Paper ID #23409

Structured Programing Methodology and Its Role in Cognitive Development in Problem Solving Skills

Prof. Omer Farook, Purdue University Northwest

Omer Farook is a member of the faculty of Electrical and Computer Engineering Technology at Purdue University, Nothwest. Farook received the diploma of licentiate in mechanical engineering and B.S.M.E. in 1970 and 1972, respectively. He further received B.S.E.E. and M.S.E.E. in 1978 and 1983, respectively, from Illinois Institute of Technology. Farook's current interests are in the areas of embedded system design, hardware-software interfacing, digital communication, networking, image processing, and biometrics, C++, Python, PHP and Java languages. He has a keen interest in pedagogy and instruction delivery methods related to distance learning. He has a deep commitment to social justice and in achieving economic and educational equity.

Dr. Jai P. Agrawal, Purdue University Northwest

Jai P. Agrawal is a professor in electrical and computer engineering technology at Purdue University, Calumet. He received his Ph.D. in electrical engineering from University of Illinois, Chicago, in 1991, dissertation in power electronics. He also received M.S. and B.S. degrees in electrical engineering from Indian Institute of Technology, Kanpur, India, in 1970 and 1968, respectively. His expertise includes analog and digital electronics design, power electronics, nanophotonics, and optical/wireless networking systems. He has designed several models of high frequency oscilloscopes and other electronic test and measuring instruments as an entrepreneur. He has delivered invited short courses in Penang, Malaysia and Singapore. He is also the author of a textbook in power electronics, published by Prentice-Hall, Inc. His professional career is equally divided in academia and industry. He has authored several research papers in IEEE journals and conferences. His current research is focused on renewable energy technology, smart energy grid.

Prof. Ashfaq Ahmed P.E., Purdue University Northwest

Ashfaq Ahmed is a Professor of Electrical and Computer Engineering Technology at Purdue University Northwest. Ahmed received his bachelor's of science degree in electrical engineering from the University of Karachi in 1973 and master's of applied science degree in 1978 from University of Waterloo. He is the author of a textbook on power electronics, published by Prentice-Hall. He is a registered Professional Engineer in the state of Indiana. He is a senior member of IEEE. Ahmed's current interests include embedded system design, electric vehicle, and VHDL design.

Dr. Wangling Yu, Purdue University Northwest

Dr. Wangling Yu is an assistant professor in the Electrical & Computer Engineering Technology Department of the Purdue University Northwest. He was a test engineer over 15 years, providing technical leadership in the certification, testing and evaluation of custom integrated security systems. He received his PhD degree in Electrical Engineering from the City University of New York in 1992, specializing in control theory and electronic technology.

Mr. Hassan Abdullah Alibrahim, Purdue University Northwest

I'm Hassan Alibrahim. A graduate teaching assistant at Purdue University Northwest Since August 2017. An active member in the national honor society for engineering technology, Tau Alpha Pi. Recognized as an outstanding student by the College of Technology at Purdue University Northwest for the 2015 -2016 academic year.

Dr. Ahmed S. Khan, Academic Platform, Lombard, IL

Dr. Ahmed S. Khan has more than thirty-five years of experience in research, instruction, curricula design and development, program evaluation and accreditation, management and supervision.

Dr. Khan received an MSEE from Michigan Technological University, an MBA from Keller Graduate School of Management, and his Ph.D. from Colorado State University. His research interests are in the areas of Nanotechnology, Fiber Optic Communications, Faculty Development, and Social and Ethical Implications of Technology. He is the author of many educational papers and presentations. He has authored/coauthored the following books:

• Nanotechnology: Ethical and Social Implications (2012) • Technology and Society: Issues for the 21st Century and Beyond 3E, (2008) • The Telecommunications Fact Book and Illustrated Dictionary 2E (2006) • Fiber Optic Communication: An Applied Approach, Prentice Hall, N.J. (2002) • Technology and Society: A Bridge to the 21st Century (2002) • Technology and Society: Crossroads to the 21st Century (1996) • Technology and Society: A Spectrum of Issues for the 21st Century (1994) • The Telecommunications Fact Book and Illustrated Dictionary (1992)

Dr. Khan is a senior member of the Institute of Electrical and Electronics Engineering (IEEE), and a member of American Society of Engineering Education (ASEE), and has been listed in Who's Who among America's Teachers. Dr. Khan also serves as a program evaluator for the Accreditation Board for Engineering and Technology (ABET).

Dr. Qudsia Tahmina, The Ohio State University

Dr. Qudsia Tahmina, The Ohio State University at Marion

Dr. Qudsia Tahmina is an Assistant Professor of Practice at The Ohio State University at Marion and teaches first year engineering courses.

Structured Programing Methodology and Its Role in Cognitive Development in Problem Solving Skills

Abstract

The paper expounds the practices utilized in teaching a two course sequence for the undergraduate curriculum, 1) Introductory C^{++} Software Design course and 2) An Embedded Systems Design course. This two course sequence is exclusively based on Structured Programing Methodology (SPM). The pedagogical underpinning for these courses is in strict adherence to the Structured Programing model, and is based on the interdependence among problem solving, cognition, and program (software design) development [1].

Presented is a learning model that these courses adhere to for the purpose of Problem Solving both in general and specific to Engineering and Technology. Cognitive skills are developed, honed, and enforced by practicing the SPM. The model aptly taps into the innate nature of C and C^{++} language syntax which requires every design application to begin and have a minimum of a single function. The SPM model relies on utilizing pseudo code design as the first step, as it is natural to human cognition and problem solving. This approach displays auto-morphism, as source and target are indistinguishable at several levels: 1) between pseudo code and language specific syntax code, 2) between software model and hardware model, 3) between physical model and virtual model in memory. The model specifically utilizes pointers structure exclusively for Inter Functional Data Communication.

As practiced in this SPM model, the authors begin with exploiting memory both directly and indirectly (with pointer). The remainder of the process is learning Program Flow Control language constructs and their appropriate usage. The model presents language constructs as an extension to physical hardware's attributes, thereby leading practitioners in the discipline of software hardware integration.

The paper serves as the pointer to fellow academicians for adopting this approach in their classrooms.

I. Introduction

The role of software in Electrical Engineering and Technology over the years has been increasing. Many of the disciplines of Electrical Engineering and Technology cannot be conceived without having a solid foundations in the discipline of software design. Our program focuses on two such courses in software design. The languages chosen for software design plays a crucial role in its application for the purpose of what it is used for. In our program we find C/C^{++} to be languages of choice for this purpose. The success of a program is to be the measure of how well the students it produces perform in the market place. C/C^{++} by design is meant to be both operating at low and high levels. Operating at low level, it is able to interface with physical hardware and memory. It operates data streams either with physical devices or with memory.

Operating at very high level it has the language constructs to deal in very abstract manner, thereby making virtual models of the physical objects. As practiced in our curriculum we are presenting a very novel pedagogical approach to software design, which is Functional Oriented view rather than Object Oriented view.

II. Learning Model Discussed is based on Structured Programing Methodology

Here the subject is briefly introduced for the purposes of relevance with the discussion at hand. Structured programing Methodology was proposed and is being practiced as a solution to the classical problems of spaghetti code. This provides economy, reusability, and security of code.

In short avoid 1) goto statements, 2) global variables and 3) monotony of huge formless code, instead 1) break code into well-defined tasks into functions, 2) replace goto statements with function calls, 3) use local variables, and 4) use inter-functional data communication with the pointers. Inter-functional data communication with the pointers provides autonomy to functions, without writing straight jacketed code specific to memory location references, instead it makes the functions more abstract and lends them versatility to operate without specific reference to specific memory location.

Structured programming Methodology as taught in our class and discussed here lends very well to embedded applications, hardware software interacting, control applications, robotics and DSP applications to name a few. Granted there is Object Oriented Programing (OOP) Methodology the choicest approach but that requires more course work in OOP, which many of Engineering / Technology students lack [1].

The authors model presented in the paper here takes a unique perspective, at looking at the Software Design (problem Solving) with a unique interpretive outlook for the C/C^{++} language construct in interpreting them from a utilitarian point of view. The utilitarian understanding of C / C^{++} constructs provides a systematic top down approach of software design. This provides a direct cognitive mapping in problems solving.

III. Interdependence among problem solving, cognition, and software design

Designers tend to use solution conjectures as the means of developing their understanding of the problem. Since ‘the problem’ cannot be fully understood in isolation from consideration of ‘the solution’, it is natural that solution conjectures should be used as a means of helping to explore and understand the problem formulation [2]. Based on scores of Protocols and other formal Studies of Design Activity and their impact on Human Cognition, the authors of the paper have developed a learning model that these courses adhere to for the purposes of Problem Solving both in general and specific to Engineering and Technology. Cognitive skills are developed, honed, and enforced by practicing the SPM.

IV. Problem Solving and Creativity

Every problem has its unique way to arrive at its solution. Many problems lend themselves to algorithmic approach and they are popular as they guarantee a correct solution. But all problems do not lend themselves to algorithmic approach that requires using heuristic techniques, such as working backwards or splitting the problem into sub-problems. So becoming familiar with multiple approaches increases the chances of success. (Expressed differently through the saying, "If your only tool is a hammer, you tend to treat every problem like a nail!"). Creativity has been defined as the new ideas that work. Creativity has elements of uniqueness associated with it. Software Application Design as a discipline is unique in providing so many ways to solve the problem. Ideas influenced from the article on, “The Cognitive Approach” [3].

V. Basic C/C^{++} Language Constructs and Key Words

The basic language constructs of C/C^{++}, provides the outermost shell with which the software designer be honing the cognitive skill sets. In their 1991 paper, Feuerstein and his colleagues note that cognitive operations may range from purely perceptual and reproductive ones, such as the operation involved in “recognition,” to more formal and abstract operations involved in inferential, inductive, and deductive reasoning. [4] The next eight items provides the software designer the operations involved in inferential, inductive, and deductive reasoning.

V.1. C++ keywords

The following are the reserved keywords in C++. Since they are used by the language, these keywords are not available for re-definition or overloading and they are case sensitive.

alignas (since C++11)	do	register(2)
alignof (since C++11)	double	reinterpret_cast
and	dynamic_cast	requires (since C++20)
and_eq	else	return
asm	enum	short
atomic_cancel (TM TS)	explicit	signed
atomic_commit (TM TS)	export(1)	sizeof(1)
atomic_noexcept (TM TS)	extern(1)	static
auto(1)	false	static_assert (since C++11)
bitand	float	static_cast
bitor	for	struct(1)
bool	friend	switch
break	goto	synchronized (TM TS)
case	if	template
catch	import (modules TS)	this
char	inline(1)	thread_local (since C++11)
char16_t (since C++11)	int	throw
char32_t (since C++11)	long	true
class(1)	module (modules TS)	try
compl	mutable(1)	typedef
concept (since C++20)	namespace	typeid

const	new	typename
constexpr (since C++11)	noexcept (since C++11)	union
const_cast	not	unsigned
continue	not_eq	using(1)
co_await (coroutines TS)	nullptr (since C++11)	virtual
co_return (coroutines TS)	operator	void
co_yield (coroutines TS)	or	volatile
decltype (since C++11)	or_eq	wchar_t
default(1)	private	while
delete(1)	protected	xor
	public	xor_eq

V.2. The main () Function.

The most basic and central to **C/C^{++}** language is the concept of a function. The main function is a unique function where the code execution starts. The open curly brace ({) indicates the beginning of main's function definition, and the closing brace (}) the end. Together they form the body of the main function.

V.3. The C/C^{++} Expression and Statement.

The statements forms the basic instructions to the underlying machine. The C/C++ syntax is case sensitive. They are terminated always by a semicolon (;). Every C/C++program comprises of Statements. Program executes statement by statement. Each statement generally, in turn, comprises of some expression. And an expression may further comprise of sub-expressions. An expression by itself is formed by a sequence of ***operators*** and their **operands** that specifies a computation. The blank line(s) are used for better readability.

V.4. Comments.

The C/C^{++} provides language constructs that are basically ignored by the compiler and are there for the benefits of humans and such have no bearing on the code designed and its execution. There are two such constructs1) // line comments; and /* multiple line comments */

V.5. Compound Statements.

The language construct of pair of curly brackets treats the number of enclosed statements as a single identifiable entity and treat them as a single block (unit).

V.6. The Preprocessor Directives.

The preprocessor directives are directives to the compilers. Whereas the rest of the code is translated to the specific underlying machine and are the instructions to the machine. They are executed at translation phase before the compilation. The result of preprocessing is to have a single file which is then passed and ready for compilation.

V.7. The Input Output Streams

This iostream class inherits all members from its two parent classes, istream and ostream, thus being able to perform both input and output operations. The class relies on a single streambuf object for both the input and output operations. This is not part of the language of $C^{++.}$ I/o streams could be envisioned as data stream that could flow in or out of the application under discourse. This is simply accomplished by including the preprocessor directive: #include <iostream>

V.8. The use of namespace std

All the objects of the standard library, and all the elements in the standard C++ library are declared within what is called a *namespace*: the namespace std . The most typical way to introduce visibility of these components is by means of *using declaration*:
using namespace std;

With the element discussed so far, the simple application that outputs a message is designed. The code is provided in Figure No 1.

```
// A very simple program in C++
#include <iostream>

int main ()
{
  std::cout << "Hello World! ";
  std::cout << "I'm a C++ program";

  char c1;
  std::cin>>c1;
}
```

Figure 1. A typical application to output three lines of code

In Figure No. 2. The code for the application that outputs the name and address of an individual’s is provided. The application outputs 4 lines.

In Figure No. 3. The code for the application that outputs the name and address of an individual is provided, "Structured Program". This application is the first example of Structured Programming, which demonstrates the declaration of function, calling of a function and the body of the function. This application also demonstrate Program flow control method of a function call.

In Figure No. 4. The code is provided for the application that is interactive program that is performing all the arithmetic operations on the data provided by the user. Non-structured example.

In Figure No. 5. The code is provided for the application that is interactive program that is performing all the arithmetic operations on the data provided by the user. Structured program example. The use of pointers for interfunctional communication.

In Figure No. 6. The code is provided for the application that is interactive program that is performing all the arithmetic operations on the data provided by the user. Structured program example, same application implemented with more functions. The use pointers for interfunctional communication.

```
/*
 Design an application that outputs your name and address
 Designed by: Omer Farook
 Date: January 12th 2017
*/
#include <iostream>
using namespace std;
int main ()
{
  cout<<"My name is:    Omer Farook"<< endl;
  cout<<"My address is: 420 Living Street\n";
  cout<<"               Hammond, IN"<<endl;
  cout<<"               46323 \n";
  return 0;
}
/*
 Typical Output:
 My name is:    Omer Farook
My address is: 420 Living Street
               Hammond, IN 46323
*/
```

Figure 2. A Typical application to output your name and address.

```
/*
Design an application that outputs your name and address
 "Structured Program"
 Designed by: Omer Farook
 Date: January 12th 2017
*/

#include <iostream>
using namespace std;

//1.Function Declaration
void my_address (void);
int main ()
{

//2. Calling the function

my_address();

  return 0;
}

//3. Body of the function

 void my_address (void)
 {
  cout<<"My name is:    Omer Farook"<< endl;
  cout<<"My address is: 420 Living Street\n";
  cout<<"               Hammond, IN"<<endl;
  cout<<"               46323 \n";

 }

 /*
 Typical Output:
 My name is:    Omer Farook
My address is: 420 Living Street
               Hammond, IN
               46323
*/
```

Figure 3. A Typical application to output your name and address "Structured Program"

```
/*
Lab 4: Design interactive (Use of input - output) application that outputs the
following:
The apllication deals with four numbers: defined by the user. The application
is to output the
(i) the addition of first and fouth numbers,
(ii) the subtraction of 4th and 3rd numbers,
(iii) the multiplication of first and fouth numbers,
(iv) the division of first and 2nd numbers,
(v) the modulo of first and 4th

 "Structured Program"
 Designed by: Omer Farook
 Date: January 19th 2017
*/
#include <iostream>
using namespace std;

//1.Function Declaration
void number_operation (void);
int main ()
{

//2. Calling the function
number_operation();

char c1;
cin>>c1;

  return 0;
}

//3. Body of the function
 void number_operation (void)
 {
 // variable declration and assignment
 int x1 = 0, x2 = 0, x3 = 0, x4 =0;
 int y1 = 0, y2 = 0, y3 = 0, y4 = 0, y5 =0;
 cout<<"Please provide the 2nd number \n";
 cin>> x2;
 cout<<"Please provide the 3rd number"<<endl;
 cin>> x3;
 cout<<"Please provide the 4th number"<<endl;
 cin>> x4;

 //manipulating in memory data with arithmetic operators
  y1 = x1 + x4;
  y2 = x4 - x3;
  y3 = x1 * x4;
  y4 = x1 / x2;
  y5 = x1 % x4;

  cout<<"Addition of: "<<x1<<'+'<<x4<<'='<<y1<<endl;
  cout<<"Subtraction of: "<<x4<<'-'<<x3<<'='<<y2<<endl;
  cout<<"Multiplication of: "<<x1<<'X'<<x4<<'='<<y3<<endl;
  cout<<"Division of: "<<x1<<"Division"<<x2<<'='<<y4<<endl;
  cout<<"Modulo of: "<<x1<<"Modulo"<<x4<<'='<<y5<<endl;
 }
 /*
 Typical Output:
 Please provide the first number
100
Please provide the 2nd number
50
Please provide the 3rd number
47
Please provide the 4th number
1000
Addition of: 100+1000=1100
Subtraction of: 1000-47=953
Multiplication of: 100X1000=100000
Division of: 100Division50=2
Modulo of: 100Modulo1000=100
 */
```

Figure 4. A Typical application to demonstrate an interactive application

```
/*
 Lab 5: Design interactive (Use of input - output) application
 that outputs the following:
 The apllication deals with four numbers: defined by the user.
 The application is to output the (i) the addition of first
 and fouth numbers, (ii) the subtraction of 4th and 3rd numbers,
(iii) the multiplication of first and fouth numbers,
(iv) the division of first and 2nd numbers,
(v) the modulo of first and 4th
 "Structured Program with pointers"
  "Interfunctional communinication with pointers"
 Designed by: Omer Farook
 Date: January 19th 2017
*/

#include <iostream>
using namespace std;

//1.Function Declaration
void getting_values(int *, int *, int *, int*);
void number_operation (int *, int *, int *, int*,int *, int *, int *, int *, int *);

int main ()
{
// variable declration and assignment
 int x1 = 0, x2 = 0, x3 = 0, x4 =0;
 int y1 = 0, y2 = 0, y3 = 0, y4 = 0, y5 =0;

 //2. Calling the functions
getting_values(&x1, &x2, &x3,&x4);
number_operation(&x1, &x2, &x3,&x4,&y1,&y2,&y3,&y4,&y5);

char c1;
cin>>c1;

  return 0;
}
```

```
//3. Body of the functions

void getting_values(int *p1, int *p2, int *p3, int *p4)
{
 cout<<"Please provide the first number"<<endl;
 cin>> *p1;
 cout<<"Please provide the 2nd number \n";
 cin>> *p2;
 cout<<"Please provide the 3rd number"<<endl;
 cin>> *p3;
 cout<<"Please provide the 4th number"<<endl;
 cin>> *p4;
}
 void number_operation (int *px1, int *px2, int *px3, int*px4,int *py1, int
*py2, int *py3, int*py4,int *py5)
 {
 //manipulating in memory data with arithmetic operators
  *py1 = *px1 + *px4;
  *py2 = *px4 - *px3;
  *py3 = *px1 * *px4;
  *py4 = *px1 / *px2;
  *py5 = *px1 % *px4;
  cout<<"Addition of: "<<*px1<<'+'<<*px4<<'='<<*py1<<endl;
  cout<<"Subtraction of: "<<*px4<<'-'<<*px3<<'='<<*py2<<endl;
  cout<<"Multiplication of: "<<*px1<<'X'<<*px4<<'='<<*py3<<endl;
  cout<<"Division of: "<<*px1<<"Division"<<*px2<<'='<<*py4<<endl;
  cout<<"Modulo of: "<<*px1<<"Modulo"<<*px4<<'='<<*py5<<endl;
 }
 /*
 Typical Output:
 Please provide the first number
400
Please provide the 2nd number
30
Please provide the 3rd number
5
Please provide the 4th number
60
Addition of: 400+60=460
Subtraction of: 60-5=55
Multiplication of: 400X60=24000
Division of: 400Division30=13
Modulo of: 400Modulo60=40
 */
```

Figure 5. Structured program example. The use pointers for interfunctional communication

```
/*
 Lab 6: Design an interactive (Use of input - output) application
 that outputs the following:
 The application deals with four numbers: defined by the user.
 The application is to output the (i) the addition of first and fourth numbers,
(ii) the subtraction of 4th and 3rd numbers,
(iii) the multiplication of first and fourth numbers,
(iv) the division of first and 2nd numbers,
(v) the modulo of first and 4th
 "Structured Program with pointers"
  "Interfunctional communication with pointers"
  Three functions 1)for input, 2) Computation, 3) outputting the results.
 Designed by: Omer Farook
 Date: January 24th 2017
*/

#include <iostream>
using namespace std;

//1.Function Declaration
void getting_values(int *, int *, int *, int*);
void number_operation (int *, int *, int *, int*,int *, int *, int *, int *, int *);
void my_output(int *, int *, int *, int*,int *, int *, int *, int *, int *);

int main ()
{
// variable declration and assignment
 int x1 = 0, x2 = 0, x3 = 0, x4 =0;
 int y1 = 0, y2 = 0, y3 = 0, y4 = 0, y5 =0;

 //2. Calling the functions
getting_values(&x1, &x2, &x3,&x4);
number_operation(&x1, &x2, &x3,&x4,&y1,&y2,&y3,&y4,&y5);
my_output(&x1, &x2, &x3,&x4,&y1,&y2,&y3,&y4,&y5);

char c1;
cin>>c1;

  return 0;
}
```

```
//3. Body of the functions

void getting_values(int *p1, int *p2, int *p3, int *p4)
{
cout<<"Please provide the first number"<<endl;
 cin>> *p1;

 cout<<"Please provide the 2nd number \n";
 cin>> *p2;

 cout<<"Please provide the 3rd number"<<endl;
 cin>> *p3;

 cout<<"Please provide the 4th number"<<endl;
 cin>> *p4;

}
 void number_operation (int *px1, int *px2, int *px3, int*px4,int *py1, int
*py2, int *py3, int*py4,int *py5)
 {

 //manipulating in memory data with arithmetic operators

  *py1 = *px1 + *px4;
  *py2 = *px4 - *px3;
  *py3 = *px1 * *px4;
  *py4 = *px1 / *px2;
  *py5 = *px1 % *px4;

 }
```

```
void my_output(int *ppx1, int *ppx2, int *ppx3, int*ppx4,int *ppy1, int
*ppy2, int *ppy3, int*ppy4,int *ppy5)
 {

  cout<<"Addition of: "<<*ppx1<<'+'<<*ppx4<<'='<<*ppy1<<endl;
  cout<<"Subtraction of: "<<*ppx4<<'-'<<*ppx3<<'='<<*ppy2<<endl;
  cout<<"Multiplication of: "<<*ppx1<<'X'<<*ppx4<<'='<<*ppy3<<endl;
  cout<<"Division of: "<<*ppx1<<"Division"<<*ppx2<<'='<<*ppy4<<endl;
  cout<<"Modulo of: "<<*ppx1<<"Modulo"<<*ppx4<<'='<<*ppy5<<endl;
 }



 /*
 Typical Output:
 Please provide the first number
200
Please provide the 2nd number
500
Please provide the 3rd number
40
Please provide the 4th number
55
Addition of: 200+55=255
Subtraction of: 55-40=15
Multiplication of: 200X55=11000
Division of: 200Division500=0
Modulo of: 200Modulo55=35
*/
```

Figure No. 6. Structured program example. The use of pointers for inter functional communication. More functions to implement.

VI. C/C^{++} Memory interfacing Language Constructs

The authors present the seven memory interfacing language constructs which provides the software designer the tools to model virtual map of the physical task or object at hand. This creates a virtual imagery with that of physical world. Other parameters or dimensions of the Cognitive Map which are clearly task focused include the content around which the task is centered, what Feuerstein calls the “universe of content” upon which the mental act is centered, or in common usage, with the subject matter. [5]

VI.1. A variable is a unique symbolic reference to a physical memory location that has a unique physical memory address. The teaching model starts with this definition of variable as such is first language constructs with which memory is accessed. The type of variable indicates how many bytes of memory it is referring to and hence determine the data size and data type. In short, a variable is a memory location that hold data.

Variables have to be declared.

Examples:
```
int x1;
float y1;
char c1;
```

VI.2. The **assignment operator** is the most important available to the application designer there by the data is placed in a variable.

Variables have to initialize with assignment operator.

Examples:
```
x1 = 111;
y1 = 25.75;
c1 = ‘z’;          //single character has to be enclosed within single quote.
```

The variables can be utilized from this point on to access or replace with other data.

Examples:
```
cout << x1 <<endl;     // 111
cout<< y1 << endl;     // 25.75
cout<< c1 <<endl;      // z
```

VI.3. A **Pointer,** is a special variable in that it can hold the address of other variable. Pointer has to be declared, it has to be initialized with the address of other variables. This is referred to as pointer pointing to. The variable whose address it is holding.

VI.4. The Pointer Operator *, is used to declare pointers

Examples:
```
int *p1;
float *q1;
char *r1;
```

VI.5. The Address of operator & is used to get the the address of a variable and then could be used in initializing the pointer.

Example:
```
p1 = &x1;
q1 = &y1;
r1 = &c1;
```

VI.6. The Indirectional or Dereferencing Operator *, is used when used with pointer to access the variable it is pointing to.

Example:

```
p1 = *x1;       // 111
q1 = *y1;       // 25.75
r1 = *c1;       // z
```

VI.7. An **Array, is** a series of Variables of the same type referred to by a common name placed in contiguous memory locations that can be individually (element of the array) referenced by adding an index to a unique identifier.

An array needs to be declared, initialized, and referred to it by its index number;

Example:

```
/*
 Lab_22
 "Generating random numbers for a particular range
  concept of Aray use to save the random numbers"

 The application will generate a set 100 numbers
 in the range of 23 - 300. Then the application is to
 reproduce all these numbers.

 Designed by: Omer Farook
 Date: March 9th 2017
*/
#include <iomanip>
#include <iostream>
#include <ctime>
#include <stdlib.h>

using namespace std;

//1.Function Declaration

void rand_gen(int *);
 void my_out(int *);

int main ()
{
// declaring an array.
 int aa1[100];
//calling function to populate this arry with random number.
rand_gen(aa1);
//2. Calling function to output the array
 my_out(aa1);

//next two lines are there for keeping output window open
char c1;
cin>>c1;

  return 0;
}

//3. Body of the function(s)
 //function having one single outpit with return statement.

 void rand_gen(int *a1)
 {
  // randomize();

  srand((unsigned)time(0));

  for (int i = 0; i < 100; i++)
            {
                        *a1 = (rand()% 278) +23;
                        a1++;
    }

 }

  void my_out(int *b1)
 {
 for (int ii = 0; ii < 100; ii++)
            {
             cout<<setw(5)<<*b1 << "   ";
              b1++;
            }
  }
```

```
/*
 Typical Output:
   44    257    270    262    167     75     50    253     46     27
  66     73    124    149    142    230    166    130    130    149
  242     48     84    165     61    157    141    126     67    113
  217    118    204    174    284    297    116    294    216    250
  245    140    111    226     61    284    120    168    114    284
   97    270    139    254    277     87    122    239    156    250
   83    152     98    141    205    137     62    236    212    278
   94    173    230    120    170    251    215    128    163    190
  271    107     61    255    201    144    191    141     40    266
  261    174     24     95     72    225     90    150     71     27
  */
```

Figure 7. Demonstration of Array

VI.h. A **data structure** is a group of variables, grouped together under one name. These data structure elements are termed as members, they could be of different types and different sizes. Data structures can be declared in C++ using the following syntax:

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
}
```

A data structure is a group of data elements grouped together under one name. These data elements, known as members, can have different types and different lengths. Data structures can be declared in C++ using the following syntax:

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} ;
struct type_name object_name_01, object_name_02 ;
```

Examples:

```
struct card
{
char title[30];
```

char author[20];
int catlog_number;
int year_published;
int number_of_copies;
};
Struct card c1, c2……………..;

VII. C/C^{++} Program Flow Control Language Constructs

C/C^{++} provides the following eight language constructs that alter the next statement (or compound statement's) execution or the order of execution. They are as follows. This process of analyzing the cognitive operations of a task is often called "task analysis." A mediator must be familiar with each of the key components (cognitive operations) required to carry out the process of task solution in order not only to mediate this process to the learner, but to encourage the learner to transfer that learning to new tasks which require similar cognitive operations[4]. The following eight elements of cognitive operations are required to carry out the process of software design.

VII. 1. Sequential execution of statement (line after line execution). This is provided by the language built in nature.

VII. 2. By a Function call. Whenever a function is called, the next execution shifts to the body of the function called.

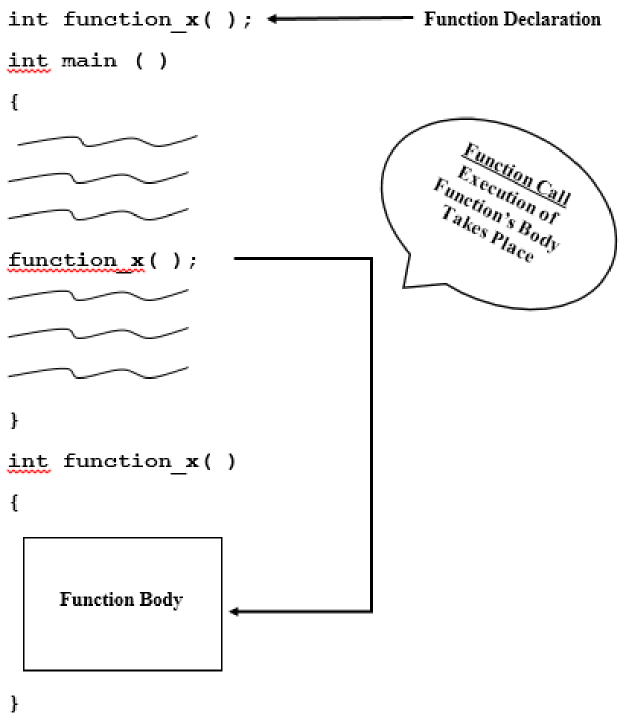


Figure 8. A Function Call

VII. 3. Use of if statement. Whenever there is a single statement or compound statement (a block) of code, and the question lies to either execute the code or skip it as a result of the condition then if statement is utilized.

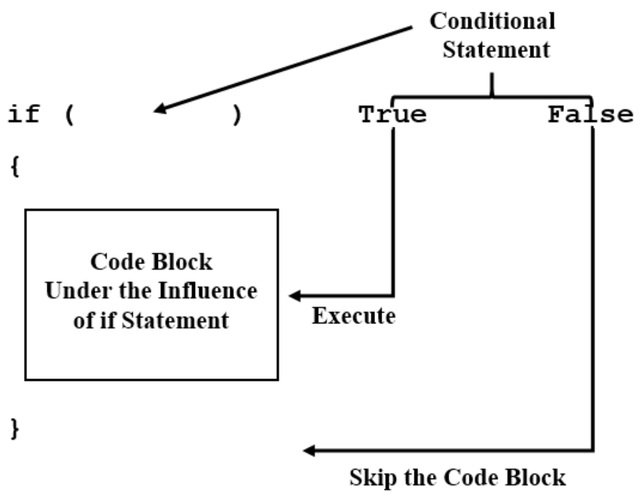


Figure 9. Use of if statement

VII. 4. Use of if – else statement. Whenever there are two blocks of code and the question lies between one or the other block being executed as a result of a condition under the binary output of the condition, then first block is being executed for condition being evaluated as true and the second block being executed for condition being false.

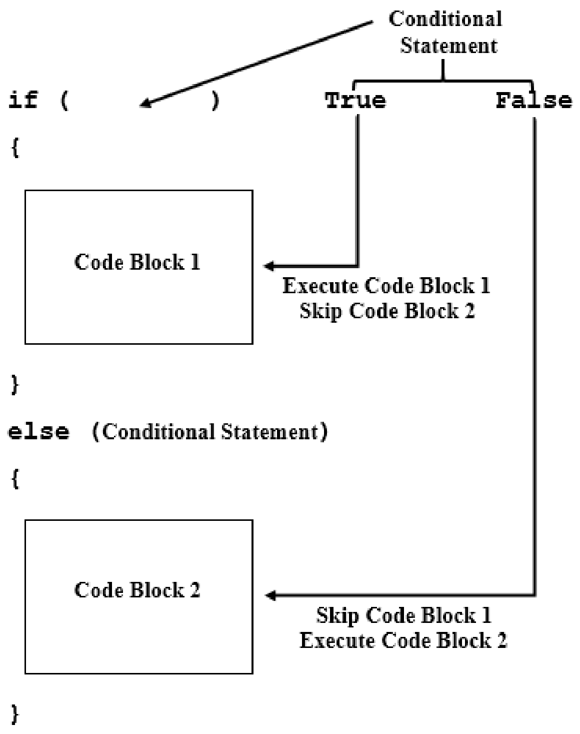
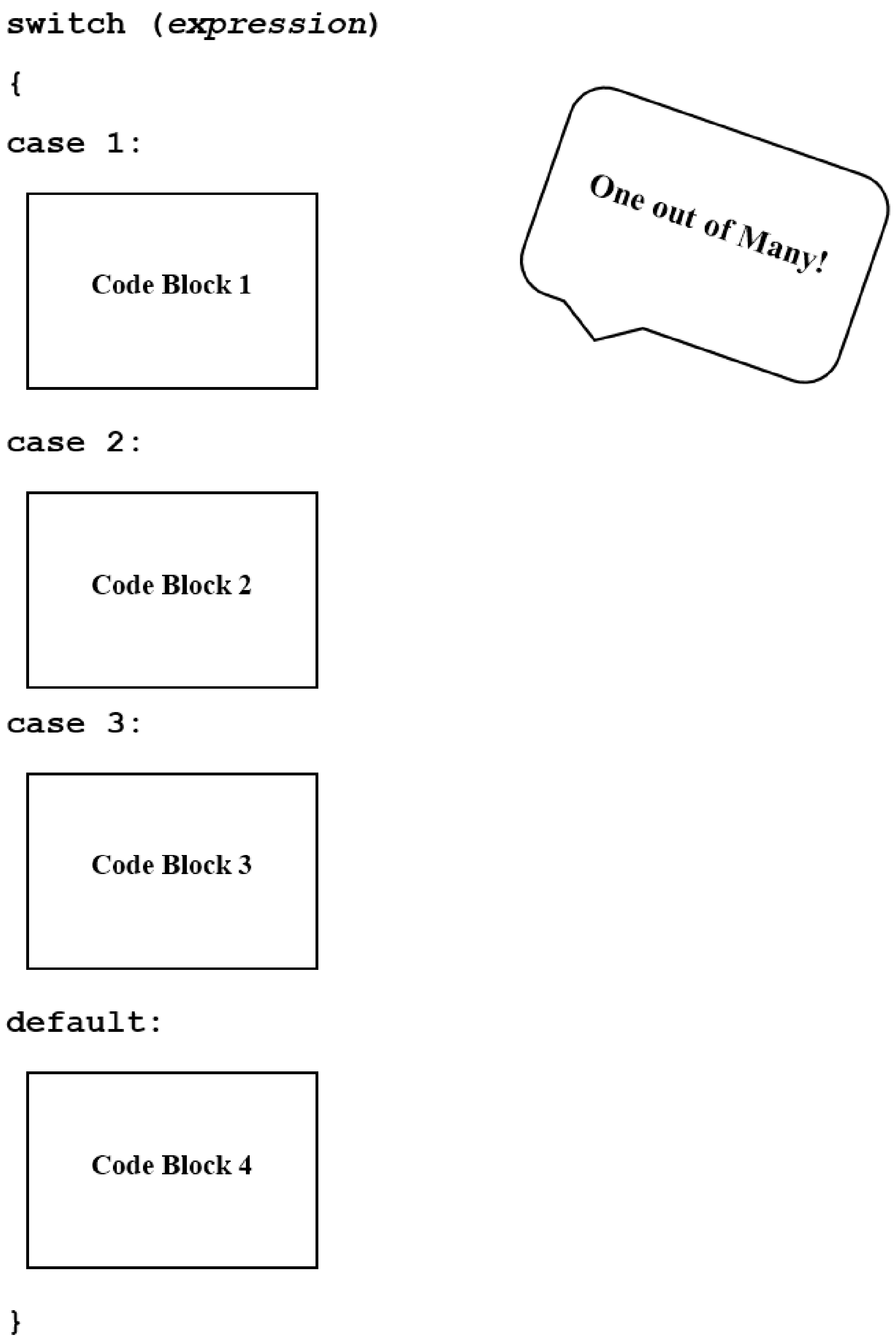


Figure 10. Use of if – else statement

VII. 5. The switch statement. If the decision is to execute one block among the many blocks as a result of expression being evaluated and tying a particular block with the value of the expression being evaluated. There is a default optional block to be executed in case of no matching outcome.



```
switch (expression)
{
case 1:
case 2:
case 3:
default:
}
```

Figure 11. Switch statement

VII. 6. For Loop. The most common of the loops for repeated execution or iteration of the loop (body of the loop). The iteration of the loop stops with the condition failure.

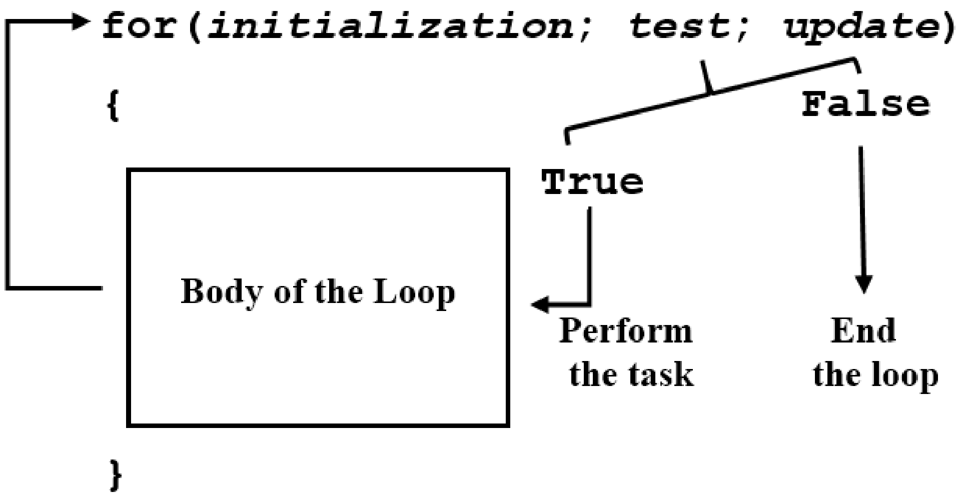


Figure 12. For Loop

VII. 7. While Loop. The loop with pre testing of the condition; loop iteration continues as long as condition is true.

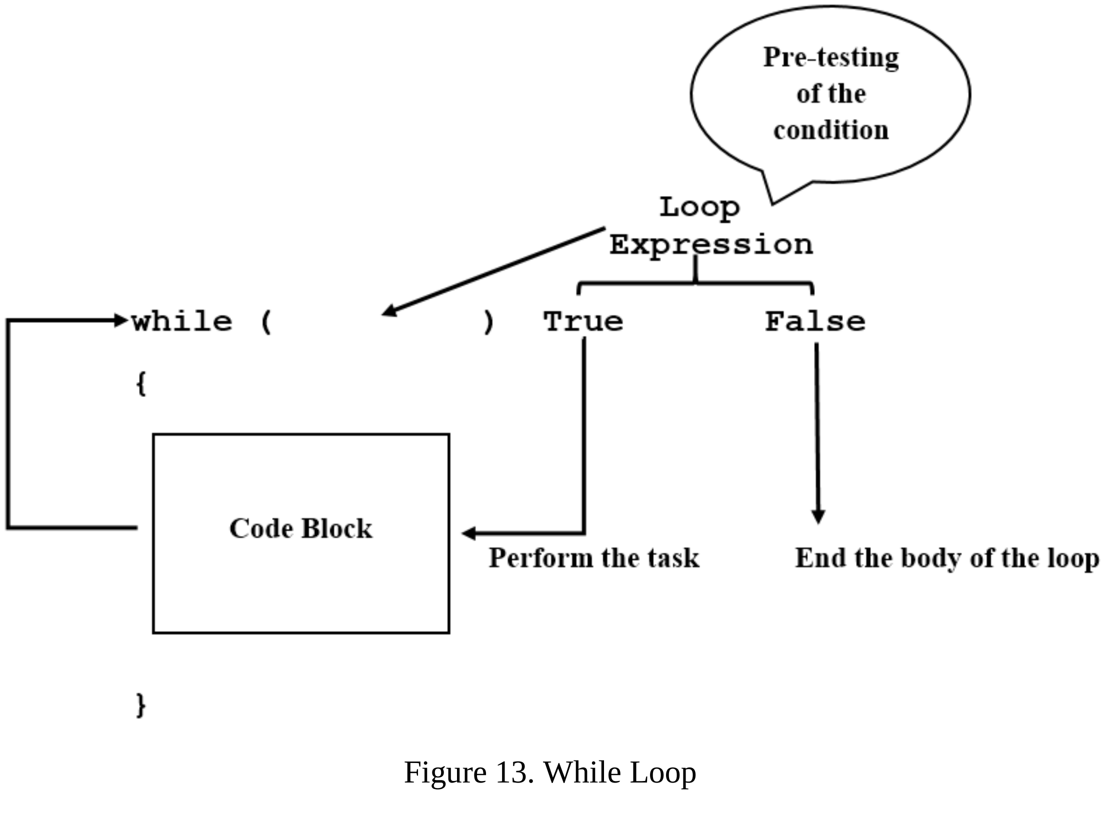


Figure 13. While Loop

VII. 8. Do – while Loop. The loop with pre testing of the condition guarantees at least a minimum of a single iteration.

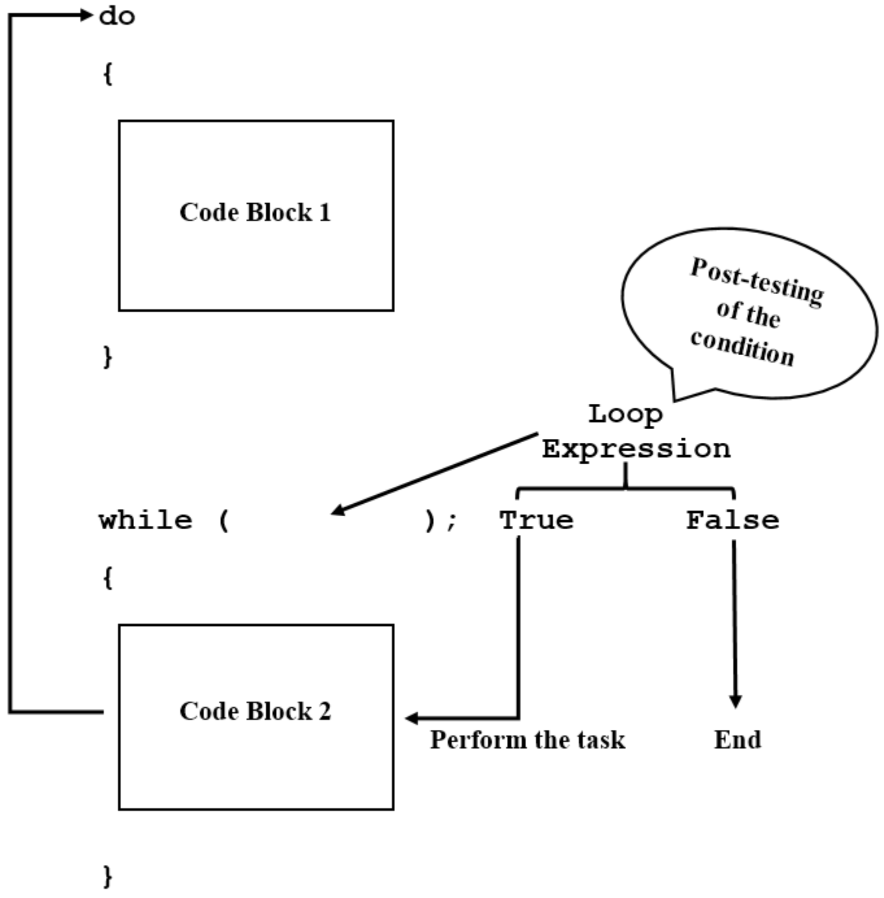


Figure 14. Do-while Loop

IX. Operators

The C/C^{++}, data operators are the very basic and most primitive of language constructs, and they are used as leg in the journey of software Application Design. They are simple but elegant as they provide single point to deviate, control and take decision to go forth in the program flow control which is critical cognitive outcomes.

Arithmetic operators
Bitwise operators
Assignment operator
Relational operators
Logical operators
Compound assignment operators
Increment and decrement operators
Subscript operator

X. Concluding Remarks

The authors have presented essentials of Structured Programing learning models that is Functional Oriented rather than Object Oriented for ease of implementation. The paper serves as a pointer to the fellow academicians to pursue and implement this in their respective classrooms. The paper tried to present with few examples as this is achieved from the onset, the application of pointers is used for Interfunctional Communication. Throughout the papers cognitive processing of problem solving that leads to software application design is discussed with reference to C/C^{++} language constructs, Program flow control elements, memory interfacing elements and Operators are discussed. The topics presented in the paper are in the same order as that of authors practices in the classroom. The authors have witnessed that subject matter of software design, when practiced in this way, keeps students engaged in this Top-down Design Approach, and will keep the cognitive process both agile and active.

KEYWORDS
Problem solving, Application (program) design and development, cognitive model, Cognition, Structured Programing, pseudo code design.

Bibliography

[1] Yu, W., & Farook, O., & Agrawal, J. P., & Ahmed, A. (2017, June), Board # 63 : Teaching Microcontrollers with Emphasis on Control Applications in the Undergraduate Engineering Technology Program Paper presented at 2017 ASEE Annual Conference & Exposition, Columbus, Ohio. https://peer.asee.org/27895

[2] Cross, Nigel (2001). Design cognition: results from protocol and other empirical studies of design activity. In: Eastman, C.; Newstatter, W. and McCracken, M. eds. Design knowing and learning: cognition in design education. Oxford, UK: Elsevier, pp. 79–103.

[3] The Cognitive Approach, https://www.ryerson.ca/~glassman/cognitiv.html, Last accessed, March 19th 2018

[4] https://www.encyclopedia.com/education/applied-and-social-sciences-magazines/cognitive-map-and-real-life-problem-solving, Last accessed, March 19th 2018

[5] On Feuerstein's Instrumental Enrichment: A Collection. Ben-Hur, Meir. ERIC Number: ED379069, Publication Date: 1994, ISBN: ISBN-0-932935-76