Unit 4

Input (cin)

More Assignment Statements

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

VARIABLES

The Need For Variables & Input

- Printing out constants is not very useful (nor exciting)
- In fact, we could just as easily compute the value ourselves in many situations
- The real power of computation comes when we introduce variables and user input
  - Variables provide the ability to remember and name a value for use at a later time
  - User input allows us to write general programs that work for "any" input values
  - Thus, a more powerful program would allow us to enter an arbitrary number and perform conversion to dozens

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  cout << "3 dozen is " << 3*12 << " items." << endl;

  // the above results in the same output as below

  cout << "3 dozen is 36 items." << endl;

  return 0;
}
```

C/C++ Variables

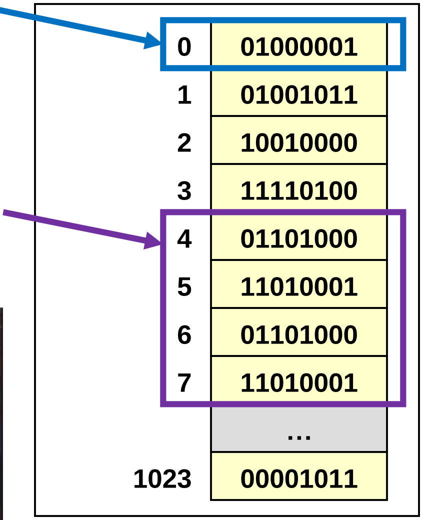
- Variables allow us to
  - **Store a value until it is needed and change its values potentially many times**
  - **Associate a descriptive name with a value**
- Variables are just memory locations that are reserved to store a piece of data of specific size and type
- Programmer indicates what variables they want when they write their code
  - Difference: C requires declaring all variables at the beginning of a function before any operations. C++ relaxes this requirement.
- The computer will allocate memory for those variables when the code starts to run
- We can provide initial values via '=' or leave them uninitialized

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

int main()
{ // Sample variable declarations
  char gr = 'A';
  int x;   // uninitialized variables
           // will have a (random) garbage
           // value until we initialize it
  x = 1;   // Initialize x's value to 1
  gr = 'B'; // Change gr's value to 'B'
}
```

char gr = 'B';
A single-byte variable

int x;
A four-byte variable

A picture of computer memory (aka RAM)



0	01000001
1	01001011
2	10010000
3	11110100
4	01101000
5	11010001
6	01101000
7	11010001
	…
1023	00001011

Variables are actually allocated in RAM when the program is run

C/C++ Variables

- Variables have a:
  - type [`int, char, unsigned int,float, double`, etc.]
  - name/identifier that the programmer will use to reference the value in that memory location [e.g. `x, myVariable, num_dozens`, etc.]
    - Identifiers must start with [A-Z, a-z, or an underscore '_'] and can then contain any alphanumeric character [0-9, A-Z, a-z, _] (but no punctuation other than underscores)
    - Use descriptive names (e.g. numStudents, doneFlag)
    - Avoid cryptic names (myvar1, a_thing)
  - location [the address in memory where it is allocated]
  - Value
- Reminder: You must declare a variable before using it

What's in a name?

To give descriptive names we often need to use more than 1 word/term. But we can't use spaces in our identifier names. Thus, most programmers use either camel-case or snake-case to write compound names

Camel case: Capitalize the first letter of each word (with the possible exception of the first word)

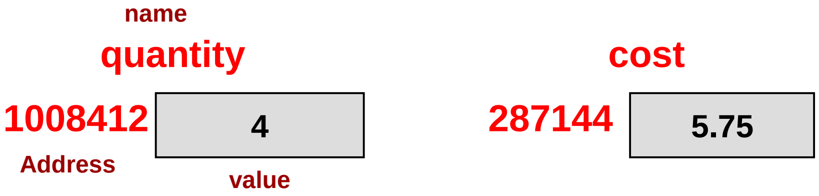
myVariable, isHighEnough

Snake case: Separate each word with an underscore '_'

my_variable, is_high_enough

Code

```
int quantity = 4;
double cost = 5.75;
cout << quantity*cost << endl;
```



name

quantity

1008412 | 4

Address

value

cost

287144 | 5.75

Know Your Common Variable Types

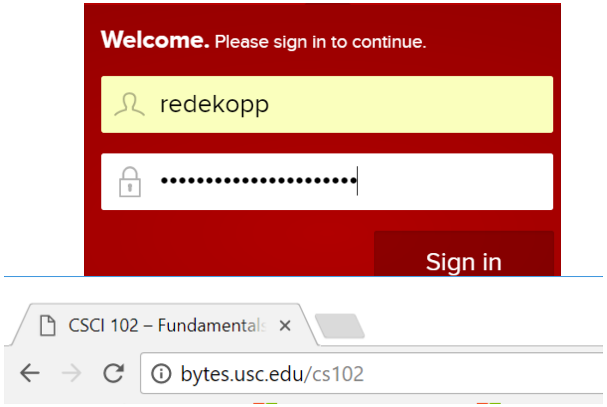
- Variables are declared by listing their type and providing a name
- They can be given an initial value using the '=' operator

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  int w = -400;
  double x = 3.7;
  char y = 'a';
  bool z = false;
  cout << w << " " << x << " ";
  cout << y << " " << z << endl;
  return 0;
}
```

C Type	Usage	Bytes	Bits	Range
char	Text character Small integral value	1	8	ASCII characters -128 to +127
bool	True/False value	1	8	true / false
int unsigned int	Integer values	4	32	-2 billion to +2 billion 0 to +4 billion
double	Rational/real values	8	64	±16 significant digits * $10^{+/-308}$
string	Arbitrary text	-	-	-

When Do We Need Variables?

- When a value will be supplied and/or change at run-time (as the program executes)

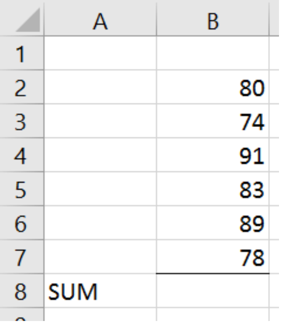


Welcome. Please sign in to continue.

redekopp

••••••••••••••••••••••

- When a value is computed/updated at one time and used (many times) later



	A	B
1		
2		80
3		74
4		91
5		83
6		89
7		78
8	SUM	

- To make the code more readable by another human

```
double area = (56+34) * (81*6.25);
// readability of above vs. below
double height = 56 + 34;
double width =  81 * 6.25;
double area = height * width;
```

What Variables Might Be Needed

- Calculator App
  - Current number input, current result
- Video playback (YouTube player)
  - Current URL, full screen, volume level

Assignment (=) Operator

- To update or change a value in a variable we use the assignment operator (=)
- Syntax:
  - `variable = expression;`
    `(Left-Side)  (Right-side)`
- Semantics:
  - Place the resulting value of 'expression' in the memory location associated with 'variable'
  - Does not mean "compare for equality" (e.g. is w equal to 300?)
    - That is performed by the == operator

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  int w;  // variables don't have to
  char x; //  be initialized when declared

  w = 300;
  x = 'a';
  cout << w << " " << x << endl;

  w = -75;
  x = '!';
  cout << w << " " << x << endl;
  return 0;
}
```

Output:

```
300 a
-75 !
```

Order of evaluation: right to left

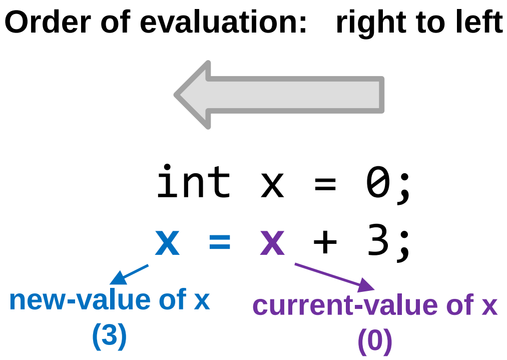
`variable = expression;`

Assignment is one of the most common operations in programs

Assignment & Expressions

- Variables can be used in expressions and be operands for arithmetic and logic
- See inset below on how to interpret a variable's usage based on which side of the assignment operator it is used

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  int dozens = 3;
  double gpa = 2.0;

  int num = 12 * dozens;
  gpa = (2 * 4.0) + (4 * 3.7);  // gpa updated to 22.8
  gpa = gpa / 6;  // integer or double division?

  cout << dozens << " dozen is " << num << " items." << endl;
  cout << "Your gpa is " << gpa << endl;
  return 0;
}
```



```
int x = 0;
x = x + 3;
```

Semantics of variable usage:

- Right-side of assignment: Substitute/use the current value stored in the variable
- Left-side of assignment: variable is the destination location where the result of the right side will be stored

Exercises

- What is printed by the following two programs?

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

int main()
{
  int value = 1;
  value = (value + 5) * (value – 3);
  cout << value << endl;

  double amount = 2.5;
  value = 7;
  amount = value + 6 / amount;
  cout << amount << endl;

  cout << value % 3 << endl;
  return 0;
}
```

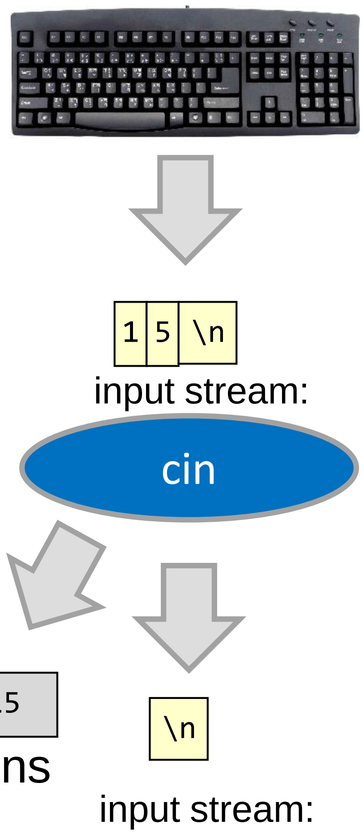
```
#include <iostream>
using namespace std;

int main()
{
  int x = 5;
  int y = 3;
  double z = x % y * 6 + x / y;

  cout << z << endl;

  z = 1.0 / 4 * (z – x) + y;
  cout << z << endl;

  return 0;
}
```

RECEIVING INPUT WITH CIN

Keyboard Input

- In C++, the 'cin' object is in charge of receiving input from the keyboard
- Keyboard input is captured and stored by the OS (in an "input stream") until cin is called upon to "extract" info into a variable
- 'cin' converts text input to desired format (e.g. integer, double, etc.)

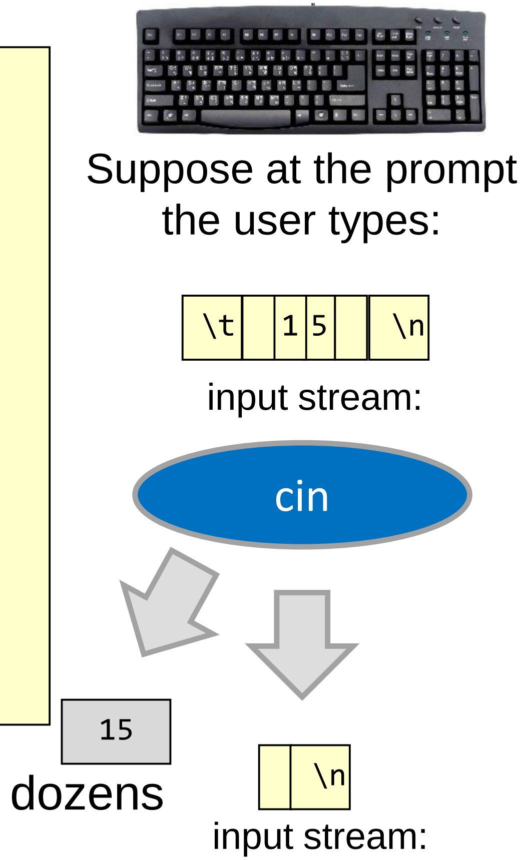
```
#include <iostream>
using namespace std;

int main()
{
  int dozens;

  cout << "Enter number of dozen: "
       << endl;
  cin  >> dozens;

  cout << 12 * dozens << " eggs" << endl;
  return 0;
}
```



Dealing With Whitespace

- **Whitespace *(def.)*:**
  - Characters that represent horizontal or vertical blank space. Examples: newline ('\n'), TAB ('\t'), spacebar (' ')
- cin sequentially scans the input stream for actual characters, discarding leading whitespace characters
- Once cin finds data to convert it will STOP at the first trailing whitespace and await the next cin command

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

int main()
{
  int dozens;

  cout << "Enter number of dozen: "
       << endl;
  cin  >> dozens;

  cout << dozens << " dozen "
       << " is " << 12*dozens
       << "items." << endl;

  return 0;
}
```



Suppose at the prompt the user types:

\t		1	5		\n

input stream:

cin

15

dozens

	\n

input stream:

Main Take-away:

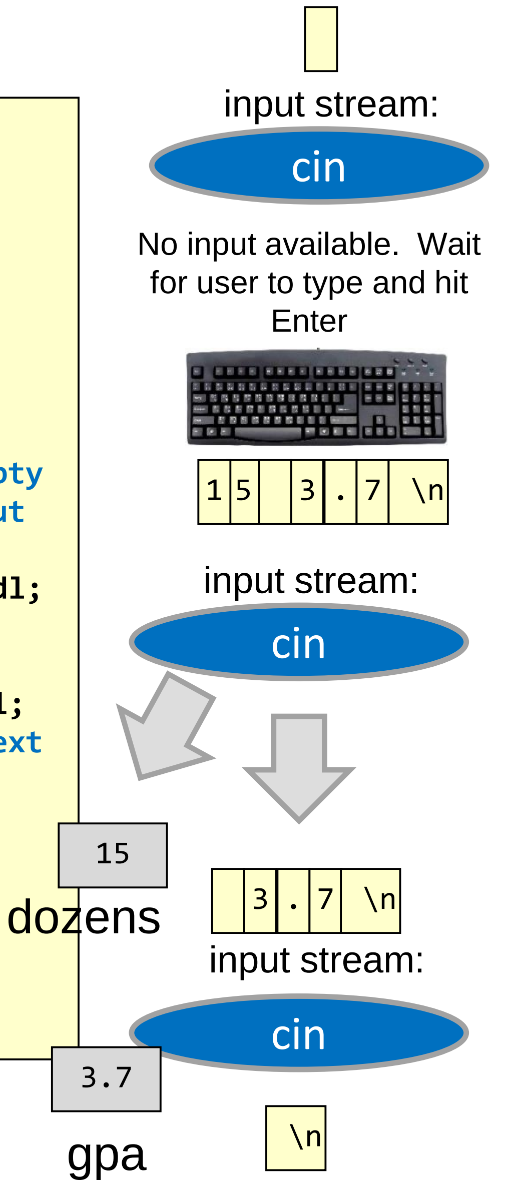
cin SKIPS leading whitespace

cin STOPS on the first trailing whitespace

Timing of Execution

- When execution hits a 'cin' statement it will:
  - Wait for input if nothing is available in the input stream
    - OS will capture what is typed until the next 'Enter' key is hit
    - User can type as little or much as desired until Enter (\n)
  - Immediately extract input from the input stream if some text is available and convert it to the desired type of data

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

int main()
{
  int dozens;

  cout << "Enter number of dozen: "
       << endl;
  cin  >> dozens; // input stream empty
                  // so wait for input

  cout << 12*dozens << " eggs" << endl;

  double gpa;
  cout << "What is your gpa?" << endl;
  cin  >> gpa; // input stream has text
               // so do not wait…
               // just use next text

  cout << "GPA = " << gpa << endl;
  return 0;
}
```



input stream:

cin

No input available. Wait for user to type and hit Enter

1	5		3	.	7	\n

input stream:

cin

15

dozens

	3	.	7	\n

input stream:

cin

3.7

gpa

\n

Excercises

- cpp/cin/building_floor

Common Idioms and Potential Pitfalls

ASSIGNMENT AND ORDERING

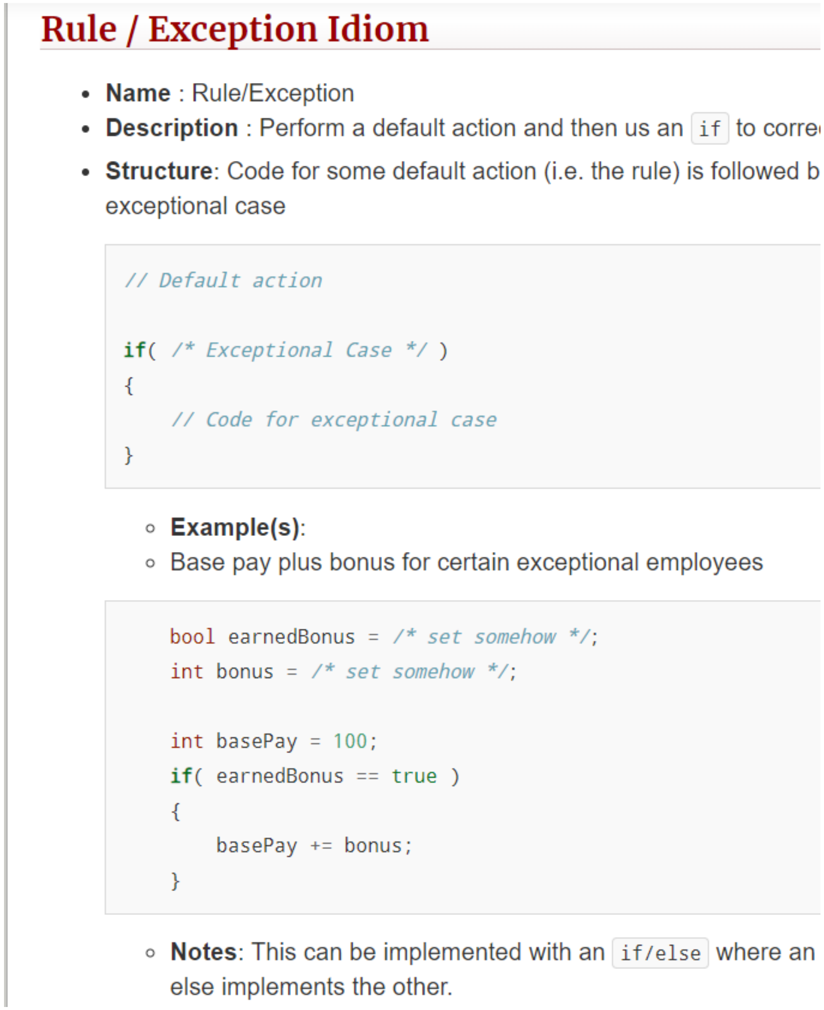
Temporal/Sequential Nature of Assignment

- It is critical to realize 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
}
```

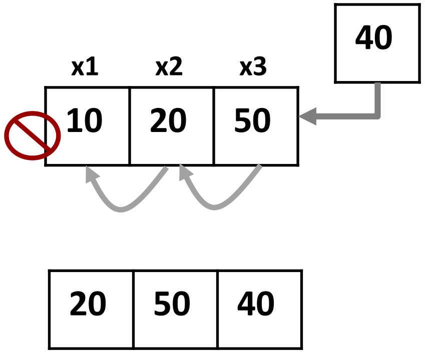
- **Example(s)**:
- Base pay plus bonus for certain exceptional employees

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

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

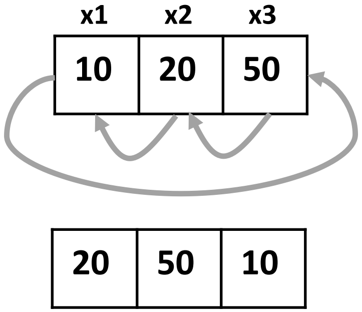
- **Notes**: This can be implemented with an `if/else` where an else implements the other.

Shifting and Rotation Assignment Idioms

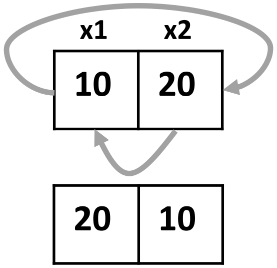
- The **shifting idiom** shifts data among variables usually replacing/dropping some elements to make room for new ones
  - The key pattern is some elements get **dropped/overwritten** and other elements are **reassigned/moved**
  - It is important to **start by assigning the variable to be replaced/dropped**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

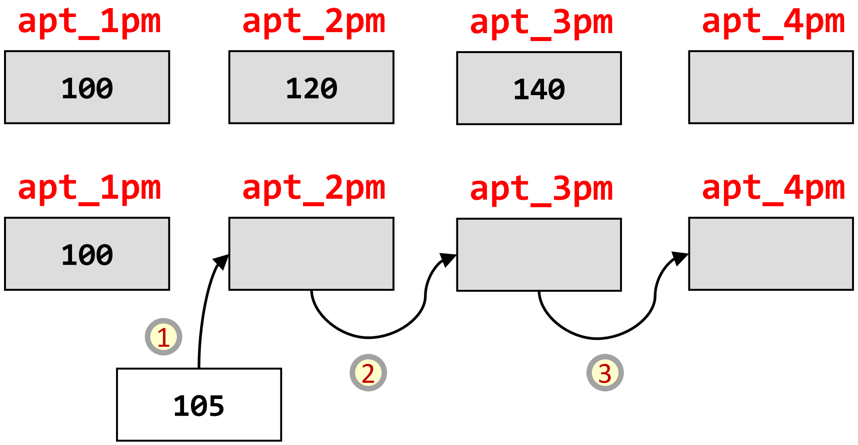
Shifting Idiom Ex. (Insertion)

- 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 = 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)

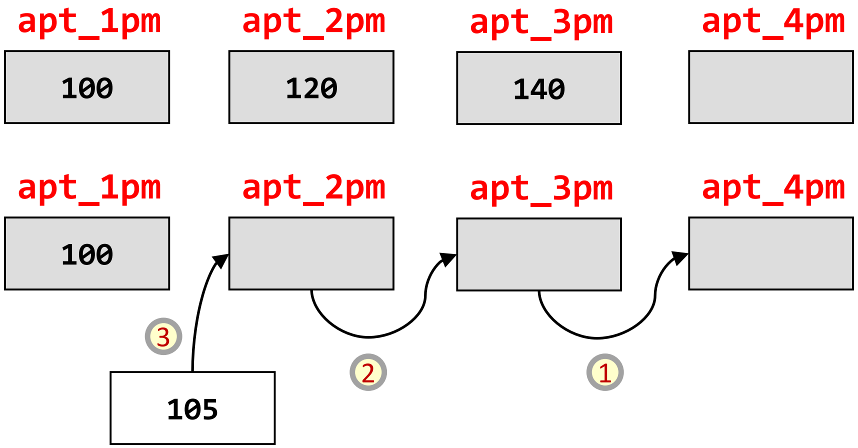
- To correctly code the shift, we must start with the variable to be dropped
- The code to the right does not follow this guideline
  - Perform each highlighted operation one at a time, marking up the diagram below to see the error that results



```
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
  - Move items in reverse order

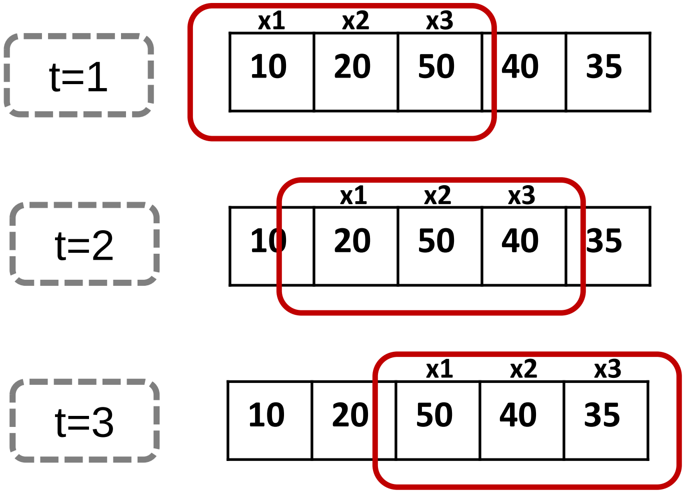


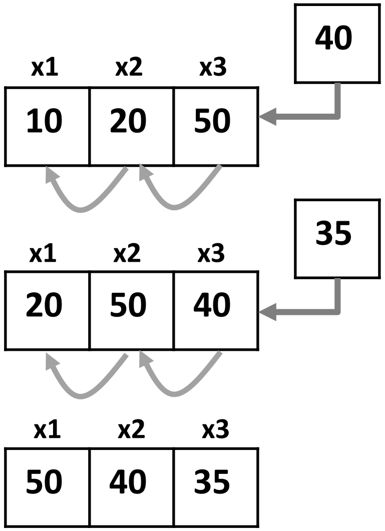
```
int main()
{
  // Original appointment
  //  schedule
  // Lower client ID gets
  //  earlier appointment
  int apt_1pm = 100;
  int apt_2pm = 120;
  int apt_3pm = 140;
  int apt_4pm = -1;

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

  return 0;
}
```

Shifting Idiom Ex. (Moving-Window)

- Suppose we only want to work with the last *k* (let *k=3* for this example) value input by the user
  - Declare k variables (i.e. x1, x2, x3)
  - As we receive new values we drop the undesired values shifting the current values as needed via assignment operations

```
int x1 = 10, x2 = 20, x3 = 50;
```

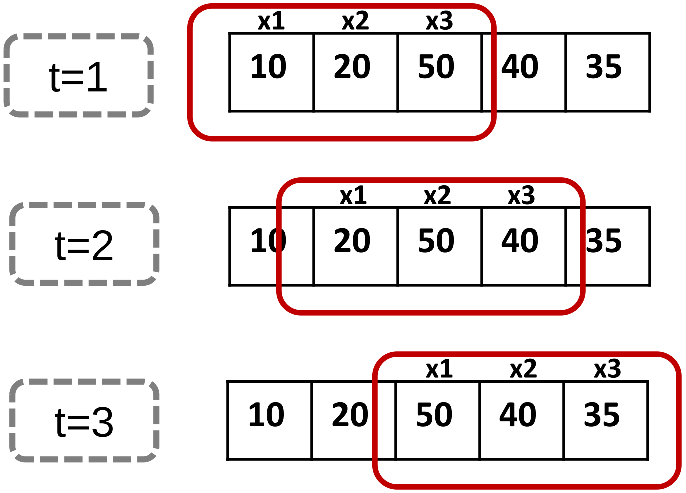


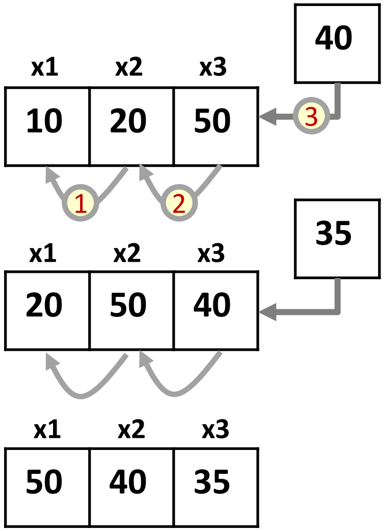


Shifting Values (Moving Window) Idiom

- Remember, ***order*** of assignment is **very important** to avoid overwriting data we still need
- Start by assigning the value to be overwritten/dropped…
- Continue assigning in order until reaching the variable that should receive the new value

```
int x1 = 10, x2 = 20, x3 = 50;
```

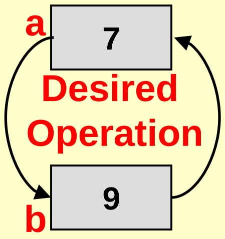


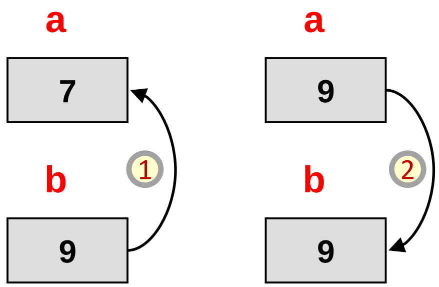


Rotation Idiom Ex. (Swap)

- Given two variables, swap their contents
  - Before: a = 7, b = 9
  - Desired Result: a = 9, b = 7
- This is rotation because we want to keep all values and just reorder them
- Since shifting requires us to start with the variable to be overwritten/dropped and we want to keep both values, no order of assignment will work without a temporary variable!
- Perform the code to the right to see the error:
  - Actual Result: a = ___, b = ___;

```
int main()
{
  int a = 7, b = 9;

  // Now suppose we want to
  //   swap the values of
  //   a and b

  // What will this do?
  a = b;
  b = a;


  return 0;
}
```

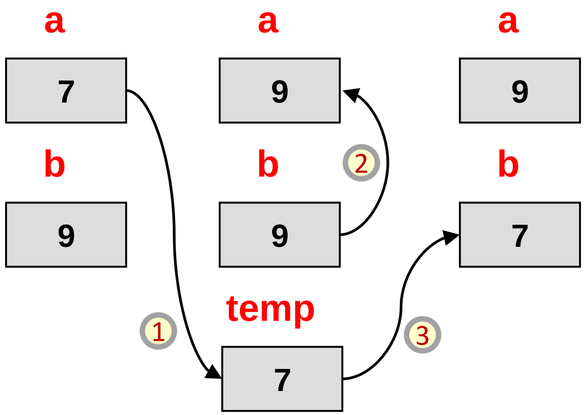




Rotation Idiom Ex. (Swap)

- We need an extra, temporary location to hold the old value of one of the variables while we update it to the new value

```
int main()
{
  int a = 7, b = 9;

  // Now suppose we want to
  //  swap the values of
  //  a and b

  // Introduce a temp var.
  int temp = a;
  a = b;
  b = temp;

  return 0;
}
```



MORE OPERATIONS AND USING MATH LIBRARY FUNCTIONS

Shortcut Assignment Statements

- A common task is to update a variable by adding, subtracting, multiplying, etc. some value to it
  - `x = x + 4;`
  - `y = y * 2.5;`
- C/C++ provide a shortcut for writing these statements:
  - `x += 4;`
  - `y *= 2.5;`
- The substitution is:
  - `var op= expr;`
  - Becomes `var = var op expr;`

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

int main()
{
  int x = 1;
  double y = 3.75;

  x += 5;    // x updates to 6
  y -= 2.25; // y updates to 1.5
  x /= 3;    // x updates to 2
  y *= 2.0   // y updates to 3.0

  return 0;
}
```

Post-Increment/Decrement

- Adding 1 to a variable (e.g. `x += 1`) and subtracting 1 from a variable (e.g. `x -= 1`) are extremely common operations (especially when we cover loops).
- The `++` and `--` operators offer a shortcut to "increment-by-1" or "decrement-by-1"
  - Performs (`x += 1`) or (`x -= 1`)
  - `x++; // If x was 2 it will be updated to 3 (x = x + 1)`
  - `x--; // If x was 2 it will be updated to 1 (x = x – 1)`

- Note: There are some nuances to this operator and an alternative known as pre-increment/decrement that we will discuss in future lectures but this is sufficient for now.

Casting Motiviation

- To achieve the correct answer for 5 + 3 / 2 we could…
- Make everything a double
  - Write 5.0 + 3.0 / 2.0 [explicitly use doubles]
- Use **implicit** casting (mixed expression)
  - Could just write 5 + 3.0 / 2
    - If operator is applied to mixed type inputs, less expressive type is automatically promoted to more expressive (int => double)
- But what if instead of constants we have variables
  - `int x=5, y=3, z=2;`
    `x + y/z;`   // Won't work & you can't write **y.0**
- We need a way to explicitly cast a variable to a different type for the sake of a computation

Casting

- To cast a variable, place the type to which you wan to cast in parentheses BEFORE the variable
- Casting is the only way to convert a **variable** to a different numeric type
  - `x +  (double) y  /  z ;` // z will be implicitly cast to a double
- This won't work
  - `x +  (double) (y  /  z) ;` // the integer division in parens goes first
- Notes:
  - Only changes the type temporarily for the sake of the expression (not a permanent type change)
  - Only works on numeric types and not strings
    - Can't cast an integer/double to a character or string
    - `double x = 1.6;  int y = (int) x / 2;     // fine !`
    - `int x = 123;     string y = (string) x;  // doesn't work`
    - `int x = (string) "123";                  // doesn't work`

Math & Other Library Functions

- C++ predefines a variety of functions for you. Here are a few of them:
  - `sqrt(x)`: returns the square root of x (in <cmath>)
  - `pow(x, y)`: returns x^y, or x to the power y (in <cmath>)
  - `sin(x)/cos(x)/tan(s)`: returns the sine of x if x is in radians (in <cmath>)
  - `abs(x)`: returns the absolute value of x (in <cstdlib>)
  - `max(x, y)` and `min(x,y)`: returns the maximum/minimum of x and y (in <algorithm>)
- You call these by writing them similarly to how you would use a function in mathematics [using parentheses for the inputs (aka) arguments]
- Result is replaced into bigger expression
- Must #include the correct library
  - #includes tell the compiler about the various pre-defined functions that your program may choose to call

```
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

int main()
{
  // can call functions
  //  in an assignment
  double res = cos(0); // res = 1.0

  // can call functions in an
  //  expression
  res =  sqrt(2) / 2; // res = 1.414/2

  cout << max(34, 56) << endl;
  // outputs 56

  return 0;
}
```

http://www.cplusplus.com/reference/cmath/

Statements

- C/C++ programs are composed of statements
- Most common kinds of statements end with a semicolon
- Declarations (e.g. `int x=3;`)
- Assignment + Expression (suppose **int x=3; int y;**)
  - `x = x * 5 / 9;` // compute the expression & place result in x
    `// x = (3*5)/9 = 15/9 = 1`
- Assignment + Function Call (+ Expression)
  - `x = cos(0.0) + 1.5;`
  - ~~`sin(3.14)`~~`;` // Must save or print out the result (x = sin(3.14), etc.)
- cin, cout statements
  - `cout << cos(0.0) + 1.5 << " is the answer." << endl;`
- Return statement (immediately ends a function)
  - `return value;`
  - More on this in Unit 6

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 other 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) << 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

Exercises

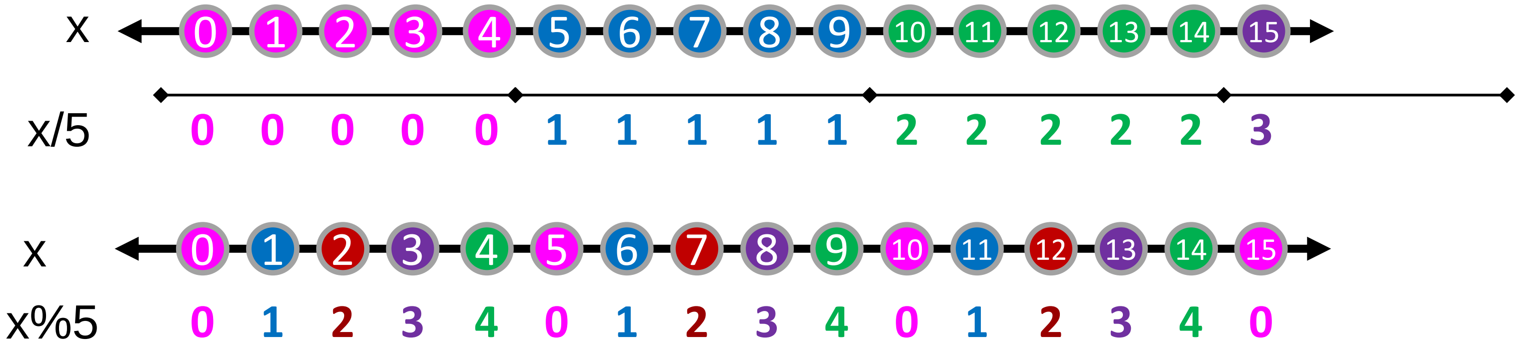
- Exercises:
  - cpp/cin/average
  - cpp/cin/rad2deg
- Write a program to convert temperature from Celsius to Fahrenheit [$F = \frac{9}{5} \cdot C + 32$]
  - Use http://cpp.sh or http://onlinegdb.com

Arithmetic Idioms

APPLICATIONS OF DIVISION AND MODULO

Integer Division and Modulo Operations

- Recall integer division discards the remainder (fractional portion)
  - Consecutive values map to the **same value**
- Modulo operation yields the remainder of a division of two integers
  - Consecutive values map to **different values**
  - *x mod m* will yield numbers in the range [0 to m-1]
- Example:



x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x/5	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	3

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x%5	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0

Unit Conversion Idiom

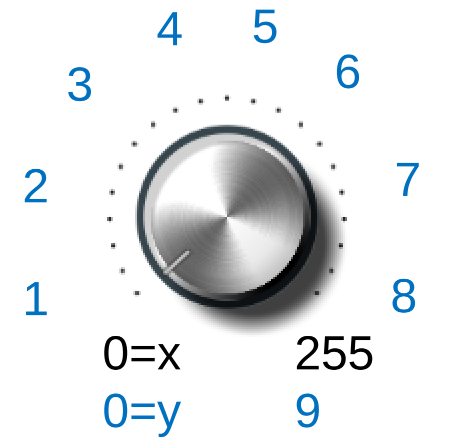
- The unit conversion idiom can be used to convert one value to integral number of larger units and some number of remaining items
  - Examples:
    - Ounces to Pounds and ounces
    - Inches to Feet and inches
    - Cents to Quarters, dimes, nickels, pennies
- Approach:
  - Suppose we have **n smaller 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)

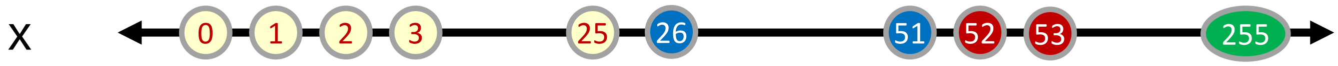
- Make change (given 0-100 cents) convert to quarters, dimes, pennies
- cpp/var-expr/change

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



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.	0	0
	100	10	1	0.1	0.01

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

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

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 = ______________________

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
```

Challenge Exercise

- 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 = ______________________;

  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