Caden Johnson

CSCI 301 – Section 03

10/5/2020

**Project 5 – Design Document**

Introduction:

This project is designed to implement a sequential stack and a recursion function. I will be using a similar recursion function to project 2. This function will find all of the prime factors of an integer and add them to the stack. The stack will store the prime factors in descending order.

Data Structures:

The data structures used in this project will be a sequential (array based) stack and a recursion function. This stack will be created using a class and will contain multiple class functions.

Functions:

This program will consist many functions. These include the following:

// Constructor

Stack()

// Push

void push(int entry)

// Pop

int pop()

// showStack - displays items in stack

void showStack()

// findPrimes - function for finding all prime factors of integer

void findPrimes(int number, Stack stack)

// main function

int main()

The Main Function:

The main function will be responsible for initializing a couple of variables and providing the user interface. It will check the integrity of the input and call the function required for finding the prime factors and outputting them.

**Project 5 – Source Code**

Below is my source code for Project 5. I will also be attaching the .cpp file to the submission as well just in case. Enjoy:

// Caden Johnson - CSCI 301 - 03

// Project 5 - Due 10/5/2020

// This project input an integer and output the integer's prime factors.

// Additionally, this program will implement an array based stack

// to store the factors and eventually output them in a decreasing order.

#include <iostream>

using namespace std;

static const int CAPACITY = 15;

bool complete = false;

// stack class

class Stack

{

private:

int data[CAPACITY];

int used;

public:

// Constructor

Stack()

{

used = -1;

}

// Push

void push(int entry)

{

if(used + 1 == CAPACITY)

{

cout << "ERROR : integer is too large, update CAPACITY" << endl;

}

else

{

used++;

data[used] = entry;

}

}

// Pop

int pop()

{

if(used == -1)

{

cout << "ERROR : stack is empty" << endl;

return -1;

}

else

{

int temp = data[used];

return temp;

}

}

// displays items in stack

void showStack()

{

int temp;

while(used >= 0)

{

temp = pop();

if(used == 0)

{

cout << temp << endl;

}

else if(used > 0)

{

cout << temp << " x ";

}

used --;

if(used == -1)

{

break;

}

}

}

};

// function for finding all prime factors of integer

void findPrimes(int number, Stack stack)

{

for(int i=2; i<=number; i++)

{

if(number%i == 0)

{

stack.push(i);

if(i != number)

{

findPrimes((number/i), stack);

break;

}

}

}

if(!complete)

{

stack.showStack();

complete = true;

}

}

// main function

int main()

{

Stack stack;

int entry = 1;

while(entry != 0)

{

cout << "Enter a positive integer (0 to stop): ";

cin >> entry;

if(entry == 1)

{

cout << "Prime factors: 1 = 1" << endl;

}

else if(entry > 1)

{

findPrimes(entry, stack);

}

else if(entry < 0)

{

cout << "ERROR : invalid input" << endl;

}

}

return 0;

}

**Project 5 – User Document**

This program must be compiled and then ran via an IDE or from a terminal/command prompt. The compiling can be done by an IDE with a graphical user interface, or you can use g++ -o “new\_name” “old\_file\_name.cpp”. Once compiled, you can run the program.

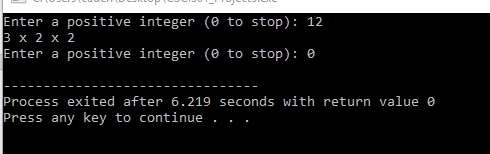
Once displayed with the prompt:

Enter a positive integer (0 to stop):

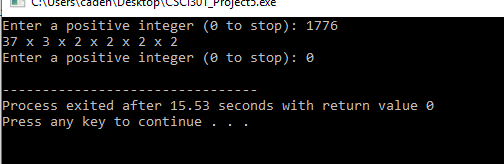
You can input an integer greater than 0. Once entered, the program will output the list of prime factors that make up the integer.

**Project 5 – Tests**

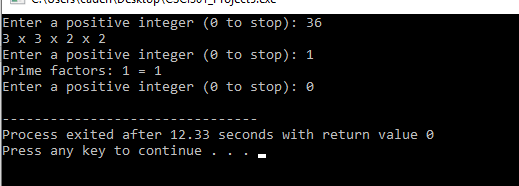
I was able to get this program to work 100% and although I came across some errors, I was able to fix them and learn from them. Below, you can see the tests for this program proving its functionality:



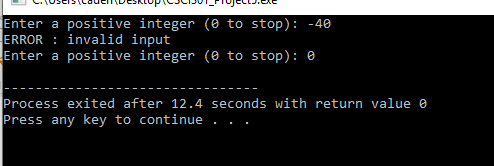
As you can see above, the program is able to find the prime factors of a simple integer and list them in descending order.



In this screenshot, you can see how the program is able to handle larger integers as well and operate smoothly.



Here, you can see how the program can repeat multiple times and even operate when the integer is 1.



Lastly, this screenshot shows how the program will make sure there are no invalid inputs. You can also see in all of the tests how 0 is used to stop the program.

**Project 5 – Summary**

For Project 5, I designed a program to implement a sequential stack and display the potential. This implementation used a recursive function to find the prime factors and added them to the stack. This is beneficial since the output will always result in a descending order.

I learned a ton from this project and I am extremely happy that I was able to master this method of stack creation and management.

Lastly, this program uses the iostream library.

**Project 5 – Project Questions:**

If we wanted to report each integer's prime factors in **increasing** order, would the stack be necessary or helpful? Explain.

* If implementing an ascending order as an output, the stack would still be beneficial. The integers would have to be added to the list backwards, but this can be done by dividing the number by the factor and adding the result rather than directly adding the factor. You would not have to change the stack, but you would have to change the way the recursive function adds items to the list.