Project 8

Queue Simulation

Full Name: Prashul Shrestha

Section #: 2

Project #: 8

Due Date: April 6, 2017

**Design** **Document**

**Introduction**

**A stack** is a basic data structure that can be logically thought as linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack.

**Data** **Structures**

The program uses a class called Class **Stack()**, an integer named **Prime**, and a stack of integer type Item Data[] to store each prime factor into the stack. First, the number is taken from the user and checked if it’s Zero or not to stop the program. If it’s non Zero, then the number is taken to check for its factor and that factor is referenced to another function **check\_prime(int),** where the factor is pushed into the stack if it’s prime or void is returned.

**Functions**

The program uses **four** functions to implement the Prime Factorization through Stack. The functions are called from main() and some are member function to return the result within the function which called it. The list of the functions are given below:

* Stack() : A constructor to initialize used to be empty.
* ~Stack(): A destructor to delete all the stacks and make used as 0.
* Void Push(Item Entry)– This will insert the item into the stack.
* Item pop(): Returns the items form the stack.
* Check\_prime() : Check if the factor is prime or not.
* S\_print()– to print the items in the list.

**Menu**() is a the main function from where we first create an object or an instance of class **Stack** called **s** and which helps invoke the functions such as s**.check\_prime**(i); and **s.s\_print ()** to print the items in the list.

**The Main Program**

**Menu**() is a the main function from where we first create an object or an instance of class **S** and which helps invoke the functions such as **s.check\_prime**(**i**); to check the factor is prime. The program is quite simple, first, the number is taken from the user and checked if it’s Zero or not to stop the program. If it’s non Zero, then the number is taken to check for its factor and that factor is referenced to another function **check\_prime(int),** where the factor is pushed into the stack if it’s prime or void is returned.

User Document

**A stack** is a basic data structure that can be logically thought as linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the s **A stack** is a basic data structure that can be logically thought as linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack.

The main program named **main**.**cpp** can be compiled and run, using the code:

**g++ main**.**cpp**

**a.out**

**g++** function will compile the function and make it ready to be run using **a.out**. The function will prompt the following output:

**Ouptut:**

Enter a positive integer (0 to stop): 1776

Prime factors: 1776 = 37 x 3 x 2 x 2 x 2 x 2

Enter a positive integer (0 to stop): 6463

Prime factors: 6463 = 281 x 23

Enter a positive integer (0 to stop): 349856

Prime factors: 349856 = 29 x 29 x 13 x 2 x 2 x 2 x 2 x 2

Enter a positive integer (0 to stop): 352170

Prime factors: 352170 = 43 x 13 x 7 x 5 x 3 x 3 x 2

**Summary**

Completing this project, I learnt the implementation of stack in the real-world experience. Pushing and popping items from the stack seemed really easy.

Since, in the Stack is First in first out, if we input an items into the list the last inputted gets printed first and the first inputted gets printed last. So, while printing factors in ascending order can be pretty easy if we use stack unlike if we store it in an array, we need to print it in opposite direction.