Output of the given code snippet would be:

- 26 4294967270 4294967269 -27 -20 65510
We can follow the calculations for each step to conclude to the above output.
Since my roll number is 2019101126. Hence, the calculation goes as following:

For line 1:

int x = 2019101126%100;
Since int can store values ranging from -2,147,483,648 to 2,147,483,647, there occurs no overflow and thus it stores the (INT) value 26. Actually during complilation to assembly code, this modulo gets calculated and stored in a register.

For line 2:

int a=(-1)*x;
Here, multiplying x with -1 results in INT value -26 which is then stored in variable a.
(actual)2's complement binary representation of -26 = 11111111111111111111111111100110.

For line 3:

unsigned int b = (unsigned int) a ;
It copies the binary value of "a" in "b" as it is, the crux here is that now it will be interpreted in unsigned form, i.e. now the value would become 2^32-26 which is equal to 4294967270 in decimal, instead of -26.

For line 4:

unsigned int c = UINT_MAX - x ;
Since UINT_MAX is equal to 2^32 -2, the value of "c" here becomes 2^32 -2 - 26 = 2^32 - 27 = 4294967269 in decimal.

For line 5:

int d = (int) c ;
Here, since "d" is a signed-bit representation and it becomes equal to signed-bit representation of "c" (explicit type-casting), the value of "d" becomes value of c - 2^32 equal to -27.

For line 6:

int p = 65490 + x ;
The value of p here becomes 65490 + 26 = 65516 in decimal form.

For line 7:

short int e = (short int) p ;
Since short is just 16-bits instead of 32-bits like int, the maximum value it can store is 2^15 - 1 = 32767(in decimal), and this limit is insufficient for "p" which is clearly more than the upper limit.
Hence, short would just store the last 2 bytes of p in binary form.
p (in binary) = 00000000 00000000 11111111 11101100
e (in binary) = 11111111 11101100

Thus, "e" in decimal becomes -20 (since it is signed and the first bit is 1, it becomes negative).

For line 8:

unsigned short f = (unsigned short) a ;
Since again "f" is just 16-bits in contrast to "a" which is 32-bits, the last two bytes gets copied and interpreted as an unsigned short integer.
a (in binary) = 11111111 11111111 11111111 11100110.
f (in binary) = 11111111 11100110
f (in decimal)= 65510.
As for the last line, it just prints the space-seperated variables a-f.