1. A null statement in C programming is valid but also mandatory in some cases, e.g.

```
if (isalpha(c))                 /* true = nonzero, false = zero */
  ;                             /* empty is ok, but ";" must be there */
else
  return(printf("You did not enter an alphabetic character\n"));
```

Instead of using the ';' character, what other ways can you define a null statement? Give a simple example like the above.

Answer:

```
if (isalpha(c))                 /* true = nonzero, false = zero */
  { }                           /* empty is ok, but ";" or { } must be there */
else
  return(printf("You did not enter an alphabetic character\n"));
```

2. A whole decimal number can be represented in exactly binary form, as follows:

$123_{10} = \underline{1}x10^2 + \underline{2}x10^1 + \underline{3}x10^0 = 01111011_2 = \underline{0}x2^7 + \underline{1}x2^6 + \underline{1}x2^5 + \underline{1}x2^4 + \underline{1}x2^3 + \underline{0}x2^2 + \underline{1}x2^1 + \underline{1}x2^0$

Why certain decimal fractions cannot be stored exactly in binary form? Explain using the examples "0.75" and "0.45" to illustrate your answer.

Answer:

$0.75 = 1x2^{-1} + 1x2^{-2} = 0.5 + 0.25$ → exact representation

$0.45 = 0x2^{-1} + 1x2^{-2} + 1x2^{-3} + 1x2^{-4} + 0x2^{-5} + 0x2^{-6} + 1x2^{-7} + \ldots$ → cannot be exactly represented as a binary fraction

3. The return value for **printf()** is incidental to its main purpose of printing output, and it usually not used. The return type of **printf()** function is **int**. Write a simple program to determine the value of printf() and infer its meaning.

Under ANSI C, printf() function returns the number of characters it printed. If there is an output error, printf() returns a negative value. The following program illustrates the fact:

```
#include <stdio.h>
int main(void)
{
  int c;
  c=printf("One");
  printf("\nc = %d",c);
  return 0;
}
```

Output

```
One
c= 3
```

4. What is the output of the following program? Explain.

```
int main(void)
{
   int i;
   for(i=5; --i; )
      printf("%d",i);
   return 0;
}
```

What is a more descriptive (understandable) way to write the for loop?

Answer:

Output 4321

For loop starts with i equals to 5, then tests the condition of the loop to execute; as part of the test, the variable **i** is decremented first, making **i** contain the value of 4. A true outcome can also be represented by a non-zero value, in this case, **i** being greater than 0. Execution proceeds into the loop (just one statement) to print **i**. After printing i = 1, it goes back into the control test which first decrements i to 0; and exits the loop.

```
int main(void)
{
   int i;
   for(i=5; i>0;--i)
      printf("%d",i);
   return 0;
}
```

5. What is the output of the following program? Explain, step-by-step how you got the answer.

```
int main()
{
   int i=3;
   for(i--; i<7; i=7)
      printf("%d",i++);
   return 0;
}
```

Answer:

Output **2**

The 'for' loop starts with i equals 3 and gets decremented by 1. The test for i<7 is true, since i is 2, and execution enters the loop to print the value of i.

Then, it executes the expression i=7, and tests the condition i<7. That is false, and execution exits the loop.