Administrivia
PA3
Forward declaration
#ifndef _parser Included_
#define _parser Included_

#include <vector>
#include <map>
#include <functional>

#include "XMLSerializable.h"

void parseXML(std::map<std::string, std::function<XMLSerializable * ()>> & mConstructors, 
              std::string sFilename, std::vector<XMLSerializable*> & vWorld);

#endif
```cpp
#ifndef _Tile_included_
#define _Tile_included_

#include <vector>
#include "Character.h"
#include "Item.h"

class Tile
{
public:
    // Class definition, etc

private:
    Character * m_pCharacter;
    std::vector<Item*> m_vItems;
};
#endif
```

```cpp
#ifndef _Tile_included_
#define _Tile_included_

#include <vector>

class Character;
class Item;

class Tile
{
public:
    // Class definition, etc

private:
    Character * m_pCharacter;
    std::vector<Item*> m_vItems;
};
```

Limitations
Only works with pointers or references
You can’t use a class that’s been forward declared until it’s been defined.
Pure virtual
class A
{
 publi:
       virtual void doSomething() = 0;
};
class B : public A
{
public:
    virtual void doSomething()
    {
        cout << "Something was done" << endl;
    }
};
A class with *any* pure virtual methods is called an abstract class.
A class with *only* pure virtual methods is called an interface.
You cannot instantiate an object of an abstract class!
Type
Parameter and return type passing semantics
By value
By reference
(pointers are kinda by reference)
But: You can pass the pointer \textit{itself} by value or by reference.
Tile *&

Tile **
Scalars are easy enough.
You should know that when you pass something by value, you get a new copy on the stack.
(via the copy constructor if an object)
This applies to return values as well.