CS2002 Week 3 Practical

Introduction to C

8th February 2021

Matriculation Number: 200002548

Tutor: Marco Caminati

Contents

1 Introduction

The aim of this practical was to introduce myself to C programming. By the end of the practical I should be able to write a simple multi source file program and write a makefile to build it. The program I was to create was to simply calculate the weighted Fibonacci sequence. The weighted Fibonacci sequence starts with 0, 1 and then each term after is generated using this formula:

$$x_n = x_{n-1} + 2 \cdot x_{n-2}$$

2 Design and Implementation

Design and implementation followed from the specification. The makefile was built to include two different build targets `stage1` and `stage2`. The first stage built an executable from the `recursive_fib.c` file whereas the second stage was built using the `iterative_fib.c` file. The makefile also included a `clean` target which removed the executables (`stage1` and `stage2`) as well as all `.o` object files. The executables `stage1` and `stage2` can be built using `make stage1` and `make stage2` respectively. The recursive approach was straightforward, with the `fibcalc` function being recursively called until the base case where $n < 2$, (i.e $n = 0$ or 1) is reached. The iterative approach was slightly harder, with three variables (`res`, `i1` and `i2`) needed to keep track of values so that none are recomputed. The sequence is then calculated iteratively using the `i1` and `i2` variables to store the two previous results. The function then returns the result unless $n = 1$, whereby 1 is returned instead.

The program has included some optional error checking by displaying `Invalid input` and exiting when the length entered is negative. The variables used to store the result of the Fibonacci sequence were given the type `long long` so that they can store integers up to a size of 64 bits. Furthermore a `safe_add` function has been provided within an extra `safe_maths.c` file and is included in both the recursive and iterative files (however technically this is not needed within the recursive definition since it is very inefficient and would take an astronomically huge amount of time to overflow). This `safe_add` function uses the `limits.h` script from the standard library to get the maximum and minimum values that can be stored in the `long long` type. If when the two `long long` operands would overflow when they are combined then the `safe_add` function would display `Overflow` and exit from the program. The `safe_maths.h` and `safe_maths.c` are included within makefile and are built alongside the rest of the program.

3 Testing

Testing was performed using stacscheck and manually bound checking both `stage1` and `stage2`. The results are shown below.

```
Testing CS2002 W03-Exercise
- Looking for submission in a directory called 'src': found in current directory
* 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
```

Figure 1: Stacscheck output

```
[user@arch src]$ ./stage1
Length? 7
[0, 1, 1, 3, 5, 11, 21]
[user@arch src]$ ./stage2
Length? 7
[0, 1, 1, 3, 5, 11, 21]
```

```
[user@arch src]$ ./stage1
Length? 0
[]
[user@arch src]$ ./stage2
Length? 0
[]
```

```
[user@arch src]$ ./stage1
Length? 1
[0]
[user@arch src]$ ./stage2
Length? 1
[0]
```

```
[user@arch src]$ ./stage1
Length? 2
[0, 1]
[user@arch src]$ ./stage2
Length? 2
[0, 1]
```

```
[user@arch src]$ ./stage1
Length? -1
Invalid input
[user@arch src]$ ./stage2
Length? -1
Invalid input
```

```
[user@arch src]$ ./stage1
Length? 30
[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, 22369621, 44739243,
89478485, 178956971]
[user@arch src]$ ./stage2
Length? 30
[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, 22369621, 44739243,
89478485, 178956971]
```

```
[user@arch src]$ ./stage2
Length? 66
[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, 22369621, 44739243,
89478485, 178956971, 357913941, 715827883, 1431655765, 2863311531, 5726623061, 11453246123,
22906492245, 45812984491, 91625968981, 183251937963, 366503875925, 733007751851, 1466015503701,
2932031007403, 5864062014805, 11728124029611, 23456248059221, 46912496118443, 93824992236885,
187649984473771, 375299968947541, 750599937895083, 1501199875790165, 3002399751580331,
6004799503160661, 12009599006321323, 24019198012642645, 48038396025285291, 96076792050570581,
192153584101141163, 384307168202282325, 768614336404564651, 1537228672809129301,
3074457345618258603, 6148914691236517205, 12297829382473034411]
```

```
[user@arch src]$ ./stage2
Length? 67
[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, 22369621, 44739243,
89478485, 178956971, 357913941, 715827883, 1431655765, 2863311531, 5726623061, 11453246123,
22906492245, 45812984491, 91625968981, 183251937963, 366503875925, 733007751851, 1466015503701,
2932031007403, 5864062014805, 11728124029611, 23456248059221, 46912496118443, 93824992236885,
187649984473771, 375299968947541, 750599937895083, 1501199875790165, 3002399751580331,
6004799503160661, 12009599006321323, 24019198012642645, 48038396025285291, 96076792050570581,
192153584101141163, 384307168202282325, 768614336404564651, 1537228672809129301,
3074457345618258603, 6148914691236517205, 12297829382473034411,
Overflow
```

Figure 2: Comprehensive tests

From testing it can be seen that the program performs exactly as tested and produces the correct error messages for negative lengths and overflows.

4 Conclusion

During this practical I greatly increased my understanding of C programming. Having moved over from C++ I had a decent understanding of header files, and the compilation and linking process which helped during this practical. Creating a working makefile that included all of the correct files proved challenging since before I have only directly used `gcc` from the terminal. I enjoy programming in a lower level language and being left to check for overflows rather than working with Java and the JVM.

References

[1] Shivam Chauhan, *How do you format an unsigned long long int using printf?*, 13-05-2015, accessed 08-02-2021,
`https://stackoverflow.com/a/30221946/7711225`

[2] Gareth Rees, *Simple method to detect int overflow*, 11-12-2013, accessed 08-02-20201,
`https://codereview.stackexchange.com/a/37178`