

Decimal	Binário Inteiro complemento de 2	Binário Real Ponto Flutuante (IEEE 754) (polarização = 011111112 =12710).
+23	000000000000000000000000010111	0x00000017
0	000000000000000000000000000000	0x00000000
NaN	NaN	0xFFFFFFFF8

Questão 2.

2 A:

```

1  #include <stdio.h>
2
3  int main(){
4      float x, y;
5      x = 0.1;
6
7      for(int i = 0; i <= 99; i++){
8          y += x;
9      }
10     printf("%f\n", y);
11     return 0;
12 }

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL Code

```

tempCodeRunnerFile
/bin/sh: 1: g++: not found

[Done] exited with code=127 in 0.007 seconds

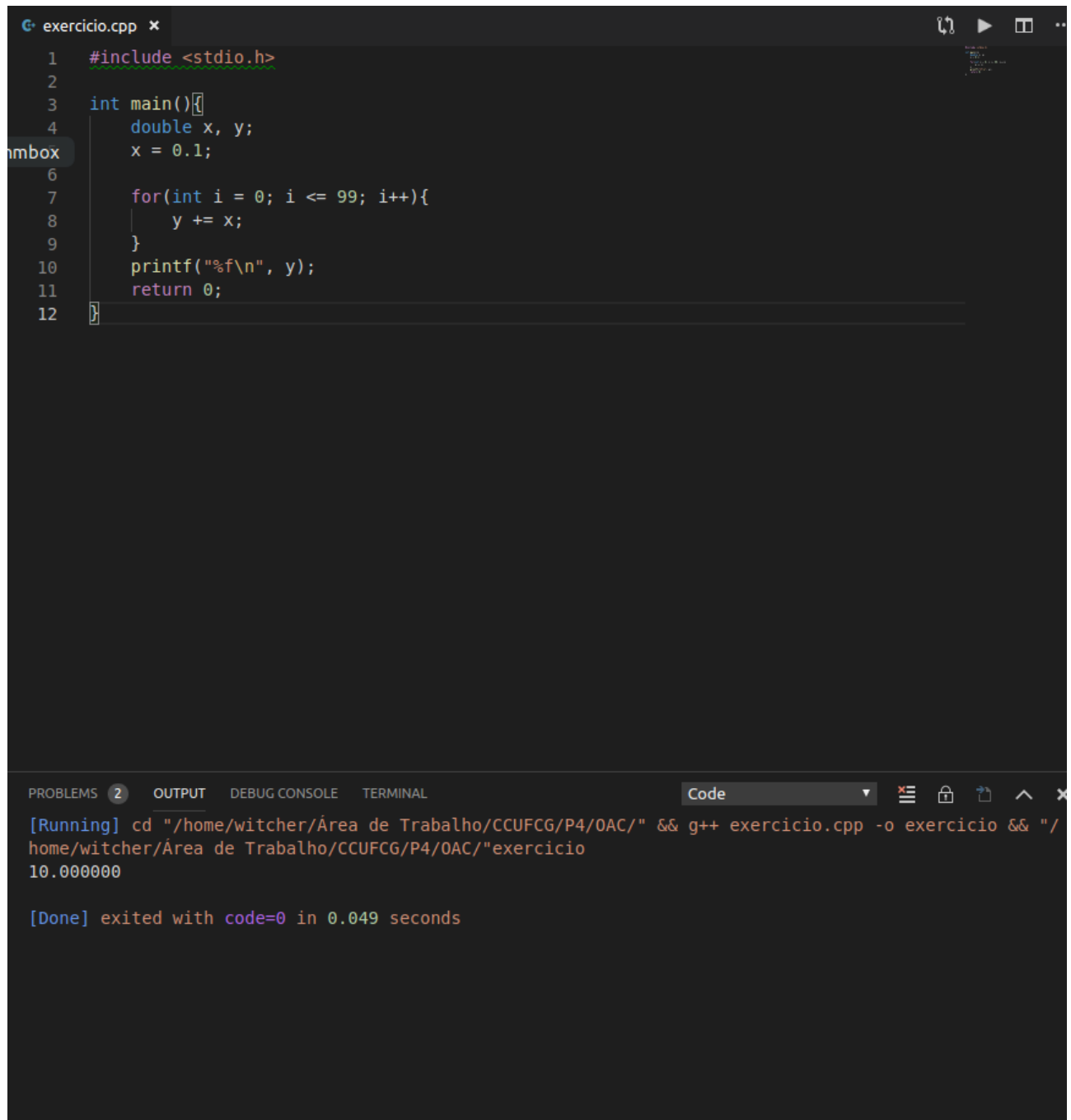
[Running] cd "/tmp/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/tmp/
tempCodeRunnerFile
10.000002
[Done] exited with code=0 in 0.194 seconds

[Running] cd "/home/witcher/Área de Trabalho/CCUFCG/P4/OAC/" && g++ exercicio.cpp -o exercicio && "/
home/witcher/Área de Trabalho/CCUFCG/P4/OAC/"exercicio
10.000002

[Done] exited with code=0 in 0.049 seconds

```

2 B:



```
exercico.cpp x
1  #include <stdio.h>
2
3  int main()
4  {
5      double x, y;
6      x = 0.1;
7
8      for(int i = 0; i <= 99; i++){
9          y += x;
10     }
11     printf("%f\n", y);
12     return 0;

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL Code

```
[Running] cd "/home/witcher/Área de Trabalho/CCUFCG/P4/0AC/" && g++ exercico.cpp -o exercico && "/home/witcher/Área de Trabalho/CCUFCG/P4/0AC/"exercico
10.000000

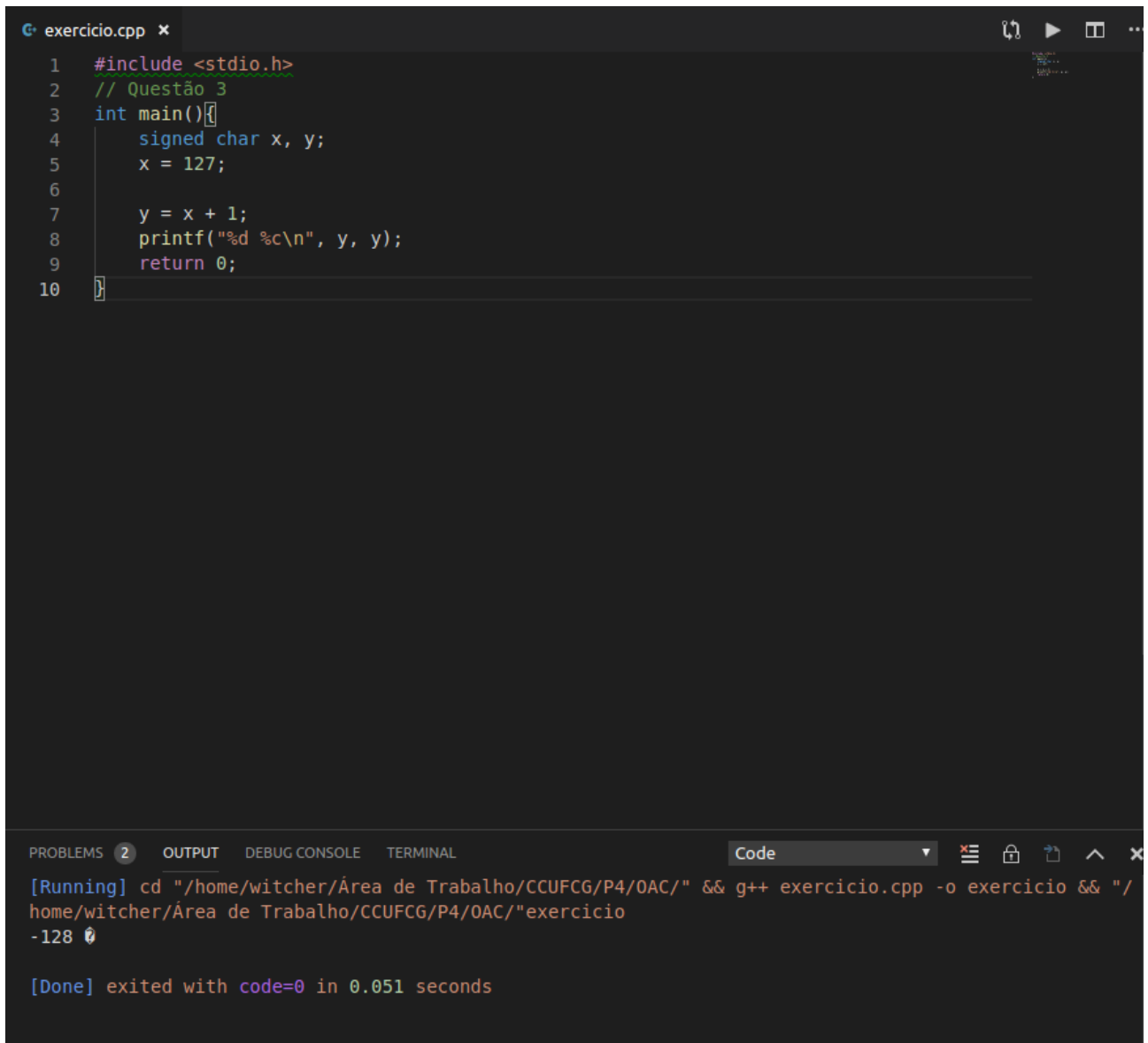
[Done] exited with code=0 in 0.049 seconds

```

Conclusão: Foi observado que em C, a soma 100x de 0.1 resultou em 10.000002, utilizando o tipo Float enquanto com o tipo Double ele mostrou o número corretamente sem a necessidade de “arredondamento”. Isso se dá pois a precisão do tipo double é maior que o float, enquanto o float necessita de 4 Bytes para armazenar um valor, o double consome o dobro disto (8 Bytes), mas consequentemente, com uma precisão melhor.

QUESTÃO 3.

3 A:



```
exercicio.cpp x
1  #include <stdio.h>
2  // Questão 3
3  int main()
4  {
5      signed char x, y;
6      x = 127;
7
8      y = x + 1;
9      printf("%d %c\n", y, y);
10     return 0;
11 }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL Code

[Running] cd "/home/witcher/Área de Trabalho/CCUFCG/P4/OAC/" && g++ exercicio.cpp -o exercicio && "/home/witcher/Área de Trabalho/CCUFCG/P4/OAC/"exercicio

-128 0

[Done] exited with code=0 in 0.051 seconds

3 B:

```
exercicio.cpp x
1  #include <stdio.h>
2  // Questão 3
3  int main() {
4      signed char x, y;
5      x = -127;
6
7      y = x - 2;
8      printf("%d %c\n", y, y);
9      return 0;
10 }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL Code

```
[Running] cd "/home/witcher/Área de Trabalho/CCUFCG/P4/OAC/" && g++ exercicio.cpp -o exercicio && "/home/witcher/Área de Trabalho/CCUFCG/P4/OAC/"exercicio
127 
[Done] exited with code=0 in 0.051 seconds
```

Conclusão: o tipo signed char em C tem o range de -128 à 127. É sabido que 127 é 0111 1111 na base 2, ao somarmos $127 + 1$, obtemos : 1000 0000 que consequentemente é o -128 pela representação do complemento de 2. O mesmo se aplica para a questão 3B, onde $-127 - 2 = -129$ o que ocasiona um estouro de capacidade e a máquina apenas considera o binário 0111 1111 = 127.