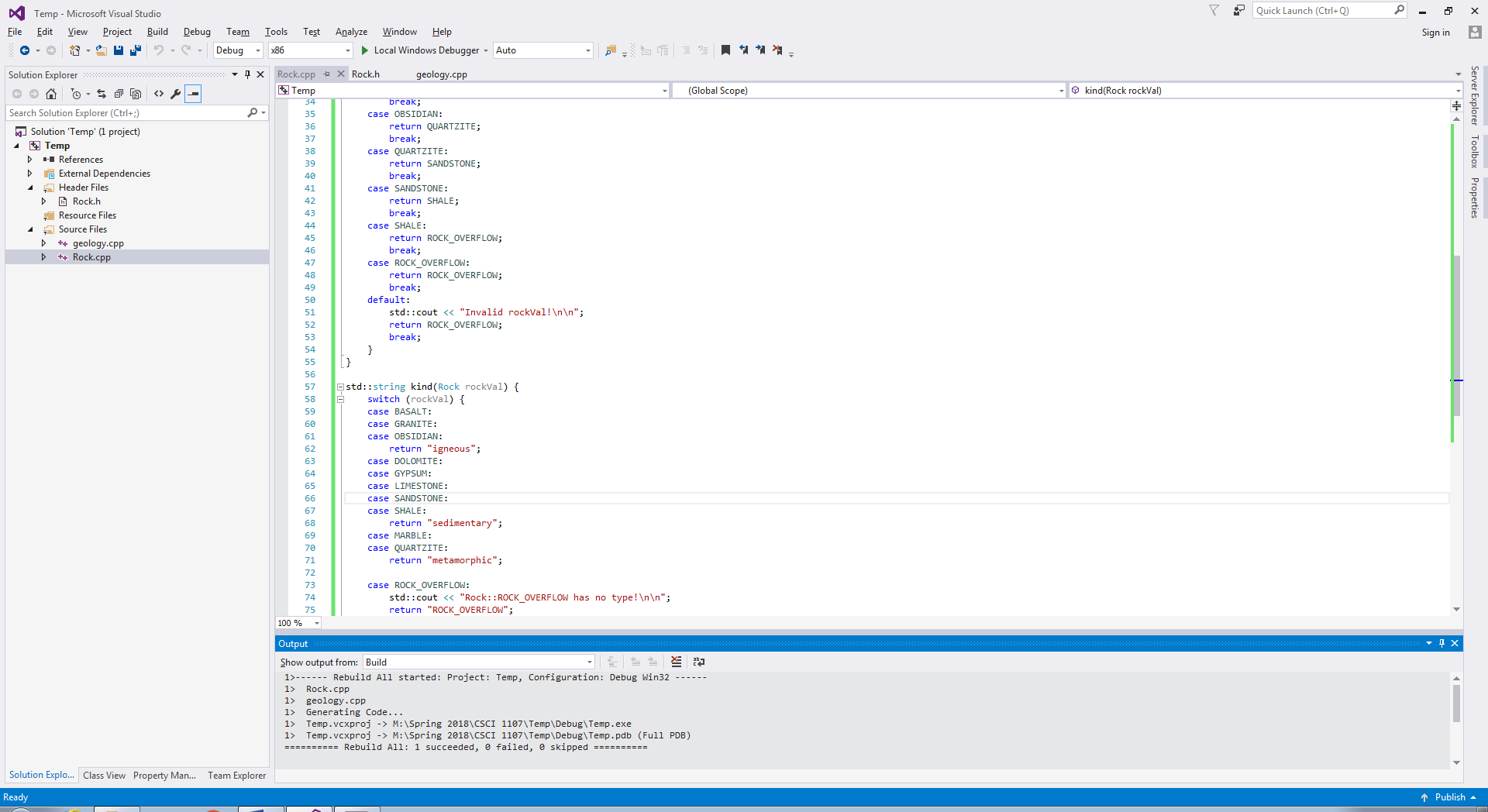
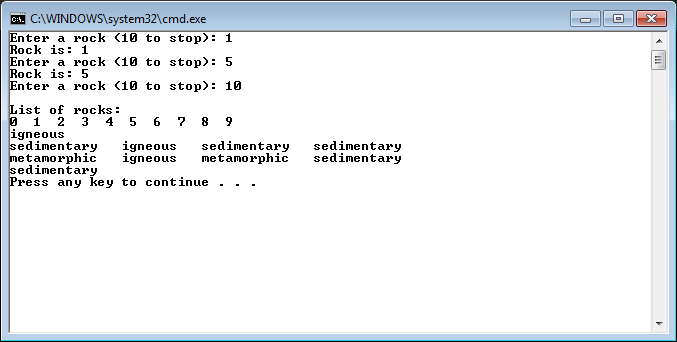
**CSCI 1107 Lab Experience Four**

You are to do the following from your lab manual. If you use a word processor to reproduce the lab manual you need to also include the question with your answer.

***Do Lab 2.2. On pages 23 – 24 you are supposed to create a function called next. Change the function name to nextOne instead of next.***





/\*--- geology.cpp ---------------------------------------------------

A program to model rocks.

Written by: Larry R. Nyhoff

Written for: Lab Manual for ADTs, Data Structures, and Problem

Solving with C++, 2E

Lab #2.2 and Project #2.2

Johnathan Lee CSCI 1107

Lab 4 Due Date: 02/06/18

---------------------------------------------------------------------\*/

#include <iostream>

using namespace std;

// Function prototypes go here

#include "Rock.h"

int main() {

/\*\*\*\*\* PART 1 \*\*\*\*\*\*/

// Declare a variable to store a rock

int sample;

// Input and display various rocks

for (;;) {

cout << "Enter a rock (" << ROCK\_OVERFLOW << " to stop): ";

int temp;

cin >> temp;

sample = Rock(temp);

if (sample == ROCK\_OVERFLOW) break;

cout << "Rock is: " << sample << endl;

}

/\*\*\*\*\* PART 2 \*\*\*\*\*\*/

// Display all the rock names

cout << "\nList of rocks:\n";

for (Rock rockVal = BASALT; rockVal < ROCK\_OVERFLOW; rockVal = nextOne(rockVal))

cout << rockVal << " ";

cout << endl;

for (Rock rockVal = BASALT; rockVal < ROCK\_OVERFLOW; rockVal = nextOne(rockVal))

// Display only 4 rocks per line (rockVal % 4)

cout << kind(rockVal) << " " << (rockVal % 4 == 0 ? "\n" : "");

cout << endl;

}

/\*--- Rock.h ------------------------------------------------------------------

\* Defines an enum Rock that models rocks of various types, and defines

\* operations for that enum.

\*

\* Johnathan Lee CSCI 1107

\* Lab 4 Due 02/06/18

\*----------------------------------------------------------------------------\*/

#include <string>

enum Rock {

BASALT,

DOLOMITE,

GRANITE, GYPSUM,

LIMESTONE,

MARBLE,

OBSIDIAN,

QUARTZITE,

SANDSTONE,

SHALE,

ROCK\_OVERFLOW

};

/// \brief Increments a rock value to the next rock in order.

/// \param rockVal The Rock to increment.

/// \returns The next Rock in the chain (Basalt->Dolomite, etc)

/// \note Incrementing Rock::ROCK\_OVERFLOW returns Rock::ROCK\_OVERFLOW.

Rock nextOne(Rock rockVal);

/// \brief Gives the kind (igneous, metamorphic, or sedimentary) of the rock input.

/// \param rockVal The rock whose type should be returned

/// \returns A string containing one of "igneous", "metamorphic", "sedimentary", or "ROCK\_OVERFLOW".

/// \note "ROCK\_OVERFLOW" is returned for an input of Rock::ROCK\_OVERFLOW or an invalid

/// rock type.

std::string kind(Rock rockVal);

/\*--- Rock.cpp ------------------------------------------------------------------

\* Defines an enum Rock that models rocks of various types, and defines

\* operations for that enum.

\*

\* IMPLEMENTATION.

\*

\* Johnathan Lee CSCI 1107

\* Lab 4 Due 02/06/18

\*----------------------------------------------------------------------------\*/

#include "Rock.h"

#include <iostream>

Rock nextOne(Rock rockVal) {

switch (rockVal) {

case BASALT:

return DOLOMITE;

break;

case DOLOMITE:

return GRANITE;

break;

case GRANITE:

return GYPSUM;

break;

case GYPSUM:

return LIMESTONE;

break;

case LIMESTONE:

return MARBLE;

break;

case MARBLE:

return OBSIDIAN;

break;

case OBSIDIAN:

return QUARTZITE;

break;

case QUARTZITE:

return SANDSTONE;

break;

case SANDSTONE:

return SHALE;

break;

case SHALE:

return ROCK\_OVERFLOW;

break;

case ROCK\_OVERFLOW:

return ROCK\_OVERFLOW;

break;

default:

std::cout << "Invalid rockVal!\n\n";

return ROCK\_OVERFLOW;

break;

}

}

std::string kind(Rock rockVal) {

switch (rockVal) {

case BASALT:

case GRANITE:

case OBSIDIAN:

return "igneous";

case DOLOMITE:

case GYPSUM:

case LIMESTONE:

case SANDSTONE:

case SHALE:

return "sedimentary";

case MARBLE:

case QUARTZITE:

return "metamorphic";

case ROCK\_OVERFLOW:

std::cout << "Rock::ROCK\_OVERFLOW has no type!\n\n";

return "ROCK\_OVERFLOW";

default:

std::cout << "Invalid Rock entered!\n\n";

return "ROCK\_OVERFLOW";

break;

}

}

Do Project 2.2 on pages 29-30. In project 2.2 you can use your overloaded extraction operator for file input since the ifstream class inherits from the istream class.

Copy and paste your program into a word document. Capture the console window for each test run and paste it below your program..

**Geology.cpp since it was modified in the course of Project 2.2:**

/\*--- geology.cpp ---------------------------------------------------

A program to model rocks.

Written by: Larry R. Nyhoff

Written for: Lab Manual for ADTs, Data Structures, and Problem

Solving with C++, 2E

Lab #2.2 and Project #2.2

Johnathan Lee CSCI 1107

Lab 4 Due Date: 02/06/18

---------------------------------------------------------------------\*/

#include <fstream>

#include <iomanip>

#include <iostream>

using namespace std;

// Function prototypes go here

#include "Rock.h"

int **main**() {

/\*\*\*\*\* PART 1 \*\*\*\*\*\*/

// Declare a variable to store a rock

Rock sample;

// Input and display various rocks

for (;;) {

cout << "Enter a rock (" << *ROCK\_OVERFLOW* << " to stop): ";

cin >> sample;

if (sample == *ROCK\_OVERFLOW*)

break;

cout << "Rock is: " << sample << endl;

}

/\*\*\*\*\* PART 2 \*\*\*\*\*\*/

// Display all the rock names

cout << "\nList of rocks:\n";

for (Rock rockVal = *BASALT*; rockVal < *ROCK\_OVERFLOW*;

rockVal = nextOne(rockVal))

cout << rockVal << " ";

cout << endl;

for (Rock rockVal = *BASALT*; rockVal < *ROCK\_OVERFLOW*;

rockVal = nextOne(rockVal))

// Display only 4 rocks per line (rockVal % 4)

cout << kind(rockVal) << " " << (rockVal % 4 == 0 ? "\n" : "");

cout << endl;

// More complete testing for project 2.2 #2

Rock input;

cout << "Please enter a rock: ";

cin >> input;

cout << "You entered: " << input << ", which is a(n) " << kind(input)

<< " rock!\n";

/\*\*\*\*\*\* Project 2.2 Program \*\*\*\*\*/

}

**The main file for Project 2.2:**

/\* Project-2.2.cpp

\*

\* Johnathan Lee CSCI 1107

\* Project 2.2 Lab 4

\* Due 02/06/18

\*/

#include <fstream>

#include <iomanip>

#include <iostream>

#include "Rock.h"

using namespace std;

bool **openFile**(string filename, ifstream& stream);

void **countRocks**(ifstream& inFile, int counts[]);

void **printArrayHistogram**(int ar[]);

int **main**() {

int count[*ROCK\_OVERFLOW*] = {0};

ifstream inFile;

if (!openFile("Rockfile.txt", *inFile*)) {

cout << "Failed to open Rockfile.txt! Exiting...\n\n";

exit(1107);

}

countRocks(*inFile*, count);

printArrayHistogram(count);

return 0;

}

/// \brief Helper for ifstream that gives simpler error checking.

/// \pre stream shouldn't already be opened

///

/// \param filename The name of the file to open

/// \param stream a REFERENCE to the ifstream to use.

///

/// \returns Whether the file opened correctly.

/// \post stream is opened, AND IF returned true, opened correctly.

bool **openFile**(string filename, ifstream& stream) {

stream.open(filename);

return !stream.bad();

}

/// \brief Counts occurences of rock names from an inFile.

///

/// \param inFile The file to read rocks from.

/// \note The file must contain the STRING names of rocks, no numbers.

///

/// \param counts The array to store the occurence count in.

/// \note counts MUST have enough space for all rocks. (a size of

/// Rock::ROCK\_OVERFLOW)

///

/// \post inFile's marker is at the end of the file, counts is filled with the

/// counts of each rock type.

void **countRocks**(ifstream& inFile, int counts[]) {

Rock rock;

while (inFile >> rock) // Should automatically stop at EOF

counts[rock]++;

}

/// \brief Prints a histogram of an array of rock counts.

///

/// \param ar The array to read from.

/// \note ar MUST be AT LEAST Rock::ROCK\_OVERFLOW in size.

///

/// \post ar Has been printed to stdout in the following format:

/// <ROCK NAME>: <X repeated once for each occurence> (<number of occurences>)

void **printArrayHistogram**(int ar[]) {

for (int i = 0; i < Rock::ROCK\_OVERFLOW; i++) { // Loop through rocks

cout << setw(12) << Rock(i) << ":";

for (int j = 0; j < ar[i]; j++) // Loop through X's

cout << "X";

cout << "(" << ar[i] << ")\n";

}

}

**Rock.h after being modified for Project 2.2:**

/\*--- Rock.h ------------------------------------------------------------------

\* Defines an enum Rock that models rocks of various types, and defines

\* operations for that enum.

\*

\* Johnathan Lee CSCI 1107

\* Lab 4 Due 02/06/18

\*----------------------------------------------------------------------------\*/

#ifndef ROCK\_H

#define ROCK\_H

#include <string>

using namespace std;

enum Rock {

*BASALT*,

*DOLOMITE*,

*GRANITE*,

*GYPSUM*,

*LIMESTONE*,

*MARBLE*,

*OBSIDIAN*,

*QUARTZITE*,

*SANDSTONE*,

*SHALE*,

*ROCK\_OVERFLOW*

};

/// \brief Increments a rock value to the next rock in order.

/// \param rockVal The Rock to increment.

/// \returns The next Rock in the chain (Basalt->Dolomite, etc)

/// \note Incrementing Rock::ROCK\_OVERFLOW returns Rock::ROCK\_OVERFLOW.

Rock nextOne(Rock rockVal);

/// \brief Gives the kind (igneous, metamorphic, or sedimentary) of the rock

/// input. \param rockVal The rock whose type should be returned \returns A

/// string containing one of "igneous", "metamorphic", "sedimentary", or

/// "ROCK\_OVERFLOW". \note "ROCK\_OVERFLOW" is returned for an input of

/// Rock::ROCK\_OVERFLOW or an invalid

/// rock type.

string kind(Rock rockVal);

/// \brief Overloads the << operator for Rock on output streams.

/// \param out The ostream we write to.

/// \param rockVal The rock whose name should be printed.

/// \post The user-friendly name of the rock has been written to out

/// \returns A reference back to the ostream for chaining.

/// \note Writes "ROCK\_OVERFLOW" to out if rockVal is invalid.

ostream& operator<<(ostream& out, Rock rockVal);

/// \brief Overloads the >> operator for Rock on input streams.

/// \param in The istream to read from

/// \param rockVal the rock to store the value in.

/// \returns A reference to the in stream for chaining.

/// \note Sets rockVal to Rock::ROCK\_OVERFLOW if invalid input.

istream& operator>>(istream& in, Rock& rockVal);

#endif

**Rock.cpp after modification for Project 2.2:**

/\*--- Rock.cpp

\*------------------------------------------------------------------ Defines an

\*enum Rock that models rocks of various types, and defines operations for that

\*enum.

\*

\* IMPLEMENTATION.

\*

\* Johnathan Lee CSCI 1107

\* Lab 4 Due 02/06/18

\*----------------------------------------------------------------------------\*/

#include "Rock.h"

#include <iostream>

using namespace std;

Rock nextOne(Rock rockVal) {

switch (rockVal) {

case *BASALT*:

return *DOLOMITE*;

case *DOLOMITE*:

return *GRANITE*;

case *GRANITE*:

return *GYPSUM*;

case *GYPSUM*:

return *LIMESTONE*;

case *LIMESTONE*:

return *MARBLE*;

case *MARBLE*:

return *OBSIDIAN*;

case *OBSIDIAN*:

return *QUARTZITE*;

case *QUARTZITE*:

return *SANDSTONE*;

case *SANDSTONE*:

return *SHALE*;

case *SHALE*:

return *ROCK\_OVERFLOW*;

case *ROCK\_OVERFLOW*:

return *ROCK\_OVERFLOW*;

default:

cout << "Invalid rockVal!\n\n";

return *ROCK\_OVERFLOW*;

}

}

string kind(Rock rockVal) {

switch (rockVal) {

case *BASALT*:

case *GRANITE*:

case *OBSIDIAN*:

return "igneous";

case *DOLOMITE*:

case *GYPSUM*:

case *LIMESTONE*:

case *SANDSTONE*:

case *SHALE*:

return "sedimentary";

case *MARBLE*:

case *QUARTZITE*:

return "metamorphic";

case *ROCK\_OVERFLOW*:

cout << "Rock::ROCK\_OVERFLOW has no type!\n\n";

return "ROCK\_OVERFLOW";

default:

cout << "Invalid Rock entered!\n\n";

return "ROCK\_OVERFLOW";

break;

}

}

ostream &operator<<(ostream &out, Rock rockVal) {

switch (rockVal) {

case *BASALT*:

out << "Basalt";

break;

case *DOLOMITE*:

out << "Dolomite";

break;

case *GRANITE*:

out << "Granite";

break;

case *GYPSUM*:

out << "Gypsum";

break;

case *LIMESTONE*:

out << "Limestone";

break;

case *MARBLE*:

out << "Marble";

break;

case *OBSIDIAN*:

out << "Obsidian";

break;

case *QUARTZITE*:

out << "Quartzite";

break;

case *SANDSTONE*:

out << "Sandstone";

break;

case *SHALE*:

out << "Shale";

break;

case *ROCK\_OVERFLOW*:

out << "ROCK\_OVERFLOW";

break;

default:

out << "ROCK\_OVERFLOW";

break;

}

return out;

}

istream &operator>>(istream &in, Rock &rockVal) {

// Simpler than if...else if...else if...etc

std::string nameMap[] = {

"BASALT", "DOLOMITE", "GRANITE", "GYPSUM", "LIMESTONE",

"MARBLE", "OBSIDIAN", "QUARTZITE", "SANDSTONE", "SHALE",

};

string buff;

rockVal = Rock::ROCK\_OVERFLOW; // Assume we can't find it.

in >> buff;

// Convert the entire string to uppercase. toupper ignores already uppercase.

for (int i = 0; i < buff.size(); i++)

buff[i] = toupper(buff[i]);

// If this can't find it, it's already ROCK\_OVERFLOW

for (int i = 0; i < *ROCK\_OVERFLOW*; i++)

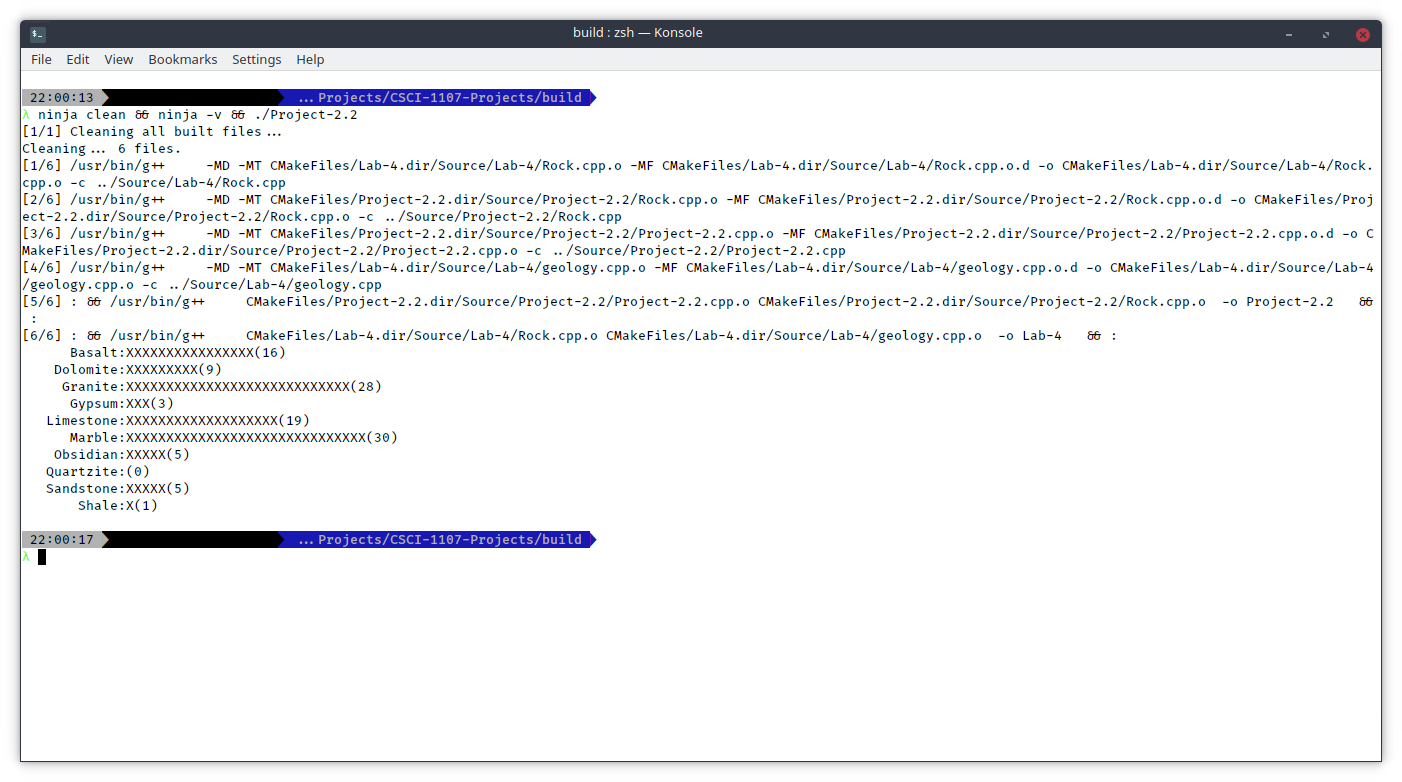
if (nameMap[i] == buff)

rockVal = Rock(i);

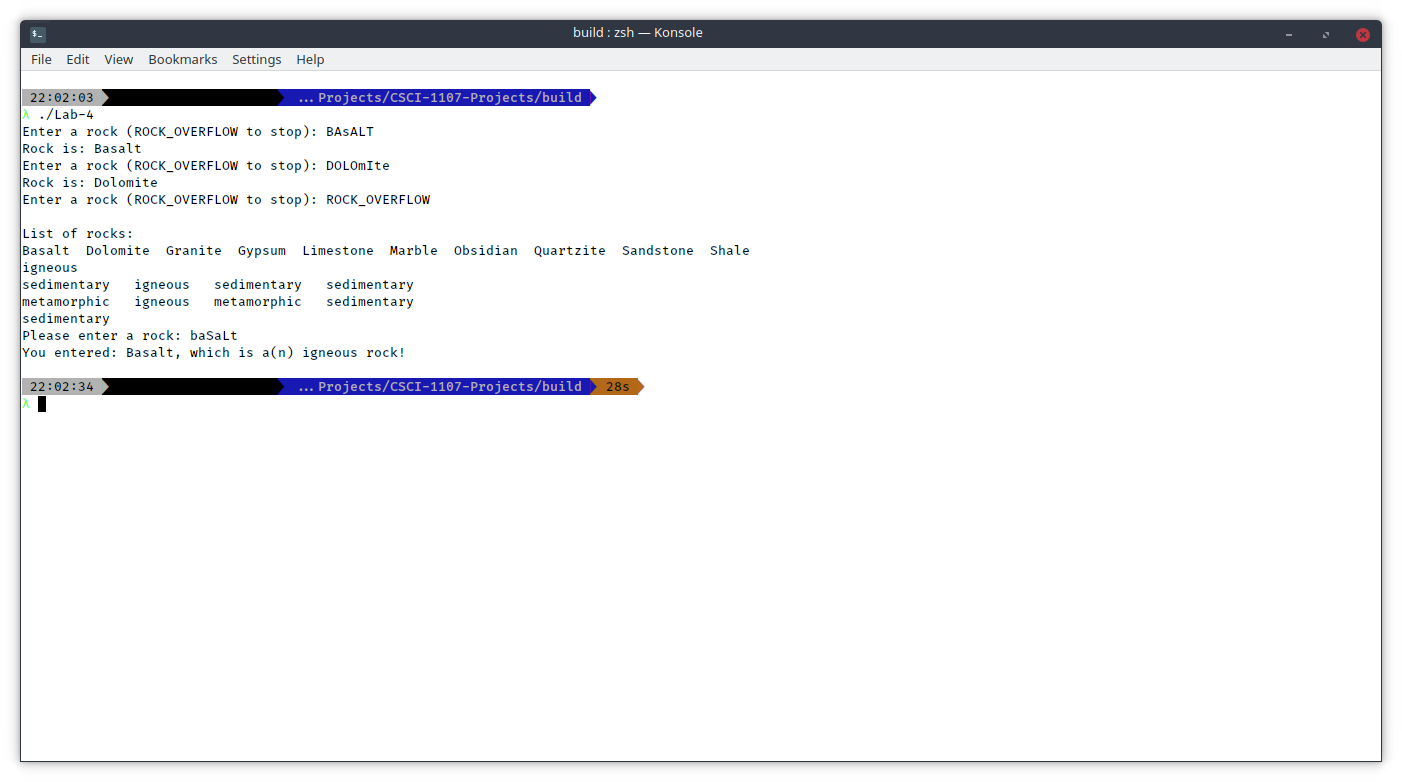
return in;

}

**Compile proof and screenshot of Project 2.2 (Both Lab 2.2 AND Project 2.2 were compiled here):**



**Lab 2.2 Screenshot:**



**What to turn in:**

1. Printouts of your word document and your source code.
2. A hard copy of the questions and answers from your lab manual.
3. Compress the word document and the source code from all of your programs and submit them to the D2L assignment folder called Lab Experience Four.