IJCSE International Journal of Computer Sciences and Engineering Open Access

Research Paper Vol.-7, Issue-6, June 2019 E-ISSN: 2347-2693

A Comparison of while, do-while and for loop in C programming language based on Assembly Code Generation

J. Makhijani[1*], M.Niranjan [2], Y. Sharma [3]

[1] Department of Computer Science & Engineering, Rustamji Institute of Technology, Tekanpur, Gwalior, India
[2] Department of Computer Applications, Rustamji Institute of Technology, Tekanpur, Gwalior, India
[3] Department of Computer Science & Engineering, Rustamji Institute of Technology, Tekanpur, Gwalior, India

**Corresponding Author: j_makhijani@yahoo.com*

DOI: https://doi.org/10.26438/ijcse/v7i6.12041211 | Available online at: www.ijcseonline.org

Accepted: 17/Jun/2019, Published: 30/Jun/2019

***Abstract*—** C is a programming language which is the most powerful and useful language ever for the programmers and developers. Like all the modern programming languages, the C language also has many control statements out of which five are iterative statements, i.e., while, do-while and for. These three statements are meant for use in same conditions, but which one is better. The performance of these three statement does not compared by the novice programmers. So, a basic knowledge of performance of these loops should be there. This comparison will guide novice programmers to use these loops efficiently. The comparison can be done by counting the execution time but that can depend on other factors also like CPU usage by other programs or services etc. But a very efficient way to compare is the comparison of Assembly Instruction generated by a program, so here we are presenting a comparison based on the assembly code generation by each loop which can be seen by the object file created just after the compilation.

***Keywords*—** while, do-while, for, assembly code

I. Introduction

C is a programming language which is even popular from more than two decades. It is a procedural programming language which provides so many features. It has low-level access of memory, it has very simple keywords and it also has very clean programming style. These features makes it very useful and popular.

A very useful feature of every programming language is control statements. A control statement specifies the flow of program control or it can be said that which instruction should be executed and which should not be. Control statements makes possible to make decision for execution of one or more statements, it make decision to perform task repeatedly, and it make decision to jump from one statement to other statement.

C programming language has four types of control statements, Decision making statements, selection statements, iteration statements and jump statements. The if-else statement, nested-if statement are the decision making statement. The switch-case statement is the selection control statement. The while statement, do-while statement and for statement are iteration statements. The goto, break, continue and return statements are the jump statements.

The documentation and examples of these control statements are easily available, but the comparison in these statements (in same category) is not available in detail. The learners and programmers does not compare them, so use any one of them randomly or as per their compatibility. In this research paper, we are going to give a comparison of iteration control statements, i.e., while statement, do-while statement and for statement.

II. Introduction to while Statement

While statement is the most basic iteration control statement of c programming language. The while statement has a conditional expression which decides the execution of statements which are given in while block. If condition is true, the while-block-statements get executed otherwise not. Basic syntax of while is as [2]:

```
        while(conditional expression)
        {
                one or more statements;
        }
```

The important thing in this statement is that if condition found false in first iteration, then statements given in while-block will not executed even once [1].

III. Introduction to do-while Statement

Do-while statement is little different from while-statement. Like while statement, the do-while statement also has a conditional expression (which are given in while block) which decides the execution of statements. If condition is true, the while-block-statements get executed otherwise not. The difference from while block is that do-while has conditional expression in the end of statement where while statement has it in the start. Basic syntax of do-while is as [2]:

```
do
{
        one or more statements;
} while(conditional expression);
```

The important thing in this statement is that if condition found false in first iteration, then also the statements will execute once. This assure executions of statements at-least once [1].

IV. Introduction to for Statement

For statement is very similar to while statement in behavior. This statement also has a conditional statement which is responsible for the execution of statements under for block. If statement is true, then statement gets executed otherwise not. Basic syntax of for statement is as under [2]:

```
for(expression 1, expression 2, expression 3)
        {
                one or more statements;
        }
```

Here expression 1 is to initialize the control variable, expression 2 is the conditional expression and expression 3 is the control variable’s value modifier statement or to increment/decrement in control variable’s value.

Although all these statements are meant for similar work, i.e. repetition of one or more statement, the performance of them may be different. Our aim is to analyze the workability of these statements [1].

V. Code for comparison

Here we are taking three types of code for each loop. First which is just printing “Hello World!”. No arithmetic or logical calculation is here. Second one is with arithmetic expression, so we are using the general logic of printing the table of *n* and *n* is provided by user at runtime and the third one is with nesting of loop where we are providing the code for sorting of 10 numbers. Here these 10 numbers are static, i.e., provided in program. The code with all three loops is as under:

A. Code for Case-1 (print “Hello World!” ten times)

While loop

```
#include<stdio.h>
int main(){
        int i=0;
        while(i<10)     {
                printf("Hello World!\n");
                i++;
        } return 0;
}
```

Do-while loop

```
#include<stdio.h>
int main(){
        int i=0;
        do      {
                printf("Hello World!\n");
                i++;
        }while(i<10);
        return 0;
}
```

For loop

```
#include<stdio.h>
int main(){
        int i=0;
        for(i=0;i<10;i++) {
                printf("Hello World!\n");
        }
        return 0;
}
```

B. Code for Case-2 (loop with arithmetic operation, i.e., printing the table)

While loop

```
# include<stdio.h>
int main(){
        int i=1,n=0;
        printf("Enter  number : ");
        scanf("%d",&n);
        while(i<=10)    {
                printf("%d * %d = %d\n",n,i,n*i);
                i++;
        }       return 0;
}
```

Do-while loop

```
#include<stdio.h>
int main(){
        int i=1,n=0;
        printf("Enter  number : ");
        scanf("%d",&n);
        do      {
                printf("%d * %d = %d\n",n,i,n*i);
                i++;
        }while(i<=10);
        return 0;
}
```

For loop

```
#include<stdio.h>
int main(){
        int i=1,n=0;
        printf("Enter  number : ");
        scanf("%d",&n);
        for(i=1;i<=10;i++)      {
                printf("%d * %d = %d\n",n,i,n*i);
        }
        return 0;
}
```

C. *Code for Case-3 (nesting of loop for sorting of 10 numbers)*

While loop

```
#include<stdio.h>
int main(){
        int i=0,j=0,n[]={4,2,5,8,1,10,9,6,7,3},temp;
        while(i<9)      {
                j=i;
                while(j<10)             {
                        if(n[i]>n[j])   {
                                temp=n[i];
                                n[i]=n[j];
                                n[j]=temp;
                        }       j++;
                }       i++;
        }
        i=0;
        while(i<10)     {
                printf("%d\n",n[i]);
                i++;
        }return 0;
}
```

Do-while loop

```
#include<stdio.h>
int main(){
        int i=0,j=0,n[]={4,2,5,8,1,10,9,6,7,3},temp;
        do      {
                j=i;
                do              {
                        if(n[i]>n[j])   {
                                temp=n[i];
                                n[i]=n[j];
                                n[j]=temp;
                        }j++;
                }while(j<10);
                i++;
        }while(i<9);
        i=0;
        do      {
                printf("%d\n",n[i]);
                i++;
        }while(i<10);
        return 0;
}
```

For loop

```
#include<stdio.h>
int main(){
        int i=0,j=0,n[]={4,2,5,8,1,10,9,6,7,3},temp;
        for(i=0;i<9;i++)  {
                for(j=i;j<10;j++)       {
                        if(n[i]>n[j])   {
                                temp=n[i];
                                n[i]=n[j];
                                n[j]=temp;
                        }}}
        for(i=0;i<10;i++) {
                printf("%d\n",n[i]);
        }       return 0;
}
```

VI. Performance Analysis

To analyze performance of any statement, the best way is to analysis of assembly code generated by a statement. So, we analyzed the assembly code of every code. Here we are using gcc compiler for compiling and generating the assembly code. The assembly code (only for executable section) of input code is as under:

A. *Assembly Code for Case-1 (print "Hello World!" ten times)*

While loop

```
00000000 <_main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 20                sub    $0x20,%esp
   9:   e8 00 00 00 00          call   e <_main+0xe>
   e:   c7 44 24 1c 00 00 00    movl
$0x0,0x1c(%esp)
  15:   00
  16:   eb 11                   jmp    29 <_main+0x29>
  18:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  1f:   e8 00 00 00 00          call   24 <_main+0x24>
  24:   83 44 24 1c 01          addl   $0x1,0x1c(%esp)
  29:   83 7c 24 1c 09          cmpl   $0x9,0x1c(%esp)
  2e:   7e e8                   jle    18 <_main+0x18>
  30:   b8 00 00 00 00          mov    $0x0,%eax
  35:   c9                      leave
  36:   c3                      ret
  37:   90                      nop
```

Do-while loop

```
00000000 <_main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 20                sub    $0x20,%esp
   9:   e8 00 00 00 00          call   e <_main+0xe>
   e:   c7 44 24 1c 00 00 00    movl
$0x0,0x1c(%esp)
  15:   00
  16:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  1d:   e8 00 00 00 00          call   22 <_main+0x22>
  22:   83 44 24 1c 01          addl   $0x1,0x1c(%esp)
  27:   83 7c 24 1c 09          cmpl   $0x9,0x1c(%esp)
  2c:   7e e8                   jle    16 <_main+0x16>
  2e:   b8 00 00 00 00          mov    $0x0,%eax
  33:   c9                      leave
  34:   c3                  ret
  35:   90                  nop
  36:   90                  nop
  37:   90                  nop
```

For loop

```
00000000 <_main>:
   0:   55                  push   %ebp
   1:   89 e5               mov    %esp,%ebp
   3:   83 e4 f0            and    $0xfffffff0,%esp
   6:   83 ec 20            sub    $0x20,%esp
   9:   e8 00 00 00 00      call   e <_main+0xe>
   e:       c7   44   24   1c   00   00   00          movl
$0x0,0x1c(%esp)
  15:   00
  16:       c7   44   24   1c   00   00   00          movl
$0x0,0x1c(%esp)
  1d:   00
  1e:   eb 11               jmp    31 <_main+0x31>
  20:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  27:   e8 00 00 00 00      call   2c <_main+0x2c>
  2c:   83 44 24 1c 01      addl   $0x1,0x1c(%esp)
  31:   83 7c 24 1c 09      cmpl   $0x9,0x1c(%esp)
  36:   7e e8               jle    20 <_main+0x20>
  38:   b8 00 00 00 00      mov    $0x0,%eax
  3d:   c9                  leave
  3e:   c3                  ret
  3f:   90                  nop
```

B. *Assembly for Case-2 (loop with arithmetic operation, i.e., printing the table)*

While loop

```
00000000 <_main>:
   0:   55                  push   %ebp
   1:   89 e5               mov    %esp,%ebp
   3:   83 e4 f0            and    $0xfffffff0,%esp
   6:   83 ec 20            sub    $0x20,%esp
   9:   e8 00 00 00 00      call   e <_main+0xe>
   e:       c7   44   24   1c   01   00   00          movl
$0x1,0x1c(%esp)
  15:   00
  16:       c7   44   24   18   00   00   00          movl
$0x0,0x18(%esp)
  1d:   00
  1e:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  25:   e8 00 00 00 00      call   2a <_main+0x2a>
  2a:   8d 44 24 18         lea    0x18(%esp),%eax
  2e:   89 44 24 04         mov    %eax,0x4(%esp)
  32:   c7 04 24 11 00 00 00    movl   $0x11,(%esp)
  39:   e8 00 00 00 00      call   3e <_main+0x3e>
  3e:   eb 30               jmp    70 <_main+0x70>
  40:   8b 44 24 18         mov    0x18(%esp),%eax
  44:   0f af 44 24 1c      imul   0x1c(%esp),%eax
  49:   89 c2               mov    %eax,%edx
  4b:   8b 44 24 18         mov    0x18(%esp),%eax
  4f:   89 54 24 0c         mov    %edx,0xc(%esp)
  53:   8b 54 24 1c         mov    0x1c(%esp),%edx
  57:   89 54 24 08         mov    %edx,0x8(%esp)
  5b:   89 44 24 04         mov    %eax,0x4(%esp)
  5f:   c7 04 24 14 00 00 00    movl   $0x14,(%esp)
  66:   e8 00 00 00 00      call   6b <_main+0x6b>
  6b:   83 44 24 1c 01      addl   $0x1,0x1c(%esp)
  70:   83 7c 24 1c 0a      cmpl   $0xa,0x1c(%esp)
  75:   7e c9               jle    40 <_main+0x40>
  77:   b8 00 00 00 00      mov    $0x0,%eax
  7c:   c9                  leave
  7d:   c3                  ret
  7e:   90                  nop
  7f:   90                  nop
```

Do-while loop

```
00000000 <_main>:
   0:   55                  push   %ebp
   1:   89 e5               mov    %esp,%ebp
   3:   83 e4 f0            and    $0xfffffff0,%esp
   6:   83 ec 20            sub    $0x20,%esp
   9:   e8 00 00 00 00      call   e <_main+0xe>
   e:       c7   44   24   1c   01   00   00          movl
$0x1,0x1c(%esp)
  15:   00
  16:       c7   44   24   18   00   00   00          movl
$0x0,0x18(%esp)
  1d:   00
  1e:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  25:   e8 00 00 00 00      call   2a <_main+0x2a>
  2a:   8d 44 24 18         lea    0x18(%esp),%eax
  2e:   89 44 24 04         mov    %eax,0x4(%esp)
  32:   c7 04 24 11 00 00 00    movl   $0x11,(%esp)
  39:   e8 00 00 00 00      call   3e <_main+0x3e>
  3e:   8b 44 24 18         mov    0x18(%esp),%eax
  42:   0f af 44 24 1c      imul   0x1c(%esp),%eax
  47:   89 c2               mov    %eax,%edx
  49:   8b 44 24 18         mov    0x18(%esp),%eax
  4d:   89 54 24 0c         mov    %edx,0xc(%esp)
  51:   8b 54 24 1c         mov    0x1c(%esp),%edx
  55:   89 54 24 08         mov    %edx,0x8(%esp)
  59:   89 44 24 04         mov    %eax,0x4(%esp)
  5d:   c7 04 24 14 00 00 00    movl   $0x14,(%esp)
  64:   e8 00 00 00 00      call   69 <_main+0x69>
  69:   83 44 24 1c 01      addl   $0x1,0x1c(%esp)
  6e:   83 7c 24 1c 0a      cmpl   $0xa,0x1c(%esp)
  73:   7e c9               jle    3e <_main+0x3e>
  75:   b8 00 00 00 00      mov    $0x0,%eax
  7a:   c9                  leave
  7b:   c3                  ret
```

For loop

```
00000000 <_main>:
   0:   55                  push   %ebp
   1:   89 e5               mov    %esp,%ebp
   3:   83 e4 f0            and    $0xfffffff0,%esp
   6:   83 ec 20            sub    $0x20,%esp
   9:   e8 00 00 00 00          call   e <_main+0xe>
   e:   c7   44   24   1c   01   00   00          movl
$0x1,0x1c(%esp)
  15:   00
  16:   c7   44   24   18   00   00   00          movl
$0x0,0x18(%esp)
  1d:   00
  1e:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  25:   e8 00 00 00 00          call   2a <_main+0x2a>
  2a:   8d 44 24 18             lea    0x18(%esp),%eax
  2e:   89 44 24 04             mov    %eax,0x4(%esp)
  32:   c7 04 24 11 00 00 00    movl   $0x11,(%esp)
  39:   e8 00 00 00 00          call   3e <_main+0x3e>
  3e:   c7   44   24   1c   01   00   00          movl
$0x1,0x1c(%esp)
  45:   00
  46:   eb 30                   jmp    78 <_main+0x78>
  48:   8b 44 24 18             mov    0x18(%esp),%eax
  4c:   0f af 44 24 1c          imul   0x1c(%esp),%eax
  51:   89 c2                   mov    %eax,%edx
  53:   8b 44 24 18             mov    0x18(%esp),%eax
  57:   89 54 24 0c             mov    %edx,0xc(%esp)
  5b:   8b 54 24 1c             mov    0x1c(%esp),%edx
  5f:   89 54 24 08             mov    %edx,0x8(%esp)
  63:   89 44 24 04             mov    %eax,0x4(%esp)
  67:   c7 04 24 14 00 00 00    movl   $0x14,(%esp)
  6e:   e8 00 00 00 00          call   73 <_main+0x73>
  73:   83 44 24 1c 01          addl   $0x1,0x1c(%esp)
  78:   83 7c 24 1c 0a          cmpl   $0xa,0x1c(%esp)
  7d:   7e c9                   jle    48 <_main+0x48>
  7f:   b8 00 00 00 00          mov    $0x0,%eax
  84:   c9                      leave
  85:   c3                      ret
  86:   90                      nop
  87:   90                      nop
```

C. *Assembly for Case-3 (nesting of loop for sorting of 10 numbers)*

While loop

```
00000000 <_main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 50                sub    $0x50,%esp
   9:   e8 00 00 00 00          call   e <_main+0xe>
   e:   c7   44   24   4c   00   00   00          movl
$0x0,0x4c(%esp)
  15:   00
  16:   c7   44   24   48   00   00   00          movl
$0x0,0x48(%esp)
  1d:   00
  1e:   c7   44   24   1c   04   00   00          movl
$0x4,0x1c(%esp)
  25:   00
  26:   c7   44   24   20   02   00   00          movl
$0x2,0x20(%esp)
  2d:   00
  2e:   c7   44   24   24   05   00   00          movl
$0x5,0x24(%esp)
  35:   00
  36:   c7   44   24   28   08   00   00          movl
$0x8,0x28(%esp)
  3d:   00
  3e:   c7   44   24   2c   01   00   00          movl
$0x1,0x2c(%esp)
  45:   00
  46:   c7   44   24   30   0a   00   00          movl
$0xa,0x30(%esp)
  4d:   00
  4e:   c7   44   24   34   09   00   00          movl
$0x9,0x34(%esp)
  55:   00
  56:   c7   44   24   38   06   00   00          movl
$0x6,0x38(%esp)
  5d:   00
  5e:   c7   44   24   3c   07   00   00          movl
$0x7,0x3c(%esp)
  65:   00
  66:   c7   44   24   40   03   00   00          movl
$0x3,0x40(%esp)
  6d:   00
  6e:   eb 57                   jmp    c7 <_main+0xc7>
  70:   8b 44 24 4c             mov    0x4c(%esp),%eax
  74:   89 44 24 48             mov    %eax,0x48(%esp)
  78:   eb 41                   jmp    bb <_main+0xbb>
  7a:   8b 44 24 4c             mov    0x4c(%esp),%eax
  7e:        8b     54     84     1c              mov
0x1c(%esp,%eax,4),%edx
  82:   8b 44 24 48             mov    0x48(%esp),%eax
  86:        8b     44     84     1c              mov
0x1c(%esp,%eax,4),%eax
  8a:   39 c2                   cmp    %eax,%edx
  8c:   7e 28                   jle    b6 <_main+0xb6>
  8e:   8b 44 24 4c             mov    0x4c(%esp),%eax
  92:        8b     44     84     1c              mov
0x1c(%esp,%eax,4),%eax
  96:   89 44 24 44             mov    %eax,0x44(%esp)
  9a:   8b 44 24 48             mov    0x48(%esp),%eax
  9e:        8b     54     84     1c              mov
0x1c(%esp,%eax,4),%edx
  a2:   8b 44 24 4c             mov    0x4c(%esp),%eax
  a6:        89     54     84     1c              mov
%edx,0x1c(%esp,%eax,4)
  aa:   8b 44 24 48             mov    0x48(%esp),%eax
  ae:   8b 54 24 44             mov    0x44(%esp),%edx
  b2:        89     54     84     1c              mov
%edx,0x1c(%esp,%eax,4)
  b6:   83 44 24 48 01          addl   $0x1,0x48(%esp)
  bb:   83 7c 24 48 09          cmpl   $0x9,0x48(%esp)
  c0:   7e b8                   jle    7a <_main+0x7a>
  c2:  83 44 24 4c 01         addl   $0x1,0x4c(%esp)
  c7:  83 7c 24 4c 08         cmpl   $0x8,0x4c(%esp)
  cc:  7e a2                  jle    70 <_main+0x70>
  ce:  c7 44 24 4c 00 00 00   movl
$0x0,0x4c(%esp)
  d5:  00
  d6:  eb 1d                  jmp    f5 <_main+0xf5>
  d8:  8b 44 24 4c            mov    0x4c(%esp),%eax
  dc:  8b 44 84 1c            mov
0x1c(%esp,%eax,4),%eax
  e0:  89 44 24 04            mov    %eax,0x4(%esp)
  e4:  c7 04 24 00 00 00 00   movl   $0x0,(%esp)
  eb:  e8 00 00 00 00         call   f0 <_main+0xf0>
  f0:  83 44 24 4c 01         addl   $0x1,0x4c(%esp)
  f5:  83 7c 24 4c 09         cmpl   $0x9,0x4c(%esp)
  fa:  7e dc                  jle    d8 <_main+0xd8>
  fc:  b8 00 00 00 00         mov    $0x0,%eax
 101:  c9                     leave
 102:  c3                     ret
 103:  90                     nop
```

Do-while loop

```
00000000 <_main>:
   0:  55                     push   %ebp
   1:  89 e5                  mov    %esp,%ebp
   3:  83 e4 f0               and    $0xfffffff0,%esp
   6:  83 ec 50               sub    $0x50,%esp
   9:  e8 00 00 00 00         call   e <_main+0xe>
   e:  c7 44 24 4c 00 00 00   movl
$0x0,0x4c(%esp)
  15:  00
  16:  c7 44 24 48 00 00 00   movl
$0x0,0x48(%esp)
  1d:  00
  1e:  c7 44 24 1c 04 00 00   movl
$0x4,0x1c(%esp)
  25:  00
  26:  c7 44 24 20 02 00 00   movl
$0x2,0x20(%esp)
  2d:  00
  2e:  c7 44 24 24 05 00 00   movl
$0x5,0x24(%esp)
  35:  00
  36:  c7 44 24 28 08 00 00   movl
$0x8,0x28(%esp)
  3d:  00
  3e:  c7 44 24 2c 01 00 00   movl
$0x1,0x2c(%esp)
  45:  00
  46:  c7 44 24 30 0a 00 00   movl
$0xa,0x30(%esp)
  4d:  00
  4e:  c7 44 24 34 09 00 00   movl
$0x9,0x34(%esp)
  55:  00
  56:  c7 44 24 38 06 00 00   movl
$0x6,0x38(%esp)
  5d:  00
  5e:  c7 44 24 3c 07 00 00   movl
$0x7,0x3c(%esp)
  65:  00
  66:  c7 44 24 40 03 00 00   movl
$0x3,0x40(%esp)
  6d:  00
  6e:  8b 44 24 4c            mov    0x4c(%esp),%eax
  72:  89 44 24 48            mov    %eax,0x48(%esp)
  76:  8b 44 24 4c            mov    0x4c(%esp),%eax
  7a:  8b 54 84 1c            mov
0x1c(%esp,%eax,4),%edx
  7e:  8b 44 24 48            mov    0x48(%esp),%eax
  82:  8b 44 84 1c            mov
0x1c(%esp,%eax,4),%eax
  86:  39 c2                  cmp    %eax,%edx
  88:  7e 28                  jle    b2 <_main+0xb2>
  8a:  8b 44 24 4c            mov    0x4c(%esp),%eax
  8e:  8b 44 84 1c            mov
0x1c(%esp,%eax,4),%eax
  92:  89 44 24 44            mov    %eax,0x44(%esp)
  96:  8b 44 24 48            mov    0x48(%esp),%eax
  9a:  8b 54 84 1c            mov
0x1c(%esp,%eax,4),%edx
  9e:  8b 44 24 4c            mov    0x4c(%esp),%eax
  a2:  89 54 84 1c            mov
%edx,0x1c(%esp,%eax,4)
  a6:  8b 44 24 48            mov    0x48(%esp),%eax
  aa:  8b 54 24 44            mov    0x44(%esp),%edx
  ae:  89 54 84 1c            mov
%edx,0x1c(%esp,%eax,4)
  b2:  83 44 24 48 01         addl   $0x1,0x48(%esp)
  b7:  83 7c 24 48 09         cmpl   $0x9,0x48(%esp)
  bc:  7e b8                  jle    76 <_main+0x76>
  be:  83 44 24 4c 01         addl   $0x1,0x4c(%esp)
  c3:  83 7c 24 4c 08         cmpl   $0x8,0x4c(%esp)
  c8:  7e a4                  jle    6e <_main+0x6e>
  ca:  c7 44 24 4c 00 00 00   movl
$0x0,0x4c(%esp)
  d1:  00
  d2:  8b 44 24 4c            mov    0x4c(%esp),%eax
  d6:  8b 44 84 1c            mov
0x1c(%esp,%eax,4),%eax
  da:  89 44 24 04            mov    %eax,0x4(%esp)
  de:  c7 04 24 00 00 00 00   movl   $0x0,(%esp)
  e5:  e8 00 00 00 00         call   ea <_main+0xea>
  ea:  83 44 24 4c 01         addl   $0x1,0x4c(%esp)
  ef:  83 7c 24 4c 09         cmpl   $0x9,0x4c(%esp)
  f4:  7e dc                  jle    d2 <_main+0xd2>
  f6:  b8 00 00 00 00         mov    $0x0,%eax
  fb:  c9                     leave
  fc:  c3                     ret
  fd:  90                     nop
  fe:  90                     nop
  ff:   90                  nop
```

For loop

```
00000000 <_main>:
   0:   55                    push   %ebp
   1:   89 e5                  mov    %esp,%ebp
   3:   83 e4 f0               and    $0xfffffff0,%esp
   6:   83 ec 50               sub    $0x50,%esp
   9:   e8 00 00 00 00          call   e <_main+0xe>
   e:        c7   44   24   4c   00   00   00           movl
$0x0,0x4c(%esp)
  15:   00
  16:        c7   44   24   48   00   00   00           movl
$0x0,0x48(%esp)
  1d:   00
  1e:        c7   44   24   1c   04   00   00           movl
$0x4,0x1c(%esp)
  25:   00
  26:        c7   44   24   20   02   00   00           movl
$0x2,0x20(%esp)
  2d:   00
  2e:        c7   44   24   24   05   00   00           movl
$0x5,0x24(%esp)
  35:   00
  36:        c7   44   24   28   08   00   00           movl
$0x8,0x28(%esp)
  3d:   00
  3e:        c7   44   24   2c   01   00   00           movl
$0x1,0x2c(%esp)
  45:   00
  46:        c7   44   24   30   0a   00   00           movl
$0xa,0x30(%esp)
  4d:   00
  4e:        c7   44   24   34   09   00   00           movl
$0x9,0x34(%esp)
  55:   00
  56:        c7   44   24   38   06   00   00           movl
$0x6,0x38(%esp)
  5d:   00
  5e:        c7   44   24   3c   07   00   00           movl
$0x7,0x3c(%esp)
  65:   00
  66:        c7   44   24   40   03   00   00           movl
$0x3,0x40(%esp)
  6d:   00
  6e:        c7   44   24   4c   00   00   00           movl
$0x0,0x4c(%esp)
  75:   00
  76:   eb 57                 jmp    cf <_main+0xcf>
  78:   8b 44 24 4c             mov    0x4c(%esp),%eax
  7c:   89 44 24 48             mov    %eax,0x48(%esp)
  80:   eb 41                 jmp    c3 <_main+0xc3>
  82:   8b 44 24 4c             mov    0x4c(%esp),%eax
  86:            8b     54     84     1c                      mov
0x1c(%esp,%eax,4),%edx
  8a:   8b 44 24 48             mov    0x48(%esp),%eax
  8e:            8b     44     84     1c                      mov
0x1c(%esp,%eax,4),%eax
  92:   39 c2                 cmp    %eax,%edx
  94:   7e 28                 jle    be <_main+0xbe>
  96:   8b 44 24 4c             mov    0x4c(%esp),%eax
  9a:            8b     44     84     1c                      mov
0x1c(%esp,%eax,4),%eax
  9e:   89 44 24 44             mov    %eax,0x44(%esp)
  a2:   8b 44 24 48             mov    0x48(%esp),%eax
  a6:            8b     54     84     1c                      mov
0x1c(%esp,%eax,4),%edx
  aa:   8b 44 24 4c             mov    0x4c(%esp),%eax
  ae:            89     54     84     1c                      mov
%edx,0x1c(%esp,%eax,4)
  b2:   8b 44 24 48             mov    0x48(%esp),%eax
  b6:   8b 54 24 44             mov    0x44(%esp),%edx
  ba:            89     54     84     1c                      mov
%edx,0x1c(%esp,%eax,4)
  be:   83 44 24 48 01          addl   $0x1,0x48(%esp)
  c3:   83 7c 24 48 09          cmpl   $0x9,0x48(%esp)
  c8:   7e b8                 jle    82 <_main+0x82>
  ca:   83 44 24 4c 01          addl   $0x1,0x4c(%esp)
  cf:   83 7c 24 4c 08          cmpl   $0x8,0x4c(%esp)
  d4:   7e a2                 jle    78 <_main+0x78>
  d6:        c7   44   24   4c   00   00   00           movl
$0x0,0x4c(%esp)
  dd:   00
  de:   eb 1d                 jmp    fd <_main+0xfd>
  e0:   8b 44 24 4c             mov    0x4c(%esp),%eax
  e4:            8b     44     84     1c                      mov
0x1c(%esp,%eax,4),%eax
  e8:   89 44 24 04             mov    %eax,0x4(%esp)
  ec:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  f3:   e8 00 00 00 00          call   f8 <_main+0xf8>
  f8:   83 44 24 4c 01          addl   $0x1,0x4c(%esp)
  fd:   83 7c 24 4c 09          cmpl   $0x9,0x4c(%esp)
 102:   7e dc                 jle    e0 <_main+0xe0>
 104:   b8 00 00 00 00          mov    $0x0,%eax
 109:   c9                    leave
 10a:   c3                    ret
 10b:   90                    nop
```

The above assembly instructions generated in these three cases can be summarized as under in Table-1:

Case	No. of Assembly Statement generated for executable section		
	while loop	do-while loop	for loop
Case-1	17	18	19
Case-2	34	31	36
Case-3	70	69	72

Table 1- Summary of Assembly Instruction in each case

VII. RESULTS

With above table it is clear that for loop is generating the highest number of assembly instructions in each case, so it is the worst performer. The do-while loop is the best loop if we are going to repeat the statements till the known number of times. The while loop tops only if arithmetic or logical operations are not used which is generally not happened, so according to our analysis, do-while loop is the best performer as it generates least number of assembly instructions.

VIII. CONCLUSION

The three loops of c language compared according to number of assembly instructions generated in executable section of program. This is a very well defined way of comparison which results do-while loop as the best. The comparison may also include some other points such as function call within the loop, or nesting of different types of loops.

REFERENCES

[1]. Mark Burgess, "*The GNU C Programming Tutorial*", Ron Hale-Evans, **Norway**, pp. 61-68, 2002

[2]. E Balagurusamy, "Programming in ANSI C", Tata McGraw-Hill, **India,** pp 154-159, 2007

[3]. Joseph Cavanagh, "X86 Assembly Language and C Fundamentals", CRC Press Taylor & Francis Group, **New York**, pp 251-266, 2013

Authors Profile

Jagdish Makhijani pursed Ph.D. in Computer Science from Barkatullah University, Bhopal in 2013. He is currently working as Assistant Professor in Department of Computer Science & Engineering, Rustamji Institute of Technology, BSF Academy, Tekanpur since 2012. He is a life member of Computer Society of India since 2017, a life member of the Vigyan Bharti since 2018. He has published more than 10 research papers in reputed international journals and conferences. His main research work focuses on Distributed Systems and Competative Programming Research. He has 19 years of teaching experience and 10 years of Research Experience.

Manoj Kumar Niranjan pursed Ph.D. in Computer Applications from Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal in 2019. He is currently working as Assistant Professor in Department of Computer Applications, Rustamji Institute of Technology, BSF Academy, Tekanpur. He is a life-time member of Vigyan Bharti. He has published more than 10 research papers in reputed international journals and conferences. His main research work focuses on Distributed Systems and Artificial Intelligence. He has 16 years of teaching experience and 10 years of Research Experience.

Yograj Sharma pursed M. E. in Computer Science & Engineering from IET, Devi Ahilya Vishwavidyalaya, Indore in 2014. He is currently working as Assistant Professor & Head in Department of Computer Science & Engineering, Rustamji Institute of Technology, BSF Academy, Tekanpur. He is a member of Computer Society of India since 2018. He has published more than 08 research papers in reputed international journals and conferences. His main research work focuses on Cryptography and Information Security. He has 05 years of teaching experience and 06 years of Research Experience.