

CPSC-402 Report

Compiler Construction

Anthony Walujono
Chapman University

May 19, 2022

Abstract

Short summary of purpose and content.

Contents

1	Introduction	1
2	Homework	1
2.1	Week 1: Searching for Strings	1
2.2	Week 2: Regular Expression and NFA	2
2.3	Week 3: Convert NFAs to DFAs by Hand	5
2.4	Week 4: Introduction to Parsing	6
2.5	Week 10: Proof Trees	9
3	Project	9
3.1	Explaining Assembly Code for the FahrenheitToCelsius Function.	11
3.2	Explaining Assembly Code for the Main Function.	13

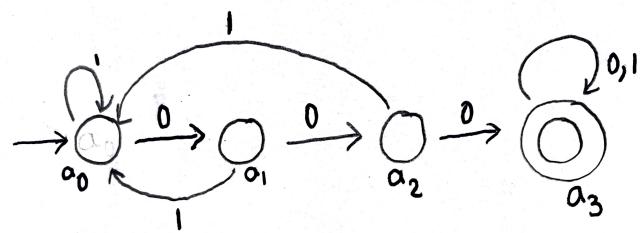
1 Introduction

This will cover everything in Compiler Construction.

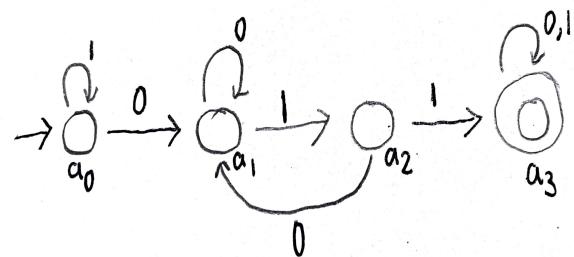
2 Homework

2.1 Week 1: Searching for Strings

- 2.24: Give DFA's accepting the the following languages over the alphabet {0,1}:
- b) The set of all strings with three consecutive 0's (not necessarily at the end).

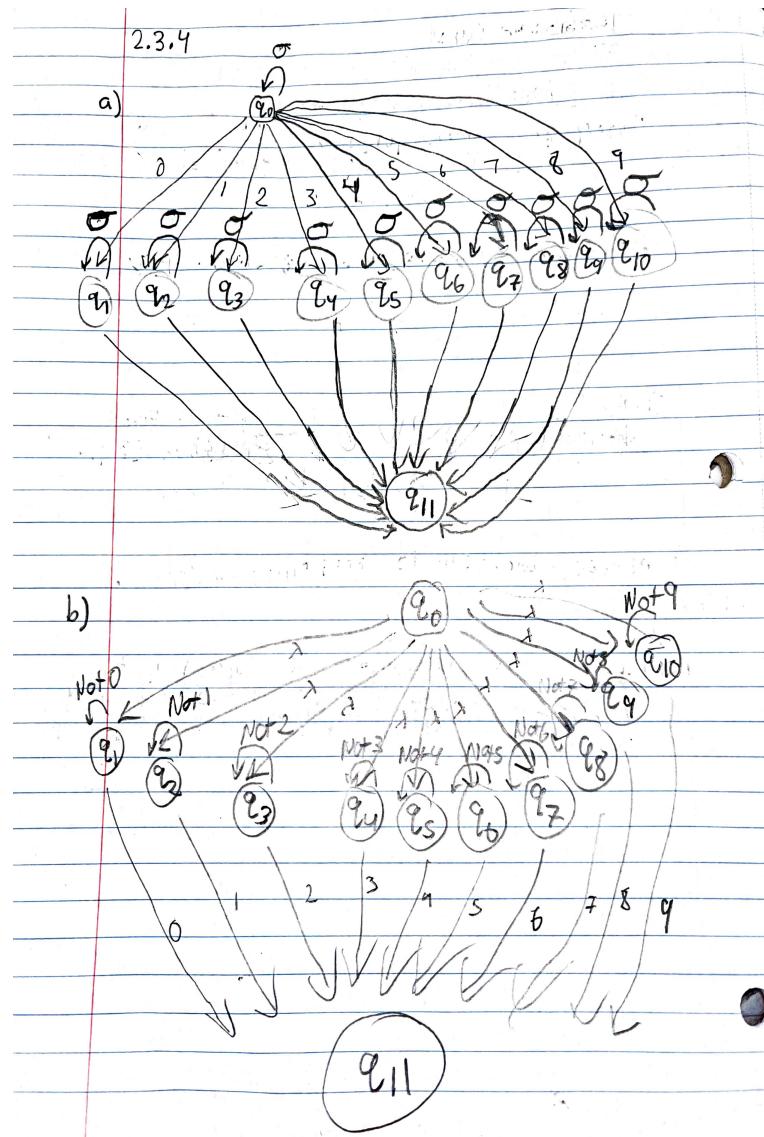


c) The set of strings with 011 as a substring.

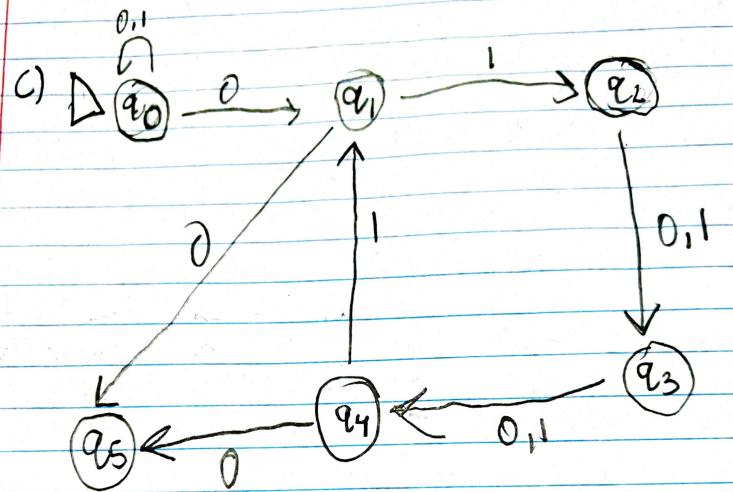


2.2 Week 2: Regular Expression and NFA

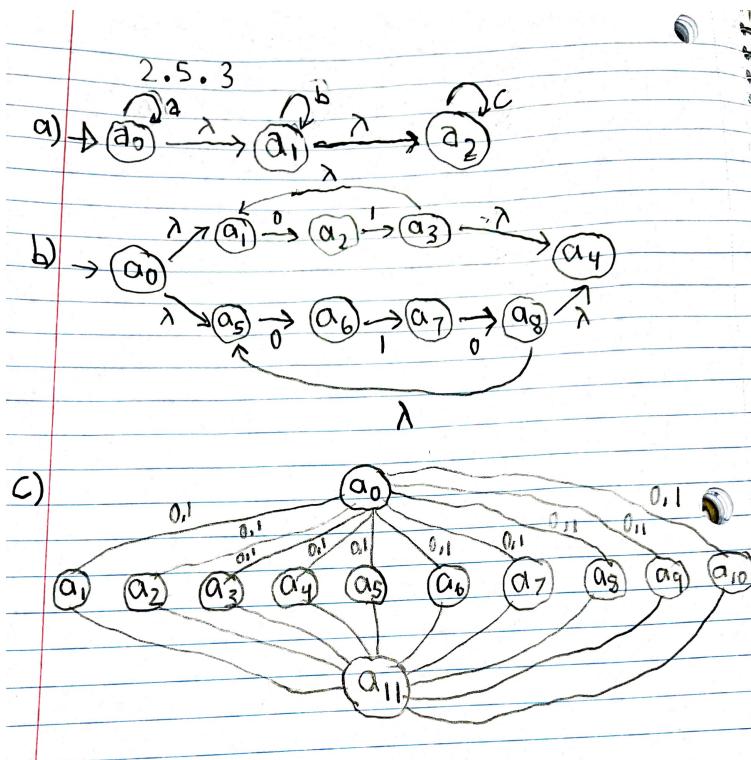
2.3.4: Give nondeterministic finite automata to accept the following languages. Try to take advantage of nondeterminism as much as possible.



2.3.4

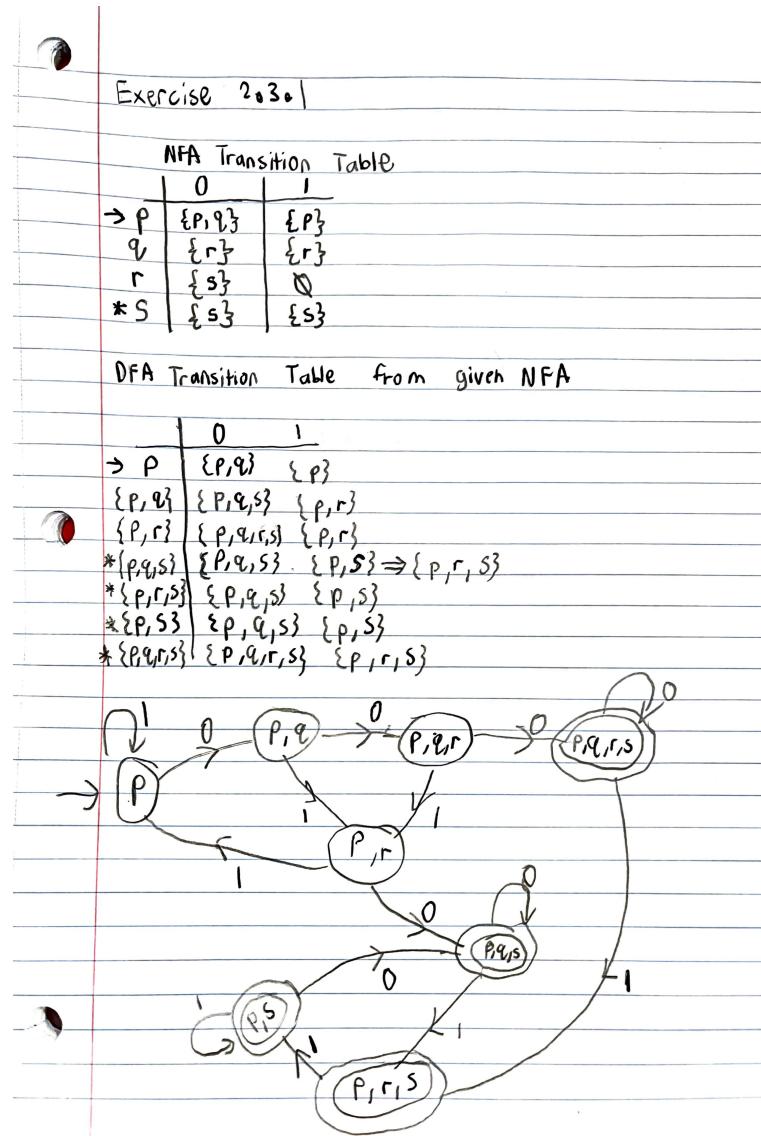


2.5.3: Design ϵ -NFA's for the following languages. Try to use ϵ transitions to simplify your design.



2.3 Week 3: Convert NFAs to DFAs by Hand

2.3.1: Convert to a DFA the following NFA



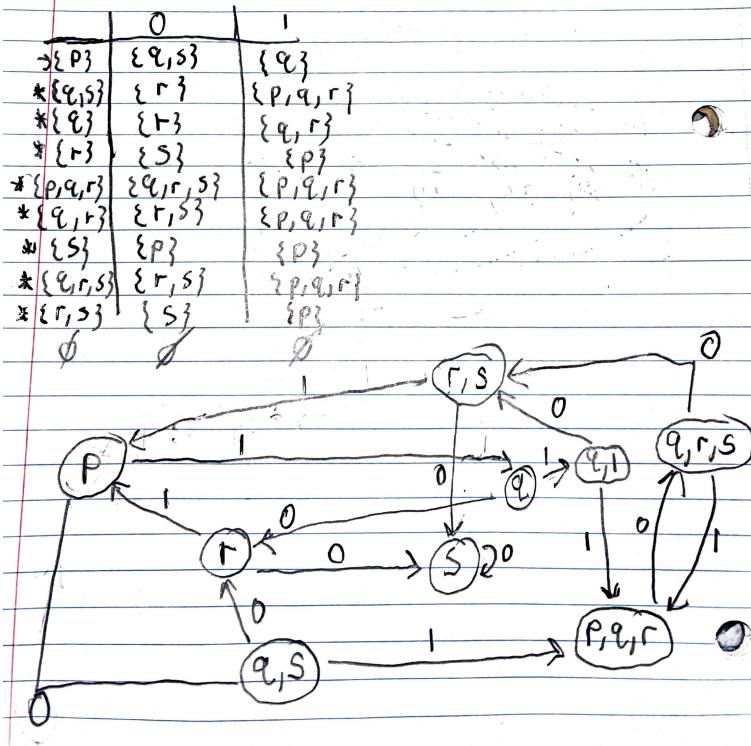
2.3.2: Convert to a DFA the following NFA

Exercise 2.3.2

NFA:

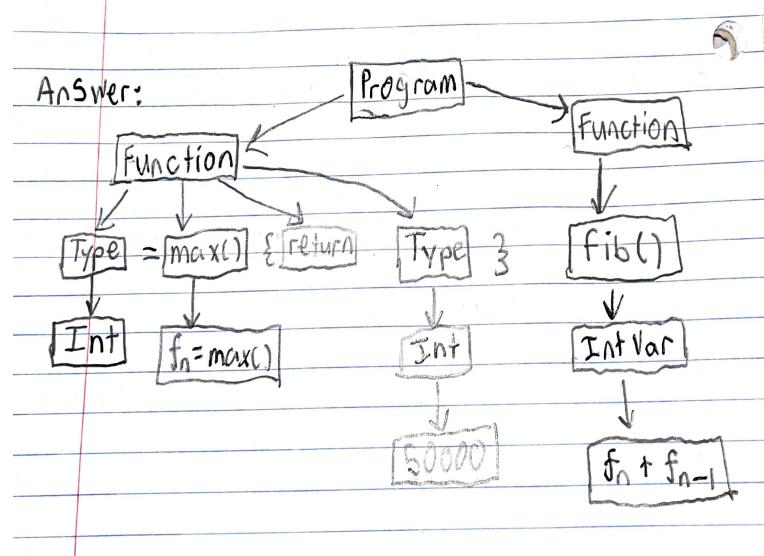
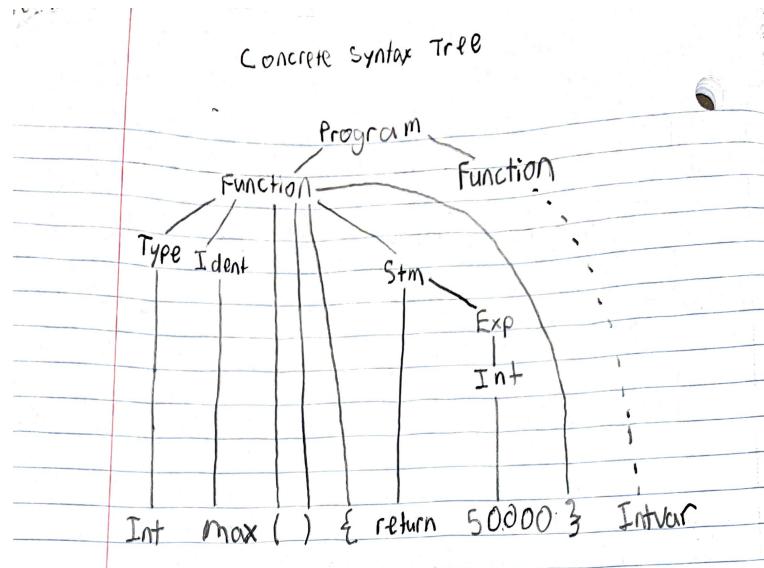
	0	1
$\rightarrow P$	$\{\epsilon, s\}$	$\{q\}$
$* q$	$\{r\}$	$\{q, r\}$
Γ	$\{s\}$	$\{p\}$
$* S$	\emptyset	$\{p\}$

NFA \rightarrow DFA

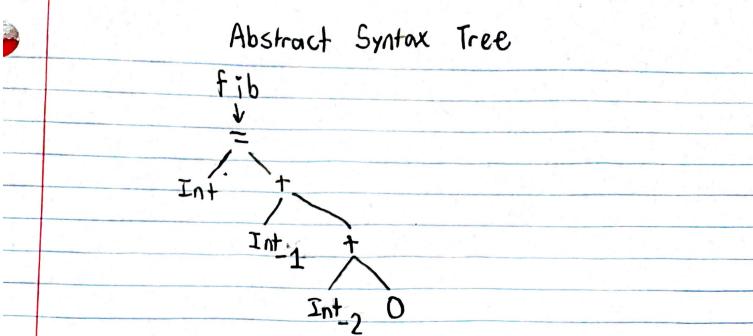


2.4 Week 4: Introduction to Parsing

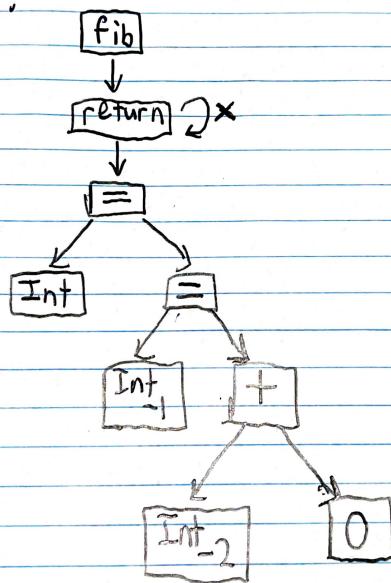
Question 1: Write out the parse tree (=concrete syntax tree) for the complete fibonacci program. Think about a question on this for the lecture.



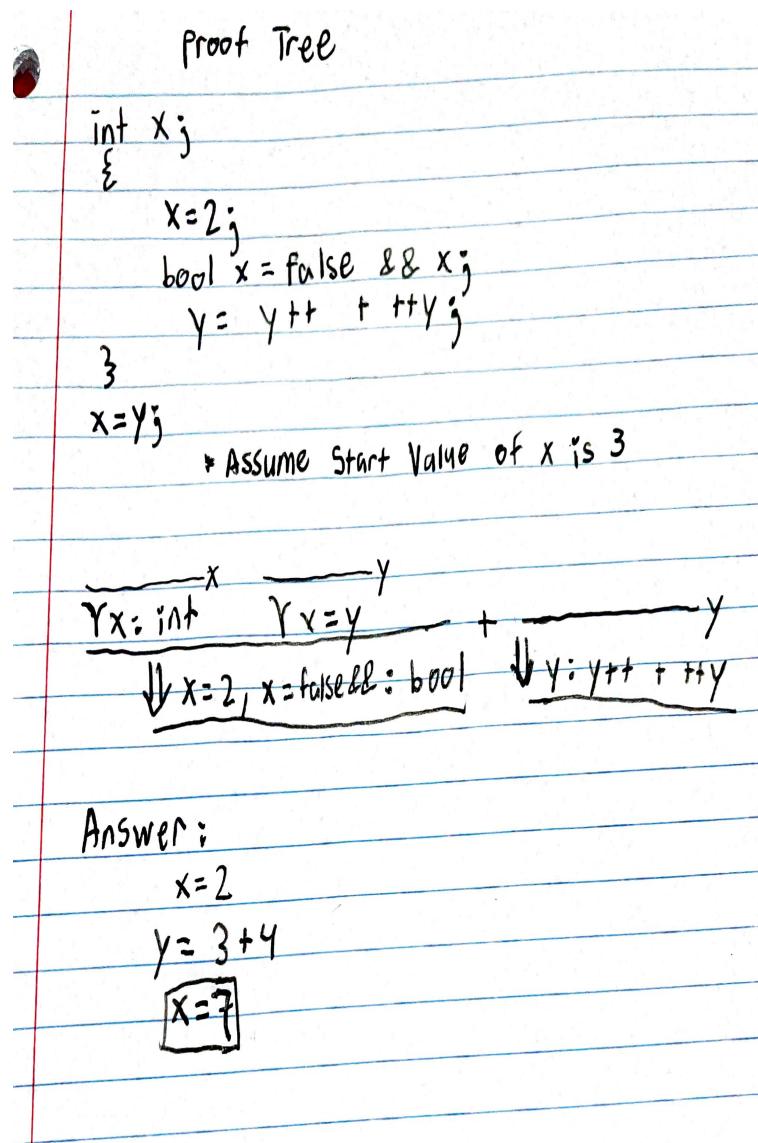
Question 2: Write out the abstract syntax tree for the complete fibonacci program.



Answer:



2.5 Week 10: Proof Trees



3 Project

For this project, I plan to explain a compiler. The language that I would use is C++, and the compiler that I would use is g++. A code that I plan to use is a C++ code that convert Fahrenheit to Celsius, which is shown below. My target language would be JVM.

```

#include <iostream>
double fahrenheitToCelsius(double fahrenheit){
    double celsius;

    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    return celsius;
}

```

```

int main(){
    double fahrenheit;

    std::cout << "Enter temperature in fahrenheit (in degrees) ";
    std::cin >> fahrenheit;
    std::cout << "Temperature in Celsius (in degrees) = "
        << fahrenheitToCelsius(fahrenheit) << std::endl;
}

```

Here is output using x86-64 gcc 12.1

```

fahrenheitToCelsius(double):
    push    rbp
    mov     rbp, rsp
    movsd  QWORD PTR [rbp-24], xmm0
    movsd  xmm0, QWORD PTR [rbp-24]
    movsd  xmm2, QWORD PTR .LC0[rip]
    movapd xmm1, xmm0
    subsd  xmm1, xmm2
    movsd  xmm0, QWORD PTR .LC1[rip]
    mulsd  xmm0, xmm1
    movsd  xmm1, QWORD PTR .LC2[rip]
    divsd  xmm0, xmm1
    movsd  QWORD PTR [rbp-8], xmm0
    movsd  xmm0, QWORD PTR [rbp-8]
    movq   rax, xmm0
    movq   xmm0, rax
    pop    rbp
    ret

.LC3:
    .string "Enter temperature in fahrenheit (in degrees) "
.LC4:
    .string "Temperature in Celsius (in degrees) = "
main:
    push    rbp
    mov     rbp, rsp
    push    rbx
    sub    rsp, 24
    mov     esi, OFFSET FLAT:.LC3
    mov     edi, OFFSET FLAT:_ZSt4cout
    call   std::basic_ostream<char, std::char_traits<char> >& std::operator<<
           <std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char
           const*)
    lea    rax, [rbp-24]
    mov    rsi, rax
    mov    edi, OFFSET FLAT:_ZSt3cin
    call   std::basic_istream<char, std::char_traits<char> >::operator>>(double&)
    mov    esi, OFFSET FLAT:.LC4
    mov    edi, OFFSET FLAT:_ZSt4cout
    call   std::basic_ostream<char, std::char_traits<char> >& std::operator<<
           <std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char
           const*)
    mov    rbx, rax
    mov    rax, QWORD PTR [rbp-24]
    movq  xmm0, rax

```

```

call    fahrenheitToCelsius(double)
movq    rax, xmm0
movq    xmm0, rax
mov     rdi, rbx
call    std::basic_ostream<char, std::char_traits<char> >::operator<<(double)
mov     esi, OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
mov     rdi, rax
call    std::basic_ostream<char, std::char_traits<char>
       >::operator<<(std::basic_ostream<char, std::char_traits<char> >&
       (*)(std::basic_ostream<char, std::char_traits<char> >&))
mov     eax, 0
mov     rbx, QWORD PTR [rbp-8]
leave
ret
__static_initialization_and_destruction_0(int, int):
push   rbp
mov    rbp, rsp
sub   rsp, 16
mov    DWORD PTR [rbp-4], edi
mov    DWORD PTR [rbp-8], esi
cmp    DWORD PTR [rbp-4], 1
jne   .L7
cmp    DWORD PTR [rbp-8], 65535
jne   .L7
mov    edi, OFFSET FLAT:_ZStL8__ioinit
call   std::ios_base::Init::Init() [complete object constructor]
mov    edx, OFFSET FLAT:_dso_handle
mov    esi, OFFSET FLAT:_ZStL8__ioinit
mov    edi, OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
call   __cxa_atexit
.L7:
nop
leave
ret
_GLOBAL__sub_I_fahrenheitToCelsius(double):
push   rbp
mov    rbp, rsp
mov    esi, 65535
mov    edi, 1
call   __static_initialization_and_destruction_0(int, int)
pop    rbp
ret
.LC0:
.long  0
.long  1077936128
.LC1:
.long  0
.long  1075052544
.LC2:
.long  0
.long  1075970048

```

3.1 Explaining Assembly Code for the FahrenheitToCelsius Function.

Here is my code for fahrenheitToCelsius function:

```
double fahrenheitToCelsius(double fahrenheit){
    double celsius;

    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    return celsius;
}
```

Here is fahrenheitToCelsius assembly code with line numbers:

```
fahrenheitToCelsius(double):
4 push    rbp
5 mov     rbp, rsp
6 movsd   QWORD PTR [rbp-24], xmm0
7 movsd   xmm0, QWORD PTR [rbp-24]
8 movsd   xmm2, QWORD PTR .LC0[rip]
9 movapd  xmm1, xmm0
10 subsd  xmm1, xmm2
11 movsd  xmm0, QWORD PTR .LC1[rip]
12 mulsd  xmm0, xmm1
13 movsd  xmm1, QWORD PTR .LC2[rip]
14 divsd  xmm0, xmm1
15 movsd  QWORD PTR [rbp-8], xmm0
16 movsd  xmm0, QWORD PTR [rbp-8]
17 movq   rax, xmm0
18 movq   xmm0, rax
19 pop    rbp
20 ret
```

Here is my explanation for the assembly code (the numbers correspond to each other).

4. Save the old base pointer so that later the base pointer value can be restored for the caller when the function returns.
5. Move the stack pointer into base pointer. This initial action maintains the base pointer to be used as a point of reference for finding parameters and local variables on the stack.
6. Move the value of the register xmm0 which is the function parameter fahrenheit to a memory address at rbp(base pointe) with offset -24 bytes.
7. the value of parameter fahrenheit is stored into a floating point register xmm0.
8. Move the value of memory address at LC0[rip], where rip pointer, which holds the value 32.0 to be stored into xmm2 register.
9. moveapd refers to Move Aligned Packed Double Precision Floating - Point value, assign register xmm1 to hold the value of register xmm0. So xmm1 = xmm0 = value of fahrenheit.
10. Subtract the value of xmm1 with the value of xmm2, this corresponds to the code fahrenheit - 32.0.
11. Move the value of memory address at LC1[rip] which holds 5.0 into xmm0 register. So xmm0 = 5.0.
12. Multiply the value of xmm0 with xmm1, the result will be in xmm0. This correspond to the value

of variable celsius being calculated as follows: celcius = (fahrenehit - 32.0) * 5

13. Move the value 9.0 (from memory address at LC2[rip]) into xmm1.

14. divide the value of xmm0 by the value of xmm1

xmm0 = celcius = (fahrenheit - 32.0) * 5.0

xmm1 = 9.0

This corresponds to the result of calculating:

celsius = ((fahrenheit - 32.0) * 5.0)/9.0

xmm0 will have the value of variable celcius.

15. The value of xmm0 (variable celsius) will be stored at memory address at rbp with offset - 8 bytes.

16. Save the value of xmm0 (= varaible celsius).

17. Move the value of xmm0 (= variable celsius) to rax register (general purpose register to hold function return value).

18. Saved the function return value in xmm0 and rax register.

19. Restore the caller's base pointer value by popping off (removing) rbp from the stack.

20. Return to the caller and remove appropriate return address from the stack.

3.2 Explaining Assembly Code for the Main Function.

Here is my code for main function:

```
18 int main()
19{
20     double fahrenheit;
21
22     std::cout << "Enter temperature in fahrenheit (in degrees) ";
23     std::cin >> fahrenheit;
24     std::cout << "Temperature in Celsius (in degrees) = "
25             << fahrenheitToCelsius(fahrenheit) << std::endl;
26 }
```

Here is main function assembly code with line numbers:

```
22 .LC3:
23     .string "Enter temperature in fahrenheit (in degrees) "
24 .LC4:
25     .string "Temperature in Celsius (in degrees) = "
26 main:
27 .LFB1746:
28     push    rbp
29     mov     rbp, rsp
30     push    rbx
31     sub    rsp, 24
32     mov     esi, OFFSET FLAT:.LC3
33     mov     edi, OFFSET FLAT:_ZSt4cout
34     call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
35     lea     rax, [rbp-8]
```

```

36      mov    rsi, rax
37      mov    edi, OFFSET FLAT:_ZSt3cin
38      call   _ZNStirsERd
39      mov    esi, OFFSET FLAT:.LC4
40      mov    edi, OFFSET FLAT:_ZSt4cout
41      call   _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
42      mov    rbx, rax
43      mov    rax, QWORD PTR [rbp-24]
44      movq  xmm0, rax
45      call   _Z19fahrenheitToCelsiusd
46      movq  rax, xmm0
47      movq  xmm0, rax
48      mov    rdi, rbx
49      call   _ZNSolsEd
50      mov    esi, OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
51      mov    rdi, rax
52      call   _ZNSolsEPFRSoS_E
53      mov    eax, 0
54      mov    rbx, QWORD PTR [rbp-8]
55      leave
56      ret

```

Here is my explanation for the assembly code (the numbers correspond to each other).

23. Assigned string value into register LC3.

.LC3 will have string "Enter temperature in fahrenheit (in degrees)"

25. Assigned string value into register LC4.

.LC4 register will have "Temperature in Celsius (in degrees) ="

28. save the old base pointer (rbp) value

29. set the new base pointer (rbp) value to be the value of stack pointer (rsp)

30. save the value of register rbx that the function will modify later.

31. make room for a 24 bytes local variable (fahrenheit);

32. move the value of register LC3 into register esi.

33. move the address of the standard library function cout into register edi.

34. call standard library function cout. This code corresponds to C++ code line 22

```
22 std::cout << "Enter temperature in fahrenheit (in degrees) ";
```

35. The value of address rbp-24 (base pointer with offset -24) is placed into register rax.

36. Move the value of register ra into register rsi.

37. Move the address of the standard library function cin into register edi.

38. Call standard library function cin. The value of register rax will be changed. Register rax will hold the

value of the input variable fahrenheit. This code corresponds to C++ code on line 23.

```
std::cin >> fahrenheit;
```

39. Move the value of register LC4 into register esi.
40. Move the address of the standard library function cout into register edi.
41. Call standard library function cout. This corresponds to C++ code line 24.

```
std::cout << "Temperature in Celsius (in degrees) = ";
```

42. Move the value of register rax(= variable fahrenheit) into register rbx.
43. Move the value at address rbp-24 (rbp offset -24 bytes) into rax register.
44. Put the value of register rax into register xmm0. This assigns registers xmm0 to hold the value of variable fahrenheit. To clarify: rax = xmm0 = fahrenheit.
45. Call the function fahrenheitToCelsius, this function will take xmm0 as parameter passed by the main function, which hold the input value of variable fahrenheit. This corresponds to C++ code line 25.

```
<< fahrenheitToCelsius(fahrenheit) << std::endl;
```

46. Move the value of register xmm0 into register rax. xmm0 holds the return value from fahrenheitToCelsius function. Register xmm0 = register rax = new value of variable fahrenheit (which is the celsius value).
47. Move register rax into register xmm0
48. Move the value of register rbx into register rdi.
49. Call the standard library cout function to display the new value of parameter fahrenheit returned by function fahrenheitToCelsius. Registers rax and xmm0 both hold the new value of fahrenheit, these registers will be accessed by the standard library cout function.
50. Move the value specified by standard library endl (end of line) into register esi.
51. Move the value of register rax into register rdi.
52. Call the standard library function endl to put end of line in the output.
53. Move 0 into register eax (initialize eax to 0)
54. Move the value of the address at rbp(base stack) with offset - 8 bytes into register rbx, which will be used for transfer control.
55. Leave the main function, pops the entire current stack and make registers available for other purposes.
56. Return pops top of stack to the calling routine, in this case end of program execution.

References

[HMU] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman: [Introduction to automata theory, languages, and computation](#), 3rd Edition. Pearson international edition, Addison-Wesley 2007