System program - Assignment 1

SFP

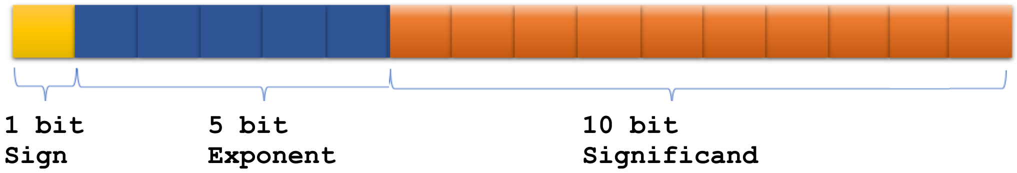
The purpose of this assignment is to become familiar with data representation of computer systems, especially about floating point numbers. You will design and implement **16-bit floating** point type, so called here *small* floating point or **sfp** that is compatible with most of IEEE standard 754 behaviors (explained in the class material 02_3_float.pdf); recall that you've learned about 32 bits (*single* precision) and 64 bits (*double* precision) standards.

In this assignment, you will implement the type cast and the addition operation of **sfp** that are defined in the below specification section. It is not permitted to use any library that is specifically designed for such different floating point types.

The format of **sfp** contains

- bit sign (s),
- 5-bit exponent (exp),
- 10-bit significand (frac).

The format of **sfp** is laid out as follows.



1. Specification of sfp :

In C, **sfp** is represented as below.

```
typedef unsigned short sfp;
```

In order to utilize such a newly implemented data type, it is required to design and implement functions supporting conversion with several conventional data types. To do so, you will implement the following four type-cast functions (type conversion with **int** and **float**). In addition, you will also implement one arithmetic operation, the addition function.

```
/* convert int into sfp */
sfp int2sfp(int input);
/* convert sfp into int */
int sfp2int(sfp input);
/* convert float into sfp */
sfp float2sfp(float input);
/* convert sfp into float */
float sfp2float(sfp input);
sfp sfp_add(sfp in1, sfp in2);
```

You will also implement the following bit stream function so that you will be able to return the bit stream of **sfp** data for the result evaluation.

```
char* sfp2bits(sfp result);
```

2. Detail specification of sfp :

sfp int2sfp(int input), sfp float2sfp(float input)

- These functions are used to convert **int** data type, and float data type into **sfp** data type, respectively. The return data type is **sfp**.
- For the value which exceeds the range of **sfp** (overflow), mark the result as ±ꝏ. (The sign must be ensured clearly. Note that +ꝏ and -ꝏ are different.)
- Use round-to-zero as rounding mode.
- For **int** 0, mark the result as **sfp** +0.0

Input (int, float)	**Output (sfp)**
>maximum of sfp	+ꝏ
<minimum of sfp	-ꝏ
int 0	+0.0
Round-to-zero mode	

Special result values for int2sfp and float2sfp

int sfp2int(sfp input)

- This function is used to convert **sfp** data type into **int** data type. The return data type is **int**.
- +ꝏ and -ꝏ is represented as TMax and TMin of **int**, respectively.
- NaN is converted into TMin.
- Use round-toward-zero as rounding mode.

Input (sfp)	**Output (int)**
+ꝏ	TMax
-ꝏ	TMin
±NaN	TMin
Round-to-zero mode	

Special result values for sfp2int

float sfp2float(sfp input)

- This function is used to convert **sfp** data type into **float** data type. The return data type is **float**.
- Note that there is no exception or error cases since **float** type is capable of covering all the value range of **sfp**.

sfp sfp_add(sfp in1, sfp in2)

- Two **sfp** variables are given as inputs. The result is a **sfp** data type value representing the sum of the inputs.
- For the result which **exceeds** the range of **sfp** (overflow), mark the result as infinity. (the sign must be ensured clearly)
- Use round-to-even rounding mode.
- Casting **sfp** to **float** or **double** for this addition are prohibited in the function. Manipulating the bits of the two **sfp** variables is required.

in1	in2	result
+ꝏ	+ꝏ	+ꝏ
+ꝏ	-ꝏ	NaN
+ꝏ	Normal Value	+ꝏ
-ꝏ	-ꝏ	-ꝏ
-ꝏ	Normal Value	-ꝏ
NaN	Any Value	NaN

Special result values for sfp_add

sfp sfp_mul(sfp in1, sfp in2)

- Two **sfp** variables are given as inputs. The result is a **sfp** data type value representing the multiplication of the inputs.
- For the result which **exceeds** the range of **sfp** (overflow), mark the result as infinity. (the sign must be ensured clearly)
- Use round-to-even rounding mode.
- Casting **sfp** to **float** or **double** for this multiplication are prohibited in the function. Manipulating the bits of the two **sfp** variables is required.

in1	in2	result
+∞	+∞	+∞
+∞	-∞	-∞
-∞	-∞	+∞
+∞	Normal Value	+∞
-∞	Normal Value	-∞
±∞	0	NaN
NaN	Any Value	NaN

Special result values for sfp_mul

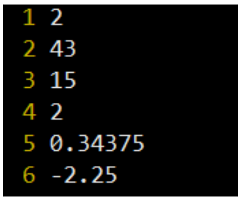
char* sfp2bits(sfp result)

- This function is used to return the bit stream of **sfp** data type. (e.g. if **sfp** val means 15 in decimal, char *string=sfp2bits(val); has “0100101110000000”)
- Use malloc() in the function for returning a string. The returned string can be freed by the caller of sfp2bits().

3. Example :

Each line of an input file contains only a number. The first line denotes the number of inputs as **int**. Given the number of inputs, the following lines include **int** type inputs. Same format of input lines is followed for the case of **float** type inputs.

Input file



```
2
43
15
2
0.34375
-2.25
```

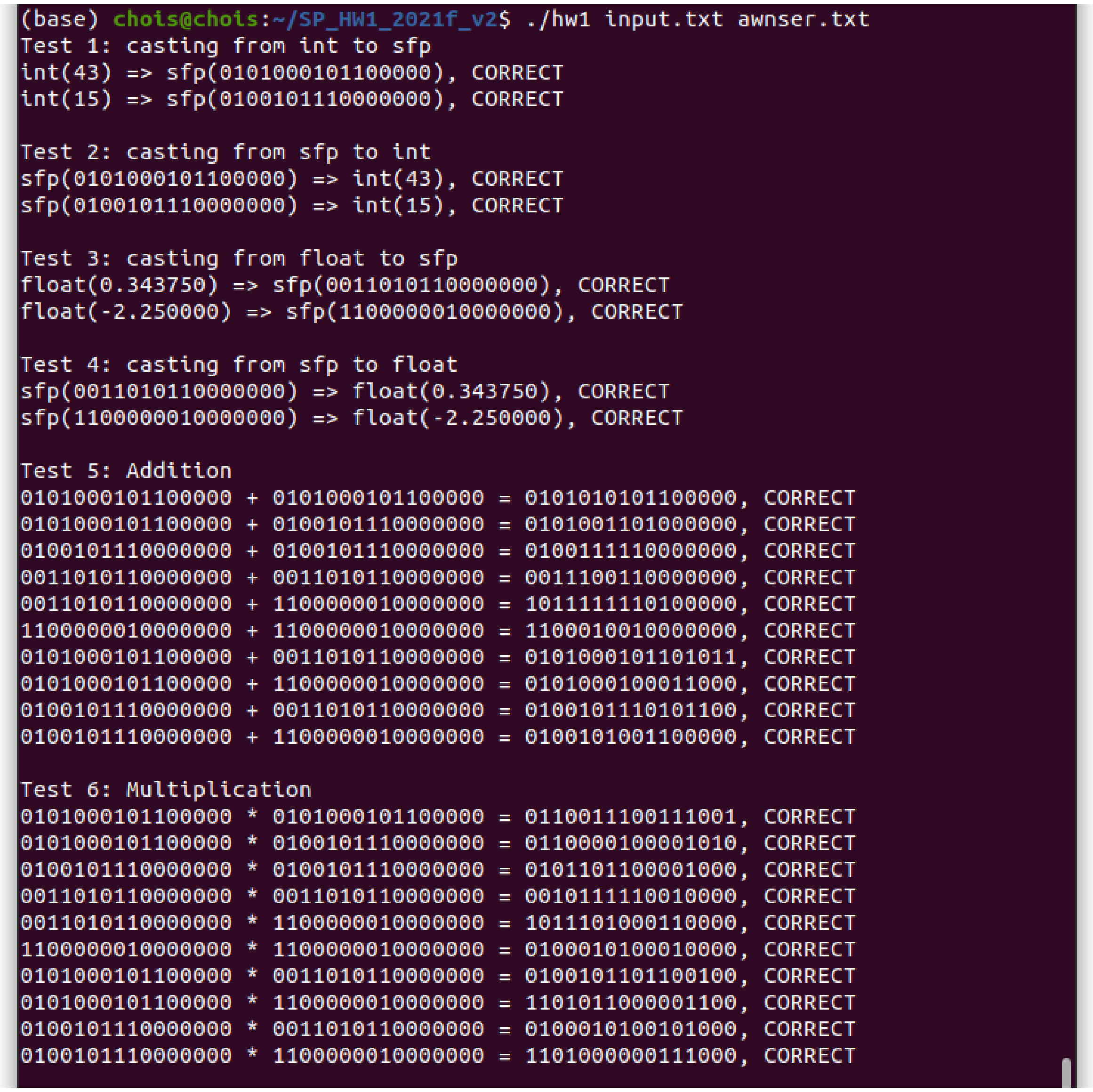
Execution result

4. Note

- Skeleton code is given in **sfp.c**. Input cases and scoring system will be implemented by TA.
- **Include a pdf file (or doc file) that explains your design and code in your sfp.c file.** The file name is studentid.pdf.
- After implementing all functions, type “make” on your terminal, and execute “hw1”.
- Zip all the files relevant to this assignment such as **sfp.c, sfp.h, hw1.c, Makefile, studentid.pdf**. The zipped

file name must have a form of “**studentid.tar**” (e.g. 2017719486.tar).

- Submit your assignment by uploading the zipped file to icampus.
- If **plagiarism** is detected, 0 point will be given and additional penalty will be considered.



```
(base) chois@chois:~/SP_HW1_2021f_v2$ ./hw1 input.txt awnser.txt
Test 1: casting from int to sfp
int(43) => sfp(0101000101100000), CORRECT
int(15) => sfp(0100101110000000), CORRECT

Test 2: casting from sfp to int
sfp(0101000101100000) => int(43), CORRECT
sfp(0100101110000000) => int(15), CORRECT

Test 3: casting from float to sfp
float(0.343750) => sfp(0011010110000000), CORRECT
float(-2.250000) => sfp(1100000010000000), CORRECT

Test 4: casting from sfp to float
sfp(0011010110000000) => float(0.343750), CORRECT
sfp(1100000010000000) => float(-2.250000), CORRECT

Test 5: Addition
0101000101100000 + 0101000101100000 = 0101010101100000, CORRECT
0101000101100000 + 0100101110000000 = 0101001101000000, CORRECT
0100101110000000 + 0100101110000000 = 0100111110000000, CORRECT
0011010110000000 + 0011010110000000 = 0011100110000000, CORRECT
0011010110000000 + 1100000010000000 = 1011111110100000, CORRECT
1100000010000000 + 1100000010000000 = 1100010010000000, CORRECT
0101000101100000 + 0011010110000000 = 0101000101101011, CORRECT
0101000101100000 + 1100000010000000 = 0101000100011000, CORRECT
0100101110000000 + 0011010110000000 = 0100101110101100, CORRECT
0100101110000000 + 1100000010000000 = 0100101001100000, CORRECT

Test 6: Multiplication
0101000101100000 * 0101000101100000 = 0110011100111001, CORRECT
0101000101100000 * 0100101110000000 = 0110000100001010, CORRECT
0100101110000000 * 0100101110000000 = 0101101100001000, CORRECT
0011010110000000 * 0011010110000000 = 0010111110010000, CORRECT
0011010110000000 * 1100000010000000 = 1011101000110000, CORRECT
1100000010000000 * 1100000010000000 = 0100010100010000, CORRECT
0101000101100000 * 0011010110000000 = 0100101101100100, CORRECT
0101000101100000 * 1100000010000000 = 1101011000001100, CORRECT
0100101110000000 * 0011010110000000 = 0100010100101000, CORRECT
0100101110000000 * 1100000010000000 = 1101000000111000, CORRECT
```