CS180 — More C++

Note Title 8/31/2011

Announcements

- Extra credit for help session - today at 4pm
- Lab tomorrow (will be short)
- HW out tonight or tomorrow, due 1 week

Comparison

Python

```python
def gcd(u, v):
   # we will use Euclid's algorithm
   # for computing the GCD
   while v != 0:
      r = u % v     # compute remainder
      u = v
      v = r
   return u

if __name__ == '__main__':
   a = int(raw_input('First value: '))
   b = int(raw_input('Second value: '))
   print 'gcd:', gcd(a,b)
```

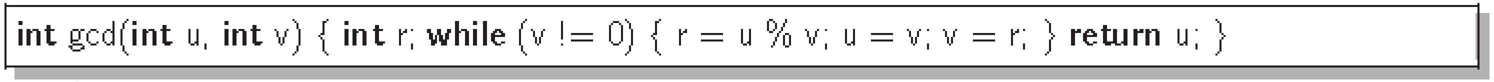
C++

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

int gcd(int u, int v) {
   /* We will use Euclid's algorithm
      for computing the GCD */
   int r;
   while (v != 0) {
      r = u % v;   // compute remainder
      u = v;
      v = r;
   }
   return u;
}

int main( ) {
   int a, b;
   cout << "First value: ";
   cin >> a;
   cout << "Second value: ";
   cin >> b;
   cout << "gcd: " << gcd(a,b) << endl;
   return 0;
}
```

Semi colon

White space

- returns, tabs, etc. are ignored in C++'



```
int gcd(int u, int v) { int r; while (v != 0) { r = u % v; u = v; v = r; } return u; }
```

↖ this is not acceptable to submit

(Recall that these were very important in python)

Here, we use () and {} to mark loops, booleans, etc.

Compiling

- In Python, you save code as gcd.py & then type "python gcd.py" to run it.

Later on - makefiles

- In C++:
  - Save as gcd.cpp

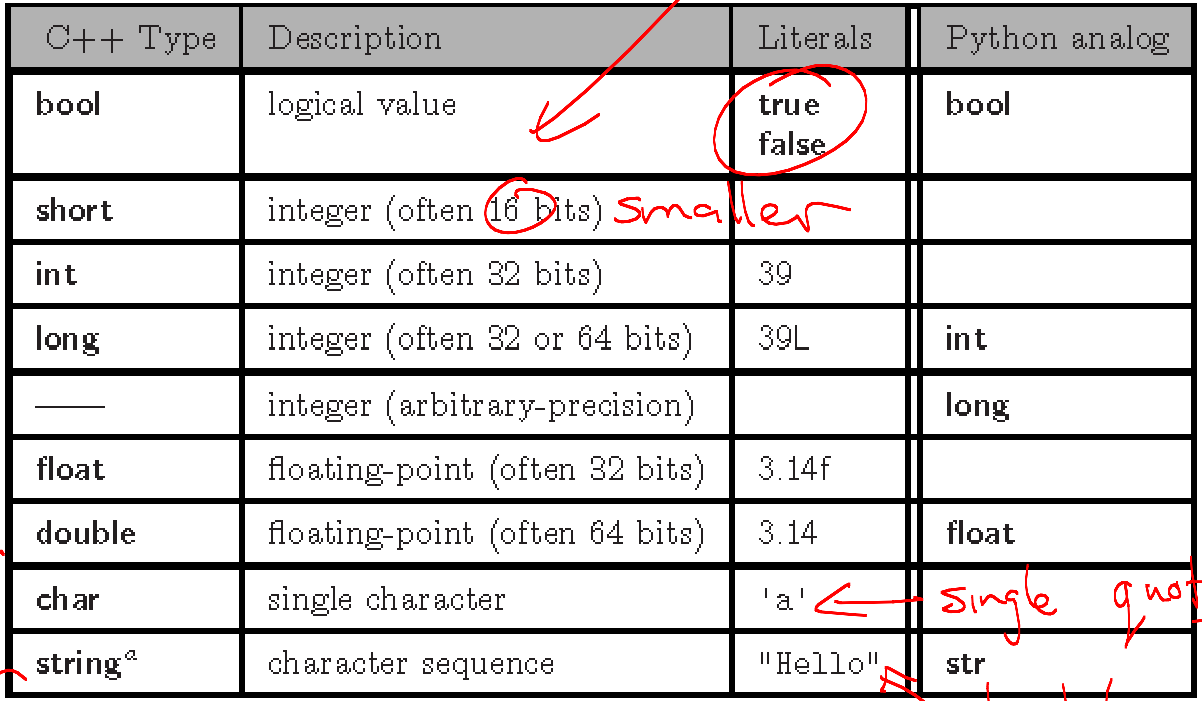
    name of a compiler program
  - type "g++ -o gcd gcd.cpp"

    optional. if omitted, defaults to "a.out"
  - type "./gcd"

    ./a.out

Data Types

a<b



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**

smaller

numerics

single quotes

double

are not a default data type

Data Types (cont)

- Ints can also be unsigned: instead of ranging from $-(2^{b-1})$ to $(2^{b-1}-1)$, go from 0 to $2^{(b-1)}$.

- Strings and chars are very different.

Char versus string

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

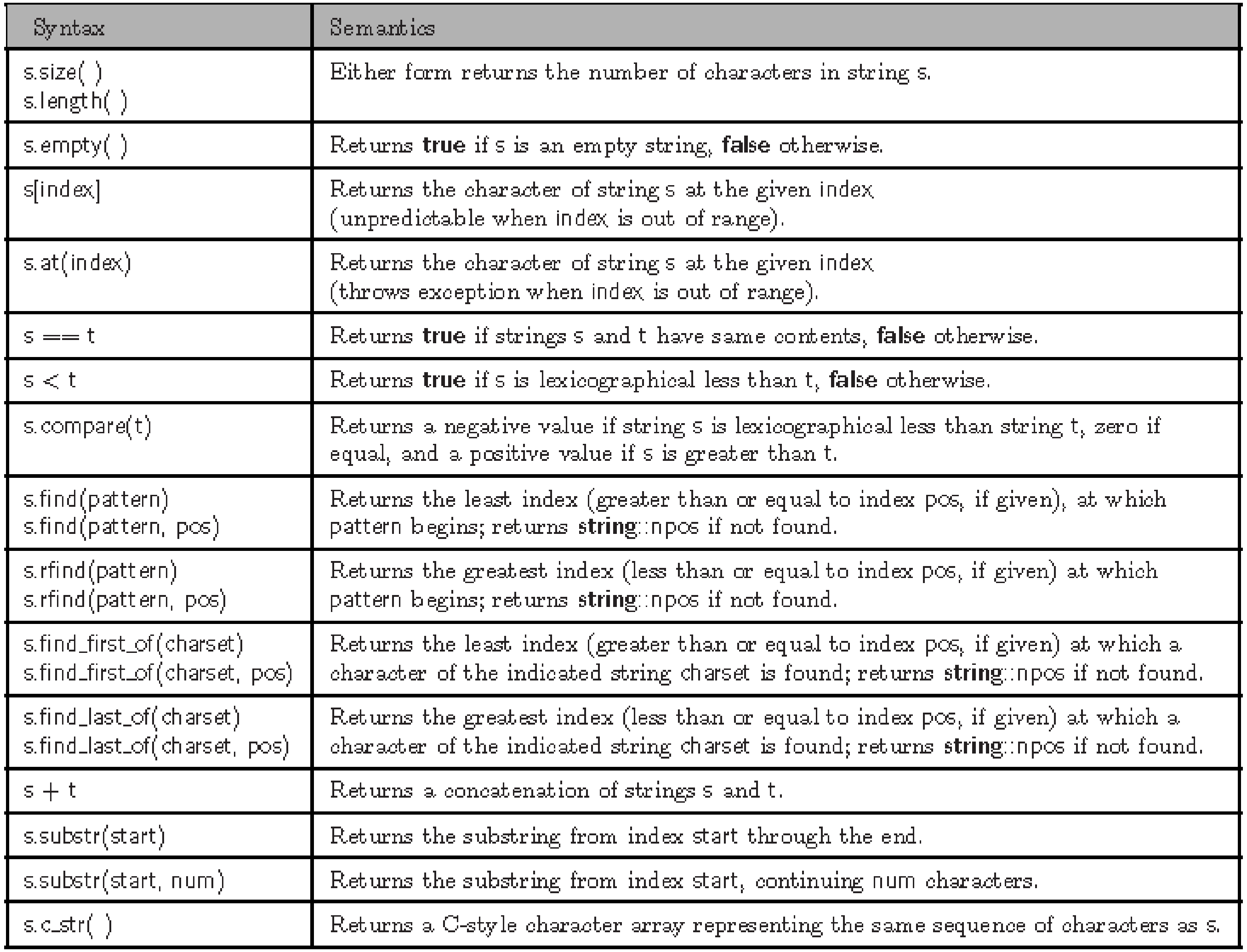
```
#include <string>
using namespace std;
```

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

Strings are not automatically included. Standard in most libraries, but need to import.

Strings

also check cplusplus.com



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

Dfn: mutable : can be changed

lists

Dfn: immutable : can't be changed

string

~~word[0] = 'k';~~

C++: Maximum flexibility

Everything is mutable by default!

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

"Jello"

Creating variables : create all at beginning of function

All variables must be explicitly created and given a type.

```
int number;
int a, b;
```

~~number = "Hello";~~ ← error

not int a, string b; ← error

```
int age(35);
int age2(currYear - birthYear);
int age3(21), zipcode(63116);
String greeting("Hello");
```

Immutable variables

We can force some variables to be immutable — use const:

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

Why?

- ease of testing
- forces the value to stay fixed

Converting between types

Be careful!

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

b = 5.0

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

a = 2 ← truncate

```
a=b+.5;
```

← round

Converting with strings

- Can't go between strings & numeric types at all.

"27" is not a number

- But chars will convert to numbers.
  How?

ASCII codes

```
int number = int(letter);
```

Control Structures

C++ has loops, conditionals, functions, & objects!

Syntax is similar, but just different enough to get into trouble.

(Remember to use your book's index in a pinch!)

While loops

```
int a= -5;
while (a<0) {
    a=a+1;
}
```

```
while (bool)
{
    body;
} // end of while (bool)
```

⟷

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

Notes:

- bool is any boolean expression
- don't need {} if only 1 command in the loop:

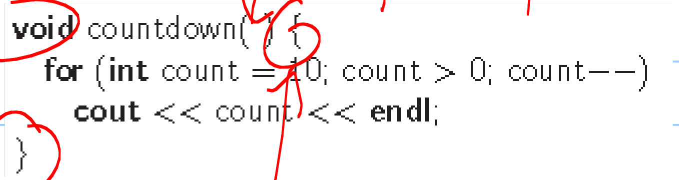
```
while (a<b)
    a++;
```

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

Defining a function : example

Remember countdown function from 150?

input parameters



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

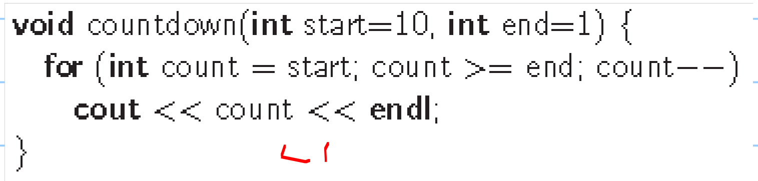
return type

use curly brackets

no def

```
void countdown (int start, int finish) {
    . . . .
}
```

Optional arguments



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

Tomorrow - lab

Friday - finish control structures