C bugs, arrays, the shell, data in memory
scanf’s arguments must be pointers to a valid location

```c
int val;
scanf("%d", val);
```
Common bug 2

Reading uninitialized memory

```c
int* matvec(int** A, int* x) {
    int* y = malloc(n * sizeof(int));
    int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            y[i] += A[i][j] * x[j];
    return y;
}
```
Allocating the (possibly) wrong sized object

```c
int** p;
p = malloc(n * sizeof(int));
for (i = 0; i < n; i++)
    p[i] = malloc(m * sizeof(int));
```
Common bug 4

Overwriting memory

```c
int** p;
p = malloc(n * sizeof(int*));
for (i = 0; i <= n; i++)
    p[i] = malloc(m * sizeof(int));
```
Misunderstanding pointer arithmetic

```c
int* search(int* p, int val) {
    while (*p && *p != val)
        p += sizeof(int);
    return p;
}
```
Common bug 6

Referencing nonexistent variables

```c
int* foo () {
    int val;
    return &val;
}
```
Common bug 7

Freeing blocks multiple times

```c
x = malloc(n * sizeof(int));
// ... manipulate x ...
free(x);
y = malloc(m * sizeof(int));
// ... manipulate y ...
free(x);
```
Common bug 8

Use after free

```c
x = malloc(n * sizeof(int));
// ... manipulate x ...
free(x);
// ...
```

```c
y = malloc(m * sizeof(int));
for (i = 0; i < m; i++)
    y[i] = x[i]++;
```
Failing to free blocks

```c
int *x = malloc(n * sizeof(int));
// ...
```
typedef struct List {
    int val;
    struct List* next;
} List;

void foo() {
    List *head = malloc(sizeof(List));
    head->val = 0;
    head->next = NULL;
    // ...
    free(head);
}

Freeing only part of a data structure
### Arrays vs. Pointers

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>holds address</td>
<td>holds data</td>
</tr>
<tr>
<td>access is indirect</td>
<td>access is direct</td>
</tr>
<tr>
<td>“dynamic” data</td>
<td>“static” data</td>
</tr>
</tbody>
</table>
Arrays vs. pointers

```c
char* s1 = "hello";
char s2[] = "hello";
```
Arrays vs. pointers

```c
int x = 42;
int* p = &x;
int a[100];

printf("%p\n", p);
printf("%p\n", &p);
printf("%p\n", a);
printf("%p\n", &a);
```
We can redirect the output from a program to a file:

```
./myProgram > out.txt
```

We can redirect stderr too:

```
find /etc -name "*color*" 2> /dev/null
```
Pipes and redirection

We can take the output from one program and give it as input to another

./myProgram | ./myOtherProgram
Given a CSV file of student names, emails, courses, and #credits:

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Course</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob Smith</td>
<td><a href="mailto:bob@rutgers.edu">bob@rutgers.edu</a></td>
<td>CS111</td>
<td>78</td>
</tr>
<tr>
<td>Carol Ford</td>
<td><a href="mailto:carol@rutgers.edu">carol@rutgers.edu</a></td>
<td>CS211</td>
<td>43</td>
</tr>
<tr>
<td>Alice Jones</td>
<td><a href="mailto:alice@rutgers.edu">alice@rutgers.edu</a></td>
<td>CS211</td>
<td>92</td>
</tr>
</tbody>
</table>

- Print names/emails of CS211 students in descending order of #credits
- Find the average number of credits of all students
Pipes and redirection

Bob Smith,bob@rutgers.edu,CS111,78
Carol Ford,carol@rutgers.edu,CS211,43
Alice Jones,alice@rutgers.edu,CS211,92

Print CS211 email list:

```bash
cat students.csv | \n  grep ',CS211,' | \n  perl -pe 's/(.*)\d*/$2 $1/' | \n  sort -rn | \n  perl -pe 's/\d* ([^,]*),([^,]*),.*$/"$1" <$2>/'
```
Find average number of credits of all students:

cat students.csv | \
  cut -d, -f4 | \
  awk '{ sum += $1; n++; } END { print sum/n }' 

18
C mask operators

• \( x \ & \ m \) - do a bitwise AND operation
• \( x \ | \ m \) - do a bitwise OR operation
• \( x \ ^\ ^\ ^\ ^\ m \) - do a bitwise XOR operation
• \( \sim x \) - do a bitwise NOT operation

Examples:

• \( 1 \ & \ 1 = 1 \)
• \( 17 \ | \ 68 = 85 \)
• \( 17 \ | \ 20 = 21 \)
• \( 85 \ ^\ ^\ ^\ ^\ 83 = 6 \)
C shift operators

- $x << n$ - shift $x$ left by $n$ bits
- $x >> n$ - shift $x$ right by $n$ bits

Note:
- A left shift = multiplying by 2
- A right shift = dividing by 2 (and discarding remainders)

Examples:

- $1 << 1 == 2$
- $25 << 2 == 100$
- $25 >> 1 == 12$
Shifts and masks

- Given the integer 1234, grab the 3rd byte
## Data sizes

<table>
<thead>
<tr>
<th>C type</th>
<th>Min size</th>
<th>Typical size</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short int</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>long int</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>pointer</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
How is a 4-byte int stored in memory?

```c
int x = 1;
```

- Most significant byte (MSB) first → big endian
- Least significant byte (LSB) first → little endian

How can we tell which our system uses?