Unit 4

Statements

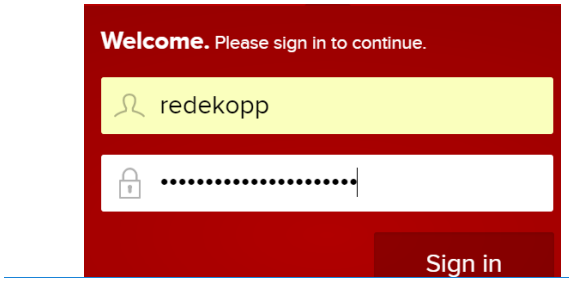
Input (cin)

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

When To Introduce a Variable

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



- 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

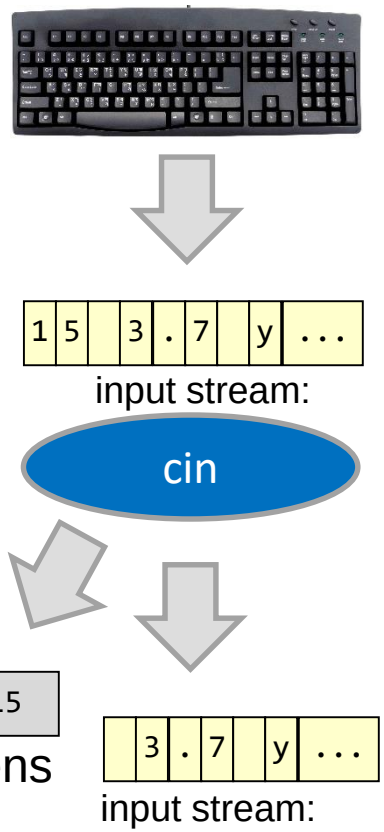
- Calculator
  - Current number input, current result
- Video playback (YouTube player)
  - Current URL, full screen, volume level

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 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 << dozens << " dozen "
       << " is " << 12*dozens
       << "items." << endl;
  return 0;
}
```

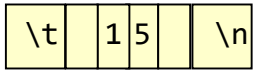


Dealing With Whitespace

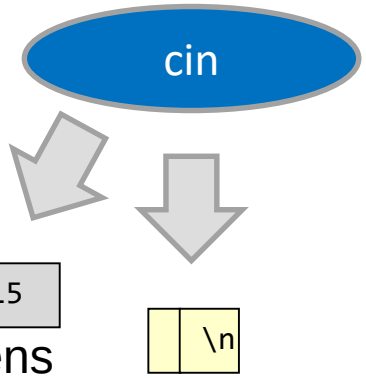
- **Whitespace *(def.):***
  - Characters that represent horizontal or vertical blank space. Examples: newline (`'\n'`), TAB (`'\t'`), spacebar (`' '`)
- cin sequentially scans the input stream for actual characters it can convert to the desired variable type (integer, double, etc.)
- **During this scan, cin:**
  - **Discards (skips) leading whitespaces**
  - **Once cin finds data to convert it will STOP at trailing whitespace**

```
#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:



input stream:

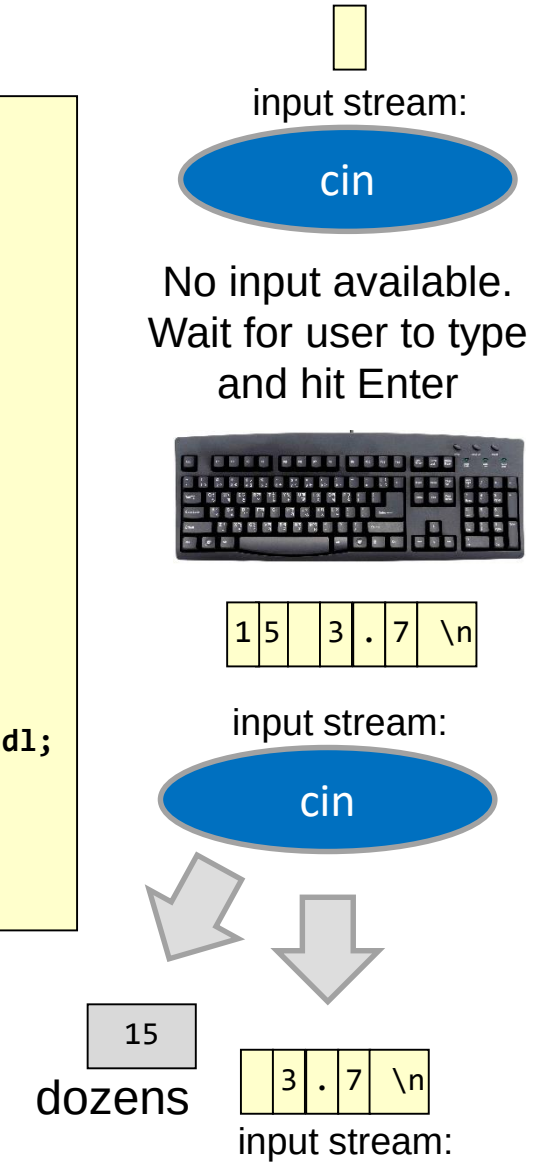


Main Take-away:
cin SKIPS leading whitespace
cin STOPS on the first trailing whitespace

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
  - 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;

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

  double gpa;
  cout << "What is your gpa?" << endl;
  cin  >> gpa;
  cout << "GPA = " << gpa << endl;
  return 0;
}
```



Excercises

- cpp/cin/building_floor

ASSIGNMENT AND ORDERING

Ordering

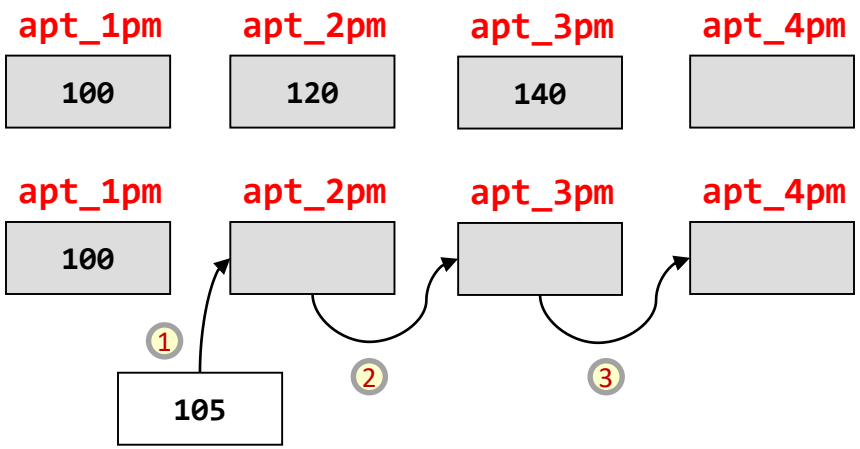
- Suppose a business assigns an integer ID to each client
  - Lower IDs are given to more important clients
  - Client's with lower ID's always get the appointment time they want
  - To reassign appointments we use the highlighted code below. Will it work?
- Recall that statements execute one at a time in sequential order
  - Earlier statement completes 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;
}
```

Ordering

- Perform each highlighted operation one at a time, marking up the diagram below

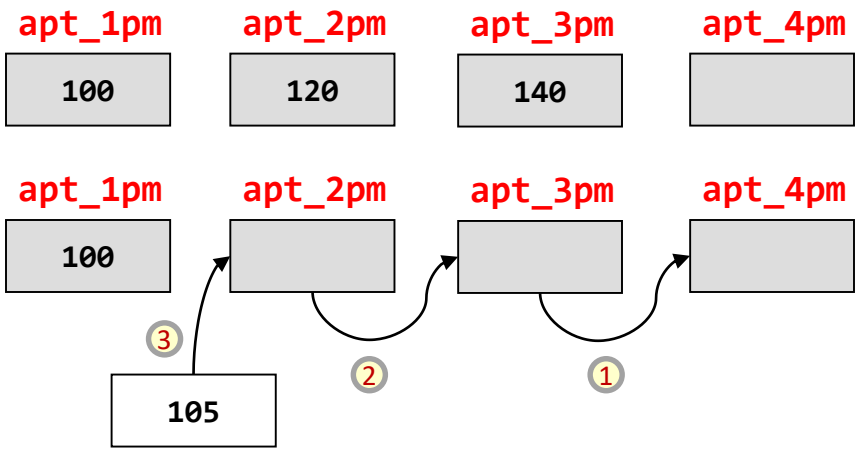
```
int main()
{
  // Original appointment
  //  schedule
  // Lower client ID gets
  //  earlier appointment
  int apt_1pm = 100;
  int apt_2pm = 120;
  int apt_3pm = 140;
  int apt_4pm = -1;

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

  return 0;
}
```



Main Point: Assignment (=) operations make copies

Ordering

- How can we fix the problem?
- Ensure we have fully used the "old" value of a variable before updating it with a new value
  - Move items in reverse order

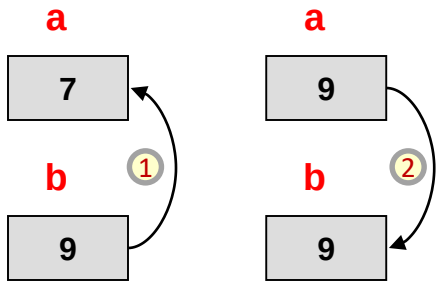



```
int main()
{
  // Original appointment
  //  schedule
  // Lower client ID gets
  //  earlier appointment
  int apt_1pm = 100;
  int apt_2pm = 120;
  int apt_3pm = 140;
  int apt_4pm = -1;

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

  return 0;
}
```

Another Example - Swap

- Given two variables, swap their contents
  - Before: a = 7, b = 9
  - Desired Result: a = 9, b = 7
  - Actual Result: a = ___, b = ___;

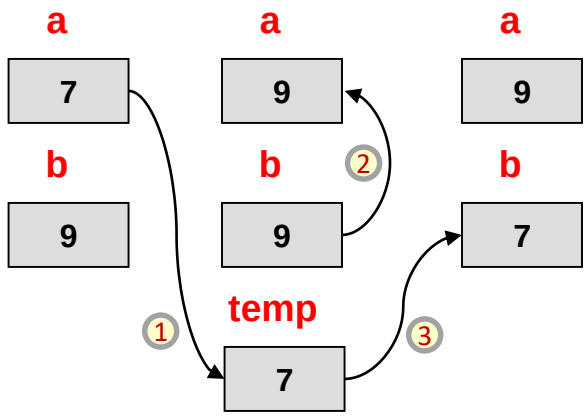
```
int main()
{
  int a = 7, b = 9;

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

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


  return 0;
}
```



Another Example - Swap

- We need a 3rd location to hold the old value of one of the variables while we update it to a 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;
}
```



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;
}
```

MORE OPERATORS AND USING MATH LIBRARY FUNCTIONS

Casting

- To achieve the correct answer for 5 + 3 / 2
- Could make everything a double
  - Write 5.0 + 3.0 / 2.0 [explicitly use doubles]
- Could 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)
- Could use **explicit** casting [Place desired type
  - 5 + (double) 3 / (double) 2 (C-Style cast)
  - 5 + (double)3 / 2 (can cast only one, rely on implicit cast of the other)

Casting

- Could use **explicit** casting
  - 5 + (double) 3 / 2 (can cast only one, rely on implicit cast of the other)
- This looks like a lot of typing compared to just writing
  5 + 3.0 / 2
- 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
- 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
- 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

Function call statements

- 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): 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/

Pre- and Post-Increment Operators

- ++ and -- operators can be used to "increment-by-1" or "decrement-by-1"
  - Performs (`+= 1`) or (`-= 1`)
  - If ++ comes before a variable it is call pre-increment; if after, it is called post-increment
  - `x++; // If x was 2 it will be updated to 3 (x = x + 1)`
  - `++x; // Same as above (no difference when not in a larger expression)`
  - `x--; // If x was 2 it will be updated to 1 (x = x – 1)`
  - `--x; // Same as above (no difference when not in a larger expression)`
- Difference between pre- and post- is only evident when used in a larger expression
- Meaning:
  - Pre: Update (inc./dec.) the variable before using it in the expression
  - Post: Use the old value of the variable in the expression then update (inc./dec.) it
- Examples [suppose we start each example with: `int y; int x = 3;`]
  - `y = x++ + 5; // Post-inc.; Use x=3 in expr. then inc. [y=8, x=4]`
  - `y = ++x + 5; // Pre-inc.; Inc. x=4 first, then use in expr. [y=9, x=4]`
  - `y = x-- + 5; // Post-dec.; Use x=3 in expr. then dec. [y=8, x=2]`

Statements

- C/C++ programs are composed of statements
- Most common kinds of statements end with a semicolon
- Assignment (use initial conditions of **int x=3; int y;**)
  - x = x * 5 / 9; // compute the expression & place result in x
    // x = (3*5)/9 = 15/9 = 1
- Function Call
  - sin(3.14); // Beware of just calling a function w/o assignment
  - x = cos(0.0);
- Mixture of assignments, expressions and/or function calls
  - x = x * y - 5 + max(5,9);
- Return statement (immediately ends a function)
  - return *value*;

I/O Manipulators

- Manipulators control HOW cout handles certain output options and how cin interprets the input data
  - 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
- Separated by `<<` or `>>` and used inline with actual data
- Other than `setw`, manipulators continue to apply to other output until changed
- See "iomanip" exercise to play with various options

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

int main()
{
  double pi = 3.14159;

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

  cout << setprecision(3) << pi << endl;
  // Prints: 3.14


  cout << fixed << pi << endl;
  // Prints: 3.142

  return 0;
}
```

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

Exercises

- Exercises:
  - cpp/cin/average
  - cpp/cin/rad2deg
- Write a program to convert temperature from Celsius to Fahrenheit [F = (9/5)*C + 32]
  - Use http://cpp.sh or http://onlinegdb.com
- Write a program to find the roots of a quadratic equation ($ax^2 + bx + c = 0$)
  - Use http://cpp.sh or http://onlinegdb.com

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$