

CS 180 - Intro to C++ (part 2)

Note Title

8/23/2010

Announcements

- Lab 1 is tomorrow!
Don't forget to do your prelab...
- HW1 posted, due next Wednesday

C++ versus Python

High level versus low level

Interpreter versus compiler

Dynamic versus static typing

In Python: `a = 10`
`a = "hello"`

Gives an error in C++

```
int a;
```

```
a = 10;
```

```
a = "hello" ← syntax error
```

```
a = 'a'; (no error) give integer ASCII #
```

Why learn C++?

Efficiency

Ubiquitous

Low level

Complex

Useful

Variables

Numerical: short, int, long
float, double

bool - true, false

char 'a' (ASCII value)
string "word"

Mutable versus immutable

Dfn: mutable - allowed to change

Ex from python: list, dictionary

Dfn: immutable - can't change

Ex: tuples, strings

word[1] = 'a'

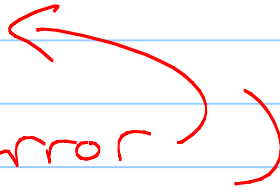
ERROR (in python)

C++ - Maximum flexibility

In C++, everything is mutable!

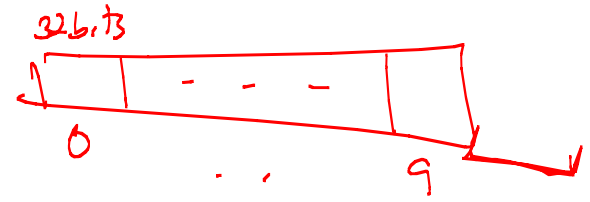
```
string word;  
word = "hello";  
word[0] = "J";
```

(in C++, no error)



So be careful!

Arrays



Python has lists, tuples, etc.

C++ only has arrays.

- size is fixed
- type is fixed (homogeneous)

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

numbers [10] = 5; Error!

Creating variables (cont.)

Allowed:
int

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

Error:
int

```
daysInMonth[];
```

↓ need a size

Allowed:
char

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

int size = 12;

```
daysInMonth[size];
```


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);
```

↑
forces value to be same

```
gravity = 10; ← Error
```

Operators

Basic numeric operators differ slightly:

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

```
int a, b;  
float c;
```

```
c = float(a) / float(b);
```

↑
OK

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; b; ← 5.0
```

```
int a;  
double b(2.67);  
a = b; ] a = 2;
```

(Can't go between strings & #s at all
although chars are given their ASCII value)

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

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

↔ while (bool) { body; }

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

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

```
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
before repeated condition is checked.

Conditionals

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

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`

Boolean conditionals in if & while statements

If statements can also be written with numeric 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;
```

in Python, get an error

in C++, sets gpa to 4.0

For loops

Example:

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

creates count & sets to 10

evaluated every time

executed at end of loop every time

Note: int declaration isn't required.

Alternate:

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

Defining a function: example

Remember our countdown function from ISO?

return
type

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

input parameters

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...