NJIT

New Jersey's Science & Technology University

THE EDGE IN KNOWLEDGE

Assignment 1 Solutions

Solution Notes

- The solutions given here are not the only solutions
- Smaller is usually better
- Not replicating code is better

Style

- Many people looked up the ASCII numeric codes for letters and used the numeric values. Using the character constant (for example, 'a') is ALWAYS more readable
- Indent your code so that it is obvious what lines of code go with what!
- Try to notice patterns in the problem that can determine what the code should look like

Frequently Seen Errors

- Check for eof() AFTER reading! Otherwise you are processing what you think is a successful read for the last read that you do… which actually fails
- Some people did not use “else if” for tests, so ended up doing multiple operations when only one was required
- Some people did not implement everything (forgot to read from standard input, or from a file)

Reading stdin vs files

- A file is an object of type ifstream
- It has most of the same methods and capabilities as the cin stream
- Writing code that is identical except for the stream read is a bad idea if you are copying and pasting large chunks of code

Stdin vs Files

- One way to not duplicate code is to write the code to read from cin or from the file, then use a common piece of code to perform the operation on what was read
- It might be a good idea to put the operation into a function

Stream references

- It is sometimes helpful to pass around references to streams (so you can, for example, tell a function to "read input from this stream")
- An istream& (reference to an istream) can refer to cin OR to a file
- Therefore you can simply pass cin or the file to a function, and it will just read the stream

Solutions

- Solution is broken up into several separate files
- Each file is its own function

Rot13

- Function takes stream to read from and stream to write to
- I pre-computed the rotations and saved them in a C-string (though a std::string would work just as well)
  - Rotation for a at position 0, rotation for b at position 1, etc.
- Just index into the pre-computed string using the input character (remember if you know a character is a lowercase letter, you know character – 'a' is the number 0-25)

Rot13 function

```
#include <iostream>
#include <cctype>
using namespace std;


void
rot13(istream& in, ostream& out)
{
    //const char *alpha = "abcdefghijklmnopqrstuvwxyz";
    const char *lower =   "nopqrstuvwxyzabcdefghijklm";
    //const char *alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    const char *upper =   "NOPQRSTUVWXYZABCDEFGHIJKLM";


    int    ch;
    for(;;) {
        ch = in.get();
        if( in.eof() ) return;

        if( islower(ch) ) {
            ch = lower[ ch - 'a' ];
        }
        else if( isupper(ch) ) {
            ch = upper[ ch - 'A' ];
        }

        out.put(ch);
    }
}
```

Pre-computed rotations

Read and check eof

Look up lowercase in one string, uppercase in the other

Note: if not a letter, ch's value is unchanged

Generate Dictionary

- Uses algorithm from recitation problem
- Takes argument for stream to write to

Generate Dictionary

```
#include <cstdlib>
#include <cstring>
#include <time.h>
#include <iostream>
using namespace std;


void mkdict(ostream& out) {
    const char *alpha = "abcdefghijklmnopqrstuvwxyz";
    char trans[27];  // 26+1 for null
    int nlets = 26;

    strcpy(trans, alpha);

    srand( time(NULL) );

    for( int i=0; i<26;  ) {
        int tix = rand() % nlets;
        if( trans[tix] == alpha[i] ) continue;


        out << alpha[i] << trans[tix] << endl;

        trans[tix] = trans[nlets-1];
        nlets--;
        i++;
    }
    return;
}
```

Space to select random letter from

Loop through all letters

Random spot

Skip match

Print entry

Shrink # of remaining random letters

Notice: increment i here

Read and Check Dictionary

- Performs all required tests
- Returns true on success, false on any error
- Takes references to maps to create for both directions of the encryption: “tocipher” for the map to ciphertext and “toclear” for the map to cleartext
- Checks for duplicates by making sure the ciphertext letter is not already a key in the toclear map; if it is, then we have a duplicate

Read and Check Dictionary

```
#include <iostream>
#include <map>
#include <string>
#include <cctype>
using namespace std;

bool
load_dict(istream& in, map<char,char>& tocipher, map<char,char>& toclear)
{
    string pair;
    for(char ch='a'; ch <= 'z'; ch++ ) {
        in >> pair;
        if( in.eof() || in.fail() )
            return false;
        if( pair.length() != 2 ) {
            cout << "FORMATTING ERROR " << pair << endl;
            return false;
        }
        if( pair[0] != ch ) {
            cout << "MISSING LETTER " << ch << endl;
            return false;
        }
        if( pair[0] == pair[1] ) {
            cout << "MAPPING ERROR " << ch << endl;
            return false;
        }
        tocipher[ pair[0] ] = pair[1];
        if( toclear.find( pair[1] ) != toclear.end() ) {
            cout << "DUPLICATE CIPHERTEXT " << pair[1] << endl;
            return false;
        }
        toclear[ pair[1] ] = pair[0];
    }

    return true;
}
```

Loop from ‘a’ to ‘z’

Read a pair at a time

Make sure there’s 2 letters only

Detect missing letter

Detect matching

Save in tocipher map

Check for duplicate

Save in toclear map

Encrypt and Decrypt

- Algorithm involves looking letter up in a map and using the value found in the map for output
  - If a is encrypted to m, tocipher[‘a’] == ‘m’
  - Also toclear[‘m’] == ‘a’
- No need to search the map: we already know it contains every letter because we checked when we loaded the dictionary
- NOTE: encrypt and decrypt are the same algorithm! The only difference is which map to use.

Encrypt and Decrypt

- Use a single piece of code for encrypt and decrypt
- Actual operation depends on which map is used
- Function takes references to map, stream to read and stream to write

Encrypt and Decrypt

```
#include <map>
#include <iostream>
using namespace std;


void caesar(map<char,char>& translate, istream& in, ostream& out)
{
    for(;;) {
        char ch;

        ch = in.get();
        if( in.eof() || in.bad() ) return;

        if( islower(ch) )
            ch = translate[ch];
        else if( isupper(ch) )
            ch = toupper( translate[tolower(ch)] );
        out.put(ch);
    }
    out << flush;
    return;
}
```

Read a character at a time

Translate lowercase

Translate uppercase

Write output: note if ch is not a letter, it is not changed

Main; pull it all together

- Make calls to the various functions that were already shown
- When reading the dictionary, do it once
- Use istream pointer to point at either cin or file: DO NOT DUPLICATE CODE

Main

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

extern void mkdict(ostream&);
extern void rot13(istream&, ostream&);
extern bool load_dict(istream& in, map<char,char>& tocipher, map<char,char>& toclear);
extern void caesar(map<char,char>& themap, istream&, ostream&);

int
main(int argc, char *argv[])
{
    map<char,char> tocipher;
    map<char,char> toclear;
    istream *in = &cin;
    ifstream file;

    if( argc == 1 ) {
          cout << "MISSING COMMAND" << endl;
          return 1;
    }
```

Extern declarations so compiler knows the functions we wrote

in defaults to point to cin. If we are reading from a file, we will open it and make in point to the file

Main - rotate

```
string cmd(argv[1]);

if( cmd == "-r" ) {
        if( argc > 3 ) {
            cout << "TOO MANY ARGUMENTS" << endl;
            return 1;
        }
        if( argc == 3 ) {
            file.open( argv[2] );
            if( !file.is_open() ) {
                cout << argv[2] << " FILE COULD NOT BE OPENED" << endl;
                return 1;
            }
            in = &file;
        }

        rot13(*in, cout);
}
```

Open file if user provided a filename

Change in to point to open file

Tell rot13 to read from *in (whatever in points to, cin or file) and write to cout

Main - generate

```
else if( cmd == "-g" ) {
        if( argc > 2 ) {
            cout << "TOO MANY ARGUMENTS" << endl;
            return 1;
        }
        mkdict(cout);
}
```

Main – encrypt/decrypt

```
else if( cmd == "-e" || cmd == "-d" ) {
        if( argc > 4 ) {
            cout << "TOO MANY ARGUMENTS" << endl;
            return 1;
        }
        if( argc < 3 ) {
            cout << "NO DICTIONARY GIVEN" << endl;
            return 1;
        }
        ifstream dict(argv[2]);
        if( !dict.is_open() ) {
            cout << argv[2] << " DICTIONARY COULD NOT BE OPENED" << endl;
            return 1;
        }

        if( !load_dict(dict, tocipher, toclear) )
            return 1;

        if( argc == 4 ) {
            file.open( argv[3] );
            if( !file.is_open() ) {
                cout << argv[3] << " FILE COULD NOT BE OPENED" << endl;
                return 1;
            }
            in = &file;
      }

      if( cmd == "-e" )
        caesar(tocipher, *in, cout);
      else
        caesar(toclear, *in, cout);
}
else {
        cout << cmd << " NOT A VALID COMMAND" << endl;
        return 0;
}

return 0;
}
```

Use function to load dictionary

Open file if user provided one

Change in to point to open file

Use function to encrypt or decrypt by passing the proper map into function

In conclusion

- Breaking work up into several files makes it easier to keep track of what you are working on
- Once a function works you never have to go change it
- Less code is easier to read


NJIT
THE EDGE IN KNOWLEDGE