Appendix: Other operations
Increment & decrement

- \( x++ \): yield old value, add one
- \( ++x \): add one, yield new value

```
int x = 10;
x++;
int y = x++;
```
```
11
```
```
int z = ++x;
```
```
13
```

- \( --x \) and \( x-- \) are similar (subtract one)
Math: Increment and Decrement Operators

- **Example 1:**
  ```c
  int x, y, z, w;
y=10; w=2;
x=++y;
z=--w;
  ```
- **Example 2:**
  ```c
  int x, y, z, w;
y=10; w=2;
x=y++;
z=w--;
  ```

What are \( x \) and \( y \) at the end of each example?
Math: Increment and Decrement Operators

- Example 1:
  ```c
  int x, y, z, w;
  y=10;  w=2;
  x=++y;
  z=--w;
  ```
  - First increment/decrement, then assign result
  - x is 11, z is 1

- Example 2:
  ```c
  int x, y, z, w;
  y=10;  w=2;
  x=y++;  
  z=w--;  
  ```
  - First assign result, then increment/decrement
  - x is 10, z is 2
Example

```c
int a[2];
int number1, number2, *p;
a[0]=1; a[1]=10;
p=a;
number1 = *p++;
number2 = *p;
```

What will `number1` and `number2` be at the end?
Math: Increment and Decrement Operators on Pointers

- Example

```c
int a[2];
int number1, number2, *p;
a[0]=1; a[1]=10;
p=a;
number1 = *p++;
number2 = *p;
```

Hint: `++` increments pointer `p` not variable `*p`

- What will `number1` and `number2` be at the end?
Logic: Relational (Condition) Operators

==  equal to
!=  not equal to
>  greater than
<  less than
>=  greater than or equal to
<=  less than or equal to
if (a == b)
    printf ("Equal.");
else
    printf ("Not Equal.");

Question: what will happen if I replaced the above with:
if (a = b)
    printf ("Equal.");
else
    printf ("Not Equal.");
if (a == b)
    printf ("Equal.");
else
    printf ("Not Equal.");

Question: what will happen if I replaced the above with:

if (a = b)
    printf ("Equal.");
else
    printf ("Not Equal.");

Perfectly LEGAL C statement! (syntactically speaking)
It copies the value of b into a. The statement will be interpreted as TRUE if b is non-zero.
**strncpy, strlen**

- `strncpy(ptr1, ptr2);`
  - `ptr1` and `ptr2` are pointers to `char`

- `value = strlen(ptr);`
  - `value` is an integer
  - `ptr` is a pointer to `char`

```c
int len;
char str[15];
strncpy (str, "Hello, world!");
len = strlen(str);
```
**strncpy**

- `strncpy(ptr1, ptr2, num);`
  - `ptr1` and `ptr2` are pointers to char
  - `num` is the number of characters to be copied

```c
int len;
char str1[15], str2[15];
strcpy (str1, "Hello, world!");
strncpy (str2, str1, 5);
```
strncpy

- `strncpy(ptr1, ptr2, num);`
  - `ptr1` and `ptr2` are pointers to char
  - `num` is the number of characters to be copied

Caution: `strncpy` blindly copies the characters. It does not voluntarily append the string-terminating null character.

```c
int len;
char str1[15], str2[15];
strcpy (str1, "Hello, world!");
strncpy (str2, str1, 5);
```
strcat

- `strcat(ptr1, ptr2);`
  - `ptr1` and `ptr2` are pointers to char

- Concatenates the two null terminated strings yielding one string (pointed to by `ptr1`).

```c
char S[25] = "world!";
char D[25] = "Hello, ";
strcat(D, S);
```
\textbf{strcat}

- \texttt{strcat(ptr1, ptr2);}
  - \texttt{ptr1} and \texttt{ptr2} are pointers to char

- Concatenates the two null terminated strings yielding one string (pointed to by \texttt{ptr1}).
  - Find the end of the destination string
  - Append the source string to the end of the destination string
  - Add a NULL to new destination string
What’s wrong with

```c
char S[25] = "world!";
strcat("Hello, ", S);
```
strcat Example

- What’s wrong with

```c
char *s = malloc(11 * sizeof(char));
/* Allocate enough memory for an array of 11 characters, enough to store a 10-char long string. */
strcat(s, "Hello");
strcat(s, "World");
```
What’s wrong with

```c
char *s = malloc(11 * sizeof(char));
    /* Allocate enough memory for an
       array of 11 characters, enough
       to store a 10-char long string. */
s[0] = 0;
strcat(s, "Hello"); or strcpy(s, "Hello");
strcat(s, "World");
```
strcat

- strcat(ptr1, ptr2);
  - ptr1 and ptr2 are pointers to char

- Compare to Java and C++
  - string s = s + " World!";

- What would you get in C?
  - If you did char* ptr0 = ptr1+ptr2;
strcat

- `strcat(ptr1, ptr2);`
  - `ptr1` and `ptr2` are pointers to char

- Compare to Java and C++
  - `string s = s + " World!";`

- What would you get in C?
  - If you did `char* ptr0 = ptr1+ptr2;`
  - You would get the sum of two memory locations!
diff = strcmp(ptr1, ptr2);
- **diff** is an integer
- **ptr1** and **ptr2** are pointers to char

**Returns**
- zero if strings are identical
- < 0 if **ptr1** is less than **ptr2** (earlier in a dictionary)
- > 0 if **ptr1** is greater than **ptr2** (later in a dictionary)

```c
int diff;
char s1[25] = "pat";
char s2[25] = "pet";
diff = strcmp(s1, s2);
```
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Parentheses (function call)</td>
<td>left-to-right</td>
</tr>
<tr>
<td>[]</td>
<td>Brackets (array subscript)</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>Member selection via object name</td>
<td></td>
</tr>
<tr>
<td>-&gt;</td>
<td>Member selection via pointer</td>
<td></td>
</tr>
<tr>
<td>++ --</td>
<td>Postfix increment/decrement</td>
<td></td>
</tr>
<tr>
<td>++ --</td>
<td>Prefix increment/decrement</td>
<td>right-to-left</td>
</tr>
<tr>
<td>+ -</td>
<td>Unary plus/minus</td>
<td></td>
</tr>
<tr>
<td>! ~</td>
<td>Logical negation/bitwise complement</td>
<td></td>
</tr>
<tr>
<td>(type)</td>
<td>Cast (change type)</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Dereference</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>sizeof</td>
<td>Determine size in bytes</td>
<td></td>
</tr>
<tr>
<td>* / %</td>
<td>Multiplication/division/modulus</td>
<td>left-to-right</td>
</tr>
<tr>
<td>+ -</td>
<td>Addition/subtraction</td>
<td>left-to-right</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>Bitwise shift left, Bitwise shift right</td>
<td>left-to-right</td>
</tr>
<tr>
<td>&lt; &lt;=</td>
<td>Relational less than/less than or equal to</td>
<td>left-to-right</td>
</tr>
<tr>
<td>&gt; &gt;=</td>
<td>Relational greater than/greater than or equal to</td>
<td>left-to-right</td>
</tr>
<tr>
<td>== ! =</td>
<td>Relational is equal to/is not equal to</td>
<td>left-to-right</td>
</tr>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
<td>left-to-right</td>
</tr>
<tr>
<td>^</td>
<td>Bitwise exclusive OR</td>
<td>left-to-right</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bitwise inclusive OR</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>left-to-right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>? :</td>
<td>Ternary conditional</td>
<td>right-to-left</td>
</tr>
<tr>
<td>=</td>
<td>Assignment</td>
<td></td>
</tr>
<tr>
<td>+= -=</td>
<td>Addition/subtraction assignment</td>
<td>right-to-left</td>
</tr>
<tr>
<td>*= /=</td>
<td>Multiplication/division assignment</td>
<td></td>
</tr>
<tr>
<td>%= &amp; =</td>
<td>Modulus/bitwise AND assignment</td>
<td></td>
</tr>
<tr>
<td>^=</td>
<td>Bitwise exclusive/inclusive OR assignment</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;&lt; &gt;&gt;&gt;</td>
<td>Bitwise shift left/right assignment</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>Comma (separate expressions)</td>
<td>left-to-right</td>
</tr>
</tbody>
</table>
Questions

- What is the value of $b[2]$ at the end?

```c
int b[3];
int* q;

b[0]=48; b[1]=113; b[2]=1;
q=b;
*(q+1)=2;
b[2]=*b;
```
Questions

- What is the value of \( b[2] \) at the end?

```c
int b[3];
int* q;

b[0]=48; b[1]=113; b[2]=1;
q=b;
*(q+1)=2;

b[2]=*b;

```

Copyright ©: University of Illinois CS 241 Staff
Questions

- What is the value of `b[2]` at the end?

```c
int b[3];
int* q;
b[0]=48; b[1]=113; b[2]=1;
q=b;
*(q+1)=2;
b[2]=*b;
```
Questions

What is the value of $b[2]$ at the end?

```c
int b[3];
int* q;
b[0]=48; b[1]=113; b[2]=1;
q=b;
*(q+1)=2;
b[2]=*b;
```
Questions

What is the value of $b[2]$ at the end?

```c
int b[3];
int* q;

b[0]=48; b[1]=113; b[2]=1;
q=b;
*(q+1)=2;
b[2]=*b;
```

```
<table>
<thead>
<tr>
<th></th>
<th>b[0]</th>
<th>b[1]</th>
<th>b[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>48</td>
<td>113</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>48</td>
<td>2</td>
<td>50</td>
</tr>
</tbody>
</table>
```