FF challenge C-test
Vijoy Sunil Kumar

1. **Write code to set bit 9 of TheRegister to be 0 without changing the other register contents**

   uint32_t TheRegister = 0xFFFFFFFF;

```
#include <stdio.h>
#include <stdint.h>

uint32_t* clear_bit_k(uint32_t* TheRegister, int k)
{
	//construct mask
	uint32_t bit_k = (1 << k);
	*TheRegister = *TheRegister &~ bit_k;
	return TheRegister;
}


int main(void)
{
	uint32_t num = 0xFFFFFFFF;
	uint32_t* TheRegister = &num;
	int k = 9;  //kth bit to clear
	TheRegister = clear_bit_k(TheRegister, k);

	printf("%d", *TheRegister);
	return 0;
}
```

2. **What does the keyword volatile mean? Give an example of its use.**

Volatile in C is used for the purpose of not caching the values of the variable automatically. It will tell the compiler not to cache the value of this variable. So it will take the value of the given volatile variable from the main memory every time it encounters it. This mechanism is used because at any time the value can be modified by the OS or an interrupt. So using volatile will help us accessing the value afresh every time.

FF challenge C-test
Vijoy Sunil Kumar

3. **Write code to extract bits 3 to 6 (inclusive) of TheRegister and put them in TheOutput starting at bit 0.**
   uint32_t TheRegister = 0xABCDEF12;
   uint32_t TheOutput =

```
#include <stdio.h>
#include <stdint.h>

int main(void)
{
    uint32_t TheRegister = 0xABCDEF12;
    uint32_t mask = 0;
    uint32_t TheOutput = 0;

    //bit positions 3 to 6
    int l = 3, r = 6;
    //construct mask
    for(int i = l; i<=r; i++)
        mask |= (1 << i);

    //extract bits
    TheOutput = TheRegister & mask;
    printf("%d", TheOutput);
    return 0;
}
```

4. **It is required to set an integer variable at the absolute address 0x67a9 to the value 0xabcd. The compiler is a pure ANSI compiler. Write code to accomplish this task.**

```
#include <stdio.h>
#include <stdint.h>

uint32_t* update_reg(uint32_t* addr, uint32_t val)
{
    if(addr != NULL){
        *addr = val;
        return addr;
    }
    else{
        printf("pointer not valid\n");
        return addr;
    }
}

int main(void)
{
    //uint32_t num = 10;
    //uint32_t* TheAddress = &num;

    uint32_t* TheAddress = 0x67a9;
    uint32_t TheValue = 0xabcd;

    TheAddress = update_reg(TheAddress, TheValue);
    printf("%d", *TheAddress);
    return 0;
}
```

5. **Write code to set bit 9 of TheRegister to be 1 without changing the other register contents**
   uint32_t TheRegister = 0x00000000;

```
#include <stdio.h>
#include <stdint.h>

uint32_t* set_bit_k(uint32_t* TheRegister, int k)
{
	//construct mask
	uint32_t bit_k = (1 << k);
	*TheRegister |= bit_k;
	return TheRegister;
}


int main(void)
{
	uint32_t num = 0x00000000;
	uint32_t* TheRegister = &num;

	//kth bit to set
	int k = 9;
	TheRegister = set_bit_k(TheRegister, k);

	printf("%d", *TheRegister);
	return 0;
}
```

6. **What is wrong with the following function?:**

```
int square(volatile int *ptr){

        return *ptr * *ptr;
}
```

It might not work because of following reason: The idea of the volatile keyword is exactly to indicate to the compiler that a variable marked as such can change in unexpected ways during the program execution. It's possible for the value of *ptr to change unexpectedly, Thus it is possible for *ptr to be different. Consequently, this code could return a number that is not a square. The correct way to do is

```
int square(volatile int *ptr)
{
  int a;
  a = *ptr;
  return a * a;
}
```

7. **Can a parameter be both const and volatile? Explain your answer.**

Yes, a parameter can be a "const volatile". For example: a read-only hardware register like an Interrupt Status register, where in the program we should not modify this variable so it should be a constant, but this variable can be changed by the processor or hardware based on the interrupt condition.

8. **What is the output of the following in terms of characters emitted to STDOUT from the printf statements.**

```
int main(void)
{
   for (int i = 0; i < 8; i++)
   {
      if ( ((i % 2) == 0)
           && (i++ > 4))
      {
         printf("A");
      }
      else
      {
         printf("B");
      }
   }

   return 0;
}
```

OUTPUT: BBBA

FF challenge C-test
Vijoy Sunil Kumar

9.

```
// The following code runs in an embedded target. Assume no major optimizations.
// Originally GetFirstCharacterIndex() is a stub that returns zero, DoSomeMath() returns 27.5 which is the desired output.
//  Adding the commented lines in GetFirstCharacterIndex() causes the result of DoSomeMath() to change unexpectedly.
//  Identify:
//   1) The bug that causes the output of unrelated code to change unexpectedly
//   2) Explain how the bug is possible
//   3) What would happen if the order of execution of GetFirstCharacterIndex() and DoSomeMath() is swapped?

#define MATH_CONSTANT_A (1.0f)
#define MATH_CONSTANT_B (2.0f)
#define MATH_CONSTANT_C (0.5f)

int GetFirstCharacterIndex(const char* string, char chr, int* index);
float DoSomeMath(void);

int main(void)
{
   float result = 0.0f;
   int indexOfG = 0;

   GetFirstCharacterIndex("Magic!", 'g', &indexOfG);
   result = DoSomeMath();

   // Result = 27.5f

   return 0;
}

int GetFirstCharacterIndex(const char* string, char chr, int* index)
{
   int charFound = 0;
   int charIndex = 0;

   // These lines gets inserted:
   /*
   while (string[charIndex] != '\0')
   {
      if (string[charIndex] == chr)
      {
         *index = charIndex;
         charFound = 1;
         break;
      }
      else
      {
         charIndex++;
      }
   }
   */
```

```
   return charFound;
}

float DoSomeMath(void)
{
   float z, q = 0.0f;

   for (int i = 0; i < 10; i++)
   {
      q += MATH_CONSTANT_A;

      z += q / MATH_CONSTANT_B;
   }

   return z;
}
```

ANSWER:

The only way DoSomeMath() would return an incorrect value, that I could think of, is because ‘z’ is not initialized to 0 – it could hold a random value. But this has nothing to do with GetFirstCharacterIndex(). I do not see any direct way where these two functions would interact. (After going through the code several times, I even ran this code in an online compiler and the result was 27.5 even after uncommenting the code in GetFirstCharacterIndex()).

Same was the case with swapping the order of function calls in main(). So, I think I failed to find the bug in the program. I could not REPRODUCE the bug.