C MINOR ASSIGNMENT- 07 (Structure)

1. Select the invalid member of the following structure;

```
struct oswcourse{
   int secid;
   float avgm;
   char present;
   int *marks();
   int teacher();
}o1,o2;
```

1. **Explanation of the Structure:**
   - `int secid;`, `float avgm;`, and `char present;` are valid data members.
   - `int *marks();` is a valid declaration of a function pointer.
   - `int teacher();` is a prototype of a function and is invalid as a structure member.
2. **Why is `int teacher();` invalid?**
   - Function prototypes cannot be members of structures because structures store data, not function definitions or prototypes.
   - Only function pointers (`int *marks();`) are valid within structures.

2. Detect any invalid member present in the given structure;

```
struct date{
    int m,d,y;
};
struct stud{
   char name[20];
   struct stud *p;
   struct date *d;
   int (*)fun(int, int);
};
```

1. **Explanation of the Structure:**
   - `char name[20];`: Valid, as it declares a character array.
   - `struct stud *p;`: Valid, as it is a pointer to the same structure (self-referential structure).
   - `struct date *d;`: Valid, as it is a pointer to another structure type.
   - `int (*)fun(int, int);`: Invalid due to missing function pointer name.
2. **Why is `int (*)fun(int, int);` invalid?**
   - A function pointer must have a name to be valid (e.g., `int (*fun)(int, int);`).
   - The current syntax is incomplete, resulting in a compilation error.

3. The following structure template is allowed or not in ANS

```
struct person{
    int a;
    struct health{
        int a;
    }h;
};
```

1. **Explanation:**
   - Nested structures are valid in ANSI C.
   - Here, `struct health` is nested inside `struct person` and is declared with the member `int a;`.
   - This syntax is correct and valid in ANSI C.
2. **Output:**
   - No errors in this code. It compiles and works as expected.

4. The following declaration is correct or wrong

```
struct person{
    int a;
    union health{
        int w;
    }h;
};
```

1. **Explanation:**
   - Nested unions within structures are valid in ANSI C.
   - Here, `union health` is nested inside `struct person` and contains the member `int w;`.
2. **Output:**
   - No errors in this code. It compiles and works as expected.

5. The following declaration is correct or wrong.

```
union person{
    int a;
    struct health{
        int e;
    }h;
};
```

1. **Explanation:**
   - Unions can contain structures as members in ANSI C.
   - Here, `struct health` is a valid member of `union person` and contains the member `int e;`.
2. **Output:**
   - No errors in this code. It compiles and works as expected.

6. Check the declaration of the structure. Write a valid ber or not.

```
struct person{
    int ht;
    float wt;
    char color;
    struct person p; /*Line- 5 */
};
```

1. **Explanation**:
   - `struct person p;` creates a recursive structure. This is invalid because:
     - Structures cannot directly contain instances of themselves; it leads to infinite memory allocation.
     - A pointer to the structure (e.g., `struct person *p;`) is valid instead.
2. **Output**:
   - Compilation error: Recursive structure definition.

7. Write valid or invalid form of the followings.

```
(1) union{....}u;
(2) union u{......};
(3) struct{.....}s;
(4) struct s{.....};
```

Detailed Output:

1. (1): `union {....} u;`:
   - Valid, unnamed union with a variable `u`.
2. (2): `union u {......};`:
   - Valid, named union `u`.
3. (3): `struct {.....} s;`:
   - Valid, unnamed struct with a variable `s`.
4. (4): `struct s {.....};`:
   - Valid, named struct `s`.

8. Decide the output of the code snippet;

```
int main(){
   struct student{
      int h;
      int w;
      int m;
   };
   struct student s1={20,40,50};
   struct student *ptr=&s1;
   printf("%d\n",*((int *)ptr+2));
   return 0;
}
```

1. **Explanation**:
   - `ptr` points to the first member of `s1`.
   - `(int *)ptr` treats the structure as a sequence of integers.
   - `*((int *)ptr + 2)` accesses the third integer, which is `m = 50`.

9. Find the output of the code snippet;

```
struct s{int *p;};
int main(){int a=200;struct s s1;
  s1.p=&a;    *(s1.p)=*(s1.p)+100;
  printf("%d %d\n",a,*(s1.p));
  return 0;}
```

Detailed Output:

1. **Explanation**:
   - `s1.p` points to `a`.
   - `*(s1.p)` updates the value of `a` by adding 100.
   - Both `a` and `*(s1.p)` hold the value `300`.
2. **Output**:

```
300 300
```

10. Draw the node connectivity of the structure s1 and determine the output of the code snippet that simulates the array of structures and also the self-referential structure;

```
int main(){
  struct s1{
   char *z;
   int i;
   struct s1 *p;
};
struct s1 a[]={{"SOA",1,a+1},
               {"ITER",4,a+2},
               {"CSE",5,a}
              };
struct s1 *ptr=a;
printf("%s%s%s\n",a[0].z,a[1].z,a[2].z);
printf("%s%s%s",(*ptr).z, ptr→z,a[2].p→z);
return 0;
}
```

1. **Explanation**:
   - **Node Connectivity**:
     - `a[0].p = &a[1]`
     - `a[1].p = &a[2]`
     - `a[2].p = &a[0]`
   - **Output Analysis**:
     - First `printf`: Concatenates `z` values: `SOAITERCSE`.
     - Second `printf`: `(*ptr).z`, `ptr->z`, and `a[2].p->z` are all `"SOA"`, `"SOA"`, and `"CSE"`.
2. **Output**:

```
SOAITERCSE
SOASOACSE
```

11. Draw the node connectivity of the structure s1 and determine the output of the code snippet that simulates the array of structures and also the self-referential structure.

```
int main(){
  struct s1{
   char *z;
   int i;
   struct s1 *p;
};
 struct s1 a[]={{"SOA",1,a+1},
                {"ITER",2,a+2},
                {"CSE",3,a}};
 struct s1 *ptr=a;
 printf("%s\n", ++(ptr→z));
 printf("%s\n", a[(++ptr)→i].z);
 printf("%s\n",a[--(ptr→p→i)].z);
 printf("%d\n",--a[2].i);
 return 0;
}
```

1. **Node Connectivity**:
   - `a[0].p = &a[1]`
   - `a[1].p = &a[2]`
   - `a[2].p = &a[0]`
2. **Explanation**:
   - `++(ptr->z)` : Moves pointer in `"SOA"` to `"OA"`. Output: `"OA"`.
   - `a[(++ptr)->i].z` : Increments `ptr` to `a[1]`. `a[1].i = 2`, so `a[2].z = "CSE"`. Output: `"CSE"`.
   - `a[--(ptr->p->i)].z` : `ptr->p = a[2]`, decrements `a[2].i` to 2, so `a[1].z = "ITER"`. Output: `"ITER"`.
   - `--a[2].i` : `a[2].i` decrements to `1`. Output: `1`.

12. An initialization of array of structures given in the following code snippet. Find the output with pointer manipulation and operator precedence rules.

```
int main(){
  struct test{
   int i;
   char *c;
};
struct test st[]={5, "Cse-Engg",
                  4, "computer",
                  6, "Electrical",
                  8, "Mechnical",
                  7, "All-Engg"
                };
struct test *p=st;
printf("%s\n", ++(p++ →c));
printf("%c\n",*p++ →c);
printf("%d\n",++p→i);
printf("%s\n",p[0].c)
printf("%s\n",p→c);
return 0;}
```

Output▼

1. **Pointer Operations**:
   - `++(p++->c)` : `p->c` points to `"Cse-Engg"`. `++` moves it to `"se-Engg"`. Then `p` points to the next element. Output: `"se-Engg"`.
   - `*p++->c` : `p->c` points to `"Computer"`. Dereferencing gives `'C'`. `p` then moves to the next element. Output: `'C'`.
   - `++p->i` : `p->i = 6`, increments to `7`. Output: `7`.
   - `p[0].c` : `p` points to `"Electrical"`, so `p[0].c = "Electrical"`. Output: `"Electrical"`.
   - `p->c` : `p` remains at `"Electrical"`. Output: `"Electrical"`.

13. Conclude the output of the code snippet based on pointer and operator precedence on a nested structure.

```
int main(){
 struct out{
       char ch[10];
       char *str;
 };
 struct b{
        char *c;
        struct out o;
 };
 struct b s2={"ODISHA", "KHURDA","JOYDEV"};
 printf("%s %s %s\n",s2.c,s2.o.str,s2.o.ch);
 printf("%s %s\n",++s2.c,++s2.o.str);
 return 0;
}
```

- `s2 = {"ODISHA", "KHURDA", "JOYDEV"}` :
  - `s2.c = "ODISHA"`.
  - `s2.o.str = "KHURDA"`.
  - `s2.o.ch = "JOYDEV"`.
- `printf("%s %s %s\n", s2.c, s2.o.str, s2.o.ch)` : Outputs all string members.
- `printf("%s %s\n", ++s2.c, ++s2.o.str)` :
  - `++s2.c` points to `"DISHA"`.
  - `++s2.o.str` points to `"HURDA"`.

Output:

```
ODISHA KHURDA JOYDEV
DISHA HURDA
```

14. Find the output of the code snippet;

```
int main(){
  union unit{
   int marks;
   int roll;
}s1,s2;
 s2.roll=23;
 s1.marks=60;
 printf("%d..%d\n",s1.marks,s2.roll);
 return 0;
}
```

1. **Explanation**:
   - Each union shares memory for all members. `s1.marks` and `s2.roll` occupy different memory spaces.
   - `s2.roll = 23` and `s1.marks = 60` do not affect each other.
2. **Output**:

```
60..23
```

15. Find the output of the code snippet;

```
int main(){
  union unit{
  int marks;
  int roll;
}s1,s2;
 s2.roll=23;
 s2.marks=60;
 printf("Check memory alloc  for union\n");
 printf("%d..%d\n",s2.marks,s2.roll);
 return 0;
}
```

Explanation:

- Union members share memory. `s2.roll` and `s2.marks` occupy the same space.
- Writing `s2.marks = 60` overwrites `s2.roll`.
- `s2.roll` will now contain the binary representation of `60`.

Output:

```
Check memory alloc for union
60..60
```

16. Declare two variable of the structure type **`planet_t`**

```
typedef struct{
  char name[30];
  double diameter;
  int moons;
  double or_time,ro_time;
}planet_t;
```

Explanation:

- The `typedef` creates an alias `planet_t` for the structure.
- Two variables of type `planet_t` can be declared as:

```
planet_t earth, mars;
```

Output:

- Two variables, `earth` and `mars`, are successfully declared.

```c
/* 17. Initialize one of the variable of the question-16 structure with values ”jupiter”, 142.34, 16, 11.9, 9.23*/
#include <stdio.h>
typedef struct {
    char name[30];
    double diameter;
    int moons;
    double or_time, ro_time;
} planet_t;

int main() {
    planet_t jupiter = {"Jupiter", 142.34, 16, 11.9, 9.23};
    printf("Planet: %s\nDiameter: %.2f\nMoons: %d\nOrbital Time: %.2f\nRotational Time: %.2f\n",
           jupiter.name, jupiter.diameter, jupiter.moons, jupiter.or_time, jupiter.ro_time);
    return 0;
}
```

```c
/* 18. Declare a pointer to the structure type planet-t and initialize the structure components with the
 help of the pointer.*/

#include <stdio.h>
typedef struct {
    char name[30];
    double diameter;
    int moons;
    double or_time, ro_time;
} planet_t;

int main() {
    planet_t mars = {"Mars", 67.45, 2, 1.88, 24.6};
    planet_t *ptr = &mars;
    // Using the pointer to modify and display values
    ptr->moons = 3; // Update the moons
    printf("Planet: %s\nDiameter: %.2f\nMoons: %d\nOrbital Time: %.2f\nRotational Time: %.2f\n",
           ptr->name, ptr->diameter, ptr->moons, ptr->or_time, ptr->ro_time);
    return 0;
}
//mars.moons is updated to 3 via the pointer.
```

19. Numeric addresses for computers on the international network Internet are composed of four parts, separated by periods, of the form

```
xx.yy.zz.mm
```

where **xx, yy, zz** , and mm are positive integers. Locally, computers are usually known by a nickname as well. You are designing a program to process a list of Internet addresses, identifying all pairs of computers from the same locality. Create a structure type called **address_t** with components for the four integers of an Internet address and a fifth component in which to store an associated nickname of ten characters. Your program should read a list of up to 100 addresses and nicknames terminated by a sentinel address of all zeros and a sentinel nickname.

```
Sample Data
111.22.3.44 platte
555.66.7.88 wabash
111.22.5.66 green
0.0.0.0 none
```

The program should display a list of messages identifying each pair of computers from the same locality, that is, each pair of computers with matching values in the first two components of the address. In the messages, the computers should be identified by their nicknames.

```
Example Message
Machines platte and green are on the same local
   network.
```

Follow the messages by a display of the full list of addresses and nicknames. Include in your program a **scan_address** function, a **print_address** function, and a **local_address** function. Function **local_address** should take two address structures as input parameters and return 1 (for true) if the addresses are on the same local network, and 0 (for false) otherwise.

```c
#include <stdio.h>
#include <string.h>

typedef struct {
    int xx, yy, zz, mm;
    char nickname[10];
} address_t;

int is_local(address_t a, address_t b) {
    return (a.xx == b.xx && a.yy == b.yy);
}

void print_addresses(address_t addr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d.%d.%d.%d %s\n", addr[i].xx, addr[i].yy, addr[i].zz, addr[i].mm, addr[i].nickname);
    }
}

int main() {
    address_t addresses[] = {
        {111, 22, 3, 44, "platte"},
        {555, 66, 7, 88, "wabash"},
        {111, 22, 5, 66, "green"},
        {0, 0, 0, 0, "none"}
    };
    int count = 3;

    for (int i = 0; i < count; i++) {
        for (int j = i + 1; j < count; j++) {
            if (is_local(addresses[i], addresses[j])) {
                printf("Machines %s and %s are on the same local network.\n",
                       addresses[i].nickname, addresses[j].nickname);
            }
        }
    }

    printf("\nFull List of Addresses:\n");
    print_addresses(addresses, count);
    return 0;
}
```

```c
/*20. Youknowthatasinglelinkedlistconsists of several nodes that are connected through pointers. Design
 a program to create a singly linked list comprising integer elements for the given n nodes. A node
 contains, an integer number and a self-referential structure of the structure type node. Additionally,
 sort this linked list in ascending order.*/

 #include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int data;
    struct node *next;
} node_t;

node_t *insert_sorted(node_t *head, int value) {
    node_t *new_node = (node_t *)malloc(sizeof(node_t));
    new_node->data = value;
    new_node->next = NULL;

    if (!head || head->data >= value) {
        new_node->next = head;
        return new_node;
    }

    node_t *current = head;
    while (current->next && current->next->data < value) {
        current = current->next;
    }

    new_node->next = current->next;
    current->next = new_node;
    return head;
}

void display(node_t *head) {
    while (head) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    node_t *head = NULL;
    int values[] = {3, 1, 4, 2, 5};
    int n = 5;

    for (int i = 0; i < n; i++) {
        head = insert_sorted(head, values[i]);
    }

    printf("Sorted Linked List: ");
    display(head);

    return 0;
}
```