CS2100 - Intro to C++

Note Title 8/26/2013

- HW1 is posted
- Lab is due today
- Email me to set up office hours if needed this week

Comparison

input
output

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++

imports

```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;
}
```

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

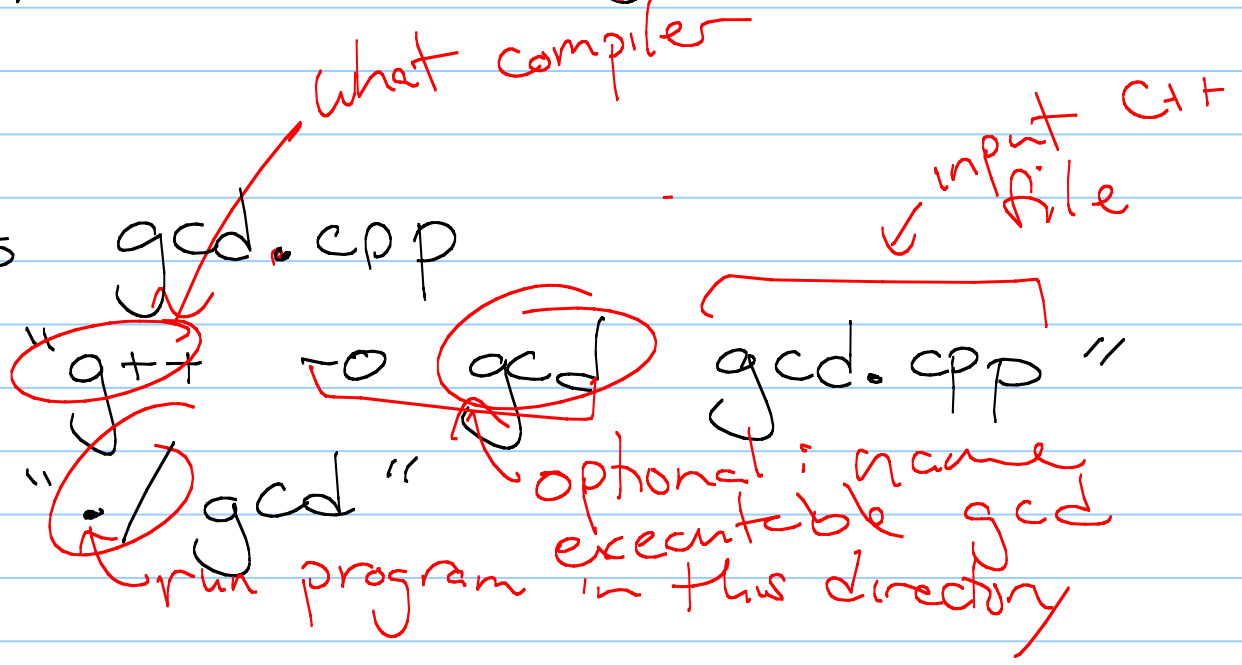
(Recall that these were very important in Python)

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"



Other ways to compile:

> g++ gcd.cpp
  ↳ save executable as a.out
> ./a.out

make gcd
  ↳ using a makefile

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**

Data Types (cont)

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

if you exceed length, it wraps around

- Strings and chars are very different.
  - strings must be imported
  - chars are ' ', strings " "

Char versus String

```
import <string>
using namespace std;
char a;
a = 'a';
a = 'h';

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

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

Strings

cplusplus.com search for string

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

change the value: lists

Dfn: immutable

opposite: value is fixed: tuples & strings

C++ : Maximum flexibility

Everything is mutable by default!

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

↳ word is now "Jello"

Even ints:
Can move bits of an int over
x<<2

Creating variables

All variables must be explicitly created and given a type.

generally at beginning of function, main, or program

```
int number;
int a, b;
int age(35);
int age2(currYear - birthYear);
int age3(21), zipcode(63116);
string greeting("Hello");
```

← NOT: int a, char b;

same as int age = 35;

Immutable variables

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

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

Why? Enforced by compiler!

```
gravity = 12;
```

Compile error

Converting between types

Be careful!

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

} b is 5.0

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

} a = 2

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

} a = 100 something

Converting with strings

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

```
word = 125;   <- NO
char x = 125;   <- ok
```

- But chars will convert to numbers. How?

ASCII

OK:

```
word[2]++;
word[0]=105;
```

Control Structures

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

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

(Remember to use cplusplus.com or transition guide in a pinch!)

Note: a++ is a = a+1

While loops

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



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

Careful:

```
while (x < 0)
    x = x+5;
cout << x;
```

Notes:

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

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

Booleans

Python C++

Boolean Operators		
and	&&	logical and
or	||	logical or
not	!	logical negation
a **if** cond **else** b	cond ? a : b	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

For loops

Python: iterator based

Example:

declaring control (loop) variable

repeat while bool exp is true

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

after every repeat of loop, do this to LCV

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

Defining a function : example

Remember countdown function from 1 to 50?

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

Optional arguments

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

If statements

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

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

So nesting can get ugly!

```
if  (     )
if  (     )
   {code}
else
    {  }
if  (   )
if  (    )
if  (   )
   {  }
else
    {  }
```

Booleans & if/whiles

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

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

0 $\Longleftrightarrow$ false

Strange quirk in C++:

Consider

```
int x;
x = 20;

if (x = 10)
   cout << "x is 10" << endl;
else
   cout << "x is not 10" << endl;
```