ANALYSIS OF ASSEMBLY CODE OF BASIC C PROGRAM (GCC 9.3.0)

Concepts of Programming Languages
Assignment -4

C P SHIVA REDDY
BT19CSE019

Computer Science and Engineering
Visvesvaraya National Institute of Technology, Nagpur

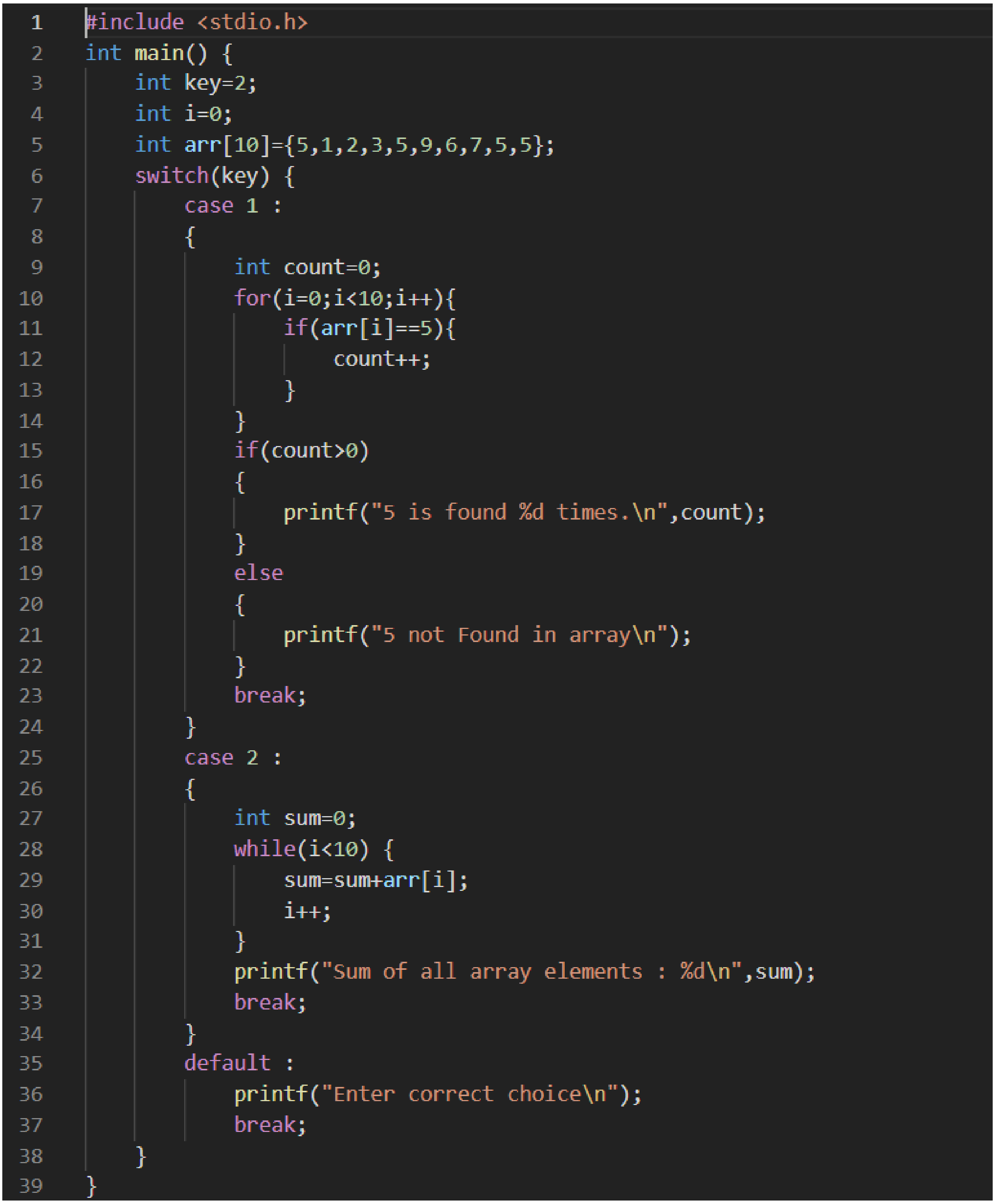
10th April 2021

Contact:
shivareddy1001@gmail.com

*** C Program used for demonstration ***

* Key features included in Program: For and While loop, Conditional Statement
Switch Statement and assignment

</> Below is the Code, it is just for demonstration purpose, switch case 1 calculates occurrences of 5 in array using linear search and case 2 finds sum of all array elements.



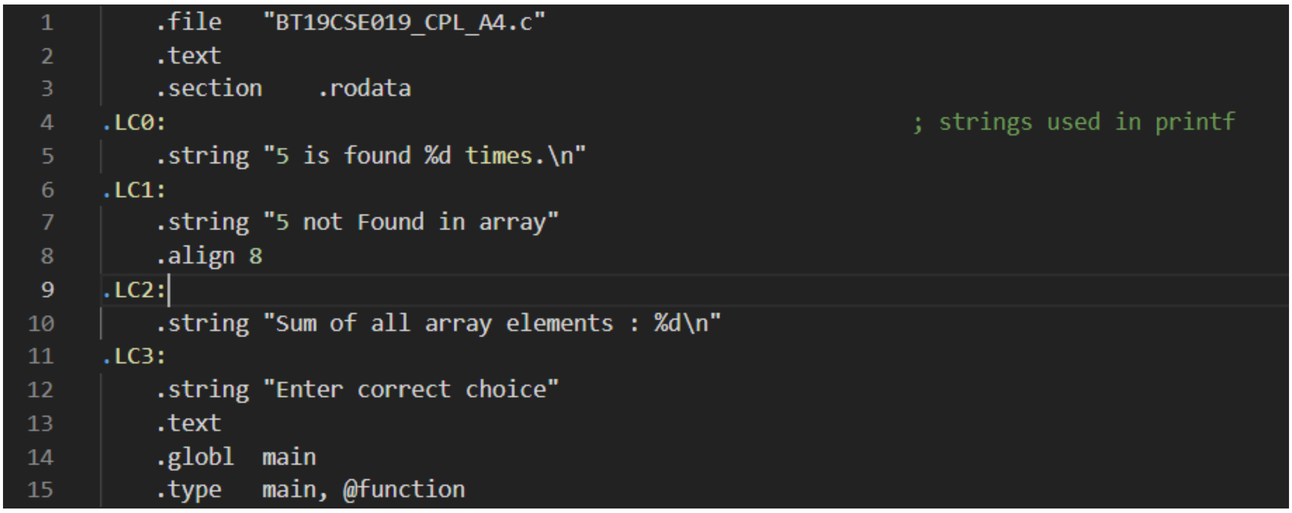
```c
#include <stdio.h>
int main() {
    int key=2;
    int i=0;
    int arr[10]={5,1,2,3,5,9,6,7,5,5};
    switch(key) {
        case 1 :
        {
            int count=0;
            for(i=0;i<10;i++){
                if(arr[i]==5){
                    count++;
                }
            }
            if(count>0)
            {
                printf("5 is found %d times.\n",count);
            }
            else
            {
                printf("5 not Found in array\n");
            }
            break;
        }
        case 2 :
        {
            int sum=0;
            while(i<10) {
                sum=sum+arr[i];
                i++;
            }
            printf("Sum of all array elements : %d\n",sum);
            break;
        }
        default :
            printf("Enter correct choice\n");
            break;
    }
}
```

*** Assembly Program ***

</>
Generated Assembly equivalent by compiler using the command
gcc -S File_name.c
which generated a “File_name.s” file containing our required assembly code.
The same is shown and explained briefly below in parts.

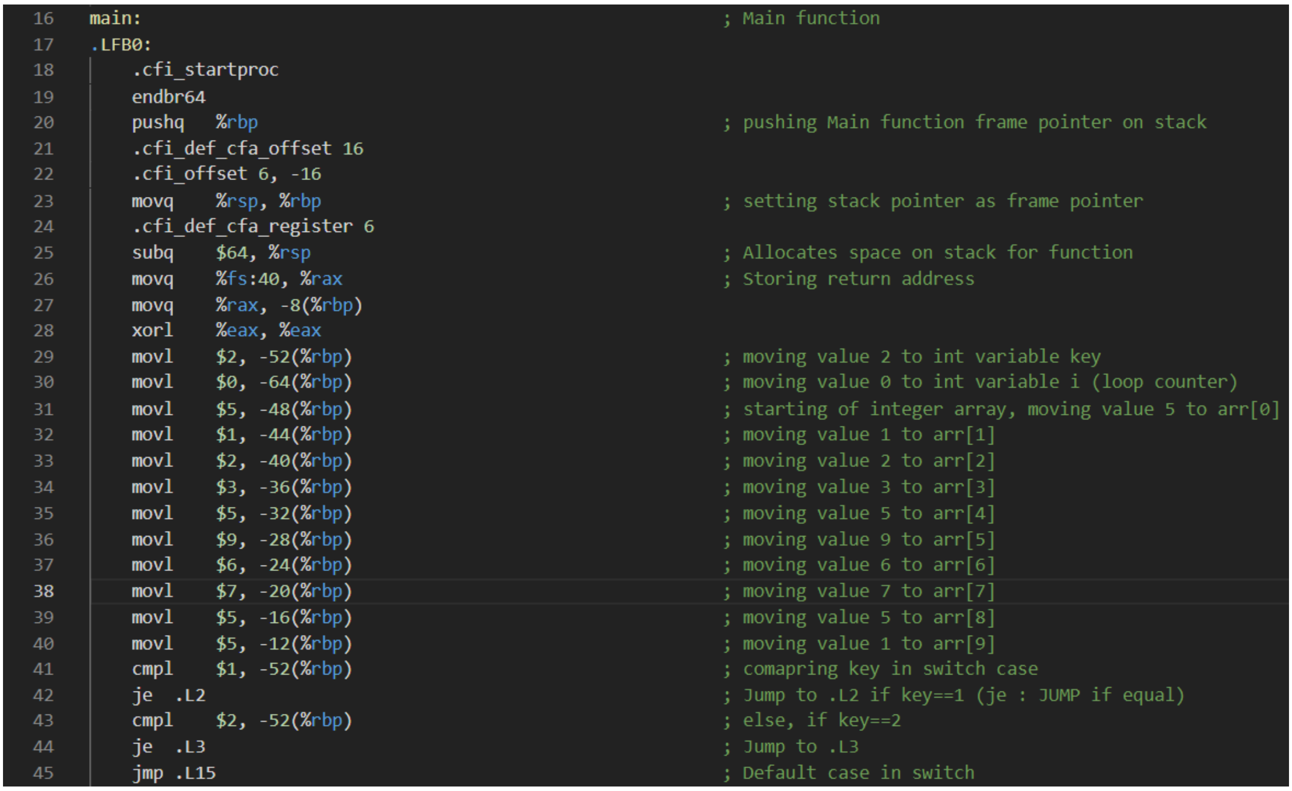
</> 1

Below shown snippet is the very beginning of the file, it contains all typed strings in printf statements, these will be used later when printf is called, also indicates main function below



```
    .file   "BT19CSE019_CPL_A4.c"
    .text
    .section    .rodata
.LC0:                                                   ; strings used in printf
    .string "5 is found %d times.\n"
.LC1:
    .string "5 not Found in array"
    .align 8
.LC2:
    .string "Sum of all array elements : %d\n"
.LC3:
    .string "Enter correct choice"
    .text
    .globl  main
    .type   main, @function
```

</> 2



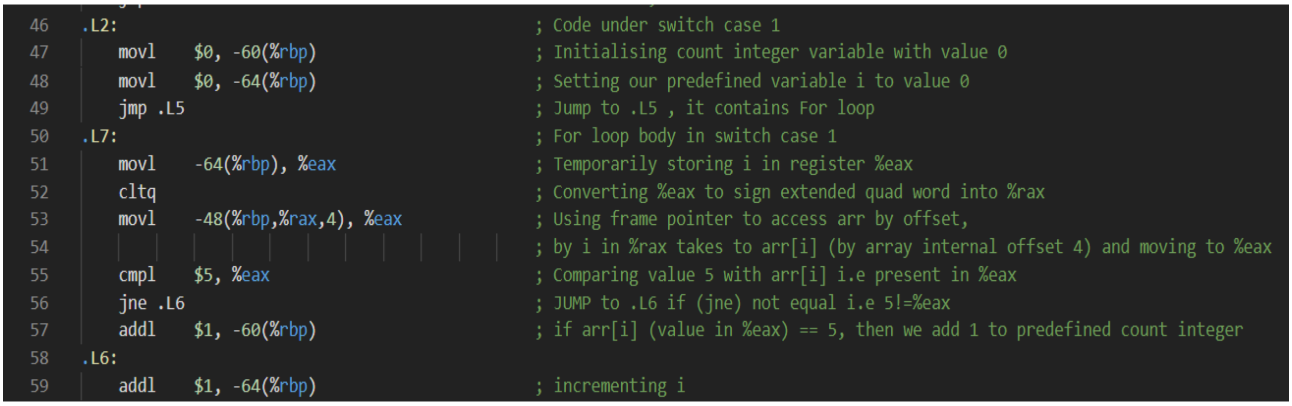
```
main:                                                   ; Main function
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp                                        ; pushing Main function frame pointer on stack
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp                                  ; setting stack pointer as frame pointer
    .cfi_def_cfa_register 6
    subq    $64, %rsp                                   ; Allocates space on stack for function
    movq    %fs:40, %rax                                ; Storing return address
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    movl    $2, -52(%rbp)                               ; moving value 2 to int variable key
    movl    $0, -64(%rbp)                               ; moving value 0 to int variable i (loop counter)
    movl    $5, -48(%rbp)                               ; starting of integer array, moving value 5 to arr[0]
    movl    $1, -44(%rbp)                               ; moving value 1 to arr[1]
    movl    $2, -40(%rbp)                               ; moving value 2 to arr[2]
    movl    $3, -36(%rbp)                               ; moving value 3 to arr[3]
    movl    $5, -32(%rbp)                               ; moving value 5 to arr[4]
    movl    $9, -28(%rbp)                               ; moving value 9 to arr[5]
    movl    $6, -24(%rbp)                               ; moving value 6 to arr[6]
    movl    $7, -20(%rbp)                               ; moving value 7 to arr[7]
    movl    $5, -16(%rbp)                               ; moving value 5 to arr[8]
    movl    $5, -12(%rbp)                               ; moving value 1 to arr[9]
    cmpl    $1, -52(%rbp)                               ; comapring key in switch case
    je  .L2                                             ; Jump to .L2 if key==1 (je : JUMP if equal)
    cmpl    $2, -52(%rbp)                               ; else, if key==2
    je  .L3                                             ; Jump to .L3
    jmp .L15                                            ; Default case in switch
```

This part of code is the crux of the Main function's control flow, as mentioned in comments frame pointer (%rbp) is pushed onto program stack activation record and required offsets are also set, stack pointer (%rsp) is set to frame pointer.

After that required size according to data is allocated on stack which is 64 in this case, essential data like return address are stored.
Data initialising starts then, starting with integer key used for toggling switch statement which is set to value to 2 (just for demonstration) initially.

Key is at -52(%rbp) 52 here denotes offset and negative sign is because stack grows downwards.
Next integer variable initialised is i, which is used as loop counter is set to value 0, it is present at -64(%rbp), and finally our array of 10 integers is stored as continuos blocks starting from-48(%rbp). Then key is compared to 1, if it is equal to 1 then program control jumps to .L2, else if key is equal to 2, program control jumps to case 2 at .L3, else default is executed in switch and jumps to .L15 to print a statement.

</> 3

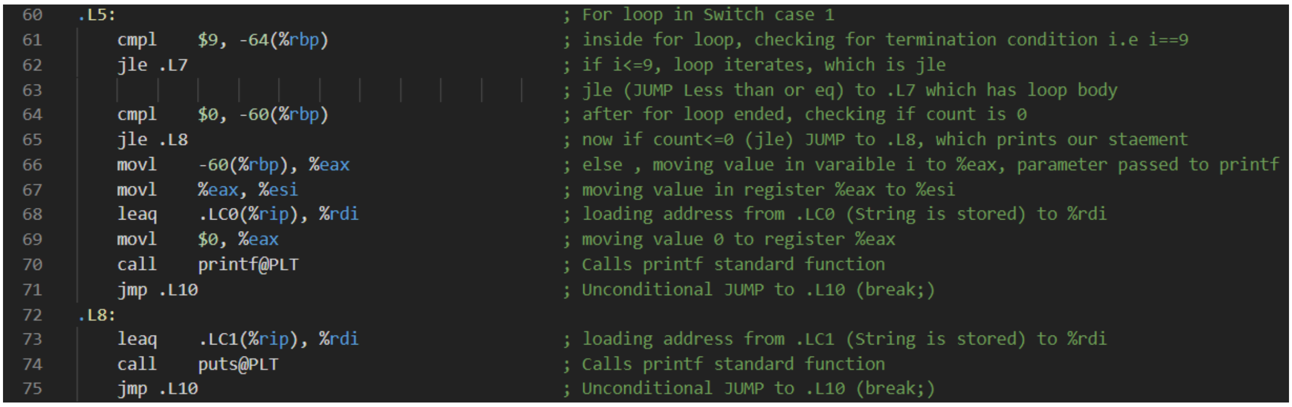


```
.L2:                                            ; Code under switch case 1
    movl    $0, -60(%rbp)                       ; Initialising count integer variable with value 0
    movl    $0, -64(%rbp)                       ; Setting our predefined variable i to value 0
    jmp .L5                                     ; Jump to .L5 , it contains For loop
.L7:                                            ; For loop body in switch case 1
    movl    -64(%rbp), %eax                     ; Temporarily storing i in register %eax
    cltq                                        ; Converting %eax to sign extended quad word into %rax
    movl    -48(%rbp,%rax,4), %eax              ; Using frame pointer to access arr by offset,
                                                ; by i in %rax takes to arr[i] (by array internal offset 4) and moving to %eax
    cmpl    $5, %eax                            ; Comparing value 5 with arr[i] i.e present in %eax
    jne .L6                                     ; JUMP to .L6 if (jne) not equal i.e 5!=%eax
    addl    $1, -60(%rbp)                       ; if arr[i] (value in %eax) == 5, then we add 1 to predefined count integer
.L6:
    addl    $1, -64(%rbp)                       ; incrementing i
```

When switch case 1 is used, this part of code works namely labels with L2,L7,L6,L5,L8
As mentioned in comments first it initialises a new count integer to 0 at -60(%rbp) and starts our loop. wherein iterator integer i is set to 0 again and control is transferred to L5 which is discussed below.
L7: it stores i in a temporary register and then sign extends that to a quad word word in register %rax, now frame pointer and offset it access array first element and then takes i from %rax and increments i*4 to address to obtain arr[i] value.
This is stored in register %eax again, now as per algorithm it compares arr[i] in %eax with $5 and then based on result jump is conditionally called
i.e. if jne (jump if not equal) is called is arr[i]!=5, this transfers control to L6 which increments i. Else i.e. if jne is not called count is incremented.

Variables offset :	**name**	**offset**
	i	-64(%rbp)
	count	-60(%rbp)
	sum	-56(%rbp)
	key	-52(%rbp)
	arr (arr[0])	-48(%rbp)

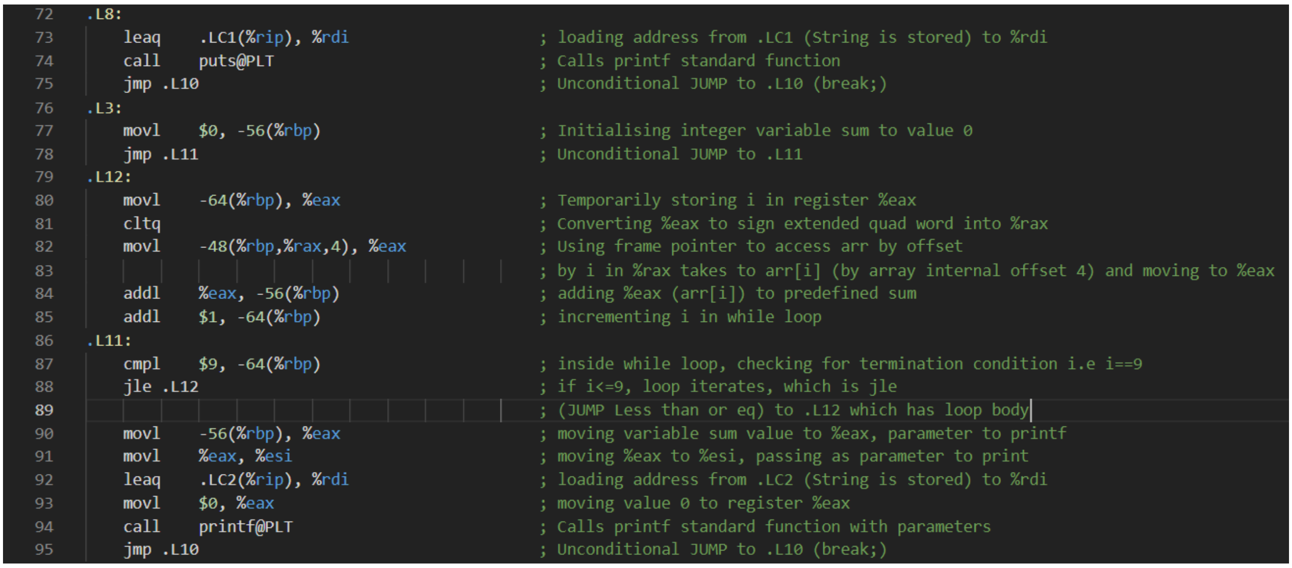
</> 4



```
.L5:                                              ; For loop in Switch case 1
    cmpl    $9, -64(%rbp)                         ; inside for loop, checking for termination condition i.e i==9
    jle .L7                                       ; if i<=9, loop iterates, which is jle
                                                  ; jle (JUMP Less than or eq) to .L7 which has loop body
    cmpl    $0, -60(%rbp)                         ; after for loop ended, checking if count is 0
    jle .L8                                       ; now if count<=0 (jle) JUMP to .L8, which prints our staement
    movl    -60(%rbp), %eax                       ; else , moving value in varaible i to %eax, parameter passed to printf
    movl    %eax, %esi                            ; moving value in register %eax to %esi
    leaq    .LC0(%rip), %rdi                      ; loading address from .LC0 (String is stored) to %rdi
    movl    $0, %eax                              ; moving value 0 to register %eax
    call    printf@PLT                            ; Calls printf standard function
    jmp .L10                                      ; Unconditional JUMP to .L10 (break;)
.L8:
    leaq    .LC1(%rip), %rdi                      ; loading address from .LC1 (String is stored) to %rdi
    call    puts@PLT                              ; Calls printf standard function
    jmp .L10                                      ; Unconditional JUMP to .L10 (break;)
```

As discussed above here we see the code under label L5, where comparison for loop is taking place, at start it compares i with 9, if i<=9 then control is passed on to loop body at .L7.
After loop terminates, count is compared to 0, if count<=0 then control is shifted to .L8 where string from .LC1 is loaded into %rdi and puts (works as printf) is called and then unconditionally jump to .L10 is called indicating label where break statement is executed.
If count is not <=0 then, count is stored in %eax and is sent as parameter to printf next along with string at LC0.

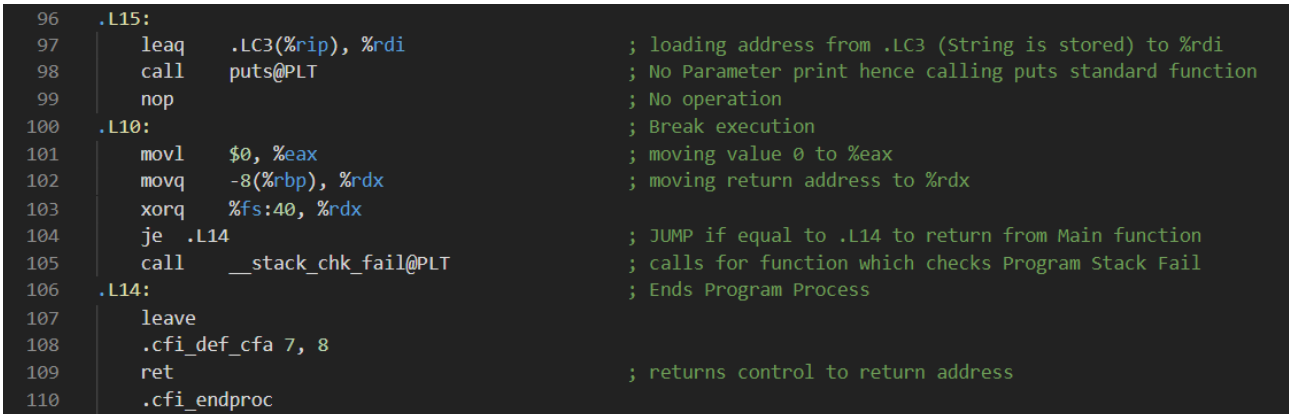
</> 5



```
.L8:
    leaq    .LC1(%rip), %rdi                      ; loading address from .LC1 (String is stored) to %rdi
    call    puts@PLT                              ; Calls printf standard function
    jmp .L10                                      ; Unconditional JUMP to .L10 (break;)
.L3:
    movl    $0, -56(%rbp)                         ; Initialising integer variable sum to value 0
    jmp .L11                                      ; Unconditional JUMP to .L11
.L12:
    movl    -64(%rbp), %eax                       ; Temporarily storing i in register %eax
    cltq                                          ; Converting %eax to sign extended quad word into %rax
    movl    -48(%rbp,%rax,4), %eax                ; Using frame pointer to access arr by offset
                                                  ; by i in %rax takes to arr[i] (by array internal offset 4) and moving to %eax
    addl    %eax, -56(%rbp)                       ; adding %eax (arr[i]) to predefined sum
    addl    $1, -64(%rbp)                         ; incrementing i in while loop
.L11:
    cmpl    $9, -64(%rbp)                         ; inside while loop, checking for termination condition i.e i==9
    jle .L12                                      ; if i<=9, loop iterates, which is jle
                                                  ; (JUMP Less than or eq) to .L12 which has loop body
    movl    -56(%rbp), %eax                       ; moving variable sum value to %eax, parameter to printf
    movl    %eax, %esi                            ; moving %eax to %esi, passing as parameter to print
    leaq    .LC2(%rip), %rdi                      ; loading address from .LC2 (String is stored) to %rdi
    movl    $0, %eax                              ; moving value 0 to register %eax
    call    printf@PLT                            ; Calls printf standard function with parameters
    jmp .L10                                      ; Unconditional JUMP to .L10 (break;)
```

Here, L3 is shown which is first called when switch is under case 2, and it initialises integer sum to 0 at -56(%rbp) and unconditional transfer happens to L11 where while loop body is present.
In While loop (L11) first ‘i’ is compared with 9, if i<=9 loop is executed else terminated.
Inside loop body L12 where our main algorithm works, in L12 ‘i’ is temporarily stored in %eax and again sign extended to quad word in %rax, and now as we have seen in for loop similar access to array element is made using offset and frame pointer and arr[i] is stored into %eax, now arr[i] is added to sum and i is incremented.
After loop terminates sum is stored in %eax ad sent as parameter for printf along with string from .LC2 (obtained from address: leaq) and then printf is called and control transferred to L10 which is break statement.

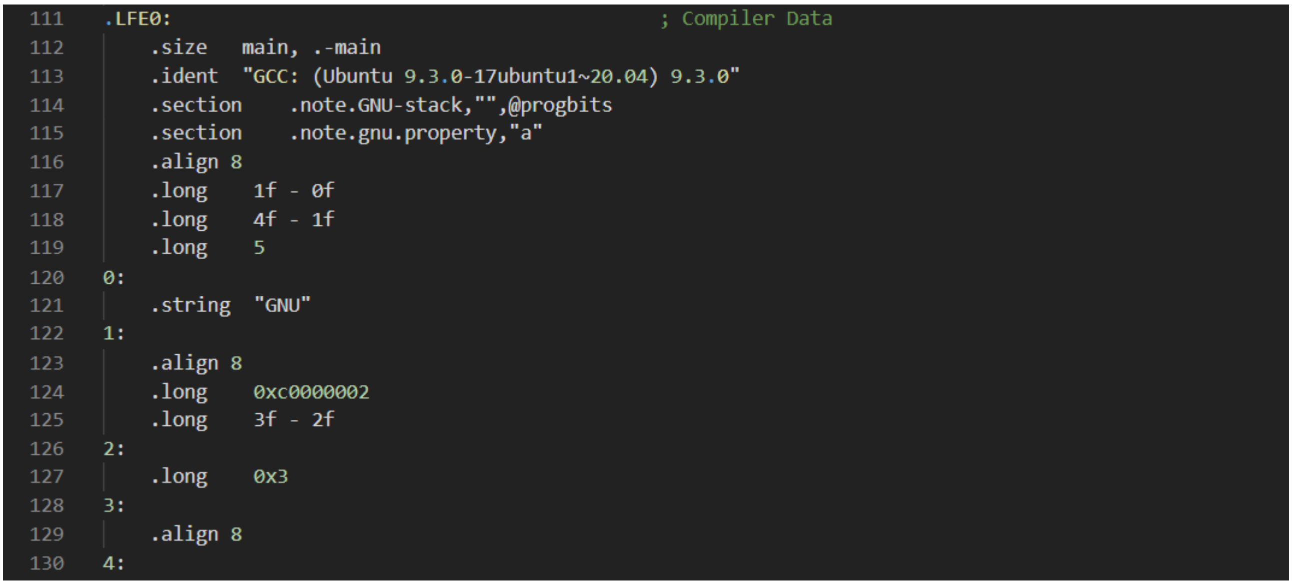
</> 6



```
.L15:
    leaq    .LC3(%rip), %rdi                        ; loading address from .LC3 (String is stored) to %rdi
    call    puts@PLT                                ; No Parameter print hence calling puts standard function
    nop                                             ; No operation
.L10:                                               ; Break execution
    movl    $0, %eax                                ; moving value 0 to %eax
    movq    -8(%rbp), %rdx                          ; moving return address to %rdx
    xorq    %fs:40, %rdx
    je  .L14                                        ; JUMP if equal to .L14 to return from Main function
    call    __stack_chk_fail@PLT                    ; calls for function which checks Program Stack Fail
.L14:                                               ; Ends Program Process
    leave
    .cfi_def_cfa 7, 8
    ret                                             ; returns control to return address
    .cfi_endproc
```

Now, L15 here is for default case in switch which prints a statement stored in LC3 and then puts function is called which works like printf and at the end nop indicates no operation.
In L10, which is our break statement gets the return address from activation record and compares using XOR and then conditionally L14 is called which returns control to address pointed by return address and ends program process.
Else if jump L14 doesn't take place call to function is made which checks program stacks failure.

</> 7



```
.LFE0:                                              ; Compiler Data
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string  "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

This data is regarding the compiler options and its details.

-- THE END --