CS 180 - Intro to C++

Note Title 8/20/2010

Announcements

This is CS 180!

C++ versus Python

High level versus low level

Interpreter versus compiler

Dynamic versus static typing

Why learn C++?

- faster
- ubiquitous
- understand low level details
- control

A comparison:

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++:

```
#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;
    } //end while
    return u;
} //end gcd function

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 is irrelevant!

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

Python used returns & indentation to separate commands & loops.

parens & {} are important!

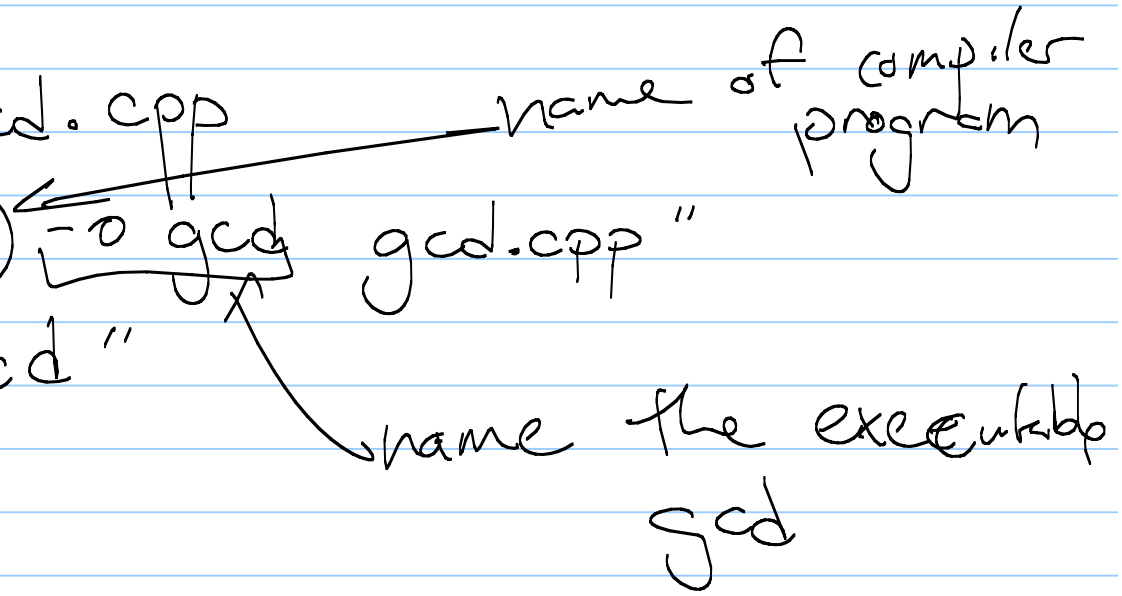
Please continue to indent.

Executing code

In Python, we could save the 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"



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.)

- Each integer type can also be unsigned.
  - Instead of ranging from $-(2^{b-1})$ to $(2^{b-1}-1)$ goes from 0 to 2^b-1.

Char versus String

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

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

Strings are not automatically included! They are standard in most libraries, but need to import that library.

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

Dfn: immutable

C++ - Maximum flexibility

In C++, everything is mutable!

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

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;
```

Creating variables (cont.)

Allowed:

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

Error:

```
int daysInMonth[];
```

Allowed:

```
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);
```