Unit 1c – Idioms and Algorithmic Thinking Examples

Mark Redekopp
Unit Objectives

• Understand chars and ints and how cout uses types to determine how it will interpret the numbers being stored.
• Dive deeper into C++ aspects of cin and cout
• Understand assignment and correctly identify errors when using assignment
• See applications of division and modulo such as unit conversion, extracting digits/coordinates, divisibility and factoring
Review of Data Types

- **bool**
  - true or false values

- **int** or **unsigned int**
  - Integer values

- **char**
  - A single ASCII character
  - Or a small integer (but just use 'int')

- **double**
  - A real number (usually if a decimal/fraction is needed) but also for very large numbers

- **string**
  - Multiple text characters, ending with the null ("\0" = 00) character
MORE CIN AND COUT
I/O Streams

- C++ and the OS use the notion of **streams** to temporarily store (aka buffer) data to be input or output and then uses the **cin** and **cout** objects (from the `<iostream>` library) to access those streams.
- **cin** extracts data from the input stream [stdin] (skipping over preceding whitespace then stopping at following whitespace).
- **cout** inserts data into the output stream [stdout] for display by the OS.

```cpp
#include <iostream>

int main()
{
    int x;
    std::cin >> x;
    return 0;
}
```

```cpp
#include <iostream>

int main()
{
    std::cout << "It was the" << std::endl;
    std::cout << "best of times."
        << std::endl;
    return 0;
}
```
std:: and the using namespace statement

• Most C++ library components "live" in the **std** namespace
  – Think of a namespace like folders on your laptop or a classification hierarchy
  – So cout and endl are technically **std::cout** and **std::endl**
  – To avoid all that typing, we can tell the C++ compiler to look for components in the std namespace when it can't find any definition earlier in our code by writing the **using namespace std**;
• Demo: Try to compile the top program WITHOUT the **using** statement.

```cpp
#include<iostream>
using namespace std;
int main()
{
    cout << "It was the" << endl;
    cout << "best of times.";
    return 0;
}
```

```cpp
#include<iostream>
// no using namespace std; statement
int main()
{
    std::cout << "It was the" << std::endl;
    std::cout << "best of times.";
    return 0;
}
```
Error without 'using' statement

```cpp
#include <iostream>

int main()
{
    cout << "Hello world" << endl;
    return 0;
}
```

```bash
$ make lec02-cout
make[1]: Entering directory `/path/to/project'
g++  lec02-cout.cpp  -o lec02-cout
lec02-cout.cpp: In function `int main()':
lec02-cout.cpp:6:5: error: `cout' was not declared in this scope; did you mean `std::cout'?
  6 |    cout << "Hello world" << endl;
     |     ^~~~
  5 |    std::cout
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.
Newlines, endl, and Flushing

• To move the cursor to the next line we need to print a new line, '\n' (char)
• `cout` only gives the characters to the OS which then copies them to the screen.
• The OS may choose to delay and not print immediately causing strange issues (see bottom)
• `endl = '\n' + a flush of the output stream which forces the OS to print immediately

```cpp
// cout only gives the characters to the OS
cout << "Hi\n";
```

```cpp
// endl = '\n' + a flush of the output stream which forces the OS to print immediately
cout << "Hi" << endl;
```
Newlines, `endl`, and Flushing

- To move the cursor to the next line we need to print a new line, `'\n'` (char)
- `cout` only gives the characters to the OS which then copies them to the screen.
- The OS may choose to delay and not print immediately causing strange issues (see bottom)
- `endl = '\n' + a flush of the output stream`

```cpp
int main() {
    task_that_might_crash(); // Doesn't crash
    cout << "Got Here 1";
    task_that_might_crash(); // Does crash!
    cout << "Got Here 2";
    return 0;
}
```

```cpp
int main() {
    task_that_might_crash(); // Doesn't crash
    cout << "Got Here 1" << endl;
    task_that_might_crash(); // Does crash!
    cout << "Got Here 2" << endl;
    return 0;
}
```

Use descriptive messages and `endl`s when debugging.
I/O Manipulators

- Manipulators control HOW `cout` handles certain output options and how `cin` interprets the input data (but print nothing themselves)
  - Must `#include <iomanip>`
- Common examples
  - `setw(n)`: Separate consecutive outputs by `n` spaces
  - `setprecision(n)`: Use `n` digits to display doubles (both the integral + decimal parts)
  - `fixed`: Uses the precision for only the digits after the decimal point
  - `boolalpha`: Show Booleans as `true` and `false` rather than 1 and 0, respectively
- Separated by `<<` or `>>` and used inline with actual data
- Other than `setw`, manipulators continue to apply to later output until changed

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    double pi = 3.14159;

    cout << pi << endl;  // Prints: 3.14159

    cout << setprecision(2) << pi << endl;  // Prints: 3.1

    cout << setprecision(2) << fixed << pi << endl;  // Prints: 3.14

    return 0;
}
```

[See "iomanip" in-class exercise to explore various options](http://en.cppreference.com/w/cpp/io/manip)
Understanding ASCII and chars

• A char is just an integer type that
  – Is only 1 byte (limited range 0 to 255 or -128 to +127)
  – cout uses the \textbf{type}, char or int, to infer if we want the
    ASCII character or integer

• We can perform arithmetic/comparison operations on
  ASCII chars since they are converted to integers

```cpp
char c = 'a'; // same as char c = 97;
cout << c << endl; // prints 'a'
c = 97;
cout << c << endl; // prints 'a'
int x = c;
cout << x << endl; // prints 97
char d = 'a' + 1; // d now contains 98 (ASCII 'b')
cout << d << endl; // prints 'b' on the screen
if(c > 'a' && c <= 'z') { } // && means AND
  // better than if(c >= 97 && c <= 122)
c = '1'; // c stores decimal 49, d stores 1
assert(c < ' '); // only prints: "1 ", not "1 1"
```

<table>
<thead>
<tr>
<th>ASCII printable characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 space</td>
</tr>
<tr>
<td>33 !</td>
</tr>
<tr>
<td>34 '</td>
</tr>
<tr>
<td>35 #</td>
</tr>
<tr>
<td>36 $</td>
</tr>
<tr>
<td>37 %</td>
</tr>
<tr>
<td>38 &amp;</td>
</tr>
<tr>
<td>39 '</td>
</tr>
<tr>
<td>40 (</td>
</tr>
<tr>
<td>41 )</td>
</tr>
<tr>
<td>42 *</td>
</tr>
<tr>
<td>43 +</td>
</tr>
<tr>
<td>44 ,</td>
</tr>
<tr>
<td>45 -</td>
</tr>
<tr>
<td>46 .</td>
</tr>
<tr>
<td>47 /</td>
</tr>
<tr>
<td>48 0</td>
</tr>
<tr>
<td>49 1</td>
</tr>
<tr>
<td>50 2</td>
</tr>
<tr>
<td>51 3</td>
</tr>
<tr>
<td>52 4</td>
</tr>
<tr>
<td>53 5</td>
</tr>
<tr>
<td>54 6</td>
</tr>
<tr>
<td>55 7</td>
</tr>
<tr>
<td>56 8</td>
</tr>
<tr>
<td>57 9</td>
</tr>
<tr>
<td>58 ;</td>
</tr>
<tr>
<td>59 :</td>
</tr>
<tr>
<td>60 &lt;</td>
</tr>
<tr>
<td>61 =</td>
</tr>
<tr>
<td>62 &gt;</td>
</tr>
<tr>
<td>63 ?</td>
</tr>
<tr>
<td>64 @</td>
</tr>
<tr>
<td>96 \</td>
</tr>
<tr>
<td>97 a</td>
</tr>
<tr>
<td>98 b</td>
</tr>
<tr>
<td>99 c</td>
</tr>
<tr>
<td>100 d</td>
</tr>
<tr>
<td>101 e</td>
</tr>
<tr>
<td>102 f</td>
</tr>
<tr>
<td>103 g</td>
</tr>
<tr>
<td>104 h</td>
</tr>
<tr>
<td>105 i</td>
</tr>
<tr>
<td>106 j</td>
</tr>
<tr>
<td>107 k</td>
</tr>
<tr>
<td>108 l</td>
</tr>
<tr>
<td>109 m</td>
</tr>
<tr>
<td>110 n</td>
</tr>
<tr>
<td>111 o</td>
</tr>
<tr>
<td>112 p</td>
</tr>
<tr>
<td>113 q</td>
</tr>
<tr>
<td>114 r</td>
</tr>
<tr>
<td>115 s</td>
</tr>
<tr>
<td>116 t</td>
</tr>
<tr>
<td>117 u</td>
</tr>
<tr>
<td>118 v</td>
</tr>
<tr>
<td>119 w</td>
</tr>
<tr>
<td>120 x</td>
</tr>
<tr>
<td>121 y</td>
</tr>
<tr>
<td>122 z</td>
</tr>
<tr>
<td>123 {</td>
</tr>
<tr>
<td>124</td>
</tr>
<tr>
<td>125 }</td>
</tr>
<tr>
<td>126 ~</td>
</tr>
</tbody>
</table>
Unexpected Inputs

- The '>>' operator can be used to input any number of variables you want to read
- If unexpected non-whitespace characters are encountered, cin simply stops and leaves the variable values unchanged
  - It does not discard the unexpected characters so they will likely cause another error on the next read, too.
  - More on error handling and input validation in CS103

```cpp
#include <iostream>
using namespace std;

int main()
{
    double b, h;
    cin >> b >> h;
    cout << "Area of rect: " << b * h << endl;
    cout << "Area of triangle: " << 0.5 * b * h << endl;
    return 0;
}
```

Input stream:
```
xy 1.2 ...
```

Output:
```
x y 1.2 ...
xy 1.25 2
Area of rect: 5.428E35
Area of triangle: 2.714E35
```
cin Question

- What do you think would happen if the user typed a double when an integer was expected?

- What happens if you type numeric digits when a string is expected?

```cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int x;
    cin >> x; // User types 1.5 42

    double y, z;
    cin >> y >> z;

    string s;
    cin >> s; // User types 103.25

    cout << "x = " << x << endl;
    cout << "y,z= " << y << " " << z << endl;
    cout << "s = " << s << endl;
    return 0;
}
```

1.5 42
103.25
x =
y,z= s =
Common Idioms and Potential Pitfalls

ASSIGNMENT AND ORDERING
Temporal/Sequential Nature of Assignment

- It is critical to remember that assignment:
  - Does **NOT** create a permanent relationship that causes one variable to update if another does
  - Uses the variable values at the time the line of code is executed
  - Copies (not moves) data to the destination variable

- So, the result of assignment statements depend on the order (timing) in which they are executed because one statement may affect the next

```c
int main()
{
    int x = 5;

    // Performs a one-time update of y to 2*5+1=11
    int y = 2 * x + 1;

    // This assignment will NOT cause y to be re-evaluated
    x = 7;

    // y is still 11 and not 15
    cout << "y = " << y << endl;

    // Copies the value of x into y
    y = x;

    // both x and y are 7 now
    cout << x << " " << y << endl;
    return 0;
}
```
Problem Solving Idioms

- An idiom is a colloquial or common mode of expression
  - Example: "raining cats and dogs"
- Programming has common modes of expression that are used quite often to solve problems algorithmically
- We have developed a repository of these common programming idioms. We STRONGLY suggest you
  - Reference them when attempting to solve programming problems
  - Familiarize yourself with them and their structure until you feel comfortable identifying them

### Rule / Exception Idiom

- **Name**: Rule/Exception
- **Description**: Perform a default action and then use an `if` to correspond.
- **Structure**: Code for some default action (i.e. the rule) is followed by an exceptional case.

```c
// Default action
if( /* Exceptional Case */ )
{
    // Code for exceptional case
}
```

- **Example(s)**:
  - Base pay plus bonus for certain exceptional employees

```c
bool earnedBonus = /* set somehow */;
int bonus = /* set somehow */;

int basePay = 100;
if( earnedBonus == true )
{
    basePay += bonus;
}
```

- **Notes**: This can be implemented with an `if/else` where an `if/else` implements the other.
Shifting and Rotation Assignment Idioms

- The **shifting idiom** shifts data among variables usually replacing/dropping some elements to make room for new ones
  - The key pattern is some elements get dropped/overwritten and other elements are reassigned/moved
  - It is important to **start by assigning the variable to be replaced/dropped** and then move in order to variables receiving newer data
  - Examples: Top k items (high score list)

- The **rotation idiom** reorders or rearranges data among variables without replacing/dropping elements
  - Swap is simply a rotation of 2 elements
  - The key pattern is all elements are kept but just reordered
  - It is usually necessary to declare and maintain some temporary variable to avoid elements getting dropped/overwritten
Shifting Idiom Ex. (Insertion)

- Suppose a business represents each client with a 3-digit integer ID (and -1 to mean "free")
  - Lower IDs are given to more important clients
  - Client's with lower ID's always get the appointment time they want
  - Suppose client 105 calls and wants a 2 p.m. appointment, will the highlighted code below work?
- Shifting or rotation?
  - Are we adding/dropping values or keeping all the originals?
- Recall that statements execute one at a time in sequential order
  - Earlier statements complete fully before the next starts

```c
#include <stdio.h>

int main()
{
    // Original appointment schedule
    // Lower client ID gets earlier appointment
    int apt_1pm = 100;
    int apt_2pm = 120;
    int apt_3pm = 140;
    int apt_4pm = -1;

    // Now client 105 wants a 2 p.m. appointment
    apt_2pm = 105;
    apt_3pm = apt_2pm;
    apt_4pm = apt_3pm;

    return 0;
}
```
Shifting Idiom Ex. (Insertion)

- To correctly code the shift, we must start with the variable to be dropped.
- The code to the right does not follow this guideline.
  
  - Perform each highlighted operation one at a time, marking up the diagram below to see the error that results.

```c
int main()
{
    // Original appointment
    // schedule
    // Lower client ID gets
    // earlier appointment
    int apt_1pm = 100;
    int apt_2pm = 120;
    int apt_3pm = 140;
    int apt_4pm = -1;

    // Now client 105 wants
    // a 2 p.m. appointment
    apt_2pm = 105;
    apt_3pm = apt_2pm;
    apt_4pm = apt_3pm;

    return 0;
}
```

```
<table>
<thead>
<tr>
<th>apt_1pm</th>
<th>apt_2pm</th>
<th>apt_3pm</th>
<th>apt_4pm</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>120</td>
<td>140</td>
<td></td>
</tr>
</tbody>
</table>

100/apt_1pm

105
```
Shifting Idiom Ex. (Insertion)

• To correctly code the shift, we must start with the variable to be dropped
  – Move items in reverse order

```c
int main()
{
    // Original appointment
    // schedule
    // Lower client ID gets earlier appointment
    int apt_1pm = 100;
    int apt_2pm = 120;
    int apt_3pm = 140;
    int apt_4pm = -1;

    // Now client 105 wants a 2 p.m. appointment
    apt_4pm = apt_3pm;
    apt_3pm = apt_2pm;
    apt_2pm = 105;

    return 0;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.
Arithmetic Idioms

APPLICATIONS OF DIVISION AND MODULO
Integer Division and Modulo Operations

• Recall integer division discards the remainder (fractional portion)
  – As we apply division to consecutive values, they map to the same output
• Modulo operation yields the remainder of a division of two integers
  – As we apply modulo to consecutive values, they map to different output
  – $x \mod m$ will yield numbers in the range [0 to m-1]
• Example:

<table>
<thead>
<tr>
<th>x</th>
<th>x/5</th>
<th>x%5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

What if we had used 10 rather than 5? What would / and % operations yield?
Integer Division and Modulo Operations

• What if we had replaced 5 with 10?
• Example:

<table>
<thead>
<tr>
<th>input</th>
<th>x</th>
<th>output x/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input</th>
<th>x</th>
<th>output x%10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>
Unit Conversion Idiom

- The unit conversion idiom can be used to convert one value to an integral number of larger units and some number of remaining items.
  - Examples:
    - Ounces to Pounds and ounces
    - Inches to Feet and inches
    - Cents to Quarters, dimes, nickels, pennies

- Approach:
  - Suppose we have \( n \) smaller units (e.g. 15 inches) and a conversion factor of \( k \) small units = 1 large unit, (e.g. 12 inches = 1 foot) then...
  - Using integer division \( (n/k) \) yields the integral number of larger units (15/12 = 1 foot)
  - Using modulo \( (n\%k) \) will yield the remaining number of smaller units (15 % 12 = 3 inches)
Extracting Coordinates

• Suppose you check into a hotel and are told you are in room 632.
  – What floor do you go to?

• A city has odd addresses on one side of the street and even on the other.
  – Given an address (e.g. 3749), how could you determine what side of the street you are on?
Extracting/Isolating Digits Idiom

- To extract or isolate individual digits of a number we can simply divide by the base

- Use modulus (%) to extract the least-significant digits

- Use integer division (\(/\)) to extract the most-significant digits
Consider a 2D grid with 3 rows and 4 columns.

Suppose we assign a linear number to each location as shown.

Given the cell number, how can we determine which row and column it is in?

Given a row and column, can we construct the cell number?
Exercise 1: Unit Conversion Idiom Ex. (Making Change)

- Make change (given 0-100 cents) convert to quarters, dimes, pennies
- cpp/var-expr/change
Exercise 2: Unit Conversion

• Suppose a knob or slider generates a number $x$ in the range 0-255
• Use division or modulo to convert $x$ to a new value, $y$, in the range 0-9 proportionally
• $y = \frac{x}{255}$

Each of the 10 bins = ______ small units
Exercise 3: Isolating Digits Idiom

- Simulate 2 random coin flips producing 2 outcomes (H or T with 50/50 prob.)
- Use `rand()` to generate a random number.
  - `rand()` is defined in `<cstdlib>`
  - Returns a random integer between 0 and \( 2^{31} \)
    - Really \( +2^{31}-1 \)
  - Your job to convert r1 and r2 to either 0 or 1 (i.e. heads/tails) and save those values in flip1 and flip2

```cpp
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
    // Generate a random number
    int r1 = rand();
    // And another
    int r2 = rand();
    int flip1 = ______________
    int flip2 = ______________
    cout << flip1 << flip2 << endl;
    return 0;
}
```

flip1 = ______________
flip2 = ______________
Divisibility / Factoring Idiom

- **Modulo** can be used to check if \( n \) is divisible by \( k \)
  - Definition of divisibility is if \( k \) divides \( n \), meaning remainder is 0
- To factor a number we can divide \( n \) by any of its divisors

\[
12 \mod 5 = 2 \\
\Rightarrow 12 \text{ is NOT divisible by } 5
\]

\[
12 \mod 3 = 0 \\
\Rightarrow 12 \text{ is divisible by } 3
\]

\[
12 / 3 = 4 \\
\Rightarrow 4 \text{ remains after factoring } 3 \text{ from } 12
\]
Challenge Exercise: Weekdays

- `cpp/var-expr/in_n_days`
  - Given the current day of the week (1-7) add n days and indicate what day of the week (1-7) it will be then
- Write out table of examples
  - Input => Desired Output
- Test any potential solution with some inputs
  - `Cday = 1, n = 2...desired outcome = 3`
  - `Cday = 1, n = 6...desired outcome = 7`
- Plug in several values, especially edge cases

```cpp
int main()
{
    int cday, n;
    cin >> cday >> n;
    int day_plus_n = __________________________;
    cout << day_plus_n << endl;
    return 0;
}
```

<table>
<thead>
<tr>
<th>n (assuming c_day=1)</th>
<th>Day_plus_n (desired)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n (assuming c_day=4)</th>
<th>Day_plus_n (desired)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
Extracting Coordinates

• Suppose you check into a hotel and are told you are in room 632.
  – What floor do you go to?
  – Room 632 / 100 rooms/floor = 6th floor

• A city has odd addresses on one side of the street and even on the other.
  – Given an address (e.g. 3749), how could you determine what side of the street you are on?
  – 3749 % 2 rooms
Dimensions

- Consider a 2D grid with 3 rows and 4 columns.
- Suppose we assign a linear number to each location as shown.
- Given the cell number, how can we determine which row and column it is in? \[ \text{row} = \text{cell} \div 4 \] and \[ \text{column} = \text{cell} \mod 4 \]
- Given a row and column, can we construct the cell number? \[ \text{cell} = 4 \times \text{row} + \text{column} \]