Design & Implementation

Much of the design of this simple program was to manage errors and undefined behaviour in a reasonable manner. For the input in the print_fib method, values acceptable are only positive ints, meaning I need to check for negative values, as overloading an integer just leaves the integer value as -1. For characters, I was unsure of the best method to check for character invalid inputs. However, the current functionality just leaves an empty array printed which I am satisfied with.

Next, I needed to deal with undefined behaviour from overloading integers in the fib_calc functions both recursive and iterative. For the weighted fib sequence, all values are positive, leaving negative values free for use as failure codes, and if an integer is overloaded by adding two positive values together, the resulting value is always negative. Therefore, at each stage of recursion and iteration, I could check for negative values and then return another negative value, which the print_fib function could detect. Implementation in the recursive_fib file required checking for negative return values whilst getting previous values, and if found, also return a negative value which runs back up the recursive call tree to the print_fib call.

In implementing the check for overflow, I wanted to print out the valid values as I go so the user knows where exactly the sequence overflowed, whilst having the previous index values still clear.

Testing

```
Testing CS2002 W03-Exercise
- Looking for submission in a directory called 'src': Already in it!
* BUILD TEST - public/build-clean : pass
* BUILD TEST - public/stage1/build-stage1 : pass
* COMPARISON TEST - public/stage1/prog-stage1-length-0.out : pass
* COMPARISON TEST - public/stage1/prog-stage1-length-1.out : pass
* COMPARISON TEST - public/stage1/prog-stage1-length-10.out : pass
* COMPARISON TEST - public/stage1/prog-stage1-length-3.out : pass
* BUILD TEST - public/stage2/build-stage2 : pass
* COMPARISON TEST - public/stage2/prog-stage2-length-0.out : pass
* COMPARISON TEST - public/stage2/prog-stage2-length-1.out : pass
* COMPARISON TEST - public/stage2/prog-stage2-length-10.out : pass
* COMPARISON TEST - public/stage2/prog-stage2-length-3.out : pass
11 out of 11 tests passed
```

Here is the stacscheck output for my program, in which all passed.

```
[george@thinker src]$ ./stage1
Length? 98938129038129839012831239381293129O3
Invalid input
[george@thinker src]$ ./stage2
Length? 98938129038129839012831239381293129O3
Invalid input
```

I also added in tests for overflow behaviour when entering an integer too large for an integer to hold. This lead to an invalid input message being printed as expected for stage 1 and 2.

```
[george@thinker src]$ ./stage1
Length? -10
Invalid input
[george@thinker src]$ ./stage2
Length? -10
Invalid input
```

This test shows invalid input when negative values are entered which prints the invalid input message as specified for both stage 1 and 2.

```
[george@thinker src]$ ./stage1
Length? 50
[0, 1, 1, 3, 5, 11, 21, 43, 85, 171, 341, 683, 1365, 2731, 5461, 10923, 21845,
43691, 87381, 174763, 349525, 699051, 1398101, 2796203, 5592405, 11184811, 2236
9621, 44739243, 89478485, 178956971, 357913941, 715827883, 1431655765, Overflow
[george@thinker src]$ ./stage2
Length? 50
[0, 1, 1, 3, 5, 11, 21, 43, 85, 171, 341, 683, 1365, 2731, 5461, 10923, 21845,
43691, 87381, 174763, 349525, 699051, 1398101, 2796203, 5592405, 11184811, 2236
9621, 44739243, 89478485, 178956971, 357913941, 715827883, 1431655765, Overflow
```

The next test was to check for overflow behaviour when the input was a valid int, but the calculations lead to an overflow on integers. This lead to printing up until the overflow was reached as desired for stage 1 and 2.