HPPS assignment 2

August Pallesen, Sky Wong og Alexander Husted

December 2023

Indhold

1 Introduction **2**

2 Tasks **2**

2.1 Implement read_uint_be() and write_uint_be() 2

2.2 Implement read_double_bin() and write_double_bin() 2

2.3 Implement write_double_ascii() 2

2.4 Implement read_double_ascii() 2

2.5 Implement avg_doubles 2

3 Questions **3**

3.1 3

3.2 3

3.3 3

3.4 3

1 Introduction

In general the quality of our functions are good when used properly. We have used assertions, which means that we will get errors for better handling.
One of the tests can be run with our makefile using the following command:

```
make
```

and

```
echo 1.2 2.3 3.4 4.5 5.6 6.7 | ./double_ascii_to_double_bin | ./avg_doubles
```

then you should get: 3.950000
Or

```
echo 123.456 | ./double_ascii_to_bin | ./double_bin_to_ascii
```

then you should get: 123.456000

2 Tasks

2.1 Implement read_uint_be() and write_uint_be()

Functionality
The functions read_uint_be() and write_uint_be() both works correctly, as long as the input doesn't exceed the byte limit. (Passes test)
Optimization
For a function with such a niche utilization, we see no room for improvement.

2.2 Implement read_double_bin() and write_double_bin()

Functionality
The functionality for read_double_bin() is correct. (Passes test)
Optimization
Since we didn't implement one if the functions correctly, we certainly have room for improvement here.

2.3 Implement write_double_ascii()

Functionality
The function works as it should as long as the inout stays within the limit of ascii characters. (Passes test)
Optimization
We see no room for improvement.

2.4 Implement read_double_ascii()

Functionality
The function works correctly. (Passes test)
We convert from ASCII charater into an integer by subtracting $'0' = 48$ such that $(50 - \textbf{'0'}) = 2$
Optimization
We see no room for improvement.

2.5 Implement avg_doubles

Functionality
The function works correctly. (Passes test)
Optimization
The function only works for values of type double_bin. This could be improved by making it more generic or by making an assertions for the input type.

3 Questions

3.1

The ASCII integer format are used human interpretation with different characters to represent different elements, and the binary integer format is contained within a more compact format hence it being binary. As a result of this we often see that the data files in the ASCII integer format are larger that the binary integer format.

3.2

Along the lines of the previous answer wee see that the ASCII floating-point format tends to be larger than the binary floating-point format. This is due to the fact that the ASCII floating-point format spends more storage on overhead. While the binary floating-point format only needs 4 bytes to write a 32 bit single precision floating point number, the ASCII floating-point format might need more to express the a decimal point or other characters.

3.3

This really depends on the input. If the input is not divisible by 4, it necessarily must be of ASCII integer format. However if the input indeed is divisible by 4, then we don't have a way of telling which format the input is of.

3.4

We have no limitations in expressing floating point values in ASCII. This means that everything that we cant express in binary floating-point format can also be expressed in ASCII floating-point format.