Question 1

Consider the following C snippet

```
void Test(int arr[], int e, int s)
{
    int temp;
    if(s>=e)
   {
       return;
   }

   temp = arr[s+1];
   arr[s+1] = arr[e];
   arr[e] = temp;
   Test(arr, s+1, e-1);
}
```

Let arr be an array[0 to 4] which initially holds the elements {1,2,3,4,5}. Test(arr, 0, 3) is called. The number of elements that will maintain their initial position after the end of the code is:……

Question 2

In a doubly linked list, how many pointers need to be modified when deleting a node from the middle?

Question 3

What is the primary advantage of a doubly linked list over a singly linked list?

Question 4

Which of the following operations is not constant time, O(1), in a circular singly linked list?

A: Insertion at the beginning
B: Deletion at the beginning
C: Insertion at the end
D: Searching for an element

Question 5

What is the time complexity for accessing an element in an array by index?

Question 6

What is the primary disadvantage of using an array over a linked list?

Question 7

In C programming, what is the correct way to declare an array of 10 integers?

Question 8

Given the following code snippet, what will be the output?

```
int arr[5] = {1, 2, 3, 4, 5};
printf("%d", arr[3])
```

Question 9

Which function is used to dynamically allocate memory for an array in C?

Question 10

What will be the output of the following code?

```
struct Node {
    int data;
    struct Node* next;
};

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 1;
    head->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3;
    third->next = NULL;

    printf("%d", head->next->data);
    return 0;
}
```

Question 11

In a linked list implementation in C, what does the following code do?

```
struct Node {
    int data;
    struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
```

```
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
```

Question 12

What is the purpose of the following code in the context of a linked list?

```
struct Node {
    int data;
    struct Node* next;
};

void printList(struct Node* n) {
    while (n != NULL) {
        printf("%d ", n->data);
        n = n->next;
    }
}
```

Question 13

What will be the output of the following code?

```
struct Node {
    int data;
    struct Node* next;
};

void printList(struct Node* n) {
    while (n != NULL) {
        printf("%d ", n->data);
        n = n->next;
    }
}

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 1;
    head->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3;
    third->next = NULL;

    printList(head);
    return 0;
```

Question 14

Which of the following statements correctly initializes an empty linked list in C?

A: **`struct Node* head = (struct Node*)malloc(sizeof(struct Node));`**
B: **`struct Node* head = NULL;`**
C: **`struct Node head;`**
D: **`struct Node head = {0};`**

Question 15

Given the following function to reverse a singly linked list, what is the value of **`head->data`** after executing **`reverse(&head)`**?

```
struct Node {
    int data;
    struct Node* next;
};

void reverse(struct Node** head_ref) {
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
}

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    struct Node* second = (struct Node*)malloc(sizeof(struct Node));
    struct Node* third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 1;
    head->next = second;
    second->data = 2;
```

```
    second->next = third;
    third->data = 3;
    third->next = NULL;

    reverse(&head);
    printf("%d", head->data);
    return 0;
```

Question 16

What will be the output of the following code?

```
int arr[] = {10, 20, 30, 40, 50};
int *ptr = arr + 2;
printf("%d", *ptr);
```

Question 17

What will be the output of the following code?

```
int arr[] = {1, 2, 3, 4, 5};
int sum = 0;
for (int i = 0; i < 5; i++) {
    sum += arr[i];
}
printf("%d", sum);
```

Question 18

Which of the following statements correctly declares a pointer to an array of 10 integers in C?

A: **`int (*arr)[10];`**
B: **`int *arr[10];`**
C: **`int *arr = [10];`**
D: **`int *arr = new int[10];`**

Question 19

What will be the output of the following code snippet?

```
int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
printf("%d", arr[1][1]);
```

Question 20

What is the correct syntax to dynamically allocate memory for an array of 10 integers in C?

A: **`int *arr = malloc(10 * sizeof(int));`**
B: **`int *arr = malloc(10 * int);`**
C: **`int arr = malloc(10 * sizeof(int));`**
D: **`int arr = malloc(10 * int);`**