Problem Set 0 Solutions

This handout contains sample solutions to the problems on Problem Set 0. There are many solutions to these problems, so don't worry if your answers don't precisely match those in this handout.

Problem 0

There are many valid ways to solve this problem, perhaps the easiest of which is as follows:

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

int main()
{
    ifstream input("datafile.dat"); // Or whatever file we want.
    /* ... validate user input ... */

    while(true)
    {
        int value;
        input >> value;

        /* See note below about why eof() and fail() are both checked here. */
        if(input.eof() && input.fail()) break;

        /* If we failed without eof, we must have hit a string.
         * Read it and ignore it.
         */
        if(input.fail())
        {
            input.clear();
            string dummy;
            input >> dummy;
        }
        else cout << value << ' ';
    }
}
```

Note that to break out of the loop, we need both `eof` and `fail` to return true. In class I incorrectly stated that `eof` indicates whether a stream operation failed due to no more data. This is incorrect – `eof` returns true if there is no more data left irrespective of whether the last operation succeeded. Therefore it is actually possible for `eof` to be true after a read operation even if data was read correctly. To check if an operation failed due to no more data being available, we need to check for both `eof` and `fail`.

Problem 1

Here's a working version of `GetReal`:

```
double GetReal()
{
    while(true)
    {
        stringstream converter;
        converter << GetLine(); // Assume this is defined.

        double result;
        converter >> result;

        if(converter.fail())
            cout << "Please enter a real number." << endl;
        else
        {
            char leftover;
            converter >> leftover;

            if(converter.fail()) return result;
            else cout << "Unexpected character: " << leftover << endl;
        }
        cout << "Retry: ";
    }
}
```

There is very little code that needs to be changed from `GetInteger` to convert it into `GetReal`. We only need to change the return type, the definition of the `result` variable, and text "Please enter an integer" and we have a completely working input function. Isn't the IOStream library great?

Problem 2

Since some of the pointers alias one another, there are many solutions to this problem. Here's one possibility:

```
int climber, floater;
int* blocker = &climber;
int* bomber = new int;
int* builder = new int;
int* basher = builder;
int* miner = blocker;
int* digger = new int;
*digger = *blocker;
delete builder;
builder = miner;
*bomber = *blocker;
basher = &floater;
climber = *blocker;
delete bomber;
bomber = blocker;
delete digger;
```

Problem 3

```
int murphy, snikSnak;
int* infotron = &murphy;
int* electron;
int zonk = snikSnak + *infotron;

if(electron != NULL)
    *electron = *infotron; // <-- Crash here
else
    electron = infotron;
*electron = snikSnak;
```

The crash occurs at the indicated line because `electron` is uninitialized. Consequently, the check `if(electron != NULL)` will likely succeed, since `electron` holds a garbage memory address. The indicated line then executes, causing a runtime crash.

Problem 4

```
int rogerWilco = 4;
int* sequelPolice = new int;
int* sludgeVohaul = sequelPolice;
*sequelPolice = rogerWilco;
*sludgeVohaul = *sequelPolice + rogerWilco;
delete sequelPolice;
*sludgeVohaul = 0; // <-- Crash here
```

`sludgeVohaul` points to the same location as `sequelPolice`, so after executing the line `delete sequelPolice` the memory referenced by `sludgeVohaul` has been deallocated. Writing to `*sludgeVohaul` is thus a bad idea.

Problem 5

```
int* commanderKeen = new int[137];
int* vorticons = &commanderKeen[42];
int shadowlands = *vorticons + commanderKeen[12];
vorticons = &shadowlands;
delete commanderKeen; // <-- Crash here
*vorticons = 2718;
```

Notice that the memory referenced by `commanderKeen` was allocated using `new[]` instead of `new`. We thus need to deallocate it using `delete[]` instead of `delete`.

Problem 6

```
/* Replaces all instances of a given character in a string with another
 * character.  This code uses the at() member function of the string, which
 * returns a reference to the character stored at that position.
 */
void ReplaceAll(string* input, char what, char with)
{
    for(int index = 0; index < input->length(); ++index)
        if(input->at(index) == what)
            input->at(index) = with;
}
```

There are many advantages of pass-by-reference over pass-by-pointer and vice-versa. One possible solution is to remark that pass-by-reference makes it easier for the function implementer (since there is no need for an explicit pointer dereference), while pass-by-pointer makes it easier for the function client to determine whether or not the argument can be modified (since there must be an explicit pointer to the argument for any changes to occur).