Research Article
Volume 10:5, 2021

Industrial Engineering & Management

ISSN: 2169-0316 Open Access

A Designed System for Providing Solutions to Basic Engineering Problems

Paul Tamaragaibi*

Paul Tamaragaibi, Applied Mechanics and Design/Production Research Group, Department of Mechanical Engineering, Nigeria Maritime University, Okerenkoko, Delta State, Nigeria.

Abstract

Engineering and the manner in which engineers think is largely visual and functional, and yet engineers are typically provided with a lot of other basic engineering problems to be solved and applied to complex ones. This paper present a designed software for providing solutions to some basic engineering problems. C++ programming language was used in writing the codes. The paper explains the activities involved in C++program development and touched areas such as the general concept of programming, programming languages, compilation and interpretation, storage and execution, functional lines to which computer programs maybe categorized. Several engineering problems and their various real-world applications were considered to accomplish the aim of the work. The problem requirements were properly understood, and the program adequately developed using the Dev-C++ software. The program was coded in a modular fashion, and constructed in a manner that can be fairly understood. Algorithms and flow charts for the basic engineering problems considered were obtained and using formula translation, solutions to such issues were dynamically programmed using the Dev C++ software. The results from the developed software when compared with other methods such as numerical and approximate analytical solutions were observed to be the same. The developed software is however, more user friendly and can be implemented at A Level Higher Institutions, as well as have many other real-world applications.

Keywords: Design, Software, Basic Engineering, Programming, C++ Program.

Introduction

Generally, a computer program is a recipe, a structured list of instructions to be sequentially executed by a CPU. In programming, every detail is clearly specified, as there are no assumptions (i.e. the CPU is "dumb"). Instructions also cover abnormal inputs, outputs or conditions. A computer program is therefore a collection of instruction that can be executed by a computer to perform a specific task (Rochki nd, 2004). Most computer devices require programs to function properly. A computer programmer usually writes a computer program using a programming language. From the program in its human-readable form of source code, a compiler or assembler can derive machine code (a form consisting of instructions that the computer can directly execute). Alternatively, an interpreter may execute the computer program.

It is also the process of writing or editing source code. Editing source code involves testing, analyzing, refining, and sometimes coordinating with other programmers on a jointly developed program. A person who practices this skill is referred to as a computer programmer, software developer, and sometimes coder. Code breaking algorithms have existed for centuries. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code; in a manuscript on deciphering cryptographic messages. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm [**1**].

Computer programs are stored and executed as binary digits in the RAM and CPU. The binary numbers represent instructions and data in the program. Programs are written in high-level languages (e.g. C, C++, Python, etc.). They are either compiled or interpreted into assembly code, then converted into binary machine code, and finally executed. Compiled languages have the high-level program permanently converted and stored as binary (i.e. they are converted directly into machine code that the processor can execute), while Interpreted languages have them stored as it is and only converted at runtime (meaning that they run through a program line by line and execute each command). In the implementation of numerical methods, any mathematical problem reduced to a numerical problem, can be solved with a computer program **[2].** An example is to implement Newton-Raphson's method using programming pseudocode. Here it includes finite series expansion, transformations, search, sort and recursive problems.

A collection of computer programs, libraries, and related data referred to as software. Computer programs may be categorized along functional lines, such as application software and system software. The underlying method used for some calculation or manipulation is known as an algorithm. The sometimes lengthy process of computer programming is usually referred to as software development. The term software engineering is becoming popular as the process is seen as an engineering discipline.

Software design is the process of envisioning and defining software solutions to one or more sets of problems. One of the main components of software design is the software requirements analysis (SRA). SRA is a part of the software development process that lists specifications used in software engineering.

The C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming. It is a middle-level language, as it comprises of high-level and low-level language features. This language was developed by BjarneStroustrup starting in 1979 at Bell Labs, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.

Programming languages

According to Wilson (1993) **[3]** computer programs can be categorized by the programming language paradigm used to produce them. Two of the main paradigms are imperative and declarative. Imperative programming languages specify a sequential algorithm using declarations, expressions, and statements:

***Address for Correspondence:** Paul Tamaragaibi, Applied Mechanics and Design/Production Research Group, Department of Mechanical Engineering, Nigeria Maritime University, Okerenkoko, Delta State, Nigeria, Tel: 07061649506, E-mail: paul.elijah@nmu.edu.ng*

Received 07 May 2021; **Accepted** 21 May 2021; **Published** 28 May 2021

1. A declaration couples a variable name to a data-type (for example: var x: integer;)

2. An expression yields a value (for example: 2 + 2 yields 4)

3. A statement might assign an expression to a variable or use the value of a variable to alter the program's control flow (for example: x := 2 + 2; if x = 4 then do something ();)

One criticism of imperative languages is the side effect of an assignment statement on a class of variables called non-local variables.

Declarative programming languages describe what computation should be performed and not how to compute it. Declarative programs omit the control flow and are considered sets of instructions. Two broad categories of declarative languages are functional languages and logical languages. The principle behind functional languages (like Haskell) is to not allow side effects, which makes it easier to reason about programs like mathematical functions (Wilson, 1993). The principle behind logical languages (like Prolog) **[4]** is to define the problem to be solved and leave the detailed solution to the Prolog system itself (Wilson, 1993). A list of sub goals defines the goals. Then each sub-goal is defined by further providing a list of its sub-goals, etc. If a path of sub-goals fails to find a solution, then that sub-goal is backtracked and another path is systematically attempted.

Compilation and interpretation

A computer program in the form of a human-readable, computer programming language is a source code. A source code may be converted into an executable images by a compiler or assembler, or executed immediately with the aid of an interpreter.

According to Silberschatz (1994) **[5]** compilers translate source code from a programming language into either object code or machine code. Object code needs further processing to become machine code, and machine code consists of the central processing units native instructions, ready for execution. Compiled computer programs are commonly referred to as executables, binary images, or simply as binaries (a reference to the binary file format used to store the executable code).Some compiled and assembled object programs need to be combined as modules with a linker utility in order to produce an executable program.

Interpreters execute a source code from a programming language line-by-line. The interpreter decodes each statement and performs its behavior. One advantage of interpreters is that they can be easily extended to an interactive session. The programmer is presented with a prompt, and individual lines of code are typed in and performed immediately.

The main disadvantage of interpreters is that computer programs run slower than when compiled. Interpreting a code is slower because the interpreter must decode each statement and then perform it. However, software development may be faster using an interpreter because testing is immediate when the compiling step is omitted. Another disadvantage is that an interpreter must be present on the executing computer. By contrast, compiled computer programs need no compiler present during execution. Just in time compilers pre-compile computer programs just before execution. For example, the Java virtual machine hotspot contains a just in time Compiler which selectively compiles Java bytecode into machine code – but only codes which the hotspot predicts is likely to be used many times. Either compiled or interpreted programs might be executed in a batch process without human interaction. Scripting languages are often used to create batch processes. One common scripting language is Unix shell, and its executing environment is the command-line interface.

No properties of a programming language require it to be exclusively compiled or exclusively interpreted. The categorization usually reflects the most popular method of language execution. For example, Java is thought of as an interpreted language and C a compiled language, despite the existence of Java compilers and C interpreters.

Storage and Execution

Typically, computer programs are stored in non-volatile memory until requested either directly or indirectly to be executed by the computer user. Upon such a request, the program is loaded into random-access memory; by an operating system, where the central processor can directly access it. The central processor then executes ("runs") the program, instruction by instruction, until termination. A program in execution is called a process (Forsythe et. al., 1977). **[6]** Termination is either by normal self-termination, by user intervention, or by error – software or hardware error.

Simultaneous execution

Many operating systems support multitasking which enables many computer programs to appear to run simultaneously on one computer. Operating systems may run multiple programs through process scheduling (a software mechanism to switch the CPU among processes often so users can interact with each program while it runs (Omijeh, 2009). Within hardware, modern day multiprocessor computers or computers with multicore processors may run multiple programs.

Self-modifying programs

A computer program in execution is normally treated differently from the data the program operates on. However, in some cases, this distinction is blurred when a computer program modifies itself. The modified computer program is subsequently executed as part of the same program. Self-modifying code is possible for programs written in machine, assembly language, Lisp, C, COBOL, PL/1, and Prolong.

Functional Categories

Computer programs may be categorized along functional lines. The main functional categories are application software and system software. System software includes the operating system which couples computer hardware with application software (Silberschatz 1994). **[7]** The operating system provides an environment in which application software executes in a convenient and efficient manner. In addition to the operating system, system software includes embedded programs, boot programs, and micro programs (Wilson, 1993). Application software designed for end users have a user interface. Application software not designed for the end user includes middleware, which couples one application with another. Application software also includes utility programs. The distinction between system software and application software is under debate.

Application software

There are many types of application software:

1. The word app came to being in the 21st century. It is a clipping of the word "application". They have been designed for many platforms, but the word was first used for smaller mobile apps. Desktop apps are traditional computer programs that run on desktop computers. Mobile apps run on mobile devices. Web apps run inside a web browser. Both mobile and desktop apps may be downloaded from the developers' website or purchased from app stores such as Microsoft Store, Apple App Store, Mac App Store, Google Play or Intel AppUp.

2. An application suite consists of multiple applications bundled together. Examples include Microsoft Office, LibreOffice, and iWork. They bundle a word processor, spreadsheet, and other applications.

3. Enterprise applications bundle accounting, personnel, customer, and vendor applications. Examples include enterprise resource planning, customer relationship management, and supply chain management software.

4. Enterprise infrastructure software supports the enterprise's software systems. Examples include databases, email servers, and network servers.

5. Information worker softwareare designed for workers at the departmental level. Examples include time management, resource management, analytical, collaborative and documentation tools. Word processors, spreadsheets, email and blog clients, personal information system, and individual media editors may aid in multiple information worker tasks.

6. Media development software generates print and electronic media for others to consume, most often in a commercial or educational setting. For instance, the advancement of novel broadband interaction services, harnessing of telecommunication and computers, recent advances in the field of interaction protocol have fostered numerous proposals for the uses of ICT to support the instruction and learning environment in higher education (Amini-Philips and Elijah, 2019). These produce graphics, publications, animations, and videos.

7. Product engineering software helps develop large machines and other application software. Examples includes computer-aided design (CAD), computer-aided engineering (CAE), and integrated development environments.

8. Entertainment Software can refer to video games, movie recorders and players, and music recorders and players.

Utility programs

Utility programs are application programs designed to aid system administrators and computer programmers.

Operating system

An operating system is a computer program that acts as an intermediary between a user of a computer and the computer hardware (Silberschatz1994).

In the 1950s, the programmer, who was also the operator, would write a program and run it. After the program finished executing, the output may have been printed, or it may have been punched onto work tape or cards for later processing (Wilson, 1993). Usually, the program did not work. The programmer then looked at the console lights and fiddled with the console switches. If less fortunate, a memory printout was made for further study. In the 1960s, programmers reduced the amount of wasted time by automating the operator's job. A program called an operating system was kept in the computer at all times. Originally, operating systems were programmed in assembly; however, modern operating systems are typically written in C.

Boot program

A stored-program computer requires an initial computer program stored in its read-only memory to boot. The boot process is to identify and initialize all aspects of the system, from processor registers to device controllers to memory contents (Wilson, 1993). Following the initialization process, this initial computer program loads the operating system and sets the program counter to begin normal operations**. [8]**

Embedded programs

Independent of the host computer, a hardware device might have embedded firmware to control its operation. Firmware is used when the computer program is rarely or never expected to change, or when the program must not be lost when the power is off.

Microcode programs

Microcode programs control some central processing units and some other hardware. This code moves data between the registers, buses, arithmetic logic units, and other functional units in the CPU. Unlike conventional programs, microcode is not usually written by, or even visible to, the end users of systems, and is usually provided by the manufacturer, and is considered internal to the device.

The intent of the work is to design a system that gives solution to some basic engineering problems, based on their formula and applications. Specifically, the objectives of the study include:

1. To understand basic engineering problem requirements and design an efficient way of solving them using algorithm.

2. To develop a system using the C++ software, that will accept user's input for certain engineering problems, and then provide adequate solutions.

Methodology

The methodology used for the development of this application was a top-down approach. The design began by specifying complex pieces of the problem, and then dividing them into successively smaller pieces. The top–down technique was used to write a main procedure/function that names all the major functions it needed.

The requirements of each of those functions were also considered and the process was repeated. These compartmentalized functions eventually performed actions so simple, easy and concise codes. After the various functions had been coded the program was ready for testing. By defining how the application comes together at a high level, lower level work could be self-contained. By also defining how the lower level abstractions integrated into higher level ones, the interface became clearly defined.

The program codes were properly written and tested using an assortment of variables to check for logical errors. Some of the basic engineering problems to be simulated were the Fibonacci series, Simultaneous equations, Factorials, Cubic Polynomial equations, and Pascal's Triangle, etc.

Dev-C++ was used in carrying out this project. Dev-C++ software is a free full-integrated development environment distributed under the GNU General Public License for programming C and C++. For the purpose of this work, C++was used as the programming language for the design of the software.

C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming. It is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features. It was developed by Bloodshed software until 2005, Orwell (Johan Mes) since 2011.

The following steps are used as procedure for this work:

Installation of Dev-C++ software:

Dev-C++ software was used and installed on a laptop. It comes in different version with almost same features and functions. The 2019 version was the latest when this research was carried out (with code name "Dev 16" and latest update version "16.6.4") and the researchers believe it has more advantages in terms of software compilation, as such it is used for this work.

Algorithm

An algorithm is a special case of an iterative method, which generally need not converge in a finite number of steps (Chipperfield and Fleming, 1994; Cody and Smith, 2006; Elijah and Etebu; Elijah et. al., 2020). Instead, an iterative method produces a sequence of iterates from which some subsequence converges to a solution.

It is procedure for finding a valve (say x) such that f (x) is as small (or as large) as possible, for a given function f, possibly with some constraints on x. Here, x can be a scalar or vector of continuous or discrete valves. An algorithm terminates in a finite number of steps with a solution (Fehlberg, 1977).

Designing the Software

For the purpose of clarity, some of the algorithm as well as codes used in designing the software are respectively shown below:

The Algorithm

Step 1: Start

Step 2: Declare variables

Step 3: Print the engineering problems menu, and ask the user to make a choice

Step 4: Enter choice

If choice is 1(Fibonacci Series):

Print the Fibonacci series problem menu, and ask the user to make a choice

Enter choice

If selection choice is 1(Generate list of Fibonacci numbers):

Enter the amount of numbers to be displayed

Calculate for eachnumber

Print list of numbers

Else if selection choice is 2(Find the nth number in the Fibonacci series):

Enter the nth term you want to find

Generate the nth term

Print the nth term

Prompt the user if they want to calculate another engineering problem

Enter answer

If answer is yes: Go to Step 3

Else: Go to Step 5

Else if choice is 2 (Simultaneous Equations):

Enter the coefficients of the first equation Enter the coefficients of the second equation

Calculate for the solutions to each variable (x and y)

Print solutions to x and y

Prompt the user if they want to calculate another engineering problem

Enter answer

If answer is yes:

Go to Step 3

Else:

Go to Step 5

Else if choice is

Else if choice is 3(Factorial):

.

.

.

Else:

Print an Invalid Entry message

Prompt the user if they want to calculate another engineering problem Enter answer

If answer is yes:

Go to Step 3

Else:

Go to Step 5

Else:

Go to Step 5

Step5:End program.

The Codes

```
#include <fstream>
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
using namespace std;
int fib(){
        fstreampaul;
        paul.open("ElijahProject.doc", ios::out | ios::app);
        doublei,a,n,f;
        stringfibChoice;
        vector<int> fib;
        cout<<"1. Generate first n numbers in the fibonacci series.\n2. Find the nth term in the fibonacci series.\nSelect your choice:";
        cin>>fibChoice;
        paul<<"\n1. Generate first n numbers in the fibonacci series.\n2. Find the nth term in the fibonacci series.\nSelect your choice: "<<fibChoice<<endl;
if(fibChoice=="1"){
                cout<<"Enter the amount of fibonacci numbers you want displayed: ";
                cin>>n;
                paul<<"\nEnter the amount of fibonacci numbers you want displayed: "<<n<<endl;
                fib.push_back(0);
                cout<<fib[0];
                fib.push_back(1);
                cout<<", "<<fib[1];
                paul<<fib[0]<<", "<<fib[1];
                for(i=0; i<=n-3;i++){
                        f = fib[i]+fib[i+1];
                        fib.push_back(f);
                        cout<<", "<<fib[i+2];
                        paul<<", "<<fib[i+2];

else if(fibChoice=="2"){
                cout<<"Enter the term number you want to find: ";
                cin>>n;
                paul<<"\nEnter the term number you want to find: "<<n<<endl;
                fib.push_back(0);
                fib.push_back(1);
                for(i=0; i<=n-3;i++){
                        f = fib[i]+fib[i+1];
                        fib.push_back(f);
                }
                cout<<"The "<<n<<"th term in the fibonacci series is
```

```
"<<fib[n-1]<<".";
                paul<<"\nThe "<<n<<"th term in the fibonacci series is "<<fib[n-1]<<"."<<endl;}
        else {return 0;}
        paul.close();} intsimeq(){
        fstreampaul;
        paul.open("ElijahProject.doc", ios::out | ios::app);
        double a,b,x1,x2,y1,y2,x,y;
        cout<<"\nEnter the coefficients of equation 1(x, y, and constant), each separated with a space: ";cin>>x1>>y1>>a; paul<<"\nEnter the coefficients of equation 1(x, y, and constant), each separated with a space: "<<x1<<" "<<y1<<" "<<a<<endl;  cout<<"Enter the coefficients of equation 2(x, y, and constant), each separated with a space: ";cin>>x2>>y2>>b;
paul<<"\nEnter the coefficients of equation 2(x, y, and constant), each separated with a space: "<<x2<<" "<<y2<<" "<<b<<endl;
        x = (y2*a - y1*b)/(x1*y2 - x2*y1);
        y = (-x2*a + x1*b)/(x1*y2 - x2*y1);
        cout<<"\nx = "<<x<<", y = "<<y<<".";
        paul<<"\nx = "<<x<<", y = "<<y<<"."<<endl;
        paul.close();
int fact(){
        fstreampaul;
        paul.open("ElijahProject.doc", ios::out | ios::app);
        inti,n,product;
        cout<<"\nEnter the factorial number: ";
        cin>>n;
        paul<<"\nEnter the factorial number: "<<n<<endl;
        product=1.0;
        for(i=n; i>0; i--){
product=product*i;
                .
                .
                .
}else{
                cout<<"Invalid entry.";
                paul<<"\nInvalid entry."<<endl;
                selection:
                cout<<"\nMake another engineering problem selection?(y/n) ";
                cin>>sel;
                paul<<"\nMake another engineering problem selection?(y/n) "<<sel<<endl;
                if(sel == "y"){
                        goto list }
                else if(sel == "n"){
                        cout<<"OK.";
                        paul<<"OK...\nEND OF PROGRAM.";
                        return 0;
        }else{
                        return 0;}} paul.close();
```

Flowcharts for the program

The Figures below show the flowchart for different solutions and instances of invalid entries **(Figures 1-7).**

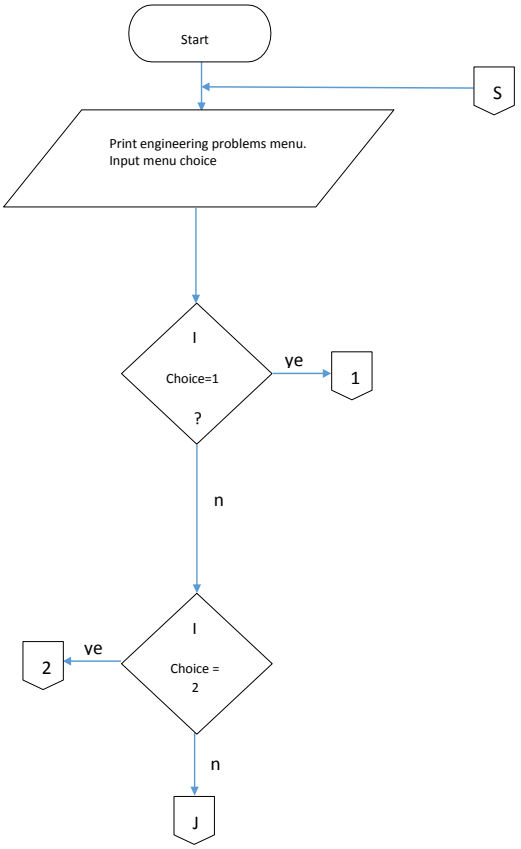

Figure 1: Main Menu and Fibonacci Series.

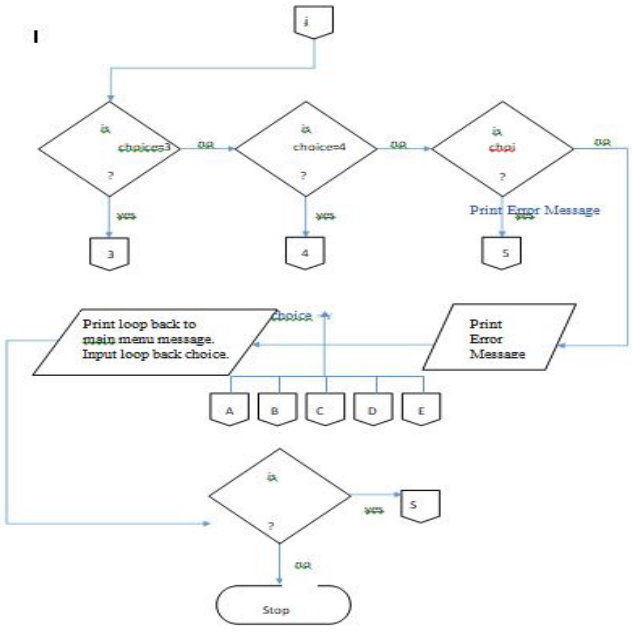

Figure 2: Main Menu and Simultaneous Equations Solutions.

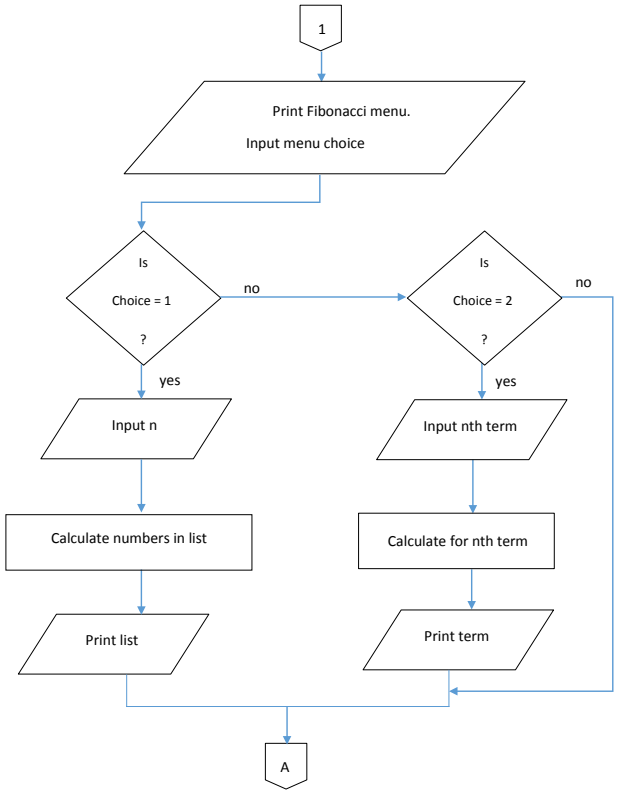

Figure 3: Main Menu and Factorial Solutions.

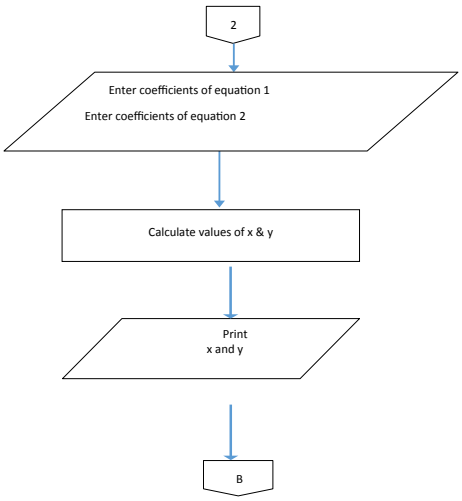

Figure 4: Main Menu and Cubic Polynomial Equations Solutions.

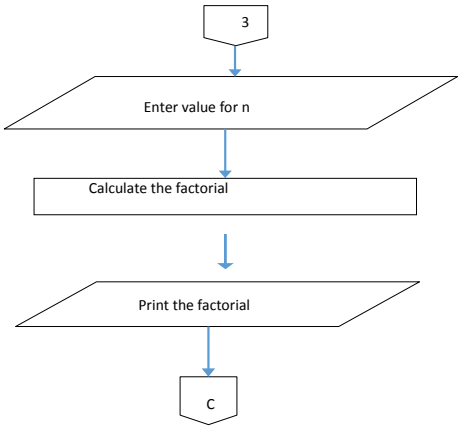

Figure 5: Main Menu and Pascal Triangle Solutions.

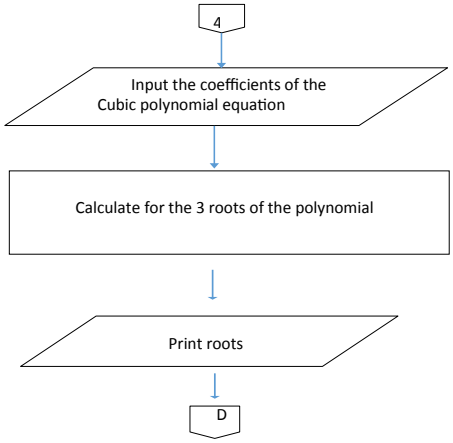

Figure 6: Main Menu and Instances of Invalid Entries.

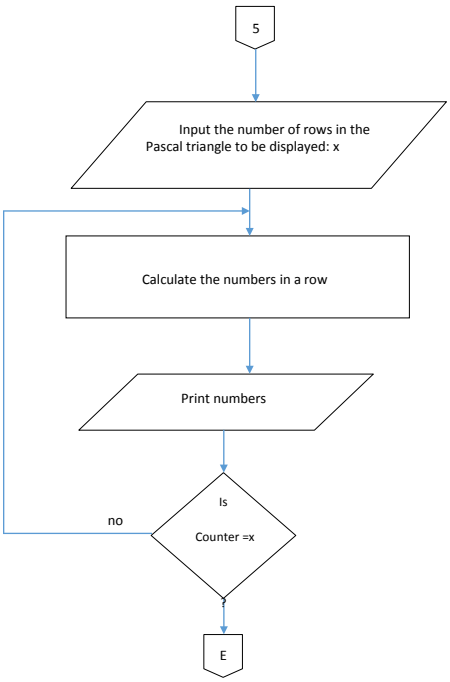

Figure 7: Main Menu and Instances of Invalid Entries.

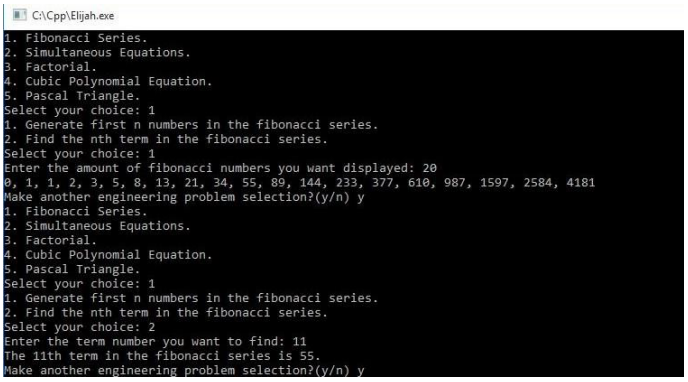

Figure 8: Plate1: A Screenshot of Fibonacci Series Solutions.

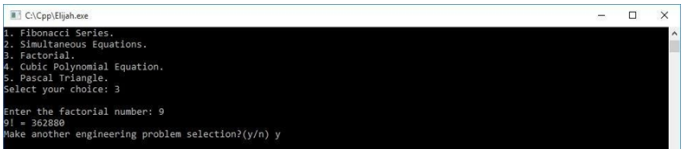

Figure 9: Plate2: A Screenshot of Simultaneous Equations Solutions.

Results and Discussion

The implementation of this system was done using the C++ software provided by the researchers. The program codes were written in a modular nature, where each mathematical problem was handled by a separate user-defined function. For the purpose of clarity, some parts of the console's output for the complete program along with sample solution as well as screen shots are respectively shown below:

1. Fibonacci Series.

2. Simultaneous Equations.

3. Factorial.

4. Cubic Polynomial Equation.

5. Pascal Triangle.

Select your choice: 1

1. Generate first n numbers in the fibonacci series.

2. Find the nth term in the fibonacci series.

Select your choice: 1

Enter the amount of fibonacci numbers you want displayed: 20

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181

Make another engineering problem selection?(y/n) y

1. Fibonacci Series.

2. Simultaneous Equations.

3. Factorial.

4. Cubic Polynomial Equation.

5. Pascal Triangle.

Select your choice: 1

1. Generate first n numbers in the fibonacci series.

2. Find the nth term in the fibonacci series.

Select your choice: 2

Enter the term number you want to find: 11

The 11th term in the fibonacci series is 55.

Make another engineering problem selection?(y/n) y

1. Fibonacci Series.

2. Simultaneous Equations.

3. Factorial.

4. Cubic Polynomial Equation.

5. Pascal Triangle.

Select your choice: 2

Enter the coefficients of equation 1(x, y, and constant), each separated with a space: 1 2 3

Enter the coefficients of equation 2(x, y, and constant), each separated with a space: 3 4 5

x = -1, y = 2.

Make another engineering problem selection?(y/n) y

.

.

.

Select your choice: a

Invalid entry.

Make another engineering problem selection?(y/n) n

OK...

END OF PROGRAM **(Figure 8 -13)**

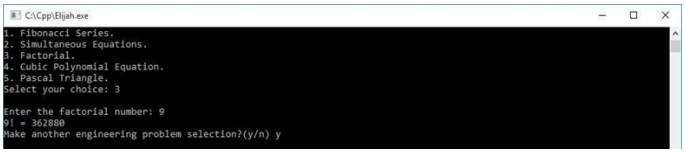

Figure 10: Plate3: A Screenshot of Factorial Solutions.

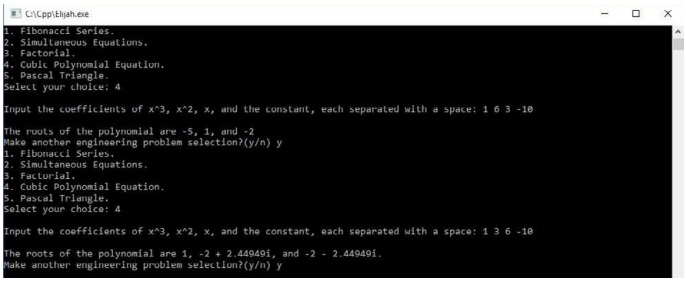

Figure 11: Plate4: A Screenshot of Cubic Polynomial Equations Solutions.

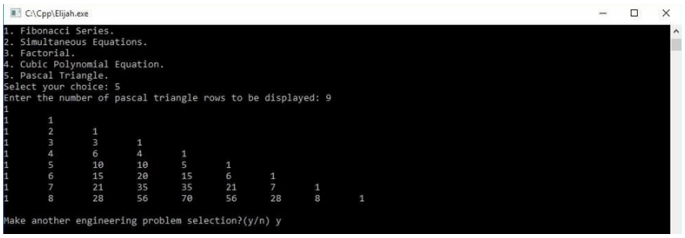

Figure 12 : Plate5: A Screenshot of Pascal Triangle Solutions.

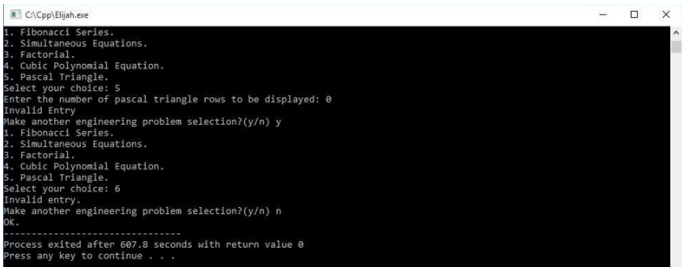

Figure 13: Plate6: A Screenshot of Instances of Invalid Entries.

Conclusion

With the completion of the program, it is obvious that the objectives have been met and that the development of a software for providing solutions to basic engineering problems was concluded successfully. The problem requirements were properly understood, as the program was adequately developed using the Dev-C++ software. The program was coded in a modular fashion, and was constructed in a manner that can fairly be understood. Parts of the algorithm, source code and file-stream outputs contained in the body of this article are provided to aid the reader in understanding the program flow.

This work substantially added to the respective area of programming and coding techniques. The programming solution to the engineering problems

was done successfully utilizing the Dev C++ which gives modern experts a flexible approach to anticipate conceivable coding dependability through time management for debugging errors. This investigation has given a novel structure to software development, which help:

i. minimizing the time of solving engineering problems.

ii. the use of each task/engineering problem designated to its unique function/ module gives a superior comprehensive evaluation of helping to eradicating difficulties in the coding activities that can incur more resources (such as time, money, and so on) to the programmer or organization (as the case may be).

The researchers therefore recommend that this work be used as guide to providing solutions to tens and hundreds of engineering problems. Also, other programming languages should explored.

References

1. Dooley, John F. "A Brief History of Cryptology and Cryptographic Algorithms." Springer Science & Business Media (2013).
2. Elijah P, Elijah E and Ojong E. "Design of Engineering Project Planning Software: A Case Study". IOSR Journal of Mechanical and Civil Engineerig (2020) 17 : 38-47.
3. Amini-Philip C and Elijah P."Impact of Information and Communication Technologies (ICTS) On Higher Education in Nigeria in the 21st Century". Journal of Humanities and Social Science (ISOR)(2019) 24: 837-845.
4. ChipperfieldA and Fleming P. " Genetic algorithm toolbox user's guide. Department of Automatic Control and System Engineering, University of Sheffield."((1994).
5. Cody R and Smith J. "Applied statistics and the SAS programming language".. Prentice Hall, fifth edition (2006).
6. Elijah Pand Etebu O." Project Planning Using Heuristics Approach: A Case Study of GAP International Limited".World Journal of Engineering Research and Technology (WJERT) (2019) 5:154-163.
7. Erwin K. "Advanced Engineering Mathematics". Wiley International edition, tenth edition. (2011).
8. Fehlberg G. "Low-order classical Runge-Kutta formulas with step size control and their application". (1977).

How to cite this article: Paul Tamaragaibi. "A Designed System for Providing Solutions to Basic Engineering Problems." *Ind Eng Manage* 10 (2021): 297.