Alan S. Thomas

CSI 130-01CA

Pg. 4-34: 1,3,4,5,7,9,12,14,15,18

09/15/2014

1.) Four types of data; numeric, character, image, and sound.

3.) Data hierarchy highest to lowest: Database, file, record, field, and character.

4.) Capacity is measured in the amount of 1s and 0s a computer can hold.

5.) 8,388,608 bits in one megabyte.

7.) 1,048,576 TB in an Exabyte.

9.) There are 1,000 zettabytes in 1 Yotabyte.

12.) int num[256][8];

14.) int n[10], i = 0;

i = 0;

for (i = 0; i<10; i = i + 1)

{

n[i]=i;

}

//print indices in reverse order

i = 9;

for (i = 9; i >= 0; i = i - 1)

{

cout << n[i] << endl;

}

15.) See attached paperwork.

18.)

Array a

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| P | A | S | C | A | L |

|  |  |
| --- | --- |
| Index | i |
| ~~-1~~  ~~1~~  4 | ~~0~~  ~~1~~  ~~2~~  ~~3~~  ~~4~~  5 |

Alan Thomas

CSI 130 – 01CA

Pg4-34:Problem# 15

09/28/2014

Five Steps of Problem Solving for Problem Number Fifteen

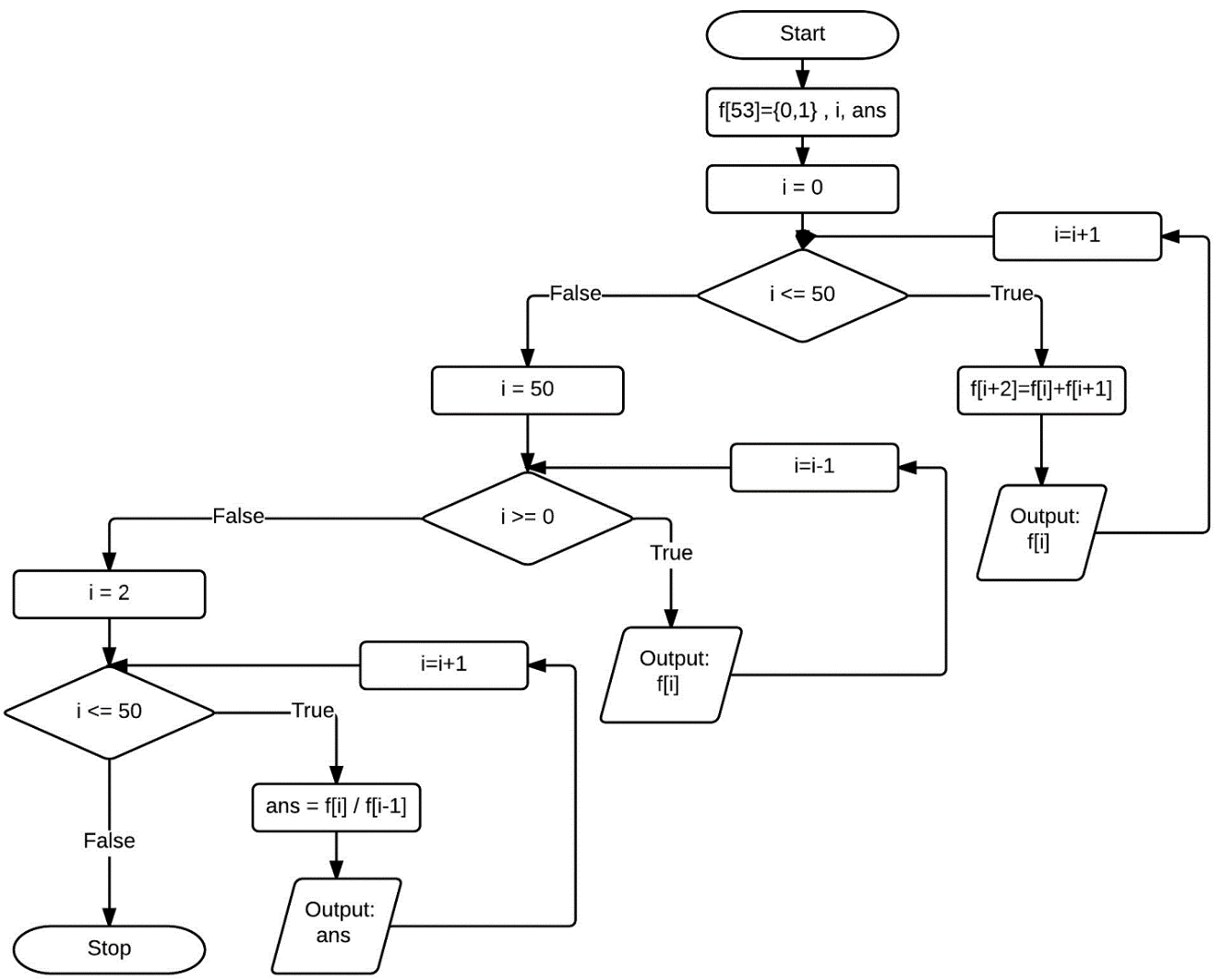
**1.) Understand the Problem:**

When first confronted with the problem I had quite a few issues to resolve. I understood what was being asked of us. However, I wasn’t sure how to develop a solution to the problem. One issue being the fact I haven’t taken a math class in over ten years. So, I set out to learn more about Fibonacci numbers. It was during my reading that I saw a lot of similaraties between using arrays and how Fibonacci numbers work. Once I saw the formula to determine Fibonacci numbers it was more clear how to implement it into code.

**2.) Develop a Solution:**

Our solution requires us to print Fibonacci numbers till the fiftieth number, print those numbers in reverse order, than print the results of dividing Fibonacci numbers by the number that proceeded them. Using Fibonacci’s formula as a base of our solution we were able to develop an algorithm which fulfills the requirements of the problem.

Fibonacci Program Flowchart



**3.) Verify Solution:**

Following the flowchart from the previous page, I was able to trace the code to ensure that in fact, the algorithm works as intended.

Flowchart Trace

|  |  |  |
| --- | --- | --- |
| i | ans | Output |
| ~~0~~  ~~1~~  ~~2~~  ~~3~~  ~~4~~  ~~5~~  ~~6~~  ~~7~~  ~~7~~  ~~6~~  ~~5~~  ~~4~~  ~~3~~  ~~2~~  ~~1~~  ~~0~~  ~~2~~  ~~3~~  ~~4~~  ~~5~~  ~~6~~  7 | ~~1~~  ~~2~~  ~~1.5~~  ~~1.66667~~  ~~1.6~~  1.625 | 0  1  1  2  3  5  8  13  13  8  5  3  2  1  1  0  1  2  1.5  1.66667  1.6  1.625 |

Array f

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 |

**4.) Implement Best Solution:**

Using the flowchart, I wrote the code to implement the algorithm that was developed in the planning phase. The initial code was consistent with Fibonacci’s original formula he developed. Some changes were made in an effort to improve the readability of the output. For instance, copying results into a separate float array from the Fibonacci array which was an integer. This allowed for the printing of numbers in plain form rather than scientific notation. For the subsequent requirements, minor changes were made to the code. The following page contains the code.

**4.) Implement Best Solution Con’t:**

//Alan Thomas CSI130-01CA p4-30 Problem#15

#include <iostream>

//#include <iomanip>

using namespace std;

int main()

{

//using long long to eliminate garbage in last fib numbers

//array size set to 53 to eliminate out of bounds error.

//Used 2 arrays for the fib numbers to increase readability. Eliminated scientific notation and provided float for division.

int long long f[53] = { 0, 1 }, i, j;

float ans = 0, h[53] = { 0, 1 };

//fib sequence begins

i = 0;

j = 0;

for (i = 0; i <= 50; i = i + 1)

{

f[i + 2] = f[i] + f[i + 1];

h[i + 2] = h[i] + h[i + 1];

cout << "Fib#" << j <<": "<< f[i] << endl;

j = j + 1;

}

//fib numbers in reverse order

cout << endl << endl << endl << "Fibonacci numbers in reverse:"<<endl;

i = 50;

j = 50;

for (i = 50; i >= 0; i = i - 1)

{

cout << "Fib#" << j << ": " << f[i] << endl;

j = j - 1;

}

//dividing fib numbers

cout << endl << endl << endl << "Fibonacci numbers divided:" << endl;

i = 2;

for (i = 2; i <= 50; i = i + 1)

{

ans = h[i]/h[i-1];

//cout << setprecision(30);

//Precision is set to 6 digits by default. Uncommenting the header and setprecision would allow for more digits in this division problem.

cout << ans << endl;

}

return 0;

}

**5.) Test the Solution**:

The strategy I used to test the program was to test pieces independently once I finished coding it. Meaning I ensured the first while loop functioned as intended before continuing to the next while loop. However, during testing I realized some tweaking was required to ensure the code functioned as intended. Originally I used a normal length integer. Doing so presented garbage as the output during the last few numbers of the sequence. To remedy this I had to expand the length that was allowed. Adding ‘long long’ to the integer I was able to remedy this. A buffer overflow was also problematic as the array was attempting to reach indices which did not exist. By increasing the length of the array by three I was able to compensate for the out of bounds error. During the testing phase I also realized that the output of the larger numbers was being represented in scientific notation. Though not problematic, it wasn’t aesthetically pleasing. I created a second array named ‘h’ which I applied the same algorithm to as I did for the original Fibonacci array. Using an integer for the first portion of the code provided standard number formatting. Using a float for the division provided the ability to see the subsequent decimal places. Though, it wasn’t entirely necessary I felt it provided a friendlier user experience.