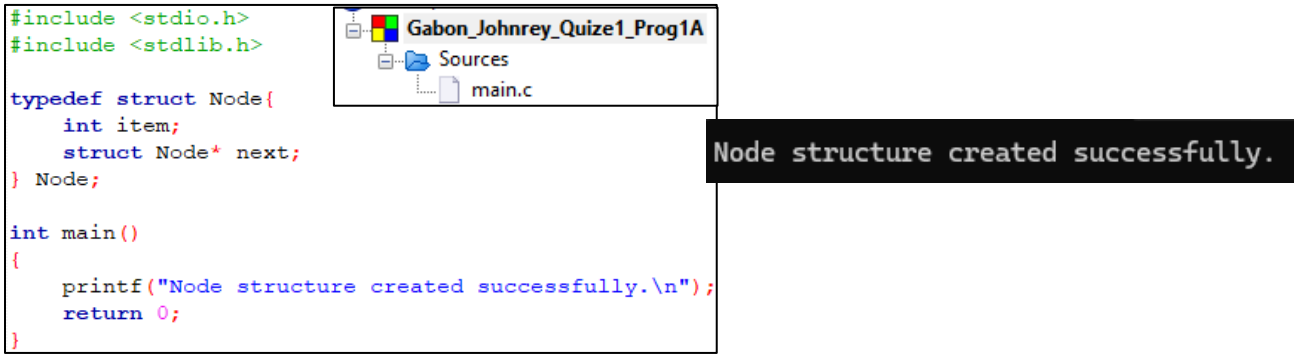
QUIZE 1

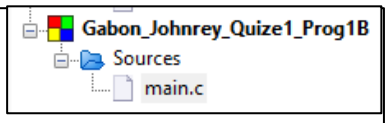
PROBLEM 1A:




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

typedef struct Node{
    int item;
    struct Node* next;
} Node;

int main()
{
    printf("Node structure created successfully.\n");
    return 0;
}
```

```
Node structure created successfully.
```

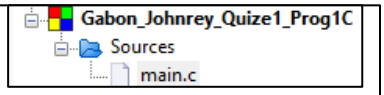
PROBLEM 1B:




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

typedef struct Node{
    int item;
    struct Node* next;
} Node;

int main()
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->item = 20;
    newNode->next = NULL;

    printf("First node created: Item = %d, Next = %p\n", newNode->item,  newNode->next);
    return 0;
}
```

```
First node created: Item = 20, Next = 0000000000000000
```

PROBLEM 1C:




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

typedef struct Node{
    int item;
    struct Node* next;
} Node;
struct Node* head;

int main()
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->item = 20;
    newNode->next = NULL;

    head = newNode;

    printf("Head node created: Item = %d, Next = %p\n", head->item,  head->next);
    return 0;
}
```

```
Head node created: Item = 20, Next = 0000000000000000
```

PROBLEM 2:

Gabon_Johnrey_Quize1_Prog2
Sources
main.c

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

typedef struct Node {
    int item;
    struct Node* next;
} Node;
Node* head = NULL;
void insertNode(int item) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->item = item;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void displayNode() {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->item);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    insertNode(10);
    insertNode(20);
    displayNode();

    return 0;
}
```

```
10 -> 20 -> NULL
```

PROBLEM 3:

```
5 -> 8 -> 15 -> 100 -> NULL
```

Gabon_Johnrey_Quize1_Prog3
Sources
main.c

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

typedef struct Node {
    int item;
    struct Node* next;
} Node;
Node* head = NULL;
void insertNode(int item) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->item = item;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void displayNode() {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->item);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    insertNode(5);
    insertNode(8);
    insertNode(15);
    insertNode(100);
    displayNode();

    return 0;
}
```

PROBLEM 4:

Gabon_Johnrey_Quize1_Prog4
Sources
main.c

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

typedef struct Node {
    int item;
    struct Node* next;
} Node;

Node* head = NULL;

void insertNode(int item) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->item = item;
    if (head == NULL) {
        head = newNode;
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}
void insertAfterNode(int target, int item) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    do {
        if (temp->item == target) {
            Node* newNode = (Node*)malloc(sizeof(Node));
            newNode->item = item;
            newNode->next = temp->next;
            temp->next = newNode;
            return;
        }
        temp = temp->next;
    } while (temp != head);
    printf("Node with item %d not found.\n", target);
}
void displayNode() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    do {
        printf("%d -> ", temp->item);
        temp = temp->next;
    } while (temp != head);
    printf("(head)\n");
}
int main() {
    insertNode(20);
    insertNode(40);
    insertNode(100);
    //Inserted node 30 after 20
    insertAfterNode(20, 30);
    displayNode();

    return 0;
}
```

```
20 -> 30 -> 40 -> 100 -> (head)
```

PROBLEM 5A:

Gabon_Johnrey_Quize1_Prog5
Sources
main.c

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

typedef struct Node {
    int item;
    struct Node* next;
    struct Node* prev;
} Node;
Node* head = NULL;
void insertNode(int item) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->item = item;
    newNode->next = NULL;
    newNode->prev = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}
void insertAfterNode(int target, int item) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    while (temp != NULL) {
        if (temp->item == target) {
            Node* newNode = (Node*)malloc(sizeof(Node));
            newNode->item = item;
            newNode->next = temp->next;
            newNode->prev = temp;
            if (temp->next != NULL) {
                temp->next->prev = newNode;
            }
            temp->next = newNode;
            return;
        }
        temp = temp->next;
    }
    printf("Node with item %d not found.\n", target);
}
void removeNode(int item) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    // If head node is to be removed
    if (head->item == item) {
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        free(temp);
        return;
    }
    // Remove non-head node
    while (temp != NULL) {
        if (temp->item == item) {
            if (temp->next != NULL) {
                temp->next->prev = temp->prev;
            }
            if (temp->prev != NULL) {
                temp->prev->next = temp->next;
            }
            free(temp);
            return;
        }
        temp = temp->next;
    }
    printf("Node %d not found.\n", item);
}
void displayNode() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->item);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    insertNode(5);
    insertNode(10);
    insertNode(100);
    printf("Before removal:\n");
    displayNode();

    removeNode(10);

    printf("After removing 10:\n");
    displayNode();
    return 0;
}
```

```
Before removal:
5 -> 10 -> 100 -> NULL
After removing 10:
5 -> 100 -> NULL
```

PROBLEM 5B:

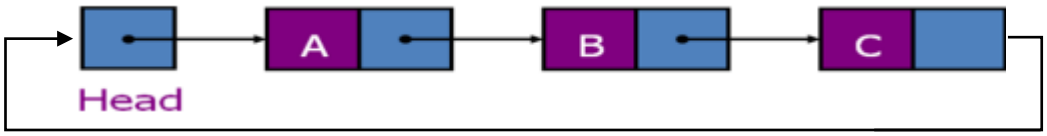
Gabon_Johnrey_Quize1_Prog5B
Sources
main.c

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

typedef struct Node {
    int item;
    struct Node* next;
    struct Node* prev;
} Node;
Node* head = NULL;
void insertNode(int item) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->item = item;
    newNode->next = NULL;
    newNode->prev = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}
void insertAfterNode(int target, int item) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    while (temp != NULL) {
        if (temp->item == target) {
            Node* newNode = (Node*)malloc(sizeof(Node));
            newNode->item = item;
            newNode->next = temp->next;
            newNode->prev = temp;
            if (temp->next != NULL) {
                temp->next->prev = newNode;
            }
            temp->next = newNode;
            return;
        }
        temp = temp->next;
    }
    printf("Node with item %d not found.\n", target);
}
void removeNode(int item) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    // If head node is to be removed
    if (head->item == item) {
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        free(temp);
        return;
    }
    // Remove non-head node
    while (temp != NULL) {
        if (temp->item == item) {
            if (temp->next != NULL) {
                temp->next->prev = temp->prev;
            }
            if (temp->prev != NULL) {
                temp->prev->next = temp->next;
            }
            free(temp);
            return;
        }
        temp = temp->next;
    }
    printf("Node %d not found.\n", item);
}
void displayNode() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->item);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    insertNode(5);
    insertNode(10);
    insertNode(100);
    printf("Before removal:\n");
    displayNode();

    removeNode(100);

    printf("After removing 100:\n");
    displayNode();
    return 0;
}
```

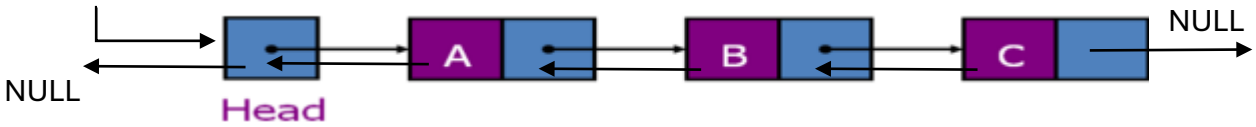
```
Before removal:
5 -> 10 -> 100 -> NULL
After removing 100:
5 -> 10 -> NULL
```

PROBLEM 6: Consider the linked list below. Make the list

a. Circular Linked List



b. Doubly Linked List



c. PROBLEM 7: Write one advantage of linked list compared to array.

- One advantage of a linked list compared to an array is that linked lists allow for efficient insertion and deletion of elements, as these operations can be performed in constant time (O(1)) if the position is known. In contrast, arrays require shifting elements, which can take linear time (O(n)). Additionally, linked lists do not require contiguous memory allocation, making them more flexible in memory management.