CLASS 1 HOMEWORK

18. Invoking gcc command initiates 4 steps, which are loosely referred to as ‘compilation’. The steps are described below:

    (i.) **Preprocessing**: This step produces an equivalent .c source code and performs various operations like removing comments, removes the #include header files and replaces it with the corresponding contents from the .h files, etc.

    (ii.) **Compiling**: This step turns our code into assembly code for the specific processor. This assembly code is a set of instructions for the processor to perform specific operation step by step. At this step the assembly code can still be read and programmed directly through a text editor.

    (iii.) **Assembling**: At this stage the assembler takes our assembly code/instructions into processor dependent machine level binary object code. At this point we can’t examine this with a text editor. The object code is called a relocatable.

    (iv.) **Linking**: At this step, the linker takes one or more object codes and produces a single executable code.

19. The main function’s return type is *int*. This means that the function should end with a value returned usually 0, indicating that our program terminated successfully.

21. We’re given, unsigned chars i = 60, j = 80 and k = 200. sum is also an unsigned char.

    (a.) **sum = i + j**; sum = 60 + 80 = 140.

    (b.) **sum = i + k**; sum = 60 + 200 = 4. This is because an unsigned char takes values from 0 to 255, so any value lager than 255 can’t be represented by an unsigned char and we get integer overflow.

    (c.) **sum = j + k**; sum = 80 + 200 = 24. This is because an unsigned char takes values from 0 to 255, so any value lager than 255 can’t be represented by an unsigned char and we get integer overflow.

22. We’re given,

    ```
    int a = 2, b = 3, c;
    float d = 1.0, e = 3.5, f;
    ```

    (a.) **f = a/b**; f = 2/3 = 0

    (b.) **f = ((float) a)/b**; f = 2/3 = 0.666667

    (c.) **f = (float) (a/b)**; f = 2/3 = 0

(d.) **c = e/d**; f = 2/3 = 3

(e.) **c = (int) (e/d)**; f = 2/3 = 3

(f.) **f = ((int) e)/d**; f = 2/3 = 3

27. My systematic approaching of debugging:

(i.) **What is the bug**: This would be the first step I would take, i.e. what kind of bug is it? If you know what the correct output should be, but you receive something different, depending on what you receive you can make estimated guesses on the type of bug, like is it a calculation error, typing mistake (which would usually lead to errors/warning during compiling), etc.

(ii.) **Narrow down sections of the code**: After we've narrowed down what kind of bug to look for, we can break the code down into parts to narrow down where the bug is. If you have several functions in your code and you are calling them all through main () function individually and they're exclusive, test the functionality of each function individually by hard coding value and getting correct results. You can also comment out other blocks of codes or function that you feel might disturb your test along the way.

(iii.) **Correct the bugs**: After narrowing down the bug, it's time to implement a method to correct it. Sometimes it's as easy as correcting a typing mistaking or replacing a variable, but sometimes it can get tricky on the logical front, like logics in nested loops, if-else statements, etc.

(iv.) **Line by line inspection**: After you exhausted all of your options above and beyond, then it's time to take a more meticulous approach. You should inspect the whole code starting from line 1 and examine each line and its output until you arrive at the bug.

28. Modified code is attached with the submission.

30. Given the array definition and initialization,

int [4] = {4, 3, 2, 1};

(a.) x[1] = 3

(b.) *x = 4

(c.) *(x + 2) = 2

(d.) (*x) + 2 = 6

(e.) *x[3]: "error/unknown".

(f.) x[4] : "error/unknown".

(g.) *(&(x[1]) + 1) = 2

31. Code given,

```
int i, k = 6;
i = 3*(5>1) + (k=2) + (k==6)
```

For this code, i = 5. This is because (5>1) returns 1 if the expression is true otherwise it returns 0 if it's false. So, the first term is 3*(5>1) = 3. The second term (k=2) means k will be assigned a value 2 and our expression becomes 3 + 2. Finally, the last term (k==6) takes the value 1 if it's true and 0 if it's false. Now because earlier k was assigned the value 2, this makes (k==6) false So, our expression evaluates to, i = 3 + 2 + 0 = 5. So, our output is i = 5.

32. unsigned char a = 0x0D, b = 0x03, c;

```
c = ~a;          // (a) c = 0xF2
c = a & b;       // (b) c = 0x01
c = a | b;       // (c) c = 0x0F
c = a ^ b;       // (d) c = 0x0E
c = a >> b;      // (e) c = 0x01
c = a << b;      // (f) c = 0x68
c &= b;          // (g) c = 0x00
```

34. Source code is attached with the submission.

35. Source code is attached with the submission.