

CS 180 - Intro to C++

Note Title

8/26/2013

Announcements

- Lab Friday, normal lecture tomorrow
- Overview of lab + software
passwd

Comparison

Python

```
1 def gcd(u, v):
2     # we will use Euclid's algorithm
3     # for computing the GCD
4     while v != 0:
5         r = u % v # compute remainder
6         u = v
7         v = r
8     return u
9
10 if __name__ == '__main__':
11     a = int(raw_input('First value: '))
12     b = int(raw_input('Second value: '))
13     print 'gcd:', gcd(a,b)
```

C++

```
1 #include <iostream>
2 using namespace std;
3
4 int gcd(int u, int v) {
5     /* We will use Euclid's algorithm
6     for computing the GCD */
7     int r;
8     while (v != 0) {
9         r = u % v; // compute remainder
10        u = v;
11        v = r;
12    }
13    return u;
14 }
15
16 int main() {
17     int a, b;
18     cout << "First value: ";
19     cin >> a;
20     cout << "Second value: ";
21     cin >> b;
22     cout << "gcd: " << gcd(a,b) << endl;
23     return 0;
24 }
```

loading
libraries

output

input

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; }
```

↳ NEVER submit this

(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.

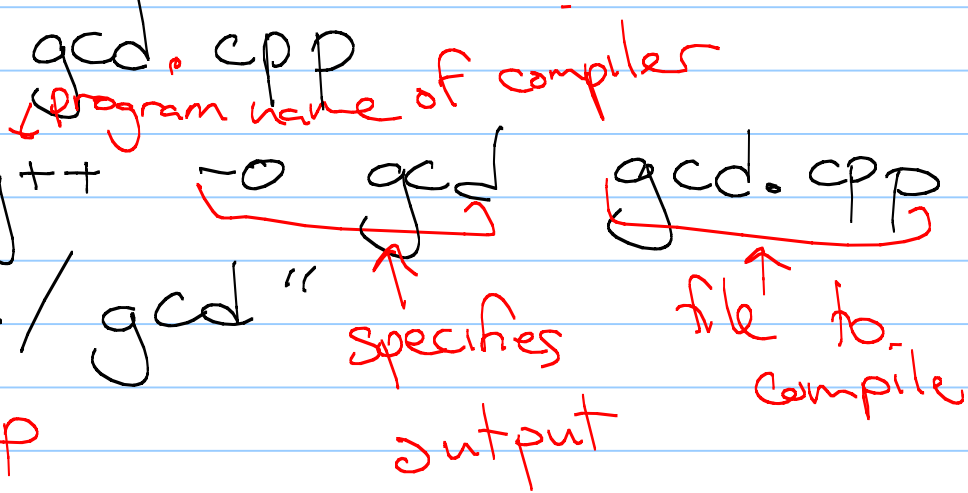
- In C++:

- Save as `gcd.cpp`

- type `g++ -o gcd gcd.cpp`

- type `./gcd`

or: `g++ file.cpp`
`> ./a.out`



Data Types

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

← not a default

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

```
#include <string>
```

```
char a;
```

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

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

← use single quote!

```
string word;
```

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

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

Strings

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

changeable : list

Dfn: immutable

not changeable : int, string, tuple

→ Default in C++!

C++: Maximum flexibility

Everything is mutable by default!

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

word is "Jello"

Creating variables

All variables must be explicitly created and given a type.

```
int number;
```

```
int a, b;
```

← not int a, char b;

```
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?

Compiler enforces no changes.

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

(8 bit ASCII)

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

↪ x = 100-something

Converting with strings

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

x = "67"

- But chars will convert to numbers.
how?

ASCII

Control Structures

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

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

(Remember to use ~~you~~ ~~book's~~ index
in a pinch!) ~~or~~ ~~transition guide~~
Cplusplus.com

While loops

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

↔ while (bool) {body;}

```
while (x < 5)
    x = x * 2;
    y = y - 1;
    ;
```

Notes:

- bool is any boolean expression

- don't need {} if only 1 command in the loop: while (a < b)

a++;
a = a + 1 or a += 1

Booleans

Python C++

Boolean Operators		
<code>and</code>	<code>&&</code>	logical and
<code>or</code>	<code> </code>	logical or
<code>not</code>	<code>!</code>	logical negation
<code>a if cond else b</code>	<code>cond ? a : b</code>	conditional expression

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

error in C++

For loops

in Python, iterator-based

Example:

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

initialization checks at each beginning time update rule

10
9
8
:
1
Blastoff!

Note: int declaration isn't required
(as long as variable already
was declared!)

```
for (int i = 0; i < bound; i++) {  
  
}
```

Defining a function : example

Remember countdown function from 150?

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

↙ type (no return type)
↙ inputs
↙ begin function
↙ end function

```
int main() {  
    countdown();  
}
```

Optional arguments

← inputs w/ default values

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

!
countdown();
countdown(20, 10);

If statements

```
if (bool) {  
    body 1;  
}  
else {  
    body 2;  
}
```

Ex:

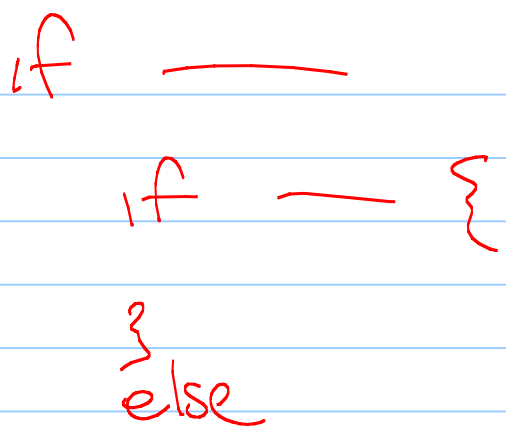
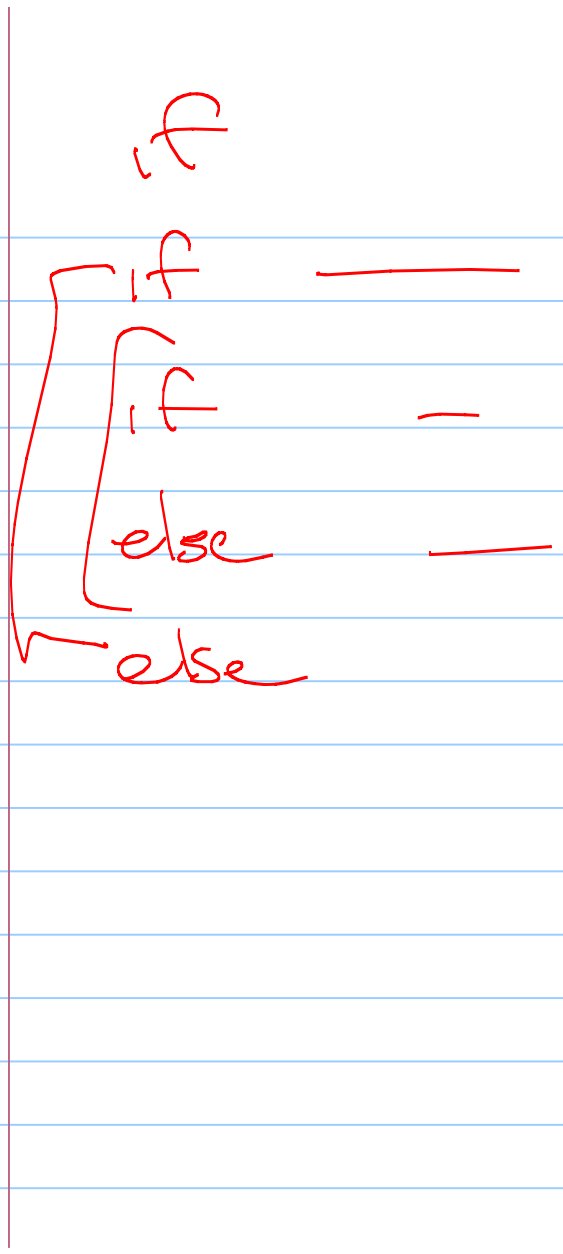
```
if (x < 0)  
    x = -x;
```

```
if (groceries.length() > 15)  
    cout << "Go to the grocery store" << endl;  
else if (groceries.contains("milk"))  
    cout << "Go to the convenience store" << endl;
```

Note:

- Don't need brackets if 1 line
- don't need else
- no elif

if —
else
if ()



Booleans & if/whiles

If & while statements can be written with numeric conditions (which are really booleans).

Ex: `if (mistakeCount)`
`cout << "Error!" << endl;`

0 \iff false

anything else is true

