CS 180 - Variables

Announcements

- Schedule page is up on website
- Next hw up today
Objects & Memory Management

In Python, variables were pointers to data.

\[
\begin{align*}
   & a \rightarrow \text{Point} \\
   & x = 3.2 \\
   & y = -5.8 \\
   & b = \text{Point(3, 4)} \\
   & c = a \cdot b \\
   & c = c + b \rightarrow \text{creates new point}
\end{align*}
\]
C++: A more versatile setup

C++ allows 3 different models for storing & passing information.

1. Value
2. Reference
3. Pointer

(remember that strange & a few slides ago?)
Value Variables (standard)

When a variable is created, a precise amount of memory is set aside:

Point \(a\), Point \(b\) \((5, 7)\).

\[
\begin{array}{|c|c|}
\hline
\text{a : Point} & \text{b : Point} \\
\hline
x = 0.0 & x = 5.0 \\
y = 0.0 & y = 7.0 \\
\hline
\end{array}
\]

This is more efficient, both for space and speed.
Now suppose we set \( a = b \):

<table>
<thead>
<tr>
<th>a : Point</th>
<th>b : Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = 5.0 )</td>
<td>( x = 5.0 )</td>
</tr>
<tr>
<td>( y = 7.0 )</td>
<td>( y = 7.0 )</td>
</tr>
</tbody>
</table>

They stay separate!
Different than Python:

(in Python):

```
Point
5.9
7.0
```
Functions: Passing by Value

```cpp
bool isOrigin(Point pt) {
    return pt.getX() == 0 && pt.getY() == 0;
}
```

When someone calls `isOrigin(myPoint)` later, the value of `pt` in the function is initialized as though a new variable was created:

```cpp
Point pt(myPoint);
```

So changes in function to `pt` don't affect `myPoint`!
Reference Variables

Syntax:

```
Point& c(a); // reference variable
```

- c is created as an alias for a
- More like Python model, but can't be changed later

Ex: `c=b;` will not rebind c to point to b, but will change the value of c (and a).

(Not in C++)
Ex:

```c
int a;
a = 35;
int &b(a);
b = 63;
a = 50;
```

bends to a

Memory

```
  131
  137
  138
  139
  140
  141
  142
  143
  144
```

Seems useless...
Passing by reference:

Reference variables aren’t usually needed in main program.

Instead, they are primarily used for passing to functions.

Ex:

```cpp
bool isOrigin(Point& pt) {
    return pt.getX() == 0 && pt.getY() == 0;
}
```

this runs `Point & pt(myPoint)`. 
Passing by reference (cont.)

- Sometimes we want changes to the variable to persist outside the function.

- Value variables copy all the data which uses both time & space.
If we want the speed of passing by reference but don’t want our object mutated, use `const`.

```cpp
bool isOrigin(const Point& pt) {
    return pt.getX() == 0 && pt.getY() == 0;
}
```

This makes it a constant reference, so the function cannot change that variable. Compiler will ensure that `pt` isn’t modified.

```cpp
pt.x = 5; // compiler error
```
Recall: `Point output`

```cpp
ostream& operator<<(ostream& out, Point p) {
    out << "<" << p.getX() << ", " << p.getY() << ">";  // display using form <x,y>
    return out;
}
```

Here, `&` is required because streams cannot be copied.

Note that we **don't** use `const` since we are changing the stream by adding data.
Pointer variables

Syntax: `int *d;`  // d is a pointer variable

d is created as a variable that stores a memory address.

So: `int b[6];`

```
  d = &b;
```

gives

```
  memory address
  of b
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>6</td>
<td>263</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>263</td>
<td>266</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>264</td>
<td>265</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

But d is not an int! Can't say d = b.
Using pointers in a class:

Suppose we need an array as private data, but don't know size.

class MyObject {
private:
    int size;  // array?
    int* myArray;

public:
    MyObject (int s = 10) : size(s) {
        myArray = new int[size];

        // initializes some other spot
    }
}
This lets you create arrays later!

Then, if need to resize:

```c
void resize_array(int newsize) {
    int * newarray = new int[newsize];
    for (int i = 0; i < size; i++)
        newarray[i] = myArray[i];
    delete[] myArray;
    myArray = newarray;
}
```
Using pointer variables to get to data

2 options: called dereferencing

(*d).get Y();

d -> get Y();
Using pointer variables to get to data

With an array, even better:

d[i] works;

\(\text{when } d \text{ is a pointer to an array} \quad i \text{ is an array}\)

(no need to dereference)
Passing by Pointer

```c
Point *pt = NULL

bool isOrigin(Point *pt) {
    return pt->getX() == 0 && pt->getY() == 0;
}
```

This is similar to passing by reference but allows you to also pass a null pointer.

\[\text{Diagram}\]
Garbage Collection
- In Python, variables that are no longer used will be automatically destroyed

Pros: lets programmer ignore deletes
Cons: time - writing it yourself is much faster
C++:

In C++, things are sometimes handled for you:

```cpp
int n = 5;
for (int i = 1; i <= n; i++) {
    int value; // value is deleted at end of loop!
    value = i;
    cout << value << endl;
}
```

Output: Error "value is not declared in this scope"
In a C++ program, all value variables are destroyed for you. The problem is pointers.

Rule: If you use `new`, you need to use `delete`!
More on Classes:  (next time)

Destructors:

If your class opens files or allocates memory, then can't just use delete.
Must create a destructor:
~ClassName() - no arguments, no return type
~Point()