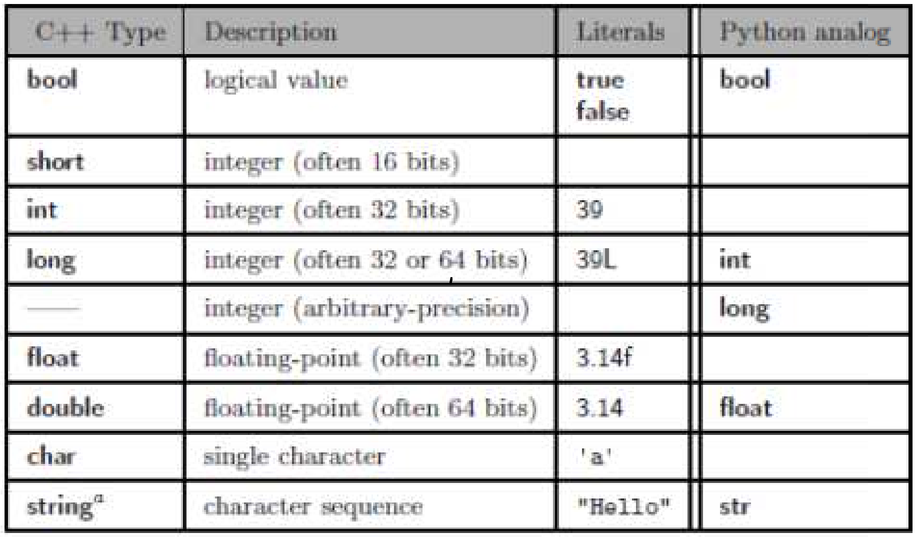
CS180 - Lecture 2

Note Title 8/25/2009

<u>Announcements</u>

- HW1 posted - due in class next Friday
- First lab tomorrow, due Monday
- Programming projects - checkpoints this year

Data types and operators



C++ Type	Description	Literals	Python analog
bool	logical value	true false	**bool**
short	integer (often 16 bits)		
int	integer (often 32 bits)	39	
long	integer (often 32 or 64 bits)	39L	**int**
——	integer (arbitrary-precision)		**long**
float	floating-point (often 32 bits)	3.14f	
double	floating-point (often 64 bits)	3.14	**float**
char	single character	'a'	
string[a]	character sequence	"Hello"	**str**

Figure 2: The most common primitive data types in C++.

[a]Not technically a built-in type; included from within standard libraries.

You'll probably use bool, int, long, float, or string the most.

Char versus String

```
char a;
a = 'a';
```

Be careful of single versus double quotes

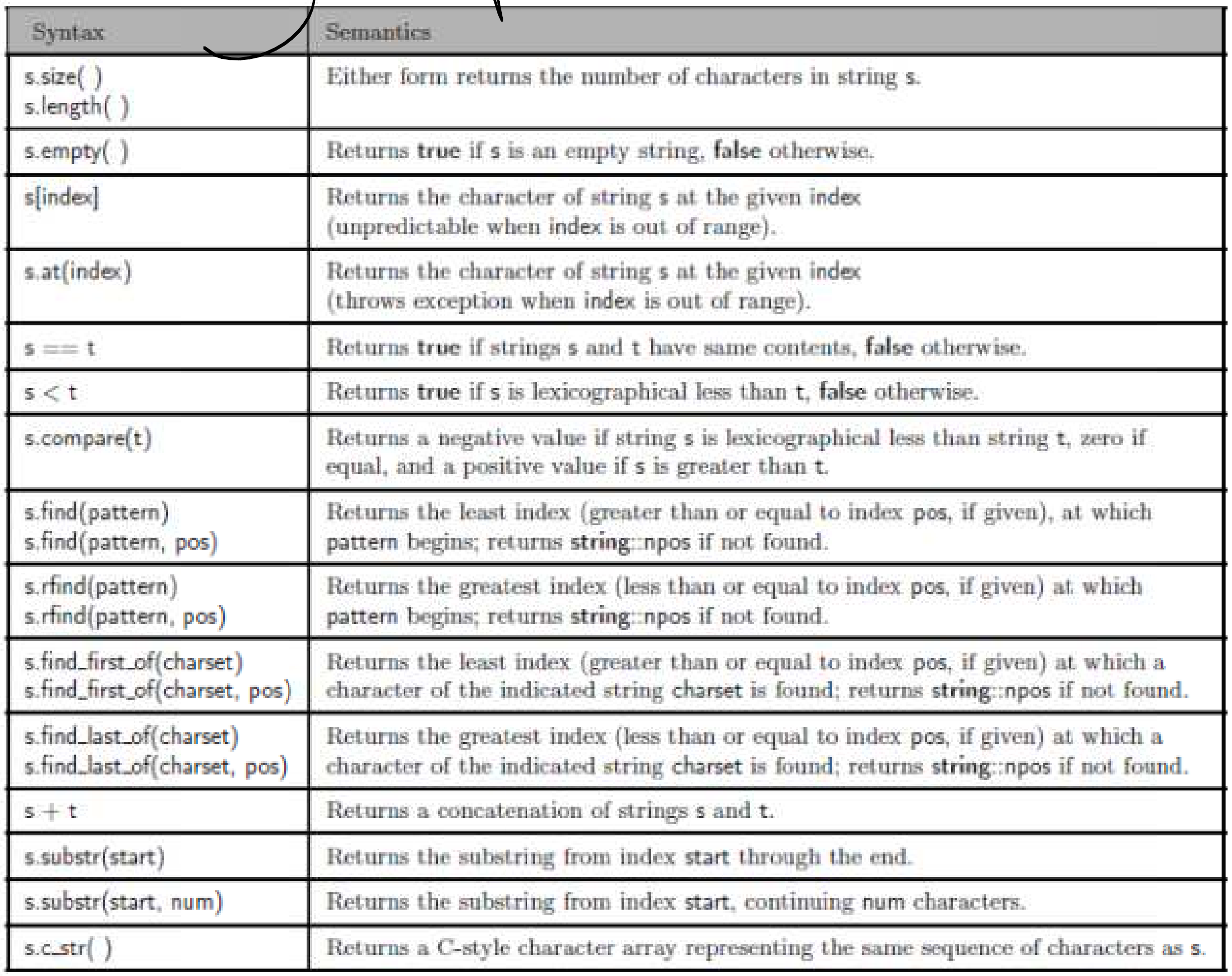
```
string word;
word = "CS 180";
```

(String is not predefined, but is standard in most C++ distributions.)

```
string myword;
myword = 'hello';
```
$\longleftarrow$ error

String operations

(see 4.3 in transition guide)



Syntax	Semantics

s.size( )
s.length( )
	Either form returns the number of characters in string s.

s.empty( )	Returns **true** if s is an empty string, **false** otherwise.
s[index]	Returns the character of string s at the given index (unpredictable when index is out of range).
s.at(index)	Returns the character of string s at the given index (throws exception when index is out of range).
s == t	Returns **true** if strings s and t have same contents, **false** otherwise.
s < t	Returns **true** if s is lexicographical less than t, **false** otherwise.
s.compare(t)	Returns a negative value if string s is lexicographical less than string t, zero if equal, and a positive value if s is greater than t.

s.find(pattern)
s.find(pattern, pos)
	Returns the least index (greater than or equal to index pos, if given), at which pattern begins; returns **string**::npos if not found.

s.rfind(pattern)
s.rfind(pattern, pos)
	Returns the greatest index (less than or equal to index pos, if given) at which pattern begins; returns **string**::npos if not found.

s.find_first_of(charset)
s.find_first_of(charset, pos)
	Returns the least index (greater than or equal to index pos, if given) at which a character of the indicated string charset is found; returns **string**::npos if not found.

s.find_last_of(charset)
s.find_last_of(charset, pos)
	Returns the greatest index (less than or equal to index pos, if given) at which a character of the indicated string charset is found; returns **string**::npos if not found.

s + t	Returns a concatenation of strings s and t.
s.substr(start)	Returns the substring from index start through the end.
s.substr(start, num)	Returns the substring from index start, continuing num characters.
s.c_str( )	Returns a C-style character array representing the same sequence of characters as s.

Mutable versus Immutable

Anyone remember these?

mutable - can be changed

immutable - can't

In C++, EVERYTHING is mutable.

```
string word;
word = "Hello";
word[0] = "J";
```

<u>Note:</u> Strings are also mutable!

See last example

Arrays

Python has lists, tuples, etc.

C++ only has arrays.

- size is fixed
- type is fixed (& homogenous)

Ex:
```
int numbers[10];
numbers[0] = 56;
numbers[9] = 11;

numbers[10] = 5; ← error
```

Creating variables (cont.)

```
int daysInMonth[] = {31, 28, 31, 30, 30, ...}
```

Create an array of appropriate size

```
int daysInMonth[]; ← ERROR
```

```
char greeting[] = "Hello";
```

Creating variables - a few examples

```
int number;
int a, b;          ← creates 2 integers.

int age(40);
int age( curYear - birthYear);

int age(40), zipcode(63116);

string greeting("Hello");
```

Forcing things to be immutable:

In some situations, there will be data that we want to be fixed.

To do this, use const:

```
const float gravity(9.8);
```

before declaration ⇒ variable is immutable

```
gravity=12; <- ERROR
```

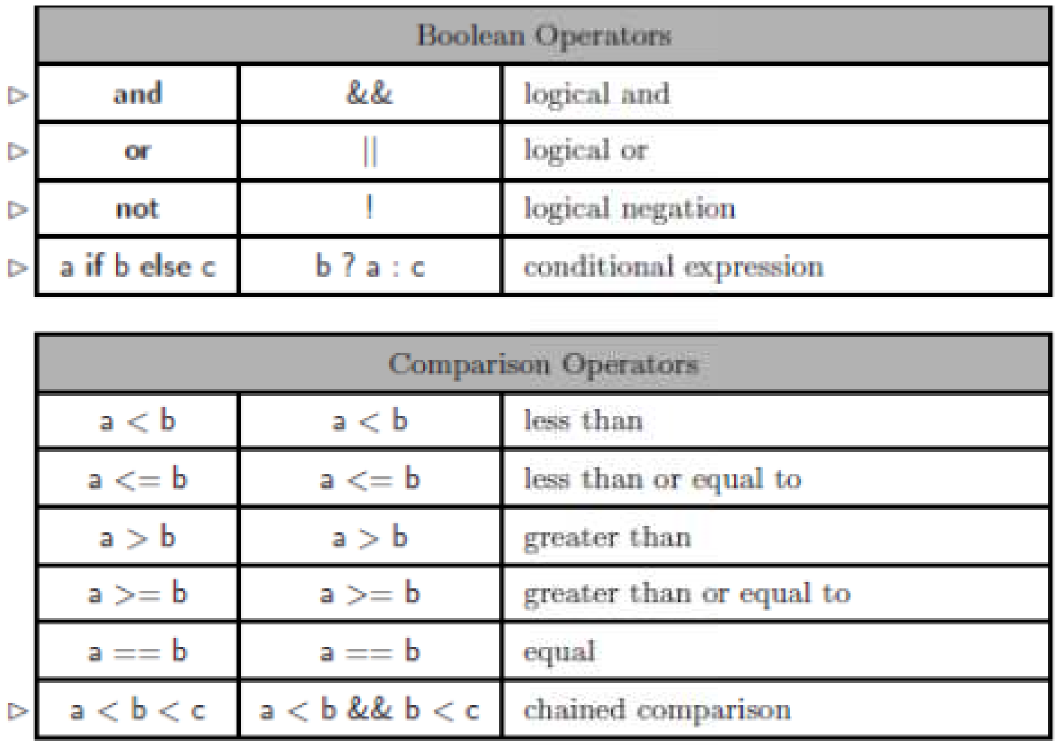
Operators

Basic numeric operators differ slightly:

	Arithmetic Operators		
	Python	C++	Description
	−a	−a	(unary) negation
	a + b	a + b	addition
	a − b	a − b	subtraction
	a ∗ b	a ∗ b	multiplication
▷	a ∗∗ b		exponentiation
	a / b	a / b	standard division (depends on type)
▷	a // b		integer division
	a % b	a % b	modulus (remainder)
▷		++a	pre-increment operator
▷		a++	post-increment operator
▷		−−a	pre-decrement operator
▷		a−−	post-decrement operator

Boolean operators & comparators -
VERY different

Python ↓ C++ ↓



	Boolean Operators		
▷	and	&&	logical and
▷	or	||	logical or
▷	not	!	logical negation
▷	a if b else c	b ? a : c	conditional expression

	Comparison Operators		
	a < b	a < b	less than
	a <= b	a <= b	less than or equal to
	a > b	a > b	greater than
	a >= b	a >= b	greater than or equal to
	a == b	a == b	equal
▷	a < b < c	a < b && b < c	chained comparison

Converting between types:

Be careful! C++ cares about type

```
int a(5);
double b;
b = a;
```

set $b = 5.0$

```
int a;
double b(2.67);
a = b;
```

set $a = 2$

(Can't go between strings & #s at all, although chars are given their ASCII values as ints.)

Control Structures

C++ has loops, conditionals, functions, + objects.

Syntax is similar — but usually just different enough to get you into trouble, also...

While loops

```
while (bool)
{
    body;
}
```

$\longleftrightarrow$ `while (bool) { body; }`

<u>Note</u>: - bool is any boolean exp: $a < b$

- don't need {} if only one command in body:

```
while (a<b)
    a++;
```

Also have do-while:

```
int number;
do {
  cout << "Enter a number from 1 to 10: ";
  cin >> number;
} while (number < 1 || number > 10);
```

This is a bit different:

body of loop is executed once <u>before</u> repeated condition is checked.

Conditionals

```
if (bool)
{
    body1;
}
else
{
    body2;
}
```

Ex:
```
if (x < 0)
    x = -x;
```

Note:
- don't need brackets if only one line in body
- don't need else
- no elif in C++ - write out else if

If statements (cont.)

If statements can also be written with numberic conditions instead of booleans:

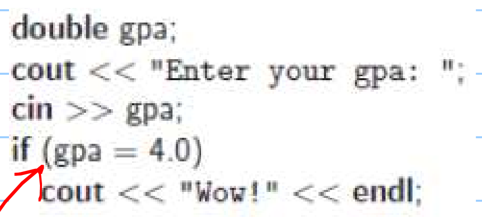
Ex

```
if (mistakeCount)
    cout << "There were " << mistakeCount
         << " problems" << endl;
```

if not = 0, true

0 always false

Common mistake - what is wrong?



```
double gpa;
cout << "Enter your gpa: ";
cin >> gpa;
if (gpa = 4.0)
 cout << "Wow!" << endl;
```

gpa == 4.0

C++ wouldn't give an error - it would reset gpa to 4.0 + then be true automatically

g = a = 0; (allowed to chain =)

For loops

Example:

run once, at beginning | checked at beginning of loop each time it runs | run at end of loop each time

```
for (int count = 10; count > 0; count --)
      cout << count << endl;
cout << "Blastoff!" << endl;
```

Note: int declaration isn't required.

```
int count;
for ( count = 10 ; count > 0; count --)
   :
```

Defining a function: example

Remember our countdown function from 150?

```
void countdown( ) {
  for (int count = 10; count > 0; count−−)
    cout << count << endl;
}
```

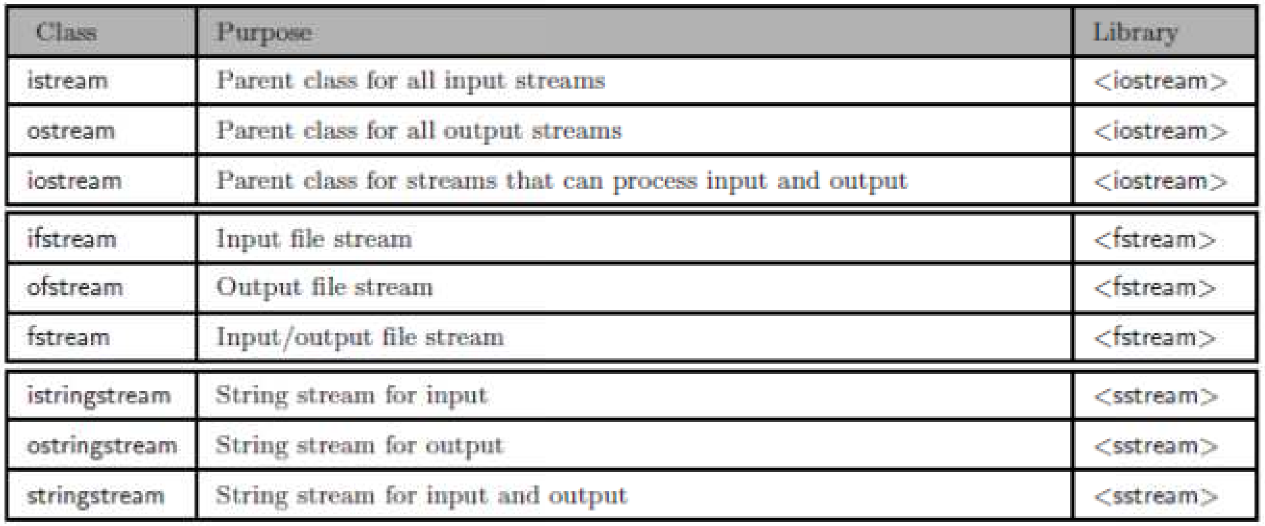
Or with optional parameters:

```
void countdown(int start=10, int end=1) {
  for (int count = start; count >= end; count−−)
    cout << count << endl;
}
```

More on functions in lab tomorrow...

Input + Output

C++ has several predefined, useful classes.



Class	Purpose	Library
istream	Parent class for all input streams	<iostream>
ostream	Parent class for all output streams	<iostream>
iostream	Parent class for streams that can process input and output	<iostream>
ifstream	Input file stream	<fstream>
ofstream	Output file stream	<fstream>
fstream	Input/output file stream	<fstream>
istringstream	String stream for input	<sstream>
ostringstream	String stream for output	<sstream>
stringstream	String stream for input and output	<sstream>

(We'll use iostream + fstream the most.)