CS180 - Variable Types in C++

Announcements

- HW due tomorrow by midnight
- Lab 2 by Sunday (don’t forget to submit even if not perfect!)
- HW2 will be up today due next Friday (?)
- Need to reschedule next Thursday’s office hours - at 9-10am.
class Point {
private:
    double _x;
    double _y; // explicit declaration of data members

public:
    Point( ) : _x(0), _y(0) { } // constructor

    double getX( ) const { // accessor
        return _x;
    }

    void setX(double val) { // mutator
        _x = val;
    }

    double getY( ) const { // accessor
        return _y;
    }

    void setY(double val) { // mutator
        _y = val;
    }

};
C++: More versatile

C++ allows for 3 different types of variables:

1. Value
2. Reference
3. Pointer
2. Reference Variables

Syntax: \( \text{Point} \ & \ c(a) \);

- \( c \) is created as an alias for \( a \);
- More like Python, but \( c \) is always the same as \( a \).

Ex: \( c = b ; \)

Will not make \( c \) point to \( b \), but will actually change value of \( a \).
Ex:

```c
int a;
a = 35;
int & b(a);
int c(7);
b = 63;
c = 11;
a = 50;
b = c;
```

<table>
<thead>
<tr>
<th>name</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>b, a</td>
<td>583588</td>
</tr>
<tr>
<td>c</td>
<td>11</td>
</tr>
</tbody>
</table>
Passing by reference

Reference variables aren't generally used in main.

Instead, primary purpose is in functions:

Ex:
```cpp
bool isOrigin(Point& pt) {
    return pt.getX() == 0 && pt.getY() == 0;
}
```
Why pass by reference?

3 main reasons

1. saves space
2. saves time
3. allows changes to persist
If we want the speed of passing by reference, but we don't want changes to variable, use const:

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

Compiler will enforce that `pt` isn't changed inside the function.
Recall: Point output

```cpp
ostream& operator<<(ostream& out, Point p) {
    out << "(" << p.getX() << ", " << p.getY() << ");
    return out;
}
```

Here, & is required since streams cannot be copied.

Note: don't use const. Why?

Goal is to change output stream!
"3 Pointer variables"

Syntax: int * d;

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

Ex: Pointer x;
    int b[8];
    int* d;

    d = &b[2]; // memory address of b
    d = &x; // error

    But d is not an int.
    Can't write d = b!
    cout << *d << endl;

    |
    | variable | contents | address |
    |---------|---------|---------|
    | b       | 8       | 281     |
    |         |         | 282     |
    | d       | 282     | 283     |
    |         |         | 284     |
    |         |         | 285     |
    |         |         | 286     |
    |         |         | 287     |
Pointers: getting to the data

Called dereferencing.

Ex: Point * d;
    Point b(3, 5);
    d = &b;

2 options:

2 = (**d).getX();

or

d -> getY();
The new command:

```c
int* c;
c = new int(12);
```

Allocates a block of memory.

Main use: the data persists even after the pointer is gone!

So can create or modify inside multiple functions.

```c
int ** x;  // Pointer to a pointer
```
Passing pointers

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

Similar to passing by reference, but allows passing a NULL pointer also.

NULL = 0
Pointers in a class

Pointers are especially useful in classes.

Often, we don't know all the details of private variables to put in the private declaration.

Example: arrays!

What do we need when creating an array?
Example class: vector of floats

A vector in $\mathbb{R}^2$: $<2, 5>$

A vector in $\mathbb{R}^4$: $<90, 0, 0, 1>$

Dynamic size!

So how to make a class?

private:

```cpp
int _size;
float* _V;
```
class MyIntVec {

private:
    int _size; // size of this array
    float* _V; // pointer to my array

public:
    MyIntVec (int s = 10) : _size(s) {
        _V = new float[_size];
    }
}
Accessing the array:

With an array, can just pretend the variable isn't a pointer.
(so no * or ->)

inside constructor to 0-out the vector:

\[
\text{for (int } i=0; i < \text{size}; i++) \\n\text{ V[i] = 0; } \]

Function to scale by int (in class):

```cpp
void operator *(int x) {
    for (int i = 0; i < _size; i++)
        _V[i] *= x;
    _i = _i + 1;
}
```

```
    \[ V[i] \] = V[i] \times x
```
Garbage Collection

In Python, variables that are no longer in use are automatically destroyed.

Pros: easy!

Cons: Slow
In 

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

Basically, any standard variable is automatically destroyed at the end of its scope.

This holds for any type of variable!
Problem: Pointers

While the pointer variable is deleted, the spot you created with a "new" is not.

```c
int main() {
    int *a = new int(5);  // a 197
    delete a;            // not destroyed
    // a is destroyed
}
```

Rule: If you have a `delete`, you must have