# **CS 410 C++ to Assembly With Loops Activity – Elliot Putnam**

**Step 1:** Explain the functionality of the C++ code.

## C++ Code Functionality

| **C++ Line of Code** | **Explanation of Functionality** |
| --- | --- |
| Int main() | Initializes the main function |
| Int num, i; | Initializes integer variables ‘num’ & ‘I’ |
| Int product = 1; | Initializes int variable ‘product’ and stores 1 |
| Cout<<”Enter a number:\n”<<endl; | Outputs a string to console |
| Cin>>num; | Takes user input and stores int in var ‘num’ |
| For (i=num;i>0;i--) | Initializes for loop to decrement ‘i’ |
| product = product \* i; | as the for loop conditions are met, store the sum of ‘product’ & ‘i’ into ‘product’ |
| cout<<”The factorial for “ << num << “is: \n”<< product; | outputs string and variable ‘num’ and string and var ‘product’ |
| return 1 | returns 1 |
|  |  |
|  |  |

**Step 2:** Convert the C++ file into assembly code.

**Step 3:** Align each line of C++ code with the corresponding blocks of assembly code.

## C++ to Assembly Alignment

| **C++ Line of Code** | **Blocks of Assembly Code** |
| --- | --- |
| cin>>num; | mov     rdi, qword ptr [rip + std::cin@GOTPCREL]  lea     rsi, [rsp + 4]  call    std::istream::operator>>(int&)@PLT |
| for(i=num;i>0; i--) | mov     eax, dword ptr [rsp + 4]  test    eax, eax  jle     .LBB0\_6  lea     ecx, [rax - 1]  cmp     eax, 1  mov     eax, ecx  jg      .LBB0\_5 |
| product = product \* i; | imul    ebx, eax |
| cout<<"The factorial for " << num << "is: \n"<< product; | mov     esi, dword ptr [rsp + 4]  mov     rdi, r14  call    std::ostream::operator<<(int)@PLT  mov     r14, rax  mov     rdi, r14  mov     esi, ebx  call    std::ostream::operator<<(int)@PLT |
| return 1; | mov     eax, 1  add     rsp, 8  pop     rbx  pop     r14  ret |

**Step 4:** Explain how the blocks of assembly code perform the same tasks as the C++ code.

## Assembly Functionality

    .file   "assignment2\_1.cpp"    ; Source file

    .text

    .section    .rodata             ; Read-only data section

    .type   \_ZStL19piecewise\_construct, @object  ; Declare piecewise\_construct object

    .size   \_ZStL19piecewise\_construct, 1    ; Object size: 1 byte

\_ZStL19piecewise\_construct:                  ; Label for piecewise\_construct

    .zero   1                              ; Allocate 1 byte of zero value

    .local  \_ZStL8\_\_ioinit                 ; Declare local symbol \_ZStL8\_\_ioinit

    .comm   \_ZStL8\_\_ioinit,1,1             ; Allocate 1 byte for \_ZStL8\_\_ioinit

.LC0:                                     ; Label for string "Enter a number"

    .string "Enter a number:\n"            ; Prompt string

.LC1:                                     ; Label for string "The factorial for"

    .string "The factorial for "

.LC2:                                     ; Label for string "is: \n"

    .string "is: \n"

    .text

    .globl  main                          ; Declare global main function

    .type   main, @function

main:

.LFB1493:                                ; Frame begin for main function

    .cfi\_startproc                        ; Start procedure

    pushq   %rbp                           ; Push base pointer

    .cfi\_def\_cfa\_offset 16                ; Define CFA offset

    .cfi\_offset 6, -16                    ; Define base pointer offset

    movq    %rsp, %rbp                     ; Set base pointer

    .cfi\_def\_cfa\_register 6               ; Set CFA register

    subq    $32, %rsp                     ; Allocate stack space

    movq    %fs:40, %rax                  ; Load stack canary value

    movq    %rax, -8(%rbp)                 ; Save stack canary value

    xorl    %eax, %eax                    ; Clear EAX register

    movl    $1, -12(%rbp)                 ; Initialize factorial result to 1

    leaq    .LC0(%rip), %rsi              ; Load "Enter a number" string

    leaq    \_ZSt4cout(%rip), %rdi         ; Load cout

    call    \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT ; Call output stream function

    movq    %rax, %rdx                    ; Move result to RDX

    movq    \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax ; Load endl function

    movq    %rax, %rsi                    ; Move endl to RSI

    movq    %rdx, %rdi                    ; Move RDX to RDI (stream)

    call    \_ZNSolsEPFRSoS\_E@PLT       ; Call stream operator to print newline

    leaq    -20(%rbp), %rax               ; Load input address

    movq    %rax, %rsi                     ; Move address to RSI

    leaq    \_ZSt3cin(%rip), %rdi           ; Load cin

    call    \_ZNSirsERi@PLT                 ; Call input stream function

    movl    -20(%rbp), %eax               ; Load input value

    movl    %eax, -16(%rbp)               ; Store input value

.L3:                                       ; Start of factorial loop

    cmpl    $0, -16(%rbp)                 ; Compare input with 0

    jle     .L2                           ; Jump if less or equal to 0 (exit loop)

    movl    -12(%rbp), %eax               ; Load factorial result

    imull   -16(%rbp), %eax               ; Multiply result by input

    movl    %eax, -12(%rbp)               ; Store new result

    subl    $1, -16(%rbp)               ; Decrement input by 1

    jmp     .L3                         ; Jump back to loop start

.L2:                                       ; End of factorial loop

    leaq    .LC1(%rip), %rsi               ; Load "The factorial for" string

    leaq    \_ZSt4cout(%rip), %rdi         ; Load cout

    call    \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT ; Print output

    movq    %rax, %rdx                    ; Move result to RDX

    movl    -20(%rbp), %eax               ; Load input value

    movl    %eax, %esi                    ; Move input to ESI

    movq    %rdx, %rdi                    ; Move RDX to RDI (stream)

    call    \_ZNSolsEi@PLT                 ; Print input value

    leaq    .LC2(%rip), %rsi              ; Load "is: \n" string

    movq    %rax, %rdi                     ; Move result to RDI (stream)

    call    \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT ; Print output

    movq    %rax, %rdx                    ; Move result to RDX

    movl    -12(%rbp), %eax               ; Load factorial result

    movl    %eax, %esi                    ; Move result to ESI

    movq    %rdx, %rdi                    ; Move RDX to RDI (stream)

    call    \_ZNSolsEi@PLT                 ; Print factorial result

    movl    $1, %eax                      ; Set return value to 1

    movq    -8(%rbp), %rcx                 ; Load stack canary

    xorq    %fs:40, %rcx                   ; Check stack canary

    je      .L5                           ; Jump if stack canary is intact

    call    \_\_stack\_chk\_fail@PLT         ; Call stack check fail if corrupted

.L5:                                       ; End of function

    leave                                 ; Restore stack

    .cfi\_def\_cfa 7, 8                     ; Define CFA

    ret                                   ; Return

    .cfi\_endproc                          ; End of procedure

.LFE1493:                                 ; End of frame

    .size   main, .-main                   ; Size of main function

    .type   \_Z41\_\_static\_initialization\_and\_destruction\_0ii, @function

\_Z41\_\_static\_initialization\_and\_destruction\_0ii:

.LFB1982:                                 ; Frame begin

    .cfi\_startproc

    pushq   %rbp                           ; Push base pointer

    .cfi\_def\_cfa\_offset 16                ; Define CFA offset

    .cfi\_offset 6, -16                    ; Define base pointer offset

    movq    %rsp, %rbp                    ; Set base pointer

    .cfi\_def\_cfa\_register 6               ; Set CFA register

    subq    $16, %rsp                     ; Allocate stack space

    movl    %edi, -4(%rbp)                 ; Store first argument

    movl    %esi, -8(%rbp)                ; Store second argument

    cmpl    $1, -4(%rbp)                   ; Compare first argument with 1

    jne     .L8                           ; Jump if not equal to 1

    cmpl    $65535, -8(%rbp)              ; Compare second argument with 65535

    jne     .L8                           ; Jump if not equal to 65535

    leaq    \_ZStL8\_\_ioinit(%rip), %rdi    ; Initialize ios\_base

    call    \_ZNSt8ios\_base4InitC1Ev@PLT   ; Call ios\_base constructor

    leaq    \_\_dso\_handle(%rip), %rdx      ; Load DSO handle

    leaq    \_ZStL8\_\_ioinit(%rip), %rsi    ; Load io init address

    movq    \_ZNSt8ios\_base4InitD1Ev@GOTPCREL(%rip), %rax ; Load ios\_base destructor

    movq    %rax, %rdi                     ; Move destructor to RDI

    call    \_\_cxa\_atexit@PLT              ; Register destructor for exit

.L8:                                      ; End of initialization

    nop                                   ; No operation

    leave                                 ; Restore stack

    .cfi\_def\_cfa 7, 8                     ; Define CFA

    ret                                   ; Return

    .cfi\_endproc                          ; End of procedure

.LFE1982:                                 ; End of frame

    .size   \_Z41\_\_static\_initialization\_and\_destruction\_0ii, .-\_Z41\_\_static\_initialization\_and\_destruction\_0ii

    .type   \_GLOBAL\_\_sub\_I\_main, @function

\_GLOBAL\_\_sub\_I\_main:

.LFB1983:                                 ; Frame begin

    .cfi\_startproc

    pushq   %rbp                           ; Push base pointer

    .cfi\_def\_cfa\_offset 16                ; Define CFA offset

    .cfi\_offset 6, -16                     ; Define base pointer offset

    movq    %rsp, %rbp                    ; Set base pointer

    .cfi\_def\_cfa\_register 6              ; Set CFA register

    movl    $65535, %esi                  ; Move 65535 to ESI

    movl    $1, %edi                       ; Move 1 to EDI

    call    \_Z41\_\_static\_initialization\_and\_destruction\_0ii ; Call static initialization

    popq    %rbp                           ; Restore base pointer

    .cfi\_def\_cfa 7, 8                     ; Define CFA

    ret                                   ; Return

    .cfi\_endproc                           ; End of procedure

.LFE1983:                                 ; End of frame

    .size   \_GLOBAL\_\_sub\_I\_main, .-\_GLOBAL\_\_sub\_I\_main

    .section   .init\_array,"aw"        ; Initialization array section

    .align 8                               ; Align to 8 bytes

    .quad    \_GLOBAL\_\_sub\_I\_main           ; Add \_GLOBAL\_\_sub\_I\_main to init array

    .hidden   \_\_dso\_handle            ; Declare \_\_dso\_handle as hidden

    .ident    "GCC: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0" ; Compiler identification

    .section   .note.GNU-stack,"",@progbits ; Stack note section