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

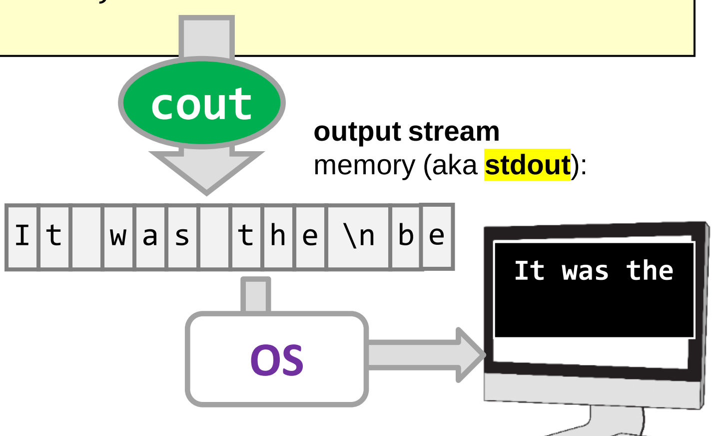
I/O Streams

- 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

input stream memory (aka **stdin**): 7 5 y ...

```
#include<iostream>
int main()
{
  int x;
  std::cin >> x;
  return 0;
}
```

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



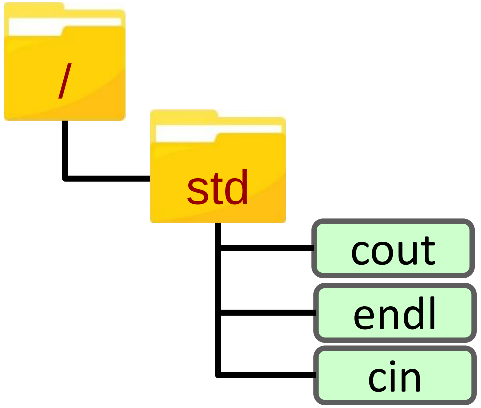
output stream memory (aka **stdout**):

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.

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

```
#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

lec02-cout.cpp

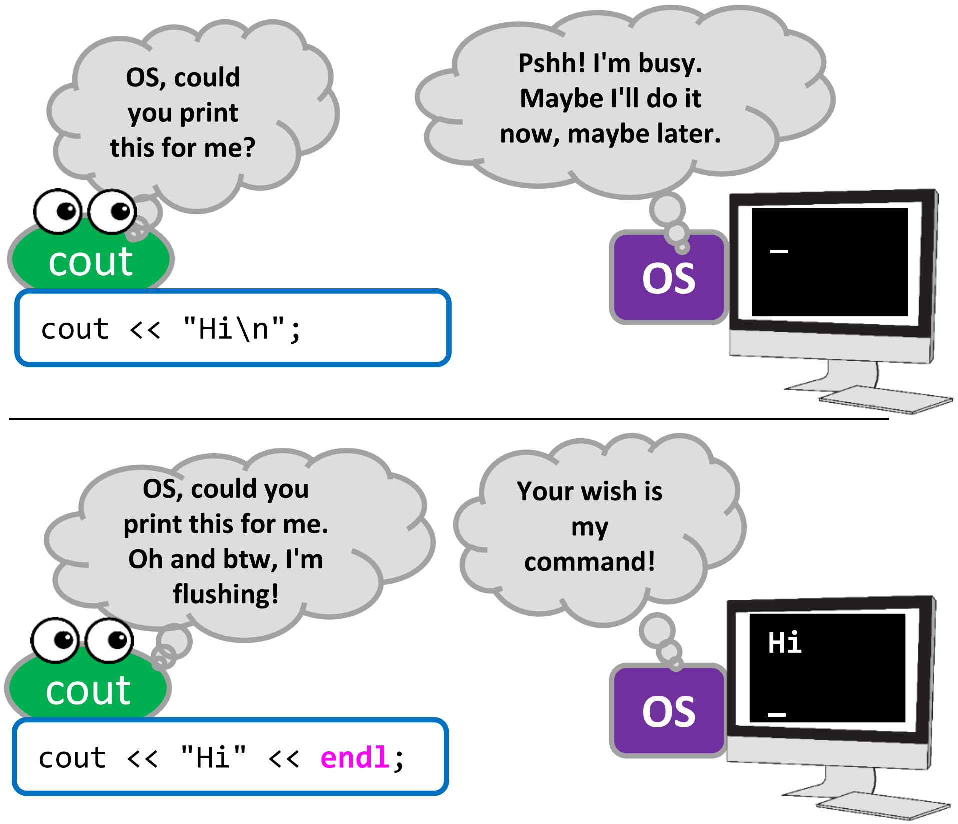
```cpp
#include <iostream>
//using namespace std;

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

user@sahara:~

```
[user@sahara ~]$ make lec02-cout
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;
      |     ^~~~
      |     std::cout
```

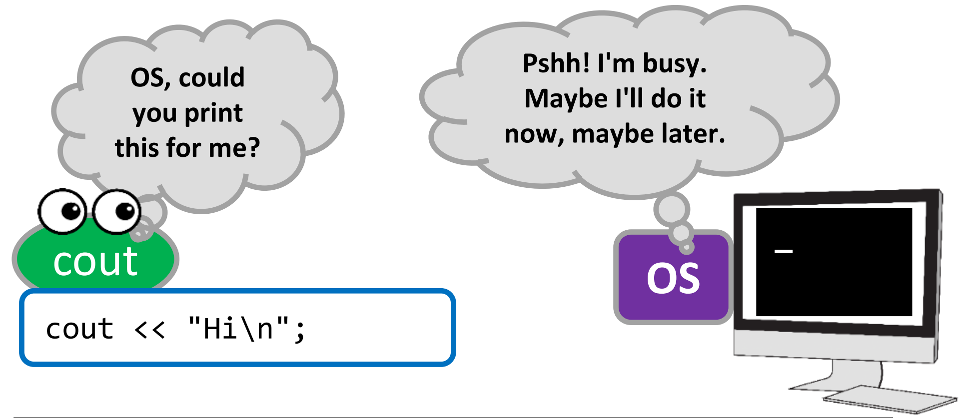
Newlines, endl, and Flushing

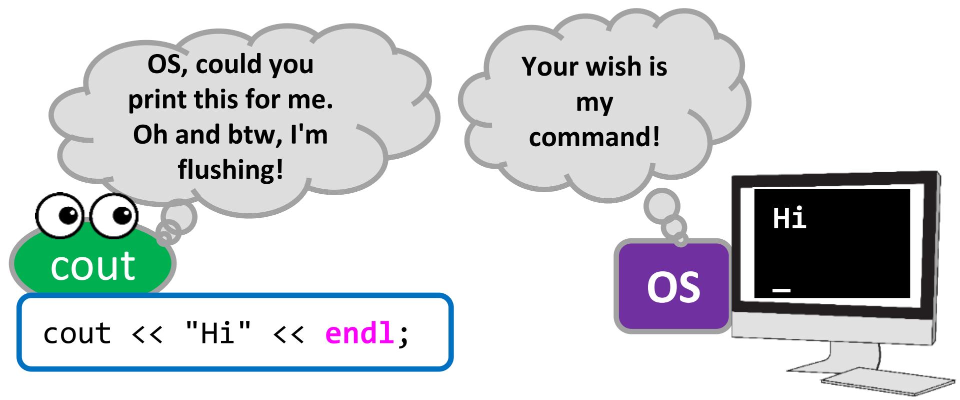
- To move the cursor to the next line we need to print a new line, '\n' (char)
- cout only gives the characters to the OS which then copies them to the screen.
- The OS may choose to delay and not print immediately causing strange issues (see bottom)
- endl = '\n' + a flush of the output stream which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



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






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

```
<Segmentation fault>
```

```
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;
}
```

```
Got Here 1
<Segmentation fault>
```

Use descriptive messages and endls 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

```
#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;
}
```

http://en.cppreference.com/w/cpp/io/manip

See "iomanip" in-class exercise to explore various options

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 **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

```
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)
x = '1'
cout << x << endl; // prints: 49
```




char c
97

int x
97

char d
98

int x
49

ASCII printable characters

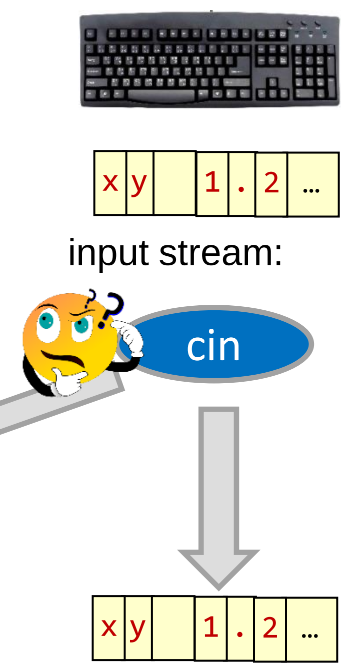
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

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

```
#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;
}
```

??	??
b	h



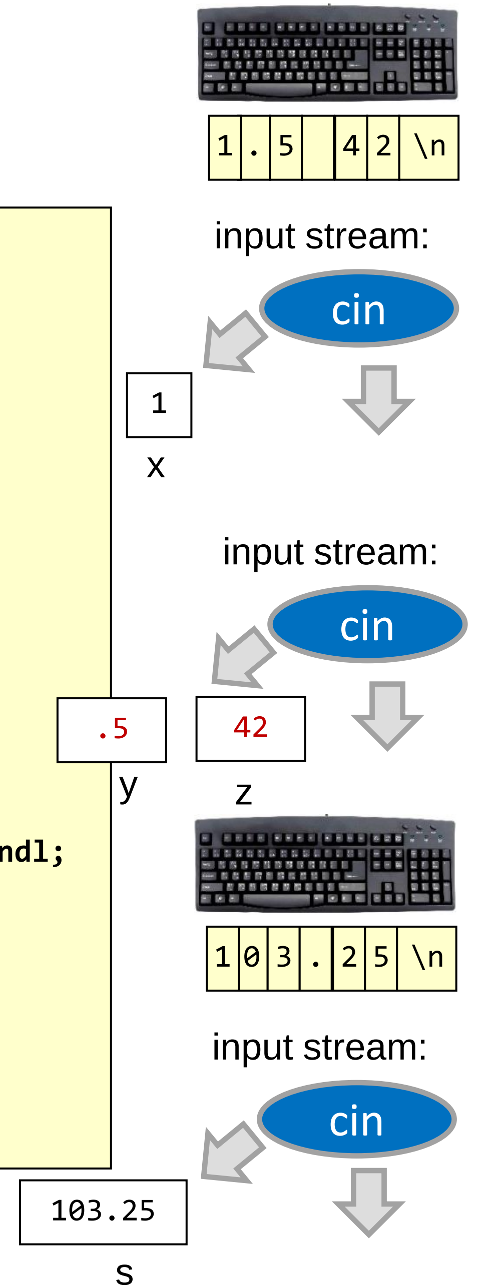

```
xy 1.25 2
Area of rect: 5.428E35
Area of triangle: 2.714E35
```

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?

```
#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

```
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 us an `if` to corre
- **Structure**: Code for some default action (i.e. the rule) is followed b exceptional case

```
// Default action

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

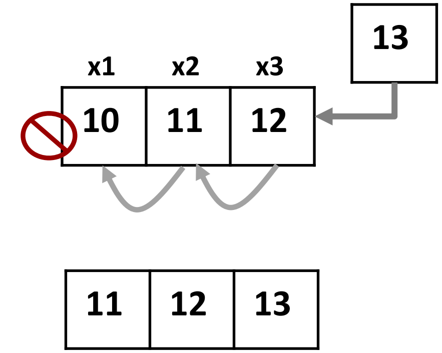
- **Example(s)**:
- Base pay plus bonus for certain exceptional employees

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

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

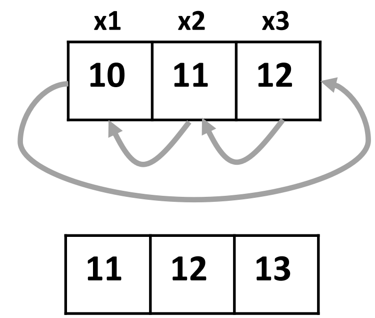
- **Notes**: This can be implemented with an `if/else` where an else implements the other.

Shifting and Rotation Assignment Idioms

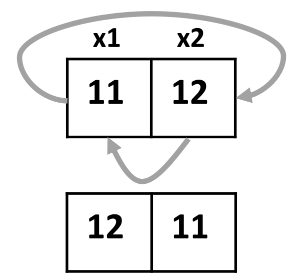
- The **shifting idiom** shifts data among variables usually replacing/dropping some elements to make room for new ones
  - The key pattern is some elements get **dropped/overwritten** and other elements are **reassigned/moved**
  - It is important to **start by assigning the variable to be replaced/dropped** 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



Rotation Idiom



Swap

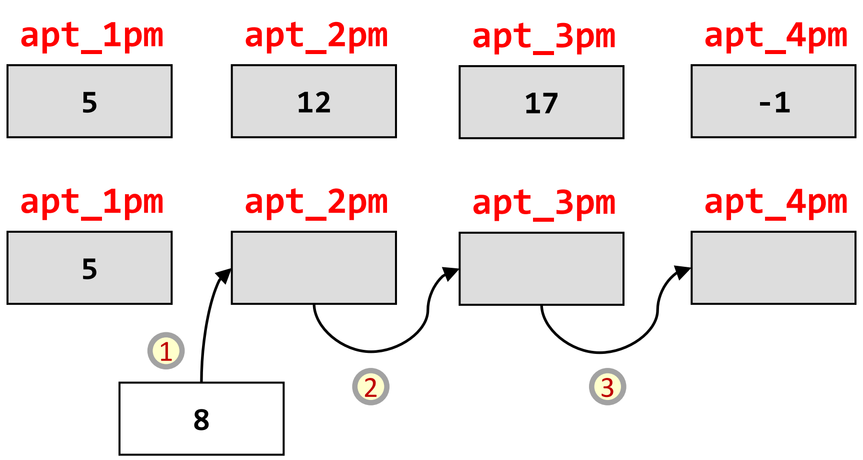
Shifting Idiom Ex. (Insertion)

- 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

```
int main()
{
  // Original appointment
  //  schedule
  // Lower client ID gets
  //  earlier appointment
  int apt_1pm = 5;
  int apt_2pm = 12;
  int apt_3pm = 17;
  int apt_4pm = -1;

  // Now client 8 wants
  //  a 2 p.m. appointment
  apt_2pm = 8;
  apt_3pm = apt_2pm;
  apt_4pm = apt_3pm;

  return 0;
}
```

Shifting Idiom Ex. (Insertion)

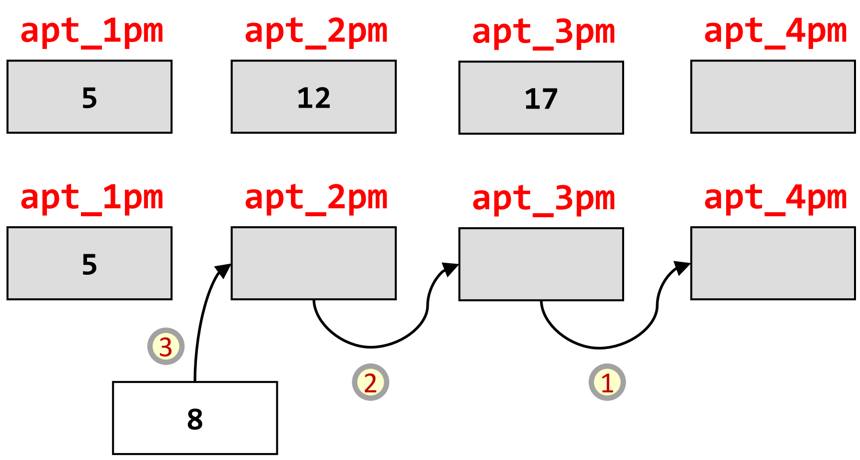
- To correctly code the shift, we must start with the variable to be dropped
- The code to the right does not follow this guideline
  - Perform each highlighted operation one at a time, marking up the diagram below to see the error that results




```
int main()
{
   // Original appointment
   //  schedule
   // Lower client ID gets
   //  earlier appointment
   int apt_1pm = 5;
   int apt_2pm = 12;
   int apt_3pm = 17;
   int apt_4pm = -1;

   // Now client 8 wants
   //  a 2 p.m. appointment
   apt_2pm = 8;
   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
  - Move items in reverse order



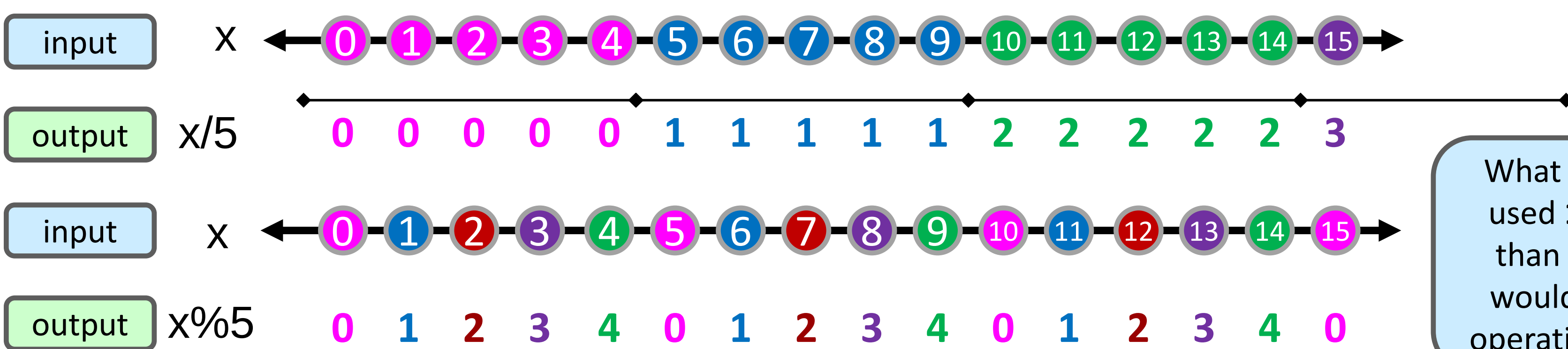

```
int main()
{
  // Original appointment
  //  schedule
  // Lower client ID gets
  //  earlier appointment
  int apt_1pm = 5;
  int apt_2pm = 12;
  int apt_3pm = 17;
  int apt_4pm = -1;

  // Now client 8 wants
  //  a 2 p.m. appointment
  apt_4pm = apt_3pm;
  apt_3pm = apt_2pm;
  apt_2pm = 8;

  return 0;
}
```

Arithmetic Idioms

APPLICATIONS OF DIVISION AND MODULO

Integer Division and Modulo Operations

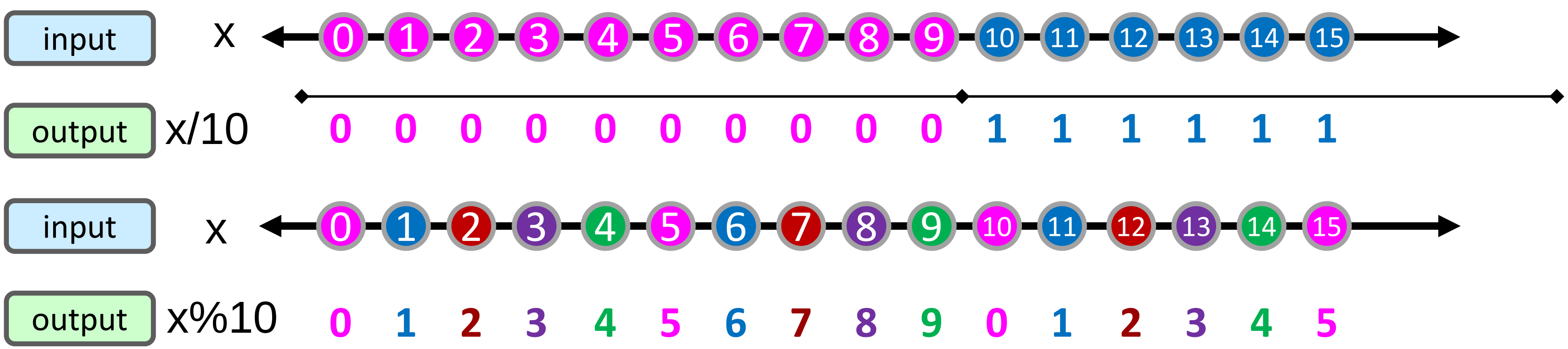
- Recall integer division yields only the quotient and discards the remainder (fractional portion)
  - As we apply **division** to consecutive values, they map to the **same output**
- Modulo operation yields the remainder (and discards the quotient)
  - As we apply **modulo** to consecutive values, they map to **different output**
  - *x mod m* will yield numbers in the range [0 to m-1]
- Example:



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:



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

957 dec. = 9 5 7.

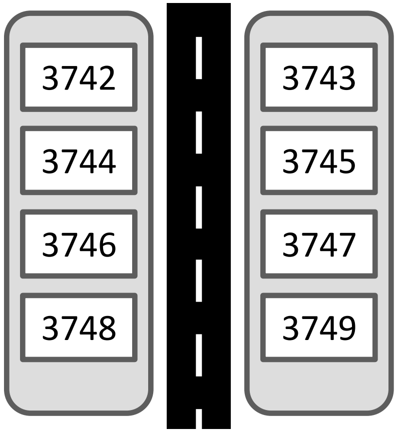
	9	5	7.
957 dec. =	100	10	1

```
957 %  10 = 7
957 /  10 = 95

957 % 100 = 57
957 / 100 = 9
```

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?

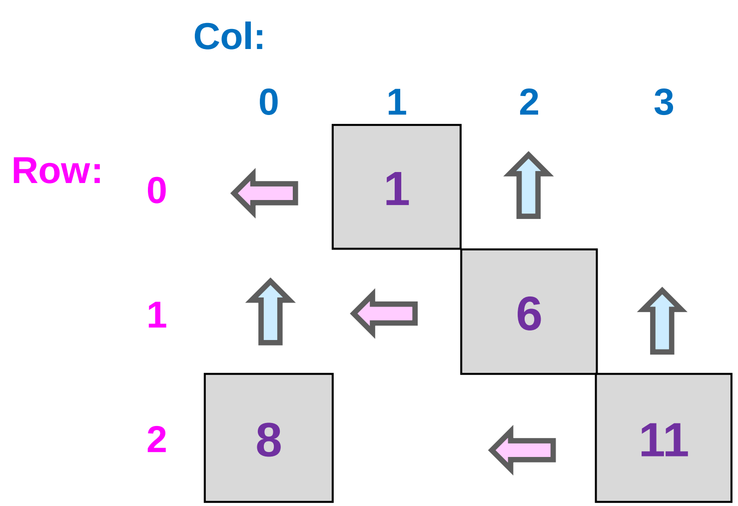


Dimensions

- 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?
- Given a row and column, can we construct the cell number?

Col:

Row:	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11



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 % 5 = 2
=> 12 is NOT divisible by 5

12 % 3 = 0
=> 12 is divisible by 3

12 / 3 = 4
=> 4 remains after
=> factoring 3 from 12
```

Unit Conversion Idiom

- The unit conversion idiom can be used to convert one value to 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)

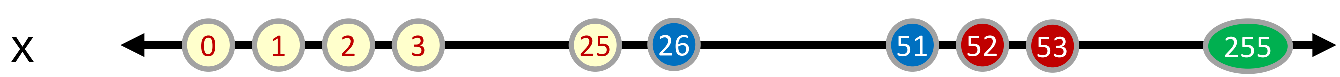
Exercise 1: Unit Conversion Idiom Ex. (Making Change)

- 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 = x ___________________

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 *about* 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

```
#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 = ______________

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

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

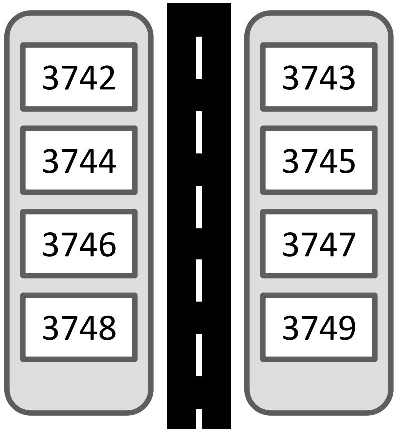
n (assuming c_day=1)	Day_plus_n (desired)
1	2
2	3
3	4
4	5
5	6
6	7
7	1
8	2

n (assuming c_day=4)	Day_plus_n (desired)
1	5
2	6
3	7
4	1
5	2
6	3
7	4
8	5

SOLUTIONS

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? [row = `cell / 4` and column = `cell % 4`
- Given a row and column, can we construct the cell number? `cell = 4*row + column`

Row: \ Col:	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11