Comparative Analysis of Programming Methodologies of Java, C and Assembly Language Programming

Khandare Nikhil B

Assistant Professor, Manav School of Engineering Technology, Akola.

Abstract: This paper is organized into four section here we discuss the c, java and assembly language programming, First section is Introduction, Second is Literature Survey, In literature survey firstly we discuss c programming language based on decision statement, loop control, case control, functions, pointer, arrays, strings, structure and file handling in c. Secondly we come to java programming language and discuss data type, operators, Control Statements, Classes, inheritance, Packages and Interfaces, Exception handling, Thirdly in literature survey we discuss Assembly language on Data transfer instruction, Arithmetic, Logical, Shift, Rotate, Flag Control, Compare, Loop and loop handling instruction. In Third Section of paper we make comparative analysis of three programming language and finally the conclusion in fourth section

Index Terms: Programming, Instruction, Factorial, Arithmetic, String, Assembly language

1. INTRODUCTION

In today's world knowledge is the power, more knowledge you have, more powerful you become. Everything around us is computerized; life is online now right from banking, booking a ticket, shopping to ordering food everything is online. What makes these things easy is the biggest revolutionary device called computer, more specifically we can say program which runs on the computer. Any program which runs on computer is written by a programmer using some programming language like C, C++, Java, Oracle, Perl, Shell, Python, Assembly Language, Cobol etc. It is up to programmer which language to choose to write a program more specifically software, here in this paper we compare the aspects of three programming language C, Java and Assembly language.

2. LITERATURE SURVEY

In the literature Survey we discuss various attributes and contents of c programming language, Java and Assembly language programming

2.1 Programming in C: In this section we discuss various aspects of c programming language

2.1.1 Decision Control in C: for taking decisions C uses various conditional statements

2.1.1.1 If Statement

The general form of **if** statement looks like this

```
if ( Condition )
{
Statement1;
Statement2;
.
..
Statement N;
}
```

The keyword **if** tells the compiler that what follows is a decision control instruction. The condition following the keyword **if** is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed. If the condition is false then the statement is not executed

2.1.1.2 if-else Statement

The general form of If else statement looks like this:

```
if ( Condition )
{
Statement1;
Statement2;
.
.
Statement N;
}
else
{
{
Statement1;
Statement2;
.
.
Statement N;
}
```

The **if** statement by itself will execute a single statement, or a group of statements, when the condition following **if** evaluates to true. It does nothing when the expression evaluates to false, but in if Else statement, when the condition is false statement or group of statement in else part will execute

2.1.1.3 Nested If-else

It is perfectly all right if we write an entire **if-else** construct within either the body of the **if** statement or the body of an **else** statement. This is called 'nesting' of **if**s. This is shown in the following program.

```
main( )
{
int k ;
printf ( "Enter 65 or 66 " ) ;
scanf ( "%d", &k ) ;
if ( k == 65 )
printf ( "You Entered 65" ) ;
else
{
if ( k == 66 )
printf ( "You Entered 66" ) ;
else
printf ( "Wrong input" ) ;
}
}
```

2.1.1.4 Else If Statement

Here is program that will demonstrate use of else if Statements or else if ladder

```
Main()
{
int a,b,c,d,e;
printf("Enter any Number between 1 to 5")
canf("%d",&a);;
if(a==1)
printf("You Entered 1");
else if(a==2)
printf("You Entered 1");
else if(a==3)
printf("You Entered 3");
else if(a==4)
printf("You Entered 4");
else
printf("You Entered 1");
}
```

2.1.2 Loops Control Structure

In C programming language mainly three loop handling instructions

1. For Loop
2. While Loop
3. Do-While Loop

2.1.2.1 For Loop

If the sequence of instruction is to be executed again and again, we use loops most easy loop is for loop, General syntax of for loop is

```
For (Initialization; Condition; Inc/Dec)
{
Body of Loop (or statements)
}
```

Example:

```
For(i=0;i<10;i++)
{
Printf("Hello World");
}
```

This piece of code will print the statement Hello World ten times

2.1.2.2 While Loop

General syntax of for loop is

```
While (Condition)
{
Body of Loop(statements)
}
```

Example: Above Example can be written using while loop as

```
Main()
{
Int i=0
While(i<10)
        {
        Printf("Hello World");
        i++
        }
}
```

2.1.2.3 Do While Loop

General Syntax of Do While Loop is

```
Do
{
Body of Loop(statements)
}While(Condition)
```

Example: Above Program can be Written using do while

```
Main()
{
Int i=0;
Do
{
Printf("Hello World");
i++;
}while(i<10);
}
```

2.1.3 Case Control Structure

Decision Using Switch, Switch is control statement that allows us to make decisions from number of choices, General Formal of Switch is:

```
Switch (integer expression )
{
case constant 1 :
{
Statement 1;
Statement 2;
.
.
Statement N;
Break;
}
case constant 2 :
{
Statement 1;
Statement 2;
.
.
Statement N;
Break;
}
case constant 3 :
{
Statement 1;
Statement 2;
.
.
Statement N;
Break;

}
default :
{
Statement 1;
Statement 2;
.
.
Statement N;
Break;
}
}
```

Example of Switch:

```
Void main( )
{
Int a;
Printf("Enter a number between 1 to 5");
scanf("%d",&a);
switch (a)
{
case 1 :
{
printf ( " you entered 1\n" ) ;
break ;
}
case 2 :
{
printf ( "You entered 2 \n" ) ;
break ;
}
case 3 :
{
printf ( "You Entered 3 \n" ) ;
break ;
}
case 4 :
{
printf ( "You Entered 4 \n" ) ;
break ;
}
case 5 :
{
printf ( "You Entered 5 \n" ) ;
break ;
}
default :
{
printf ( "out of range \n" ) ;
}
}
}
```

2.1.4 Functions

C program is a collection of functions, Every time function is called control is passed to the calling function and when the closing brace of function is encountered the control is passed back to calling function

Example:

```
#include<stdio.h>
#include<conio.h>
void func(); //Declaration of function
void main( )
{
func( ) ; //Call to Function
}

func( )  //definition of Function
{
printf ( "\nCan't imagine life without C" ) ;
}
```

Mainly there are three important things about function

1. Function Declaration
2. Function Definition
3. Function Call

2.1.5 Pointers

Pointer in c are the variables which store addresses of other variable, Consider the following piece of code

```
main( )
{
int k = 7 ;
printf ( "Address of k = %u", &k ) ;
printf ( "\nValue of k = %d", k ) ;
printf ( "\nValue of k = %d", *( &i ) ) ;
}
```

Output:

```
Address of k =65525
Value of k=7
Value of k=7
```

We can also have variables to store addresses eg. In above code we can have a pointer variable to store address of k.

```
main( )
{
int k = 7 ;
int *j ;
j = &k ;
printf ( "\nAddress of k = %u", &k ) ;
printf ( "\nAddress of k = %u", j ) ;
printf ( "\nAddress of j = %u", &j ) ;
printf ( "\nValue of j = %u", j ) ;
printf ( "\nValue of k = %d", k ) ;
printf ( "\nValue of k = %d", *( &k ) ) ;
printf ( "\nValue of k = %d", *j ) ;
}
```

```
Address of k=65524
Address of k=65524
Address of  j=655222
Value of j=65524
Value of k=7
Value of k=7
Value of k=7
```

2.1.6 Arrays

2.1.6.1 One Dimensional arrays

If we have to find the average of 25 numbers we will not declare 25 integers instead we can use array of 25 integer and one integer to store sum and another for average. We can demonstrate the example as

```
main( )
{
int avg, s = 0 ;
int i ;
int a[25] ; /* array declaration */
for ( i = 0 ; i <25 ; i++ )
{
```

```
printf ( "Enter marks " ) ;
scanf ( "%d", &marks[i] ) ; /* store data in array */
}
for ( i = 0 ; i <25 ; i++ )
{
sum = sum + marks[i] ; /* read data from an array*/
}
avg = sum / 25 ;
printf ( "\nAverage marks = %d", avg ) ;
}
```

2.1.6.2 Multi-Dimensional Array

For Example we want to perform addition or subtraction of two matrices, we know that Rectangular Array of M*N numbers into M rows and N columns is Matrix, where M=no of rows and N= number of Column

Eg. Program to Perform Addition or subtraction of two Matrices can be done using two multi-dimensional arrays

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Int a[3][3],b[3][3],c[3][3],i,j;
Printf("Enter Elements of first matrix");
For(i=0;i<3;i++)
{
        For(j=0;j<3;j++)
        {
        Scanf("%d",&a[i][j]);
        }
}
Printf("Enter Elements of Second matrix");
For(i=0;i<3;i++)
{
        For(j=0;j<3;j++)
        {
        Scanf("%d",&b[i][j]);
        }
}
Printf("Addition/Subtraction of two matrices is");
For(i=0;i<3;i++)
{
        For(j=0;j<3;j++)
        {
        C[i][j]=a[i][j] +or- b[i][j];
        }
}

// print addition or subtraction
For(i=0;i<3;i++)
{
        For(j=0;j<3;j++)
        {
        printf("%d",c[i][j]);
        }
}
```

2.1.7 Strings

Array of characters is called a string, here we discuss various library function of strings

2.1.7.1 Strlen

This function counts the number of characters present in a string. Consider the following program.

```
main( )
{
char a[ ] = "Nikhil" ;
int l;
len1 = strlen ( a ) ;
printf ( " length = %d", l ) ;
}
Output: length=6
```

2.1.7.2 Strcpy

This function copies the contents of one string into another. The base addresses of the source and target strings should be supplied to this function

```
main( )
{
char a [ ] = "Nikhil" ;
char b[20] ;
strcpy ( b, a ) ;
printf ( "\nsource string = %s", a ) ;
printf ( "\ntarget string = %s", b ) ;
}
Source string=Nikhil
Target String=Nikhil
```

2.1.7.2 Strcat

This function concatenates the source string at the end of the target string.

```
main( )
{
char a[ ] = "Nikhil" ;
char b[30] = "Khandare" ;
strcat ( b,a ) ;
printf ( "\na = %s", a ) ;
printf ( "\nb = %s", b ) ;
}
Output: a=Nikhil
        b=KhandareNikhil
```

2.1.7.4 Strcmp

This is a function which compares two strings until mismatch is found or end of string is reached, function returns zero if the string are equal and returns the difference between ascii values of two strings

```
main( )
{
char a[ ] = "nik" ;
char b[ ] = "khan" ;
int i, j, k ;
i = strcmp ( a, "nik" ) ;
j = strcmp ( string1, string2 ) ;
printf ( "\n%d %d", i, j ) ;
}
Output :0 -32
```

2.1.8 Structures

Structures provide a way of storing many different values in variables of potentially different types under the same name If data about say 3 such magazines are to be stored, then we can follow two approaches

```
main( )
{
char a[3] ;
float p[3] ;
int pg[3], i ;
printf ( "\nEnter names, prices and no. of pages of 3 books\n" ) ;
for ( i = 0 ; i <= 2 ; i++ )
scanf ( "%c %f %d", &a[i], &price[i], &pg[i] );
}
```

2.1.9 File handling in c

Two main points to be discussed in file handling in c are Reading from File and writing to file

2.1.9.1 Reading from file

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  char ch, file_name[25];
  FILE *fp;
  printf("Enter the name of file you wish to see\n");
  gets(file_name);
  fp = fopen(file_name,"r"); // read mode
  if( fp == NULL )
  {
    perror("Error while opening the file.\n");
    exit(EXIT_FAILURE);
  }
  printf("The contents of %s file are :\n", file_name);
  while( ( ch = fgetc(fp) ) != EOF )
    printf("%c",ch);
  fclose(fp);
  return 0;
}
```

2.1.9.2 Writing to file

```
#include <stdio.h>
#include <stdlib.h>  /* For exit() function */
int main()
{
  char c[1000];
  FILE *fptr;
  fptr=fopen("program.txt","w");
  if(fptr==NULL){
    printf("Error!");
    exit(1);
  }
  printf("Enter a sentence:\n");
  gets(c);
  fprintf(fptr,"%s",c);
  fclose(fptr);
  return 0;
}
```

2.2 Programming in Java

2.2.1 Primitive Types

Java defines eight primitive types of data: byte, short, int, long, char, float, double, and Boolean. The primitive types are also commonly referred to as simple types, and both terms will be used in this book. These can be put in four groups:

2.2.1.1 Integers: this group includes byte, short, int, and long, which are for whole-valued signed numbers.

2.2.1.2 Floating-point numbers This group includes float and double, which represent
numbers with fractional precision

2.2.1.3 Characters This group includes char, which represents symbols in a character set,
like letters and numbers.

2.2.1.4 Boolean This group includes boolean, which is a special type for representing true/false values.

Arrays

An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions here is a program that creates an array of the number
of days in each month.

```
// Demonstrate a one-dimensional array.
class Array {
public static void main(String args[]) {
int month_days[];
month_days = new int[12];
month_days[0] = 31;
month_days[1] = 28;
month_days[2] = 31;
month_days[3] = 30;
month_days[4] = 31;
month_days[5] = 30;
month_days[6] = 31;
month_days[7] = 31;
month_days[8] = 30;
month_days[9] = 31;
month_days[10] = 30;
month_days[11] = 31;
System.out.println("April has " + month_days[3] + " days.");
}
}
```

Above piece of code will need jdk to execute. Commands will be Javac Filename.java followed by Java Filename

2.2.2 Operators : Java Supports various operators

2.2.2.1 Arithmetic Operators: Following Arithmetic Operators are supported by Java

+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

2.2.2.2 Relational Operators

The relational operators determine the relationship that one operand has to the other

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

2.2.2.3 The Bitwise Logical Operators Bitwise And, Bitwise Or, Bitwise Not, Bitwise Xor.

2.2.3 Control Statement

2.2.3.1 Javas Selection Statement

2.2.3.1.1The if statement is Java's conditional branch statement. It can be used to route program execution through two different paths. Here is the general form of the **if** statement:

```
if (condition) statement1;else statement2;
```

2.2.3.1.3 Nested ifs

A nested **if** is an **if** statement that is the target of another **if** or **else**. Nested **if**s are very common in programming.
Here is an example:

```
if(i == 10) {
if(j < 20) a = b;
if(k > 100) c = d; // this if is
else a = c; // associated with this else
}
else a = d;
```

2.2.3.1.4 The if-else-if Ladder

A common programming construct that is based upon a sequence of nested **if**s is the
if-else-if ladder. It looks like this:

```
if(condition)
statement;
else if(condition)
statement;
else if(condition)
statement;
...
else
statement;
```

2.2.3.1.5 switch

The switch statement is Java's multiway branch statement.

```
switch (expression) {
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
...
case valueN:
// statement sequence
break;
default:
// default statement sequence
}
```

2.2.3.2 Java's iteration Statement

Java supports various Iteration Statements like while, Do-While, For.

2.2.3.2.1 while

The while loop is Java's most fundamental loop statement. It repeats a statement or block while its controlling expression is true. Here is its general form:

```
while(condition) {
// body of loop
}
```

The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true.

2.2.3.2.2 do-while

The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Its general form is

```
do {
// body of loop
} while (condition);
```

Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression

2.2.3.2.3 For Loop

Here is the general form of the traditional for statement:

```
for(initialization; condition; iteration) {
// body
}
```

If only one statement is being repeated, there is no need for the curly braces

2.2.4 Classes

2.2.4.1 Class fundamentals

A simplified general form of a **class** definition is shown here:

```
class classname {
type instance-variable1;
type instance-variable2;
// ...
type instance-variableN;
type methodname1(parameter-list) {
// body of method
}
type methodname2(parameter-list) {
// body of method
}
// ...
type methodnameN(parameter-list) {
// body of method
}
}
```

The data, or variables, defined within a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class

2.2.4.2 Declaring Objects

```
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```

The first line declares mybox as a reference to an object of type Box. After this line executes, mybox contains the value null, which indicates that it does not yet point to an actual object

2.2.4.3 Introducing Methods

classes usually consist of two things: instance variables and methods. This is the general form of a method:

```
type name(parameter-list) {
// body of method
}
```

Here, type specifies the type of data returned by the method. This can be any valid type, including class types that you create. If the method does not return a value, its return type must be void. The name of the method is specified by name. The parameter-list is a sequence of type and identifier pairs separated by commas.

2.2.4.4 Constructors

A constructor initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes consider a sample program which will demonstrate class, method, constructor and objects

```
class Box {
double w;
double h;
double d;
.
Box() {//Constructor for the box
System.out.println("Constructing Box");
w = 10;
h = 10;
d = 10;
}
 // compute and return volume
double volume() {
return w * h * d;
}
}

class BoxDemo {
public static void main(String args[]) {
// declare, allocate, and initialize Box objects
Box mybox1 = new Box();
double vol;
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume is " + vol);
}
```

2.2.5 Inheritance

2.2.5.1 Inheritance Basic

To inherit a class, you simply incorporate the definition of one class into another by using the extends keyword.

```
// A simple example of inheritance.
// Create a superclass.
class A {
int i, j;
void showij() {
System.out.println("i and j: " + i + " " + j);
}
}
// Create a subclass by extending class A.
class B extends A {
int k;
void showk() {
System.out.println("k: " + k);
}
void sum() {
System.out.println("i+j+k: " + (i+j+k));
}
}
```

2.2.5.2 Using Super Keyword

Super has two general forms.

1. The first calls the superclass’ constructor.
2. The second is used to access a member of the superclass that has been hidden by a member of a subclass.

2.2.5.3 Creating Multilevel hierarchy

We can build hierarchies that contain as many layers of inheritance as you like. As mentioned, it is perfectly acceptable to use a subclass as a superclass of another. For example, given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. When this type of situation occurs, each subclass inherits all of the traits found in all of its superclasses. In this case, C inherits all aspects of B and A

2.2.5.4 Method Overriding

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass. When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass

```
// Method overriding.
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}
// display i and j
void show() {
System.out.println("i and j: " + i + " " + j);
}
}
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
// display k – this overrides show() in A
void show() {
System.out.println("k: " + k);
}
}
class Override {
public static void main(String args[]) {
B subOb = new B(1, 2, 3);
subOb.show(); // this calls show() in B
}
}
```

2.2.6 Exception Handling

2.2.6.1 Using Try and Catch

To guard against and handle a run-time error, simply enclose the code that you want to monitor inside a try block. Immediately following the try block, include a catch clause that specifies the exception type that you wish to catch. To illustrate how easily this can be done, the following program includes a try block and a catch clause that processes the ArithmeticException generated by the division-by-zero error:

```
class Exc2 {
public static void main(String args[])
{
int d, a;
try { // monitor a block of code.
d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
} catch (ArithmeticException e)
{ // catch divide-by-zero error
```

2.3 Assembly Language Programming

Assembly language Programming is completely dependent on following Instruction

2.3.1 Data Transfer Instruction

These groups of Instruction makes possible to move data around or inside microprocessor

2.3.1.1 MOV Instruction

Syntax is MOV D(Destination),S(Source)

This Instruction is used to transfer data from source to destination, Allowed operands of move instruction

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate

Example:

1. MOV AX,BX -this instruction will move the content of 16 bit source register BX to 16 Bit Destination Register AX
2. MOV AX,1234H –This Instruction will move source immediate data 1234H to destination register AX

2.3.1.2 XCHG (Exchange Data)

Syntax XCHG Destination, Source

Use: This instruction is used to swap the data. After executing this instruction destination and source will swap

Example: XCHG AL,BL

If AL=34H and BL=34H

After the execution of this instruction the contents of AL and BL will exchange

2.3.1.3 LEA

Syntax: LEA Destination Source

This Instruction is used to load offset of Source memory operand into one processor register

```
System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
}
```

This program generates the following output:

```
Division by zero.
After catch statement.
```

2.2.6.2 Throw, Throws and Finally

It is possible for your program to throw an exception explicitly, using the **throw** statement. The general form of throw is shown here: throw ThrowableInstance; If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. You do this by including a throws clause in the method’s declaration finally creates a block of code that will be executed after a try/catch block has completed and before the code following the try/catch block. The finally block will execute whether or not an exception is thrown

Example: Suppose in ALP one label ARR is used and we want to access offset value in that label then instruction LEA AX,ARR will perform the task

Instruction	Syntax	Use
LDS/LES	LDS/LES Destination, Source	First two byte of source register are copied into destination Register and next two byte are copied into corresponding segment register(DS/ES)
XLAT	XLAT	Converts the content of AL register into number stored in memory table, this instruction performs direct lookup technique which often used to convert one code to other
IN	IN Accumulator, Port	To read from Input port and store in the Accumulator
OUT	OUT Port, Accumulator	Reverse of In Instruction, Used to send 8 bit or 16 bit data to output port

2.3.2 Arithmetic Instruction: This Group of instruction is used to perform arithmetic operation

Instruction	Syntax	Use
ADD	ADD Destination, Source	Destination = Destination + Source
ADC	ADC Destination, Source	Destination = Destination + Source+ Carry
INC	INC Destination	Destination = Destination + 1
SUB	SUB Destination, Source	Destination = Destination - Source
SBB	SBB Destination, Source	Destination = Destination - Source- Carry
DEC	DEC Destination	Destination = Destination - 1
MUL	MUL Source	AL=AL*Source
IMUL	IMUL Source	Same as MUL operand are assumed to be signed numbers
DIV	DIV Source	AX=AX/source Quotient in AL Remainder in AH
IDIV	IDIV Source	Same as DIV operand are assumed to be signed numbers
NEG	NEG Destination	Used to find the 2’s

Instruction	Syntax	Use
		complement of number in Destination
CBW/CWD	Convert Byte to Word/ Convert Word to Double Word	Used to convert Byte to Word/ Convert Word to Double Word
DAA	DAA	Decimal Adjust after Addition used to convert result of addition to decimal
DAS	DAS	Decimal Adjust after Subtraction used to convert result of subtraction to decimal
AAA	AAA	ASCII Adjust after Addition used to convert result of addition to ASCII
AAS	AAS	ASCII Adjust after Subtraction used to convert result of subtraction to ASCII
AAM	AAM	ASCII Adjust after Multiplication used to convert result of Multiplication to ASCII
AAD	AAD	ASCII Adjust after Division used to convert result of Division to ASCII

2.3.3 Logical Instruction: This group of instruction is used to perform Boolean operation on binary data

Instruction	Syntax	Use
NOT	NOT Destination	Instruction finds the complement of binary data stored in destination operand
AND	AND Destination, Source	This Instruction Performs logical AND operation on destination and source operand
OR	OR Destination, Source	This Instruction Performs logical OR operation on destination and source operand
XOR	XOR Destination, Source	This Instruction Performs logical XOR operation on destination and source operand
TEST	TEST Destination, Source	This Instruction is used to examine the individual bit or group of bit of destination operand

2.3.4 Shift Instruction: this group of instruction is used to left shift or right shift bitwise content of processor register or memory location

Instruction	Syntax	Use
SHR	SHR Destination, Count	Shifts all bits in destination operands to right, LSB shifted out is found in carry flag
SAR	SAR Destination, Count	Shifts all bits in destination operands to right, MSB shifted from left side. LSB shifted out is found in carry flag
SHL/SAL	SHL/SAL Destination, Count	Instruction is similar to SHR and SAR, Destination bit will be shifted to left both SHL and SAL will give same result

2.3.5 Rotate instruction: This group of Instruction is used to rotate left or right content of processor register or memory location

Instruction	Syntax	Use
ROL	ROL Destination, Count	Bits get rotated out of MSB position are rotated back into LSB, copy of bit rotated out of MSB is placed in carry flag
ROR	ROR Destination, Count	Bits get rotated out of LSB position are rotated back into MSB, copy of bit rotated out of LSB is placed in carry flag
RCL	RCL Destination, Count	Similar to ROL but bits are rotated through carry
RCR	RCR Destination, Count	Similar to ROR but bits are rotated through carry

2.3.6 Flag Control Instruction: Used to make changes in specific bit of flag register

Instruction	Syntax	Use
LAHF	LAHF	Transfer rightmost 8 bit content of Flag register into AH
SAHF	SAHF	Transfer Content of AH to rightmost 8 bit of flag register
CLC	CLC	Used to clear Carry flag
STC	STC	Used to set carry flag
CMC	CMC	Used to complement carry flag
CLD	CLD	Used to clear direction flag
STD	STD	Used to set direction flag
CLI	CLI	Used to clear interrupt flag
STI	STI	Used to set interrupt flag

2.3.7 Compare Instruction

Instruction	Syntax	Use
CMP	CMP Destination, Source	Perform comparison of one byte or word data, internally it performs subtraction of source operand from destination operand

2.3.8 Loop and Loop Handling Instruction: Instruction are specially designed to implement loop operation

Instruction	Syntax	Use
LOOP	LOOP <Short-Label>	CX=CX-1 If CX!=0 then jump to target address
LOOPE/LOOPZ	LOOPE/LOOPZ <Short-Label>	CX=CX-1 If CX!=0 && ZF=1then jump to target address
LOOPNE/LOOPNZ	LOOPNE/LOOPNZ <Short-Label>	CX=CX-1 If CX!=0 && ZF=0then jump to target address
JCXZ	JCXZ <Short-Label>	Jump if CX=0

3. COMPARISON ON THE BASIS OF VARIOUS ISSUES

1. We will examine the methodology of programming in different languages; consider a program to find the factorial of the number N in C, Java and ALP

C Program	Java Program	Assembly language program

#include<stdio.h>
#include<conio.h>
void main(){
int i,f=1,num;
 	
printf("Enter a number: ");
scanf("%d",&num);
 	
for(i=1;i<=num;i++)
f=f*i;
 	
printf("Factorial of %d is: %d",num,f);
getch();
}
	import java.util.Scanner;
	class Factorial
	{
	public static void main(String args[]) {
	int n, c, fact = 1;
	System.out.println("Enter an integer to calculate it's factorial");
	Scanner in = new Scanner(System.in);
	n = in.nextInt();
	if (n < 0)
	System.out.println("Number should be non-negative.");
	else {
	for (c = 1 ; c <= n ; c++)
	fact = fact*c;
	System.out.println("Factorial of "+n+" is = "+fact);
	} }}
		MOV AL,N
		MOV BL,AL
		XYZ: DEC BL
		JZ EXIT
		MUL BL
		JMP XYZ
		EXIT:MOV[FACT],AL

For loop is used to calculate factorial of a number in above piece of code, from 1 to number	For loop is used to calculate factorial of a number in above piece of code, from 1 to number, but everything is written inside class	Loops can be used but jump to a label is used here to calculate factorial of a number

3.2 Addition, Subtraction, Multiplication and division (arithmetic operation)of two numbers in C, Java and ALP

C code	**Java Code**	**ALP**

#include<stdio.h>
Void main()
{
Int a,b,c,d,e,f;
Printf("Enter 2 numbers");
Scanf("%d %d",&a,&b);
C=a+b;
D=a-b;
E=a*b;
F=a/b;
Printf("addn,sub,mul,div is %d %d %d %d",c,d,e,f);
}
	class BasicMath {
	public static void main(String args[]) {
	// arithmetic using integers
	System.out.println("Integer Arithmetic");
	int a = 1 + 1;
	int b = a * 3;
	int c = b / 4;
	int d = c - a;
	int e = -d;
	System.out.println("a = " + a);
	System.out.println("b = " + b);
	System.out.println("c = " + c);
	System.out.println("d = " + d);
	System.out.println("e = " + e);
		Addition
		MOV AX,0012H
		MOV BX,0003H
		ADD AX,BX
		SUB AX,BX
		MUL BX
		DIV BX

Here addition is stored in c, subtraction in d, multiplication in e, division in f	Here addition is stored in a, subtraction in d, multiplication in b, division in c, negation in e	Here register AX is loaded with value 12H and BX with value 03H Addition, subtraction, multiplication and division all re stored in AX (accumulator)

3.3 Use of pointers distinguished the three programming methodologies in c pointers are used, but in java there are no pointers and in assembly language we play directly with the memory but this is not specifically mentioned that assembly language uses pointers

C	Java	ALP

Here pointers are the variable that store the address/ location of another variable
 	
Consider the following code
#include<stdio.h>
Void main()
{
int a,,b,c;
a=7;
int *p;
p=&a;
//here p is pointer
Printf("value of a is %d",a);
Printf("address of a is %u",&a);
Printf("address of a is %u",p);
Printf("value of a is %d",*(&a));
Printf("value of a is %d",*p);
}
	There are No pointers in java	We save information directly in
		1. CPU registers
		2. Memory location
		
		But there are no variables which store the address of another variable
		
		We can store memory location into registers.

3.4 Graphical User Interface can be developed easily and perfectly in java using applets and abstract window toolkit, In c we can develop GUI using graphics library but it is very difficult in c, In ALP printing a message on screen needs tens of lines of code developing a nice GUI is practically impossible in assembly language.

3.5 Depending upon the Jump Statements and loops used in the programming language we can further distinguish the languages

C	Java	ALP
While Do-While For Break continue	While Do-while For For-Each Break Continue	LOOP LOOPE LOOPZ LOOPNE LOOPNZ JMP JE/JZ JNE/JNZ

3.6 Each language has its string handling function

C	Java	ALP

Strlen() – to find length
Strcmp() – to compare string
Strcat() – to concatenate
Strcpy() – to copy string
Strrev() – to reverse string
	Equals()
	Startswith()
	Endswith()
	Compareto()
	Substring()
	Concat()
		MOVSB/MPVSW – to move
		CMPSB/CMPSW – to compare
		SCASB/SCASW – to scan
		LODSB/LODSW- to load
		STOSB/STOSW – to store

3.7 Comparison on the basis of logical instruction: No matter where you go, which school you study, which university you attend, which book you read the following table will not change at all

A	B	A | B	A & B	A xor B	~ A	~(a&b)	~(a|b)
0	0	0	0	0	1	1	1
0	1	1	0	1	1	1	0
1	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0

The change is in the way these instructions are used in the programming languages. Suppose we have to perform AND operation on data, c will use (a&&b), whereas ALP will use AND AX,BX and java will use a&b, results will not differ but the way of using differs a lot.

4. Conclusion

We have discussed difference between the programming methodologies of three programming languages on different issues, We also took specific example of finding a factorial of a number in three programming languages. Coding was done to perform arithmetic operations in c, java and alp, pointers contributed significantly to the discussion of comparison of the methodologies. Array of characters ie. Strings are used in all three programming languages but library functions and their use shows significant difference, similarly for GUI Java is perfectly suitable for GUI, in c its difficult in ALP it's nearly impossible. If one language is good at one place it has flaws at other place, one liner or a closing line to this is each programming methodology is perfect at its own place and their beauty comes out depending upon the application.

References

[1] Herbert Schildt ”The Complete reference Java Seventh Edition”, Tata McGRAW-HILL publicationISBN-13:978-0-07-063677-4

[2] Yashwant Kanetkar- Let us C BPB publication ISBN 978-81-8333-163-0

[3] Pankaj Jalote, “An Integrated Approach to Software Engineering”, Springer Science Business Media, Inc, Third Edition, 2005.

[4] Grady Booch, “Object-Oriented Analysis and Design with applications”, Addison Wesley Longman, Inc, second Edition, 1998.

[5] Roger S. Pressman, “Software Engineering a practitioner‟s approach”, McGraw-Hill, 5th edition, 2001.

[6] Kernighan and Ritchie The C Ansi C Programming language ISBN:978-81-203-0596-0

[7] Barry B. Brey, “The Intel Microprocessors” ISBN-13: 978-0135026458 ISBN-10: 0135026458

Author

Khandare Nikhil B. has Batcher of technology degree in Information Technology from College of Engineering, Pune (COEP) and also has Master of Technology Degree in Computer Engineering from Veermata Jijabai Technological Institute, Mumbai (VJTI), Presently Working as a Assistant Professor at Manav School of Engineering Technology, Akola.