Problem Set 0

Due April 16, 11:59PM

This first problem should be a great way for you to play around with the C++ IOStream library and to get a feel for pointers and references. Feel free to use any resources you'd like to answer these questions, but do be sure to cite any outside aid you receive. As always, don't hesitate to email me if you have any questions.

Problem 0

In Tuesday's lecture we wrote a program which read in a file and printed out all of the contents of the file that were *not* integers. That is, if the file contained

```
137 Hello! 42 Strings are fun! 27182
```

The program would print out `Hello! Strings are fun!`

However, perhaps we're interested in solving the reverse problem – what if we *only* want to print out the integer values? For example, given the above file, we would print out `137 27182`. Write a C++ program that opens a file and prints out all of the integers it contains, skipping over all other data. For simplicity's sake, you can assume that each token (whitespace-separated sequence of characters) consists solely of digits or letters, so strings like `137Hello` and `Hello137` are not possible.

Problem 1

Modify the code for `GetInteger` covered in class to create a `GetReal` function that reads in a floating-point number from the user, reprompting as necessary. How much code did you have to modify to change the functionality?

Problem 2

The following program contains several *memory leaks*, memory allocated by `new` that is not matched by a call to `delete`. Modify the code so that it correctly deletes all allocated memory. You may want to draw a picture to help visualize what's going on.

```
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;
builder = miner;
*bomber = *blocker;
basher = &floater;
climber = *blocker;
bomber = blocker;
```

Problem 3

The following code compiles, but will almost certainly crash at runtime. At which line will it crash? Why is this?

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

if(electron != NULL)
    *electron = *infotron;
else
    electron = infotron;
*electron = snikSnak;
```

Problem 4

The following code compiles, but will almost certainly crash at runtime. At which line will it crash? Why is this?

```
int rogerWilco = 4;
int* sequelPolice = new int;
int* sludgeVohaul = sequelPolice;
*sequelPolice = rogerWilco;
*sludgeVohaul = *sequelPolice + rogerWilco;
delete sequelPolice;
*sludgeVohaul = 0;
```

Problem 5

The following code compiles but contains a serious error. Where is the error? How can you fix it? Note that the code snippet, as written, will not necessarily crash even though the error is present.

```
int* commanderKeen = new int[137];
int* vorticons = &commanderKeen[42];
int shadowlands = *vorticons + commanderKeen[12];
vorticons = &shadowlands;
delete commanderKeen;
*vorticons = 2718;
```

Problem 6

The C programming language has pointers but no references. Therefore, pure C code has no concept of "pass-by-reference." In order for a function to update the values of its parameters outside of the function, the function would have to accept pointers as arguments that point to the variables to change. For example, the `Swap` function in pure C might be written as follows:

```
void Swap(int* one, int* two)
{
    int temp = *one;
    *one = *two;
    *two = temp;
}
```

To call this function to swap two integers, you would write code as follows:

```
int one = 137, two = 42;
Swap(&one, &two);
```

Rewrite the following function to use pass-by-pointer instead of pass-by-reference:

```
/* 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;
}
```

Finally, explain one advantage of pass-by-reference over pass-by-pointer and one advantage of pass-by-pointer over pass-by-reference.

Problem 7

How long did this problem set take you? How hard was it? Any suggestions for future problem sets?

Deliverables

To submit the assignment, email any files you've created to htiek@cs.stanford.edu. If possible, please send all answers either in a plain-text format (C++ source files are plain text) or as a PDF.

Good luck!