C MINOR ASSIGNMENT- 06 (Recurrsion)

1. Consider the following ANSI C program;

```
#include <stdio.h>
int main()
{
        static int i=5;
        if(--i){
                main();
                printf("%d ",i);
        }
        return 0;
}
```

Explanation:

- The static variable `i` retains its value across recursive calls.
- `--i` decrements `i` before checking the condition.
- Recursive calls stop when `i == 0`. During unwinding, `printf` outputs `i`.

Output: `0 0 0 0`

2. Consider the following ANSI C program;

```
#include <stdio.h>
int a, b, c = 0;
void prtFun(void);
int main()
{
       static int a = 1; /* Line 1 */
        prtFun( );
        a+=1;
        prtFun( );
        printf("\n %d %d ", a, b);
        return(0);
}
void prtFun(void)
{
        static int a = 2; /* Line 2 */
        int b = 1;
        a + = ++b;
        printf(" \n %d %d ", a, b);
}
```

Explanation:

- `static int a` inside `prtFun()` retains its value between calls.
- First `prtFun()`: `a = 4, b = 2`.
- Second `prtFun()`: `a = 7, b = 2`.
- Final `printf`: Prints `main`'s `a = 2`, `b = 0` (global `b`).

Output:

```
4 2
7 2
2 0
```

3. Consider the following ANSI C program;

```
#include <stdio.h>
int a, b, c = 0;
void prtFun(void);
int main()
{ auto int a = 1; /* Line 1 */
        prtFun( );
        a+=1;
        prtFun( );
        printf("\n %d %d ", a, b);
        return(0);
}
void prtFun(void)
{ register int a = 2; /* Line 2 */
        int b = 1;
        a + = ++b;
        printf(" \n %d %d ", a, b);
}
```

Explanation:

- `register int a` is local to `prtFun()` and does not retain value.
- `prtFun()`: Executes independently with no link to `main`'s variables.
- Final `printf`: Prints `main`'s `a = 2` and global `b = 0`.

Output:

```
4 2
4 2
2 0
```

Q4)

```
#include<stdio.h>
int f(int n, int k){
        if(n==0) return 0;
        else if(n%2) return f(n/2, 2*k)+k;
        else return f(n/2, 2*k)-k;
}
int main(){
        printf( "%d",f(20,1));
        return 0;
}
```

The function `f(n, k)` processes the number n recursively by halving n at each step and adjusting k (doubling it) as it progresses. Depending on whether n is even or odd, the function adds or subtracts k to/from the result of the recursive call.

- **Base Case:** When $n = 0$, the function returns 0.
- **Recursive Case:**
  - For even n, the function subtracts k.
  - For odd n, the function adds k.

This creates a pattern of computation based on the binary representation of n. For $f(20, 1)$, the recursive evaluation results in an output of **9**.

5. What is printed by the following ANSI C program?

```
#include<stdio.h>
void f(int n){
        if(n<=1){
                printf("%d",n);
        }
        else{
                f(n/2);
                printf("%d",n%2);
        }
}
int main()
{
        f(173);
        return 0;
}
```

Explanation:

- Converts `173` to binary by recursive division by 2.
- Binary of `173`: `10101101`.

Output: `10101101`

6. Consider the following C function:

```
int f(int n)
{
   static int i = 1 ;
   if (n >=5) return n;
   n = n+1;
   i++;
   return f(n);
}
```

The value returned by **`f(1)`** is
(A) 5 (B) 6 (C) 7 (D) 8

Explanation:

- Static variable `i` increments with each recursive call but is unused in the return value.
- `n` starts at 1 and increments until it reaches 5.
- Function returns 5 when `n >= 5`.

Output: `5`

7. Consider the following C program

```
#include<stdio.h>
int r(){
  static int num=7;
  return num--;
}
int main()
{
        for(r();r();r())
           printf("%d ", r());
       return 0;
}
```

Which one of the following values will be displayed on execution of the program? (A) 41 (B) 52 (C) 63 (D) 630

The function `r()` decrements a static variable `num` starting at 7. In the `for` loop:

1. **Initialization (`r()`):** `num = 7`, decrements to 6.
2. **First Iteration:**
   - Condition (`r()`): `num = 6`, decrements to 5 (true).
   - `printf`: Prints `5` (`num` decrements to 4).
   - Update (`r()`): `num = 4`, decrements to 3.
3. **Second Iteration:**
   - Condition (`r()`): `num = 3`, decrements to 2 (true).
   - `printf`: Prints `2` (`num` decrements to 1).
   - Update (`r()`): `num = 1`, decrements to 0.
4. **Termination:** Condition (`r()`): `num = 0` (false).

Output:

```
5 2
```

8. The integer value printed by the ANSI-C progran

```
int funcp(){
  static int x = 1;
  x++;
 return x;
}
int main(){
  int x,y;
  x = funcp();
  y = funcp()+x;
  printf("%d\n", (x+y));
  return 0;
}
```

Explanation:

- Static `x` retains value and increments with each `funcp()` call.
- First call: `x = 2`.
- Second call: `x = 3`.

Output: `7`

9. Consider the following C program

```
#include<stdio.h>
int main(){
        register int a =10;
        int *ptr = NULL;
        ptr = &a;
        *ptr = 5;
        printf("%d",*ptr);
        return(0);
}
```

Explanation:

- Error: `register` variables are stored in CPU registers and do not have a memory address.
- `ptr = &a` causes a compilation error as you cannot take the address of a `register` variable.

Output: Compilation error

10. Consider the following C function;

```
file1.c
---------------
extern in count;
void write_extern(){
        count +=2;
}
```

```
file2.c
---------------
#include<stdio.h>
#include "file1.c"
int count = 5;
int main(){
        write_extern();
        write_extern();
        printf("%d\n", count);
        return(0);
}
```

Find the output if "**`file2.c`**" is compiled and executed:

Explanation:

- `count` is modified in `write_extern()`.
- Initial `count = 5`. After two calls: `count = 5 + 2 + 2 = 9`.

Output: `9`

11. Write the output of the following program;

```
#include<stdio.h>
int i=5;
int main()
{
        extern int j;
        printf("\ni=%d \nj=%d",i,j);
        int j=10;
        return 0;
}

int j =10;
```

Explanation:

- `extern int j` links to the global `j` declared outside `main()`.
- Undefined behavior as local `j` is re-declared within `main()`.

Output: Undefined (may vary)

12. Write a program to find the sum of an array elements using recursion.

```c
#include <stdio.h>
int sum_array(int arr[], int n) {
    if (n == 0) return 0;
    return arr[n - 1] + sum_array(arr, n - 1);
}
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Sum: %d\n", sum_array(arr, n));
    return 0;
}
```

13. Write a program to print "**n**" Fibonacci numbers using recursion.[N.B: The program format should be as follows]

```c
#include <stdio.h>
... print_fibo(......){
        ...
        ...
}
... main(){
        // get data from user
        print_fibo(...); // to print elements
}
```

```c
#include <stdio.h>
void print_fibo(int n, int a, int b) {
    if (n == 0) return;
    printf("%d ", a);
    print_fibo(n - 1, b, a + b);
}
int main() {
    int n;
    printf("Enter number of terms: ");
    scanf("%d", &n);
    print_fibo(n, 0, 1);
    return 0;
}
```