Finally: operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minus</td>
<td>-</td>
<td>$-2, -x$</td>
</tr>
<tr>
<td>Addition</td>
<td>+</td>
<td>$2 + 4, 5 + 3$</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>$4 - 5, 6 - 3$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$*$</td>
<td>$4 \times 5, 6 \times 3$</td>
</tr>
<tr>
<td>Division</td>
<td>$/$</td>
<td>$4/5, 6/3$</td>
</tr>
<tr>
<td>Remainder</td>
<td>$%$</td>
<td>$4%3$</td>
</tr>
</tbody>
</table>

Some notes:

1. integer division → integer
2. result is upgraded to preserve precision
3. division by zero returns undetermined result
Precedence and Associativity

Interpreting multiple operator lines:

1. -8+7/5-2*2 = ?
2. Ambiguity needs to be resolved

Solution:

1. Precedence
2. Associativity

**Precedence Level:** execute first

1. Unary + and -
2. Multiplication, division and remainder
3. Addition and Subtraction

**Associativity:** where to look first

1. Unary + and -: **Right**
2. Multiplication, division and remainder: **Left**
3. Addition and Subtraction: **Left**
Examples of precedence and associativity

Think standard mathematical notation:

1. $6 \times 2 / 3 = ?$

2. $5 + 3 + 4 + 6 \times 2 = ?$

Note: use brackets to override precedence!
For example: $5 + 6 + 7 + 8 + 3 / 2 neq (5 + 6 + 7 + 8 + 3) / 2$
Assignment

Assignment operator: ’=’

1. Assigns value of constant or other variable to a variable
   (a) Constant or Must be of same type as variable
   (b) Removes existing value

2. Variable that has been declared but no value assigned
   (a) Whatever was stored in corresponding memory value
   (b) Source of many errors!
### Abbreviated Assignment operators

<table>
<thead>
<tr>
<th>operator</th>
<th>meaning</th>
<th>example</th>
<th>equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>add and assign to self</td>
<td>$a+ = 3$</td>
<td>$a = a + 3$</td>
</tr>
<tr>
<td>-=</td>
<td>subtract and assign to self</td>
<td>$a− = 4$</td>
<td>$a = a − 4$</td>
</tr>
<tr>
<td>/=</td>
<td>divide and assign to self</td>
<td>$a/ = 4$</td>
<td>$a = a/4$</td>
</tr>
<tr>
<td>*=</td>
<td>multiply and assign self</td>
<td>$a* = 4$</td>
<td>$a = a*4$</td>
</tr>
<tr>
<td>%=</td>
<td>calculate remainder and assign to self</td>
<td>$a%= 4$</td>
<td>$a = a%4$</td>
</tr>
</tbody>
</table>
1 // example of abbreviated assignments: abbrv_assign.cpp
2 #include <iostream>
3
4 int main (void){
5     double a;
6     a = 0;
7     cout << "a=" << a << endl;
8     a += 3;
9     cout << "a=" << a << endl;
10    a -= 2;
11    cout << "a=" << a << endl;
12    a *= 4;
13    cout << "a=" << a << endl;
14    a /= 2;
15    cout << "a=" << a << endl;
16    return 0;
17 }

More magic! Increment and Decrement Operators...

<table>
<thead>
<tr>
<th>operator</th>
<th>meaning</th>
<th>example</th>
<th>equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>add 1</td>
<td>$a++$, or $++a$</td>
<td>$a = a +1$</td>
</tr>
<tr>
<td>--</td>
<td>subtract 1</td>
<td>$a --$, or $--a$</td>
<td>$a = a -1$</td>
</tr>
</tbody>
</table>

Note: The difference between $a++$, i.e. postfix, and $++a$, i.e. prefix, is order of evaluation.

For example, $c = ++a - b \rightarrow a+ = 1; c = a - b$; i.e. $++a$ is evaluated first.

However, $c = a ++ - b \rightarrow c = a - b; a+ = 1$, i.e $a++$ is evaluated afterward.
`postprefix.cpp` difference post and pre-increment

```cpp
#include <iostream>

int main(void) {
    int pre, post, a, b;

    a = 1;
    b = 3;
    pre = ++a - b;
    cout << "after pre a=" << a << endl;

    a = 1;
    b = 3;
    post = a++ - b;
    cout << "after post a=" << a << endl;

    cout << "pre=\t" << pre << endl;
    cout << "post=\t" << post << endl;

    return 0;
}
```
A note on type casting

1. Sometimes variable of given type needs to be used in context requiring other type

2. Casting:
   (a) Convert or cast one type to another
   (b) Type of variable itself is not changed, merely interpretation
   (c) Compiler attempts to preserve precision in operations and casting

3. Example:

```cpp
// Johan Bollen, CS149
// cast_example.cpp
#include <iostream.h>

int main ( void ){
    int hits = 1, atbats = 3;
    float average;

    average = hits / atbats;
}
```
cout << "average: " << average << "\n";
average = (float) hits / (float) atbats;
cout << "average (casted): " << average << "\n";
return 0;
}
The concept of streams

- Streams:
  1. Connection between devices and program: screen, keyboard and files
  2. Work in specific direction

- Output: cout
  1. Stream connects program to screen
  2. drop things in, and they will “float” to screen
  3. drop: cout << value << “string constant”

- Input: cin
  1. Stream connects keyboard to program
  2. Anything the keyboard drops in (enter drops) flows to program into variable
  3. drop: cin >> value

- Note: watch direction of << and >>
Streams
// radius.cpp: Calculates radius from circumference
#include <iostream.h>
#define PI 3.1415

int main ( void ){
    // variable declaration
    double d, r;
    // input circumference
    cout << "Circumference? " ;
    cin >> d;
    // calculate radius
    r = d / (2 * PI);
    // output radius
    cout << "Radius = " << r << endl;
    return 0;
}
cin: Example

// cin.cpp: test cin
#include <iostream.h>

int main (void){
    double a, b, c;

    cout << "Enter a, b and c:
    cin >> a >> b >> c;

    cout << "a = " << a << "b = " << b << "c = " << c << endl;

    return 0;
}

Note: cin can read more than 1 value in 1 line. Values must be entered separated by white space until all values have been entered. Return inserts values.

Try it!
cout: Formatted output

1. Manipulators
   (a) Inserted in stream
   (b) Control format of following output

2. Control:
   (a) Width (number of digits)
   (b) Precision
   (c) Fill line with character

   (d) removes white spaces

<table>
<thead>
<tr>
<th>Manipulator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>setw(size)</td>
<td>Field width</td>
</tr>
<tr>
<td>setprecision(digit)</td>
<td>Set precision</td>
</tr>
<tr>
<td>setfill (char)</td>
<td>Sets character to fill field with</td>
</tr>
</tbody>
</table>

// cout_format.cpp: test format control
#include <iostream>
#include <iomanip>

int main ( void ){
    double a, b;

    cout << "Enter a and b" << endl;
    cin >> a >> b;

    cout << "a (setfill '0', setw 10) = " << setfill ('0') << setw(10) << a << endl;
    cout << "b (setfill '0', setw 8, setprecision 3) = " << setfill ('0') << setw(8) << setprecision(3) << b << setprecision(6) << endl;
    cout << "a (setprecision 4) = " << setprecision(4) << a << endl;
    cout << "b (setprecision 6) = " << setprecision(6) << b << endl;
    cout << setfill ('x') << setw(40) << " " << endl;

    return 0;
}
A note on `endl`

1. Inserts newline and forces stream to empty
2. Stream manipulator: flushes content
3. Prevent system from caching all output

```cpp
// demonstration of endl
#include <iostream.h>

int main ( void ){
    int i, j;

    cout << "Enter two numbers (separate by white space):"; 
    cin >> i >> j;

    cout << "You entered: " << i << " \t " << j << endl;

    return 0;
}
```
String class

1. Fundamental C++ types are limited to int, char, float, double, etc.

2. Use of streams, and other string manipulations, requires more sophisticated means of dealing with strings

3. Enter: string.h
   (a) Library defines additional C++ variable type
   (b) Declaration: string variable = “string constant”;
   (c) Cout and cin streams accept strings!

4. Don’t forget to include in preprocessor directive!

5. A string is not a character, always use string constants.

6. operators are “overloaded, e.g. + adds two strings
String example

```cpp
#include <iostream.h>
#include <string>

int main (void){
    string message = "Hello", name;

    cout << "What’s your name? ";
    cin >> name;

    cout << message << name << endl;
    message += name;
    cout << message << endl;

    return 0;
}
```
String functions

1. Strings allow all kinds of manipulations
   (a) String functions belong to string object itself
   (b) Called: variable.function(operator)

2. Return string size: int n = myString.size();

3. Substring extraction: myString.substr(start,num_chars);

4. Find part of string: int n = myString.find("findthis", startposition);

5. Read entire line (not just single string): getline(cin, myString, ’char’);
String functions: Example

// Johan Bollen, string_functions.cpp
#include <iostream.h>
#include <string>

int main (void){
    int n;
    string name, studentID, login, password;

    cout << "Enter your first name and last name: ";
    getline ( cin, name, '\n' );

    cout << "What’s your full ID? ";
    cin >> studentID;

    n = name.find (" ");
    login = name.substr (0, 1) + name.substr (n+1, name.size ());
    password = name.substr (n+1, 2) + studentID;

    cout << "login: " << login << " password: " << password << endl;
20
21    return 0;
22    }

Fibonacci’s rabbits

1. We’re breeding rabbits
2. Objective: how many rabbits starting from a given population after $x$ reproductive cycles?
3. Assumptions:
   (a) Rabbits do not die
   (b) Each adult pair produces a new pair
   (c) Adult: 2 months old
4. Problem rephrased: starting with a given number of rabbits, given the above mentioned assumptions, how many rabbits will we have after $x$ cycles?
A modern twist: Dr. Strangelove. After the Doomsday machine:
OK, now let’s write a program!

Problem Solving

Generate 5 first Fibonacci numbers starting from certain number of pairs.

Problem Statement

1. Think of problem in terms of input-output values
2. Input: start value (number of pairs)
3. Output: next 5 Fibonacci numbers

Hand Example - Problem solving

1. Fibonacci series:
   \[ f(n) = f(n-1) + f(n-2) \]
2. Start with a given number
3. Next number is sum of two previous numbers, etc.

Hand Example - Work problem

1. \( f(0) = 5 \)
2. \( f(1) = 5 \)
3. \( f(2) = f(1) + f(0) = 10 \)
4. \( f(3) = f(2) + f(1) = 15 \)
5. etc...

Translate to C++ program Think of variables you need and apply method to these variables...
How to write the program

1. Edit source code file using text editor: start editor
2. Preprocessor directives
   (a) First line: write comment
   (b) include statement: think about libraries program needs
3. Start main function: int main (void){
4. Declare the variables you will use
5. Start writing the instructions of your program (think about how you manually solved the problem)
6. End your program with a return 0;
7. Close main function: }
8. Save your source code file. Use a descriptive file name
9. Compile.