Sushil Khanal

Section 1

Project 7

**PRIME FACTORIZATION USING AN ARRAY-BASED STACK**

**Introduction**

An integer is prime if it is divisible only by itself and 1. Every integer can be written as a product of prime numbers, unique except for their order, called its prime factorization. For example,

1776 = 37 x 3 x 2 x 2 x 2 x 2.

The objective of this program is to print out prime factorizations in descending order of the integers entered. This program prints out prime factorizations for all integers and will suspend the program when entered zero or any negative number.

**Data Structures**

The program contains .h file, .cpp file and main file. .h contains class module, functions and other member functions, which consists of all function prototypes. .cpp file contains the code for the functions. The main file receives instructions from the user and implements the function accordingly. Stack Abstract Data Type (ADT) is in a class called stack, it has item used and array of integer data type. The class has a capacity which defines the total size of the stack.

**Functions**

The functions used are:

Modification member functions

void push( item entry):

This function push the entered item into the Stack.

item pop():

This function returns the top item of the stack after popped.

void prime\_factor(item integer):

This function pushes the value of prime factors into the Stack.

Boolean functions

bool is\_Empty():

This function returns TRUE if the Stack is empty, else FALSE.

bool is\_InputValid(item a):

This function returns TRUE if the item is int type, else FALSE.

constant member function

item size\_t():

This function returns the total size of the stack used.

Friend Function

friend std::ostream& operator << (std::ostream& out\_s, stack& s):

This friend function has access to private contents which writes out the contents of a Stack to an output stream.

**The Main Program**

After successful compilation of the program, the program asks the user to enter an integer and when entered an integer (positive and non-zero) the program outputs the prime factors in descending order. When “0” is entered instead of a positive integer the program gets terminated.

**Code**

Project7.cpp

#include <iostream>

#include "stack.h"

#include "stack.cpp"

using namespace std;

int main()

{

stack st; //class type

int num; //integer

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

cin >> num; //enter an integer to find its prime factor

while (st.is\_InputValid(num) == true) //finds the prime factors of integer until true

{

if (num == 1) {

cout << " Prime factors: " << num << " = ";

cout << num << " " << endl;

}

else if (num > 1) {

cout << " Prime factors: " << num << " = ";

st.prime\_factor(num);

cout << st << endl;

}

else {

break; //stop if the input is less than or equal to 0

}

cout << endl;

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

cin >> num; //takes another integer until the condition is true

} //while

return 0;

}

Stack.cpp

#include <iostream>

#include "stack.h"

using namespace std;

//destructor

stack::~stack()

{

int y;

while (!is\_Empty()) {

y = pop();

}

}

//Returns TRUE if the item is "int", FALSE otherwise

bool stack::is\_InputValid(item a)

{

if (a > 0 || a <= 0) {

return true;

}

else {

return false;

}

}

//The item entry has been pushed onto the Stack.

void stack::push(item entry)

{

if (used == CAPACITY - 1) {

cout << "Stack overflow.\n";

}

else {

arr[used] = entry;

used++;

}

}

//The Stack's top item has been popped and returned.

stack::item stack::pop()

{

int top;

if (is\_Empty()) {

cout << "Empty stack.\n";

}

else {

--used;

top = arr[used];

}

return top;

}

//The prime factor of the item has been pushed onto the stack

void stack::prime\_factor(item num)

{

for (int i = 2; num != 1; ++i) {

while ((num % i) == 0) {

push(i);

num = num / i;

}

}

}

//The contents of the stack have been written to the output stream out\_s

ostream& operator<<(std::ostream& out\_s, stack& s)

{

while (s.is\_Empty() != true) {

if (s.size\_t() > 1) {

out\_s << s.pop() << " x ";

}

else {

out\_s << s.pop();

}

}

return out\_s;

}

stack.h

#include <iostream>

#ifndef STACK\_H

#define STACK\_H

using namespace std;

class stack

{

public:

// defination of data types like int, char, etc(integer in this case)

typedef int item; // enters the stack

static const item CAPACITY = 100; // maximum size of the stack

// constructor

stack()

{

used = 0;

} // inline

// Postcondition: The stack has been initialized as an empty stack.

// destructor

~stack();

// modification member function

void push(item entry);

// Postcondition: The (item entry) has been pushed into the stack.

item pop();

// Precondition: size\_t() > 0; i.e the stack is not empty.

// Postcondition: The stack's top item has been popped and returned.

// precondition: The (item integer) lies between negative infinity to positive infinity,

void prime\_factor(item integer);

// i.e, it shouldn't be a character

// postcondition: The prime factor of the item has been pushed onto the stack

// boolean function

bool is\_Empty()

{

return used == 0;

}

// Postcondition: Returns true if the stack is empty, else FALSE

bool is\_InputValid(item a);

// Postcondition: Returns true if the item is int,else FALSE

// constant member function

item size\_t() const

{

return used;

} // Inline

// postcondition: Returns the total size of the stack used

// friend function

friend std::ostream& operator<<(std::ostream& out\_s, stack& s);

// post-condition: The contents of the stack have been printed as an output

// stream out\_s

private:

// private data members

item used;

// capacity of the array to store data

item arr[CAPACITY];

};

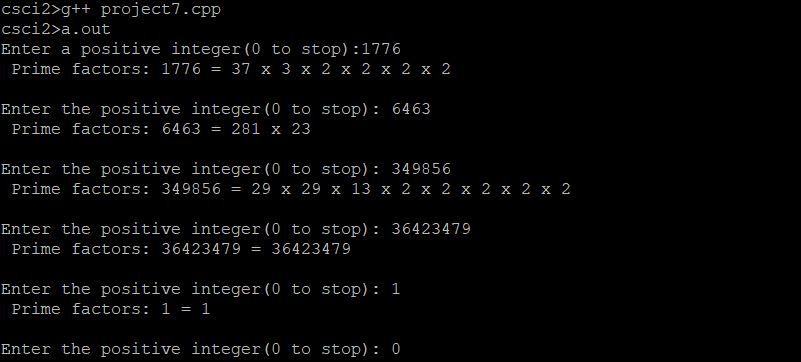
#endif

**User Document**

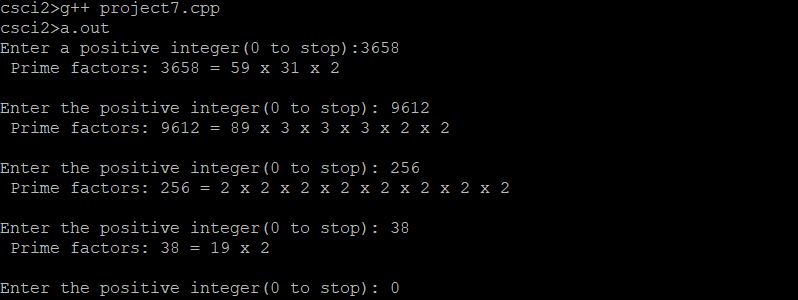
After the completion of code writing, we run the program by compiling it. In the terminal window, we should get into the directory where our codes are and compile the program by entering g++ project7.cpp. If the program has any errors, it would show up in the terminal screen, with the location of the error. If not type **a**.**out** command in the next line to run the program. Finally we got the desired output in the terminal window.

**Tests**

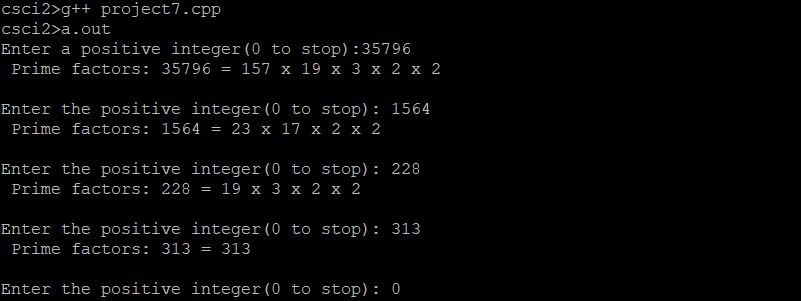
Test 1



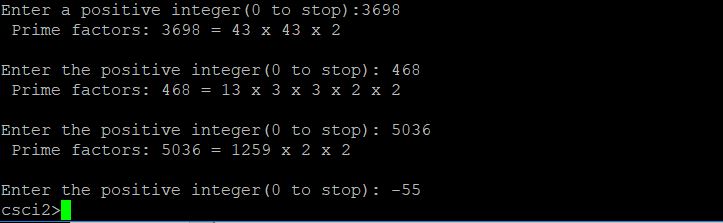
Test 2



Test 3



Test 4



**Conclusion**

We were able to design a program that takes input from user and display its prime factors in descending order. This program was based in the similar concept of our past programming assignments. I was very relieved when the program ran successfully. I had been working on this for quite some time and it was an amazing feeling to obtain the desired output.

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

* Yes, the stack would be helpful in order to display prime factors in increasing order. We could push the factors to the stack to pop it for the second time, which would result the factors to display in increasing order.