CS 180 - Intro to C++

Note Title 8/20/2010

Announcements

- HW1 is up - due next Wednesday

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

default in C++

use less space

import from a library

single quotes

double quotes

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";
```

at top of file:

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

(to import)

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

Strings

lots of functions (similar to Python)

refer to transition guide for details

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

list

mylist[2] = "word"

Dfn: immutable - fixed value

String

(can't change letter in a string)

C++ - Maximum flexibility

In C++, everything is mutable!

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

word is now "Jello"

Creating variables - a few examples

In C++, all variables must be declared and given a type!

```
int number;
int a, b;
int age(40);
int age(curYear - birthYear);
int age(40), zipcode(63116);
string greeting("Hello");
```

2 integers

NOT: int a, string b; ← error

Yes: int a; string b;

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);
```

later:

```
gravity = 12;
```

← compiler will give an error

Converting between types:

Be careful! C++ cares about type

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

a is 5

5.0

allowed

```
char a('w');
int b = a;
```

```
char letter = 'x';
b = a + x;
```

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

b is 2.67

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 (bool)
{
    body;
}
```

$\leftrightarrow$ `while (bool) { body; }`

```
x=-5;
while (x<0) {
    x=x+1;
    cout<< x <<endl;
}
```

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

`<` `>` `==` `!=`

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

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

Defining a function: example

Remember our countdown function from 150?

return type

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

must have {} at start + end

Or with optional parameters:

optional - will default to 10

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

In class exercise!

Go to webpage.

```
cp -R /Public/chambers/180/exercise1 .
cd exercise1
```

```
passwd
```

← to change your password