial segment of str1 which consists entirely of characters in str2.
char *strpbrk(const char *str1, const char *str2) Finds the first character in the string str1 that matches any character specified in str2.

Pointers and Arrays

C Additional Character Functions

- Here are some additional functions that operate on single characters of a string:

bool isspace(int ch); Returns true if ch is a whitespace character, false otherwise.
bool isdigit(int ch); Returns true if ch is a digit character, false otherwise..
bool isalpha(int ch) Returns true if ch is an upper- or lower-case character
bool isupper(int ch) Returns true if ch is an upper-case character
bool islower(int ch) Returns true if ch is a lower-case character

Pointers and Arrays

Function int atoi(const char* str)

- The atoi() C library function converts a string into an integer

```
/**
 * This function returns an int representing
 * the integer value of the string.
 *
 * @param str the input string
 * @param the integer represented by the string
 * @return the integer value for the input string
 */
int atoi(const char* str) {
```

Pointers and Arrays

Function int atoi(const char* str)

- The atoi() C library function converts a string into an integer

```
// Skip any leading blanks.
while (isspace(*str)) {
        str++;
 }
// Check for and step past sign.
bool negative = false;
if (*str == '-') {
         negative = true;
         str++;
} else if (*str == '+') {
        str++;
}
```

Pointers and Arrays

Function int atoi(const char* str)

- The atoi() C library function converts a string into an integer

```
    // accumulate positive value from successive digits
    int result = 0;
    for (; isdigit(*str); str++) {
        int digit = *str - '0';
        result = (10 * result) + digit;
    }
    // make negative if required
    if (negative) {
        return -result;
    }
    return result;
}
```

Pointers and Arrays

Function int atoi(const char* str)

- The atoi() C library function converts a string into an integer

```
/** Tests atoi() */
int main(void) {
        printf("\ntest atoi()\n");
        printf("atoi(\"%s\") = %d\n", "0", atoi("0"));
        printf("atoi(\"%s\") = %d\n", "103", atoi("103"));
        printf("atoi(\"%s\") = %d\n", "+12", atoi("+12"));
        printf("atoi(\"%s\") = %d\n", "-1", atoi("-1"));
        printf("atoi(\"%s\") = %d\n", "-199", atoi("-199"));
}
Output:
atoi("0") = 0
atoi("103") = 103
atoi("+12") = 12
atoi("-1") = -1
atoi("-199") = -199
```