Programming Projects - Characters This Year

First Lab tomorrow due Monday

HW I posted - Due in class next week

Announcement

CS180 - Lecture 3
You'll probably use bool, int, long,

Data types and operators

<table>
<thead>
<tr>
<th>C++ Type</th>
<th>Primitive name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>single character</td>
<td>1 byte</td>
</tr>
<tr>
<td>short</td>
<td>integer (16bit)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>int</td>
<td>integer (32bit)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>long</td>
<td>integer (64bit)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>float</td>
<td>floating-point (4 bytes)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>double</td>
<td>floating-point (8 bytes)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>bool</td>
<td>boolean value</td>
<td></td>
</tr>
</tbody>
</table>

Not technically a built-in type: included from within standard libraries.
myword = 'hello'

String myword

String is not predefined, but is standard in most C++ dialects

word = "hello"

String word

word = "hello"

char a = 1.0;

dchar a;
<table>
<thead>
<tr>
<th>s</th>
<th>s \textit{subset}(\textit{start}, \textit{end})</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{indexOf}(n)</td>
<td>\text{indexOf}(\text{char}, \textit{pos})</td>
</tr>
<tr>
<td>\text{lastIndexOf}(n)</td>
<td>\text{lastIndexOf}(\text{char}, \textit{pos})</td>
</tr>
<tr>
<td>\text{split}()</td>
<td>\text{split}(\text{separator})</td>
</tr>
</tbody>
</table>

**String Operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{length}()</td>
<td>Returns the length of the string</td>
</tr>
<tr>
<td>\text{charAt}(n)</td>
<td>Returns the character at the specified index</td>
</tr>
<tr>
<td>\text{substring}()</td>
<td>Returns a substring of the string from the specified start index to the specified end index</td>
</tr>
</tbody>
</table>

**Example**

```java
String s = "Hello, world!";
int length = s.length();
String substring = s.substring(0, 5);
charcharAt = s.charAt(0);
```
In C++, everything is mutable.

In mutable - can't

Mutable - can be changed

Remember these?

Immutable versus Immutable
See last example.

Note: Strings are also mutable.
```python
# Example
numbers = [10, 5, 2, 9, 4]
numbers[0] = 11
numbers[1] = 55
numbers[2] = 55.2

# Error
print(numbers[0])
```

There is a mix of homogeneous types.

- Size is fixed
- Only has arrays

C++ has lists, tuples, etc.

Arrays
Creating variables (cont.)

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

Create an array of appropriate size

```c
int daysInMonth[J]; // ERROR
```

```c
char greeting[J] = "Hello"; 
```
String greathing ("Hello")

int age (40)
int age (50)
int age (70)
in t age (over - 70 years)

int age (60)

int number
int number
int number

Creating variables - a few examples
Gravity = 1.2 \rightarrow \text{ERR0R}

Before declaration \triangleright \text{variable is immediate}

\text{const float gravity (9.8)};

To do this, use const:

\text{In some situations these will be constant. Forcing things to be untenable;
<table>
<thead>
<tr>
<th>Basic Numeric Operators</th>
<th>Pre-increment Operator</th>
<th>Pre-decrement Operator</th>
<th>Post-increment Operator</th>
<th>Post-decrement Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>++x</td>
<td>++x</td>
<td>x++</td>
<td>x++</td>
</tr>
<tr>
<td>Subtraction</td>
<td>x -</td>
<td>x -</td>
<td>x -</td>
<td>x -</td>
</tr>
<tr>
<td>Multiplication</td>
<td>x *</td>
<td>x *</td>
<td>x *</td>
<td>x *</td>
</tr>
<tr>
<td>Division</td>
<td>x /</td>
<td>x /</td>
<td>q //</td>
<td>q //</td>
</tr>
<tr>
<td>Integer Division</td>
<td>q %</td>
<td>q %</td>
<td>q %</td>
<td>q %</td>
</tr>
<tr>
<td>Integer Division (depends on type)</td>
<td>q %</td>
<td>q %</td>
<td>q %</td>
<td>q %</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>x **</td>
<td>x **</td>
<td>x **</td>
<td>x **</td>
</tr>
<tr>
<td>Unary Increment</td>
<td>++x</td>
<td>++x</td>
<td>x++</td>
<td>x++</td>
</tr>
<tr>
<td>Unary Decrement</td>
<td>--x</td>
<td>--x</td>
<td>--x</td>
<td>--x</td>
</tr>
<tr>
<td>prython</td>
<td>prython</td>
<td>prython</td>
<td>prython</td>
<td>prython</td>
</tr>
</tbody>
</table>

Strikethrough: Basic numeric operators differ.

Operators
### Boolean Operations

| Condition | Expression | Logic
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>equal</td>
<td>$a = b$</td>
<td>not</td>
</tr>
<tr>
<td>less than</td>
<td>$a &lt; b$</td>
<td>or</td>
</tr>
<tr>
<td>greater than or equal to</td>
<td>$a \geq b$</td>
<td>and</td>
</tr>
</tbody>
</table>

### Comparison Operators

<table>
<thead>
<tr>
<th>$&gt;$</th>
<th>$&gt;$</th>
<th>$&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
</tbody>
</table>

- Python `==` corresponds to `equal`
- Python `!=` corresponds to `not equal`
- Python `>` corresponds to `greater than`
- Python `>=` corresponds to `greater than or equal to`
- Python `<` corresponds to `less than`
- Python `<=` corresponds to `less than or equal to`
ASCII values as int.

Although characters are given through text, they can't be between strings as #s at all.

Let's set $a = 2$.

```
int a;
double b (2.67);
double p;
int a (5);
```

Let's set $b = 5.0$.

Be careful! C++ casts a骨头 type conversion between two types.
Control Structures

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

Syntax is similar— but usually just different enough to get you into trouble, also...
a++;
while (a < b)

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

Note: bool is any boolean exp; a < b

if (bool)

while (bool)

if (bool)

while (bool)
Also have do-while:

```
int number;
doi
    cin >> number;
    while (number < 1 || number > 10);
```
Note: don’t need brackets if only one line in body!
0 always false

If not = 0, true

Ex if (mistake count) > count "there were" "mistake count"

If statement can also be written with

If statement (cont.)
\[ g = a = \theta \] (allowed to change)

Then be true mathematically:

It wouldn't give an error -

\[ \theta = 4.0 \]

```
cour >> "\"now\""
if (\theta = 4.0)
cour >> "\"\n\n\nCommon mistake - what is wrong?
```
Note: int declaration isn't required.

For (count = 10; count > 0; count --)

Count <= 8, last() <= End
Count <= 7, Count <= 8, Count <= 7

For (int count = 10; count > 0; count --)

Example

For loops
Defining a function: example

Remember our countdown function from 150?

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

Or with optional parameters:

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

More on functions in lab tomorrow...
(We'll use io::stream at first, then the most)

<table>
<thead>
<tr>
<th>io::stream</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringStream</td>
<td>Input/Output Stream for input and output</td>
</tr>
<tr>
<td>ofstream</td>
<td>ofstream for output</td>
</tr>
<tr>
<td>ifstream</td>
<td>ifstream for input</td>
</tr>
<tr>
<td>istream</td>
<td>istream for input</td>
</tr>
<tr>
<td>&lt;iostream&gt;</td>
<td>Input/output the stream</td>
</tr>
<tr>
<td>&lt;istream&gt;</td>
<td>istream, output the stream</td>
</tr>
<tr>
<td>&lt;ostream&gt;</td>
<td>ostream, input the stream</td>
</tr>
<tr>
<td>operator &lt;&lt;</td>
<td>istream, output the stream</td>
</tr>
<tr>
<td>operator &gt;&gt;</td>
<td>istream, input the stream</td>
</tr>
<tr>
<td>library</td>
<td>istream, input the stream</td>
</tr>
<tr>
<td>C++ has several predefined useful classes</td>
<td></td>
</tr>
</tbody>
</table>

Input + Output