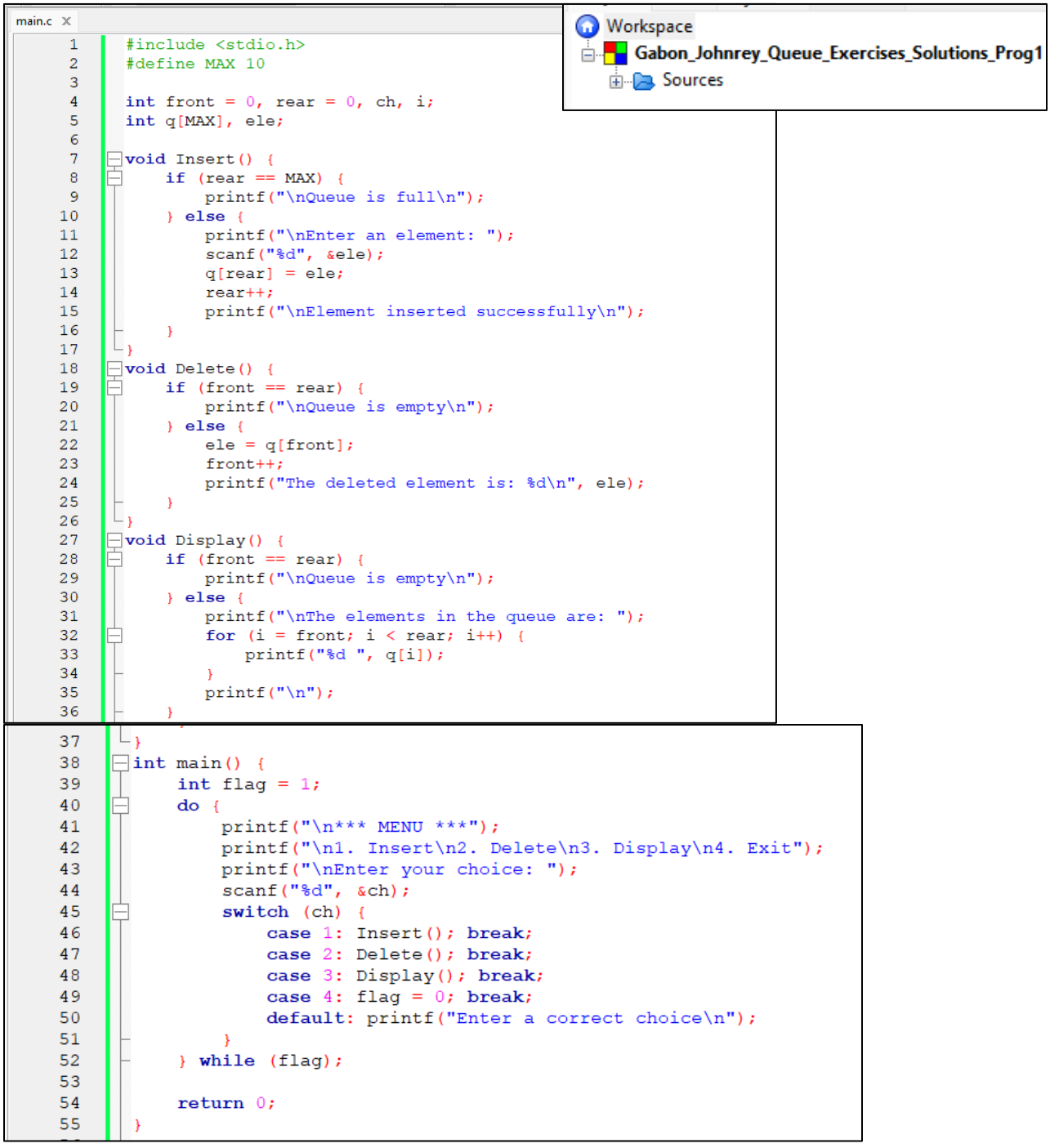
Johnrey A. Gabon BSIT2-5

FILE QUEUE EXERCISES SOLUTIONS



main.c

```c
#include <stdio.h>
#define MAX 10

int front = 0, rear = 0, ch, i;
int q[MAX], ele;

void Insert() {
    if (rear == MAX) {
        printf("\nQueue is full\n");
    } else {
        printf("\nEnter an element: ");
        scanf("%d", &ele);
        q[rear] = ele;
        rear++;
        printf("\nElement inserted successfully\n");
    }
}
void Delete() {
    if (front == rear) {
        printf("\nQueue is empty\n");
    } else {
        ele = q[front];
        front++;
        printf("The deleted element is: %d\n", ele);
    }
}
void Display() {
    if (front == rear) {
        printf("\nQueue is empty\n");
    } else {
        printf("\nThe elements in the queue are: ");
        for (i = front; i < rear; i++) {
            printf("%d ", q[i]);
        }
        printf("\n");
    }
}
int main() {
    int flag = 1;
    do {
        printf("\n*** MENU ***");
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1: Insert(); break;
            case 2: Delete(); break;
            case 3: Display(); break;
            case 4: flag = 0; break;
            default: printf("Enter a correct choice\n");
        }
    } while (flag);

    return 0;
}
```

Johnrey A. Gabon BSIT2-5

```
*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1

Enter an element: 11

Element inserted successfully

*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1

Enter an element: 12

Element inserted successfully

*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1

Enter an element: 13

Element inserted successfully
```

```
*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
The deleted element is: 11

*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

The elements in the queue are: 12 13

*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
```

```
*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
The deleted element is: 12

*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
The deleted element is: 13

*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

Queue is empty
```

```
*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

The elements in the queue are: 11 12 13
```

```
*** MENU ***
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 120.419 s
Press any key to continue.
```

STACK EXERCISES SOLUTIONS 1

main.c

Workspace
Gabon_Johnrey_Stack_Exercises_And_Solutions_Prog1
Sources
main.c

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

#define MAX 10

int top = -1, ch, i;
int stk[MAX], ele;

void Push()
{
    if (top == (MAX - 1))
    {
        printf("\nThe stack is full");
    }
    else
    {
        printf("Enter an element: ");
        scanf("%d", &ele);
        top++;
        stk[top] = ele;
        printf("\n\nElement pushed successfully\n");
    }
}
void Pop()
{
    if (top == -1)
    {
        printf("\nThe stack is empty");
    }
    else
    {
        ele = stk[top];
        top--;
        printf("\nThe deleted element is: %d\n", ele);
    }
}
void Top()
{
    if (top == -1)
    {
        printf("\nThe stack is empty");
    }
    else
    {
        printf("The top element of the stack is: %d\n", stk[top]);
    }
}
void Display()
{
    if (top == -1)
    {
        printf("\nThe stack is empty");
    }
    else
    {
        printf("\nThe elements in the stack are:");
        for (i = top; i >= 0; i--)
        {
            printf("\n%d", stk[i]);
        }
    }
}
int main()
{
    int flag = 1;
    do
    {
        printf("\n****MENU****");
        printf("\n1. Push\n2. Pop\n3. Top\n4. Display\n5. Exit");
        printf("\nEnter your Choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            Push();
            break;
        case 2:
            Pop();
            break;
        case 3:
            Top();
            break;
        case 4:
            Display();
            break;
        case 5:
            flag = 0;
            break;
        default:
            printf("Enter correct Choice\n");
            break;
        }
    }
    while (flag);
    return 0;
}
```

```
****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 1
Enter an element: 11


Element pushed successfully

****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 1
Enter an element: 22


Element pushed successfully

****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 1
Enter an element: 33


Element pushed successfully
```

```
****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 4

The elements in the stack are:
33
22
11
```

```
****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 3
The top element of the stack is: 33
```

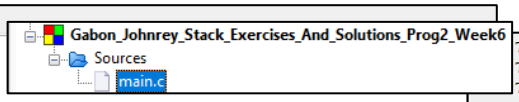
```
****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 2

The deleted element is: 33

****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 4

The elements in the stack are:
22
11
```

```
****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 2

The deleted element is: 22

****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 2

The deleted element is: 11

****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 4

The stack is empty
```

```
****MENU****
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your Choice: 5

Process returned 0 (0x0)   execution time
Press any key to continue.
```

STACK EXERCISES SOLUTIONS WEEK 6 (1)

main.c




```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define STACK_SIZE 1000

typedef struct
{
    char data[STACK_SIZE];
    int top;
} Stack;

void init_stack(Stack *stack)
{
    if (stack == NULL) return;
    stack->top = -1;
}
bool is_empty(Stack *stack)
{
    if (stack == NULL) return true;
    return stack->top == -1;
}
bool is_full(Stack *stack)
{
    if (stack == NULL) return true;
    return stack->top >= STACK_SIZE - 1;
}
void push(Stack *stack, char item)
{
    if (stack == NULL) return;

    if (is_full(stack))
    {
        fprintf(stderr, "\nStack Error: pushing on a full stack\n");
        return;
    }
    stack->data[++stack->top] = item;
}
char pop(Stack *stack)
{
    if (stack == NULL || is_empty(stack))
    {
        fprintf(stderr, "\nStack Error: Popping an empty stack\n");
        return '\0';
    }
    return stack->data[stack->top--];
}
int main()
{
    Stack equation;
    init_stack(&equation);
    char ch;
    char popped;
    bool good = true;
    int read_result;

    printf("Enter an equation followed by an s:\n");
    while ((read_result = scanf(" %c", &ch)) == 1)
    {
        if (ch == 's') break;

        if (ch == '{' || ch == '[' || ch == '(')
        {
            push(&equation, ch);
        }
        else if (ch == '}' || ch == ']' || ch == ')')
        {
            if (!is_empty(&equation))
            {
                popped = pop(&equation);
                if (!((popped == '{' && ch == '}') ||
                      (popped == '[' && ch == ']') ||
                      (popped == '(' && ch == ')')))
                {
                    good = false;
                }
            }
            else
            {
                good = false;
            }
        }
    }
    if (read_result != 1)
    {
        fprintf(stderr, "\nError reading input\n");
        return 1;
    }
    if (!is_empty(&equation))
    {
        good = false;  // Unmatched opening brackets
    }
    if (good)
    {
        printf("\nYes, it matched\n");
    }
    else
    {
        printf("\nNo, it was bad!\n");
    }
    return 0;
}
```

```
Enter an equation followed by an s:
{a{b}c}s

Yes, it matched
```

```
Enter an equation followed by an s:
{a{bc}s

No, it was bad!
```

```
Enter an equation followed by an s:
{ab}c}s

No, it was bad!
```

STACK EXERCISES SOLUTIONS WEEK 7 (1)

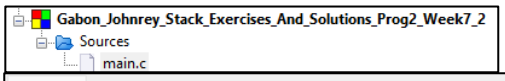
main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX 10

int stack[MAX];
int top = -1;

void push(int value) {
    if (top == MAX - 1) {
        printf("Stack is full\n");
    } else {
        stack[++top] = value;
        printf("Value %d is pushed into stack\n", value);
    }
}
int pop() {
    if (top == -1) {
        printf("Stack is empty\n");
        return -1;
    } else {
        int poppedValue = stack[top--];
        printf("Value %d is popped\n", poppedValue);
        return poppedValue;
    }
}
int evaluate(int operand1, int operand2, char operator) {
    switch (operator) {
        case '+': return operand1 + operand2;
        case '-': return operand1 - operand2;
        case '*': return operand1 * operand2;
        case '/': return operand1 / operand2;
        default: return 0;
    }
}
int main() {
    char ch;
    int operand1, operand2, result;
    while (1) {
        printf("Enter operator or operand: ");
        scanf(" %c", &ch);
        if (ch == 'x') {
            break;
        } else if (isdigit(ch)) {
            push(ch - '0');
        } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            operand2 = pop();
            operand1 = pop();
            result = evaluate(operand1, operand2, ch);
            push(result);
            printf("Result %d is pushed into stack\n", result);
        } else {
            printf("Invalid input\n");
        }
    }
    printf("\nThe result is: %d\n", stack[top]);
    return 0;
}
```

Gabon_Johnrey_Stack_Exercises_And_Solutions_Prog2_Week7_1
Sources
main.c

```
Enter operator or operand: 3
Value 3 is pushed into stack
Enter operator or operand: 4
Value 4 is pushed into stack
Enter operator or operand: 3
Value 3 is pushed into stack
Enter operator or operand: *
Value 3 is popped
Value 4 is popped
Value 12 is pushed into stack
Result 12 is pushed into stack
Enter operator or operand: +
Value 12 is popped
Value 3 is popped
Value 15 is pushed into stack
Result 15 is pushed into stack
Enter operator or operand: x

The result is: 15
```

STACK EXERCISES SOLUTIONS WEEK 7 (2)



main.c

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

float stack[10];
int top =- 1;
void push(float);
float pop();
float eval(char [],float[]);

void main()
{
    int i=0;
    char suffix[20];
    float value[20],result;
    printf("Enter a valid postfix expression: ");
    gets(suffix);
    while (suffix[i]!='\0')
    {
        if(isalpha(suffix[i]))
        {
            fflush(stdin);
            printf("Enter the value of %c: ",suffix[i]);
            scanf("%f",&value[i]); }
        i++; }
    result=eval(suffix,value);
    printf("\nThe result of %s=%f",suffix,result);
    getch();
}
float eval(char suffix[],float data[])
{
    int i=0;
    float op1,op2,res;
    char ch;
    while(suffix[i]!='\0')
    {
        ch=suffix[i];
        if(isalpha(suffix[i]))
        {
            push(data[i]);
        } else {
            op2=pop();
            op1=pop();
            switch(ch)
            {
                case '+' : push(op1+op2); break;
                case '-' : push(op1-op2); break;
                case '*' : push(op1*op2); break;
                case '/' : push(op1/op2); break;
                case '^' : push(pow(op1,op2)); break; } }
        i++; }
    res=pop();
    return(res);
}
void push(float num) {
    top=top+1;
    stack[top]=num; }
float pop() {
    float num;
    num=stack[top];
    top=top-1;
    return(num);
}
```

```
Enter a valid postfix expression: ab+c*
Enter the value of a: 5
Enter the value of b: 3
Enter the value of c: 2

The result of ab+c*=16.000000
```

```
Enter a valid postfix expression: ab-c^
Enter the value of a: 4
Enter the value of b: 1
Enter the value of c: 2

The result of ab-c^=9.000000
```

STRUCT 1

main.c

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

#define NUM_EMPLOYEES 5

struct emp
{
    char empFName[50];
    char empLName[50];
    int baseSalary;
    int overTimeHours;
};

int main()
{
    struct emp employees[NUM_EMPLOYEES];
    int i;

    // Reading employee details
    for (i = 0; i < NUM_EMPLOYEES; i++)
    {
        printf("Enter first name: ");
        scanf("%s", employees[i].empFName);

        printf("Enter last name: ");
        scanf("%s", employees[i].empLName);

        printf("Enter base salary: ");
        scanf("%d", &employees[i].baseSalary);

        printf("Enter the overtime hours: ");
        scanf("%d", &employees[i].overTimeHours);
        printf("\n");
    }

    printf("\nEmployee Monthly Salaries:\n");
    // Calculating and displaying salaries
    for (i = 0; i < NUM_EMPLOYEES; i++)
    {
        int monthlySalary = employees[i].baseSalary + (employees[i].overTimeHours * 20);
        printf("%s %s %d\n", employees[i].empFName, employees[i].empLName, monthlySalary);
    }

    return 0;
}
```

Gabon_Johnrey_Struct1_Prog1
Sources
main.c

```
Enter first name: John
Enter last name: Smith
Enter base salary: 3000
Enter the overtime hours: 10

Enter first name: Mary
Enter last name: Johnson
Enter base salary: 3500
Enter the overtime hours: 15

Enter first name: David
Enter last name: Wilson
Enter base salary: 4000
Enter the overtime hours: 5

Enter first name: Sarah
Enter last name: Brown
Enter base salary: 3800
Enter the overtime hours: 8

Enter first name: Michael
Enter last name: Davis
Enter base salary: 4200
Enter the overtime hours: 12


Employee Monthly Salaries:
John Smith 3200
Mary Johnson 3800
David Wilson 4100
Sarah Brown 3960
Michael Davis 4440
```

STRUCT 2

Gabon_Johnrey_Struct2_Prog1
Sources
main.c

main.c

```c
#include <stdio.h>
#include <string.h>
#define NUM_STUDENTS 10
struct std {
    char stdFName[50];
    char stdLName[50];
    int testScore;
    char grade;
};
// Function to calculate grade based on test score
char calculateGrade(int score) {
    if (score >= 90) return 'A';
    else if (score >= 80) return 'B';
    else if (score >= 70) return 'C';
    else if (score >= 60) return 'D';
    else return 'F';
}
int main() {
    struct std students[NUM_STUDENTS];
    int i, highestScore = -1;
    char topStudentFName[50], topStudentLName[50];
    // Reading students' data
    for (i = 0; i < NUM_STUDENTS; i++) {
        printf("Enter first name of student %d: ", i + 1);
        scanf("%s", students[i].stdFName);
        printf("Enter last name of student %d: ", i + 1);
        scanf("%s", students[i].stdLName);
        printf("Enter test score of student %d (0-100): ", i + 1);
        scanf("%d", &students[i].testScore);
        // Validate test score input
        while (students[i].testScore < 0 || students[i].testScore > 100) {
            printf("Invalid score! Enter test score of student %d (0-100): ", i + 1);
            scanf("%d", &students[i].testScore);
        }
        // Assign grade based on test score
        students[i].grade = calculateGrade(students[i].testScore);
        // Check for highest score
        if (students[i].testScore > highestScore) {
            highestScore = students[i].testScore;
            strcpy(topStudentFName, students[i].stdFName);
            strcpy(topStudentLName, students[i].stdLName);
        }
    }
    // Output students' data
    printf("\nStudent Grades:\n");
    for (i = 0; i < NUM_STUDENTS; i++) {
        printf("%s, %s: Test Score = %d, Grade = %c\n",
               students[i].stdLName, students[i].stdFName,
               students[i].testScore, students[i].grade);
    }
    // Output highest score details
    printf("\nHighest Test Score:\n");
    printf("Name: %s, %s\n", topStudentLName, topStudentFName);
    printf("Score: %d\n", highestScore);
    return 0;
}
```

```
Enter first name of student 1: John
Enter last name of student 1: Smith
Enter test score of student 1 (0-100): 95

Enter first name of student 2: Mary
Enter last name of student 2: Johnson
Enter test score of student 2 (0-100): 87

Enter first name of student 3: David
Enter last name of student 3: Wilson
Enter test score of student 3 (0-100): 73

Enter first name of student 4: Sarah
Enter last name of student 4: Brown
Enter test score of student 4 (0-100): 92

Enter first name of student 5: Michael
Enter last name of student 5: Silva
Enter test score of student 5 (0-100): 89

Enter first name of student 6: Emma
Enter last name of student 6: Taylor
Enter test score of student 6 (0-100): 88

Enter first name of student 7: James
Enter last name of student 7: Anderson
Enter test score of student 7 (0-100): 78

Enter first name of student 8: Lisa
Enter last name of student 8: Martinez
Enter test score of student 8 (0-100): 91

Enter first name of student 9: Robert
Enter last name of student 9: Thomas
Enter test score of student 9 (0-100): 83

Enter first name of student 10: Jennifer
Enter last name of student 10: White
Enter test score of student 10 (0-100): 79
```

```
Student Grades:
Smith, John: Test Score = 95, Grade = A
Johnson, Mary: Test Score = 87, Grade = B
Wilson, David: Test Score = 73, Grade = C
Brown, Sarah: Test Score = 92, Grade = A
Silva, Michael: Test Score = 89, Grade = B
Taylor, Emma: Test Score = 88, Grade = B
Anderson, James: Test Score = 78, Grade = C
Martinez, Lisa: Test Score = 91, Grade = A
Thomas, Robert: Test Score = 83, Grade = B
White, Jennifer: Test Score = 79, Grade = C

Highest Test Score:
Name: Smith, John
Score: 95

Process returned 0 (0x0)   execution time : 97.987 s
Press any key to continue.
```

MIDTERM EXAM EXAMPLE 1

PROBLEM 1:

main.c

```c
#include <stdio.h>

int main() {
    int A[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99};
    int v, found = 0;
    printf("Enter a value to search: ");
    scanf("%d", &v);

    for (int i = 0; i < 10; i++) {
        if (A[i] == v) {
            found = 1;
            break;
        }
    }

    if (found)
        printf("Yes\n");
    else
        printf("No\n");

    return 0;
}
```

Gabon_Johnrey_Midterm_Example1_Prog1
Sources
main.c

"D:\PLV\DSA\GABON DSA FIN

```
Enter a value to search: 1
No
```

"D:\PLV\DSA\GABON DSA FIN

```
Enter a value to search: 89
Yes
```

PROBLEM 2:

main.c

```c
#include <stdio.h>

int main() {
    int A[100][20];
    for (int i = 0; i < 100; i++) {
        for (int j = 0; j < 20; j++) {
            A[i][j] = (i + 1) * (j + 1);
        }
    }

    for (int i = 0; i < 100; i++) {
        for (int j = 0; j < 20; j++) {
            printf("%d ", A[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

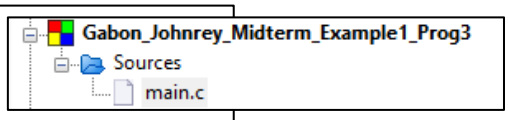
Gabon_Johnrey_Midterm_Example1_Prog2
Sources
main.c

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60
4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
6 12 18 24 30 36 42 48 54 60 66 72 78 84 90 96 102 108 114 120
7 14 21 28 35 42 49 56 63 70 77 84 91 98 105 112 119 126 133 140
8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160
9 18 27 36 45 54 63 72 81 90 99 108 117 126 135 144 153 162 171 180
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200
11 22 33 44 55 66 77 88 99 110 121 132 143 154 165 176 187 198 209 220
12 24 36 48 60 72 84 96 108 120 132 144 156 168 180 192 204 216 228 240
13 26 39 52 65 78 91 104 117 130 143 156 169 182 195 208 221 234 247 260
14 28 42 56 70 84 98 112 126 140 154 168 182 196 210 224 238 252 266 280
15 30 45 60 75 90 105 120 135 150 165 180 195 210 225 240 255 270 285 300
16 32 48 64 80 96 112 128 144 160 176 192 208 224 240 256 272 288 304 320
17 34 51 68 85 102 119 136 153 170 187 204 221 238 255 272 289 306 323 340
18 36 54 72 90 108 126 144 162 180 198 216 234 252 270 288 306 324 342 360
19 38 57 76 95 114 133 152 171 190 209 228 247 266 285 304 323 342 361 380
20 40 60 80 100 120 140 160 180 200 220 240 260 280 300 320 340 360 380 400
21 42 63 84 105 126 147 168 189 210 231 252 273 294 315 336 357 378 399 420
22 44 66 88 110 132 154 176 198 220 242 264 286 308 330 352 374 396 418 440
23 46 69 92 115 138 161 184 207 230 253 276 299 322 345 368 391 414 437 460
24 48 72 96 120 144 168 192 216 240 264 288 312 336 360 384 408 432 456 480
25 50 75 100 125 150 175 200 225 250 275 300 325 350 375 400 425 450 475 500
26 52 78 104 130 156 182 208 234 260 286 312 338 364 390 416 442 468 494 520
27 54 81 108 135 162 189 216 243 270 297 324 351 378 405 432 459 486 513 540
28 56 84 112 140 168 196 224 252 280 308 336 364 392 420 448 476 504 532 560
29 58 87 116 145 174 203 232 261 290 319 348 377 406 435 464 493 522 551 580
30 60 90 120 150 180 210 240 270 300 330 360 390 420 450 480 510 540 570 600
31 62 93 124 155 186 217 248 279 310 341 372 403 434 465 496 527 558 589 620
32 64 96 128 160 192 224 256 288 320 352 384 416 448 480 512 544 576 608 640
33 66 99 132 165 198 231 264 297 330 363 396 429 462 495 528 561 594 627 660
34 68 102 136 170 204 238 272 306 340 374 408 442 476 510 544 578 612 646 680
35 70 105 140 175 210 245 280 315 350 385 420 455 490 525 560 595 630 665 700
36 72 108 144 180 216 252 288 324 360 396 432 468 504 540 576 612 648 684 720
37 74 111 148 185 222 259 296 333 370 407 444 481 518 555 592 629 666 703 740
38 76 114 152 190 228 266 304 342 380 418 456 494 532 570 608 646 684 722 760
39 78 117 156 195 234 273 312 351 390 429 468 507 546 585 624 663 702 741 780
40 80 120 160 200 240 280 320 360 400 440 480 520 560 600 640 680 720 760 800
41 82 123 164 205 246 287 328 369 410 451 492 533 574 615 656 697 738 779 820
42 84 126 168 210 252 294 336 378 420 462 504 546 588 630 672 714 756 798 840
43 86 129 172 215 258 301 344 387 430 473 516 559 602 645 688 731 774 817 860
44 88 132 176 220 264 308 352 396 440 484 528 572 616 660 704 748 792 836 880
45 90 135 180 225 270 315 360 405 450 495 540 585 630 675 720 765 810 855 900
46 92 138 184 230 276 322 368 414 460 506 552 598 644 690 736 782 828 874 920
47 94 141 188 235 282 329 376 423 470 517 564 611 658 705 752 799 846 893 940
48 96 144 192 240 288 336 384 432 480 528 576 624 672 720 768 816 864 912 960
49 98 147 196 245 294 343 392 441 490 539 588 637 686 735 784 833 882 931 980
50 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950 1000
```

```
51 102 153 204 255 306 357 408 459 510 561 612 663 714 765 816 867 918 969 1020
52 104 156 208 260 312 364 416 468 520 572 624 676 728 780 832 884 936 988 1040
53 106 159 212 265 318 371 424 477 530 583 636 689 742 795 848 901 954 1007 1060
54 108 162 216 270 324 378 432 486 540 594 648 702 756 810 864 918 972 1026 1080
55 110 165 220 275 330 385 440 495 550 605 660 715 770 825 880 935 990 1045 1100
56 112 168 224 280 336 392 448 504 560 616 672 728 784 840 896 952 1008 1064 1120
57 114 171 228 285 342 399 456 513 570 627 684 741 798 855 912 969 1026 1083 1140
58 116 174 232 290 348 406 464 522 580 638 696 754 812 870 928 986 1044 1102 1160
59 118 177 236 295 354 413 472 531 590 649 708 767 826 885 944 1003 1062 1121 1180
60 120 180 240 300 360 420 480 540 600 660 720 780 840 900 960 1020 1080 1140 1200
61 122 183 244 305 366 427 488 549 610 671 732 793 854 915 976 1037 1098 1159 1220
62 124 186 248 310 372 434 496 558 620 682 744 806 868 930 992 1054 1116 1178 1240
63 126 189 252 315 378 441 504 567 630 693 756 819 882 945 1008 1071 1134 1197 1260
64 128 192 256 320 384 448 512 576 640 704 768 832 896 960 1024 1088 1152 1216 1280
65 130 195 260 325 390 455 520 585 650 715 780 845 910 975 1040 1105 1170 1235 1300
66 132 198 264 330 396 462 528 594 660 726 792 858 924 990 1056 1122 1188 1254 1320
67 134 201 268 335 402 469 536 603 670 737 804 871 938 1005 1072 1139 1206 1273 1340
68 136 204 272 340 408 476 544 612 680 748 816 884 952 1020 1088 1156 1224 1292 1360
69 138 207 276 345 414 483 552 621 690 759 828 897 966 1035 1104 1173 1242 1311 1380
70 140 210 280 350 420 490 560 630 700 770 840 910 980 1050 1120 1190 1260 1330 1400
71 142 213 284 355 426 497 568 639 710 781 852 923 994 1065 1136 1207 1278 1349 1420
72 144 216 288 360 432 504 576 648 720 792 864 936 1008 1080 1152 1224 1296 1368 1440
73 146 219 292 365 438 511 584 657 730 803 876 949 1022 1095 1168 1241 1314 1387 1460
74 148 222 296 370 444 518 592 666 740 814 888 962 1036 1110 1184 1258 1332 1406 1480
75 150 225 300 375 450 525 600 675 750 825 900 975 1050 1125 1200 1275 1350 1425 1500
76 152 228 304 380 456 532 608 684 760 836 912 988 1064 1140 1216 1292 1368 1444 1520
77 154 231 308 385 462 539 616 693 770 847 924 1001 1078 1155 1232 1309 1386 1463 1540
78 156 234 312 390 468 546 624 702 780 858 936 1014 1092 1170 1248 1326 1404 1482 1560
79 158 237 316 395 474 553 632 711 790 869 948 1027 1106 1185 1264 1343 1422 1501 1580
80 160 240 320 400 480 560 640 720 800 880 960 1040 1120 1200 1280 1360 1440 1520 1600
81 162 243 324 405 486 567 648 729 810 891 972 1053 1134 1215 1296 1377 1458 1539 1620
82 164 246 328 410 492 574 656 738 820 902 984 1066 1148 1230 1312 1394 1476 1558 1640
83 166 249 332 415 498 581 664 747 830 913 996 1079 1162 1245 1328 1411 1494 1577 1660
84 168 252 336 420 504 588 672 756 840 924 1008 1092 1176 1260 1344 1428 1512 1596 1680
85 170 255 340 425 510 595 680 765 850 935 1020 1105 1190 1275 1360 1445 1530 1615 1700
86 172 258 344 430 516 602 688 774 860 946 1032 1118 1204 1290 1376 1462 1548 1634 1720
87 174 261 348 435 522 609 696 783 870 957 1044 1131 1218 1305 1392 1479 1566 1653 1740
88 176 264 352 440 528 616 704 792 880 968 1056 1144 1232 1320 1408 1496 1584 1672 1760
89 178 267 356 445 534 623 712 801 890 979 1068 1157 1246 1335 1424 1513 1602 1691 1780
90 180 270 360 450 540 630 720 810 900 990 1080 1170 1260 1350 1440 1530 1620 1710 1800
91 182 273 364 455 546 637 728 819 910 1001 1092 1183 1274 1365 1456 1547 1638 1729 1820
92 184 276 368 460 552 644 736 828 920 1012 1104 1196 1288 1380 1472 1564 1656 1748 1840
93 186 279 372 465 558 651 744 837 930 1023 1116 1209 1302 1395 1488 1581 1674 1767 1860
94 188 282 376 470 564 658 752 846 940 1034 1128 1222 1316 1410 1504 1598 1692 1786 1880
95 190 285 380 475 570 665 760 855 950 1045 1140 1235 1330 1425 1520 1615 1710 1805 1900
96 192 288 384 480 576 672 768 864 960 1056 1152 1248 1344 1440 1536 1632 1728 1824 1920
97 194 291 388 485 582 679 776 873 970 1067 1164 1261 1358 1455 1552 1649 1746 1843 1940
98 196 294 392 490 588 686 784 882 980 1078 1176 1274 1372 1470 1568 1666 1764 1862 1960
99 198 297 396 495 594 693 792 891 990 1089 1188 1287 1386 1485 1584 1683 1782 1881 1980
100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600 1700 1800 1900 2000
```

PROBLEM 3:

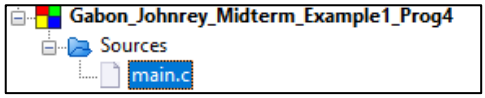
```c
#include <stdio.h>
#include <string.h>

struct House {
    int id;
    int NumberOfRooms;
    char address[50];
    int OwnerPhone;
};

int main() {
    struct House A[100]; // Array of 100 houses
    printf("Struct House created sucessfully\n");
    return 0;
}
```




```
Struct House created sucessfully
```

PROBLEM 4:

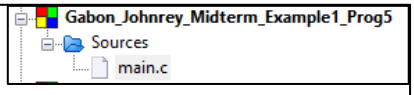
```c
#include <stdio.h>
#include <string.h>

struct House {
    int id;
    int NumberOfRooms;
    char address[100];
    int OwnerPhone;
};

int main() {
    struct House A[100];
    A[0].id = 0;
    A[0].NumberOfRooms = 5;
    strcpy(A[0].address, "Shubra Street on the front of the old hospital, Building 5, Floor 4");
    A[0].OwnerPhone = 555779922;

    // Print the details of the house
    printf("House ID: %d\n", A[0].id);
    printf("Number of Rooms: %d\n", A[0].NumberOfRooms);
    printf("Address: %s\n", A[0].address);
    printf("Owner Phone: %d\n", A[0].OwnerPhone);

    return 0;
}
```




```
House ID: 0
Number of Rooms: 5
Address: Shubra Street on the front of the old hospital, Building 5, Floor 4
Owner Phone: 555779922
```

PROBLEM 5:

```c
#include <stdio.h>

struct House {
    int id;
    int NumberOfRooms;
    char address[50];
    int OwnerPhone;
};

int main() {
    struct House A[100];

    // Initialize the NumberOfRooms for each house
    for (int i = 0; i < 100; i++) {
        A[i].NumberOfRooms = (i + 1) % 10;
    }

    // Display the number of rooms for houses with more than 5 rooms
    for (int i = 0; i < 100; i++) {
        if (A[i].NumberOfRooms > 5) {
            printf("%d\n", A[i].NumberOfRooms);
        }
    }

    return 0;
}
```

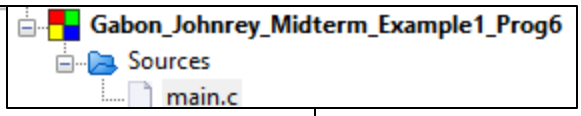



```
6
7
8
9
6
7
8
9
6
7
8
9
6
7
8
9
6
7
8
9
6
7
8
9
6
7
8
9
6
7
8
9
```

```
6
7
8
9
6
7
8
9
6
7
8
9
6
7
8
9
```

PROBLEM 6:

```c
#include <stdio.h>

int main() {
    int *ptr, a[10], x;
    x = 10;
    a[0] = 1;
    a[5] = 10;
    a[7] = 5;
    ptr = &x;
    printf("%d\n", x); // Outputs 10
    *ptr = *ptr + a[5];
    printf("%d\n", x); // Outputs 20

    return 0;
}
```




```
10
20
```

PROBLEM 7:

```c
#include <stdio.h>

int main() {
    int *ptr, a[10];
    ptr = &a[0];
    a[0] = 1;
    a[4] = 4;
    a[5] = 4;
    a[6] = 4;
    a[7] = 4;
    a[8] = -5;
    ptr += 6;
    ptr--;
    *ptr = *ptr + a[5];
    printf("%d\n", *ptr); // Outputs 8

    return 0;
}
```

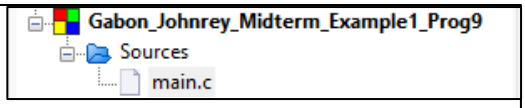

```
8
```

PROBLEM 8

```c
#include <stdio.h>

struct NodeStudent {
    int id;
    char name[20];
    int age;
    char address[20];
};
int main()
{
    printf("Created struct NodeStudent successfully\n");
    return 0;
}
```


```
Created struct NodeStudent successfully
```

PROBLEM 9:




```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct NodeStudent
{
    int id;
    char name[20];
    int age;
    char address[20];
    struct NodeStudent *next;
};
struct NodeStudent *head = NULL;
int SizeofTheList()
{
    struct NodeStudent *curr = head;
    int count = 0;
    while (curr != NULL)
    {
        count++;
        curr = curr->next;
    }
    return count;
}
void addNode(int id, const char *name, int age, const char *address)
{
    struct NodeStudent *newNode = (struct NodeStudent*)malloc(sizeof(struct NodeStudent));
    newNode->id = id;
    strcpy(newNode->name, name);
    newNode->age = age;
    strcpy(newNode->address, address);
    newNode->next = head;
    head = newNode;
}
int main()
{
    addNode(1, "Alice", 20, "123 Main St");
    addNode(2, "Bob", 21, "456 Elm St");
    addNode(3, "Charlie", 22, "789 Oak St");
    printf("Size of the list: %d\n", SizeofTheList());
    return 0;
}
```

```
Size of the list: 3
```

PROBLEM 10:

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

struct NodeStudent {
    int id;
    char name[20];
    int age;
    char address[20];
    struct NodeStudent *next;
};

struct NodeStudent *head = NULL;

void DisplayNames(int minAge, int maxAge) {
    struct NodeStudent *curr = head;

    while (curr != NULL) {
        if (curr->age >= minAge && curr->age <= maxAge) {
            printf("%s\n", curr->name);
        }
        curr = curr->next;
    }
}

void addNode(int id, const char *name, int age, const char *address) {
    struct NodeStudent *newNode = (struct NodeStudent*)malloc(sizeof(struct NodeStudent));
    newNode->id = id;
    strcpy(newNode->name, name);
    newNode->age = age;
    strcpy(newNode->address, address);
    newNode->next = head;
    head = newNode;
}

int main() {
    addNode(1, "Alice", 20, "123 Main St");
    addNode(2, "Bob", 25, "456 Elm St");
    addNode(3, "Charlie", 28, "789 Oak St");
    addNode(4, "David", 30, "101 Pine St");
    addNode(5, "Eve", 22, "202 Maple St");

    printf("Names of students aged between 24 and 30:\n");
    DisplayNames(24, 30);

    return 0;
}
```

```
Names of students aged between 24 and 30:
David
Charlie
Bob
```

MIDTERM EXAM EXAMPLE 2

PROBLEM 1:

Gabon_Johnrey_Midterm_Example2_Prog1
Sources
main.c

```c
#include <stdio.h>

int main() {
    int A[100];
    int x1, x2;
    for (int i = 0; i < 100; i++) {
        A[i] = i + 1;
    }
    printf("Enter x1: ");
    scanf("%d", &x1);
    printf("Enter x2: ");
    scanf("%d", &x2);

    printf("Values between %d and %d:\n", x1, x2);
    for (int i = 0; i < 100; i++) {
        if (A[i] >= x1 && A[i] <= x2) {
            printf("%d ", A[i]);
        }
    }
    printf("\n");

    return 0;
}
```

```
Enter x1: 2
Enter x2: 20
Values between 2 and 20:
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

PROBLEM 2:

Gabon_Johnrey_Midterm_Example2_Prog2
Sources
main.c

```c
#include <stdio.h>

int main() {
    float A[5][5] = {
        {1, 1, 0, -1},
        {1, 1, 2, -1, 5},
        {1, 1, 0, -1},
        {6, 1, 1, 0, -1},
        {1, 1}
    };

    printf("Array A content:\n");
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            printf("%.2f ", A[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
Array A content:
1.00 1.00 0.00 -1.00 0.00
1.00 1.00 2.00 -1.00 5.00
1.00 1.00 0.00 -1.00 0.00
6.00 1.00 1.00 0.00 -1.00
1.00 1.00 0.00 0.00 0.00
```

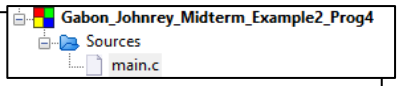
PROBLEM 3:

Gabon_Johnrey_Midterm_Example2_Prog3
Sources
main.c

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

typedef struct {
    int id;
    int NumberOfRooms;
    char address[50];
    int OwnerPhone;
} House;

int main() {
    House A[100];
    A[0].id = 1;
    A[0].NumberOfRooms = 3;
    strcpy(A[0].address, "123 Main St");
    A[0].OwnerPhone = 1234567890;

    printf("House ID: %d\n", A[0].id);
    printf("Number of Rooms: %d\n", A[0].NumberOfRooms);
    printf("Address: %s\n", A[0].address);
    printf("Owner Phone: %d\n", A[0].OwnerPhone);

    return 0;
}
```

```
House ID: 1
Number of Rooms: 3
Address: 123 Main St
Owner Phone: 1234567890
```

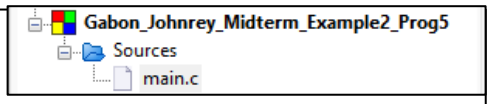
PROBLEM 4:




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

typedef struct {
    int id;
    int NumberOfRooms;
    char address[50];
    int OwnerPhone;
} House;

int main() {
    House A[100];
    printf("Enter house ID: ");
    scanf("%d", &A[0].id);

    printf("Enter number of rooms: ");
    scanf("%d", &A[0].NumberOfRooms);

    printf("Enter address: ");
    getchar();
    fgets(A[0].address, 50, stdin);
    A[0].address[strcspn(A[0].address, "\n")] = '\0';

    printf("Enter owner phone: ");
    scanf("%d", &A[0].OwnerPhone);

    printf("\nHouse details:\n");
    printf("ID: %d\n", A[0].id);
    printf("Number of Rooms: %d\n", A[0].NumberOfRooms);
    printf("Address: %s\n", A[0].address);
    printf("Owner Phone: %d\n", A[0].OwnerPhone);

    return 0;
}
```

```
Enter house ID: 191
Enter number of rooms: 3
Enter address: Valenzuela City
Enter owner phone: 0914175364

House details:
ID: 191
Number of Rooms: 3
Address: Valenzuela City
Owner Phone: 914175364
```

PROBLEM 5:

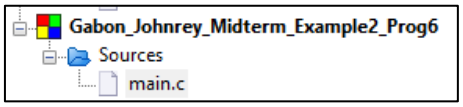



```c
#include <stdio.h>
#include <string.h>

typedef struct {
    int id;
    int NumberOfRooms;
    char address[50];
    int OwnerPhone;
} House;
int main() {
    House A[100];
    A[0] = (House){1, 1, "House 1", 1234567890};
    A[1] = (House){2, 3, "House 2", 9876543210};
    A[2] = (House){3, 2, "House 3", 1122334455};
    A[3] = (House){4, 5, "House 4", 9988776655};
    printf("Small houses (less than or equal to 2 rooms):\n");
    for (int i = 0; i < 4; i++) {
        if (A[i].NumberOfRooms <= 2) {
            printf("House ID %d: %d rooms\n", A[i].id, A[i].NumberOfRooms);
        }
    }
    return 0;
}
```

```
Small houses (less than or equal to 2 rooms):
House ID 1: 1 rooms
House ID 3: 2 rooms
```

PROBLEM 6:

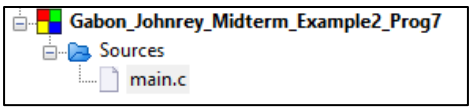
```c
#include <stdio.h>

int main() {
    int *ptr, a[10], x;
    x = 10;
    a[0] = -1;
    a[5] = -5;
    a[7] = 15;
    ptr = &x;

    printf("%d\n", x);
    *ptr = *ptr + a[5];
    printf("%d\n", x);

    return 0;
}
```




```
10
5
```

PROBLEM 7:

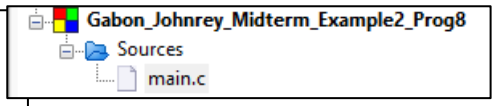
```c
#include <stdio.h>

int main() {
    int *ptr, a[10];
    ptr = &a[0];
    a[0] = 11;
    a[4] = 43;
    a[5] = -4;
    a[6] = -4;
    a[7] = -3;
    a[8] = -5;

    ptr += 6;
    ptr--;
    *ptr = *ptr + a[5];
    printf("%d\n", *ptr);

    return 0;
}
```




```
-8
```

PROBLEM 8:

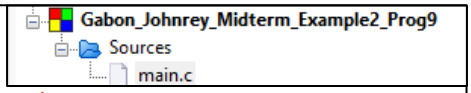
```c
#include <stdio.h>

// Define the structure for NodeEmployer
typedef struct NodeEmployer
{
    int id;
    char name[20];
    int age;
    char address[20];
} NodeEmployer;
int main()
{
    printf("Create struct NodeEmployer successfully");
    return 0;
}
```




```
Create struct NodeEmployer successfully
```

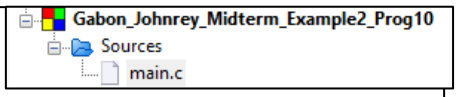
PROBLEM 9:

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

typedef struct NodeEmployer {
    int id;
    char name[20];
    int age;
    char address[20];
    struct NodeEmployer *next;
} NodeEmployer;

NodeEmployer *head = NULL;
int SizeofTheList() {
    int count = 0;
    NodeEmployer *curr = head;

    while (curr != NULL) {
        count++;
        curr = curr->next;
    }
    return count;
}
int main(){
    NodeEmployer* emp1 = (NodeEmployer*)malloc(sizeof(NodeEmployer));
    emp1->id = 1;
    strcpy(emp1->name, "Johnrey Gabon");
    strcpy(emp1->address, "Valenzuela City");
    emp1->next = NULL;

    head = emp1;

    printf("Size of the list: %d\n", SizeofTheList());
    free(emp1);
    return 0;
}
```




```
Size of the list: 1
```

PROBLEM 10:

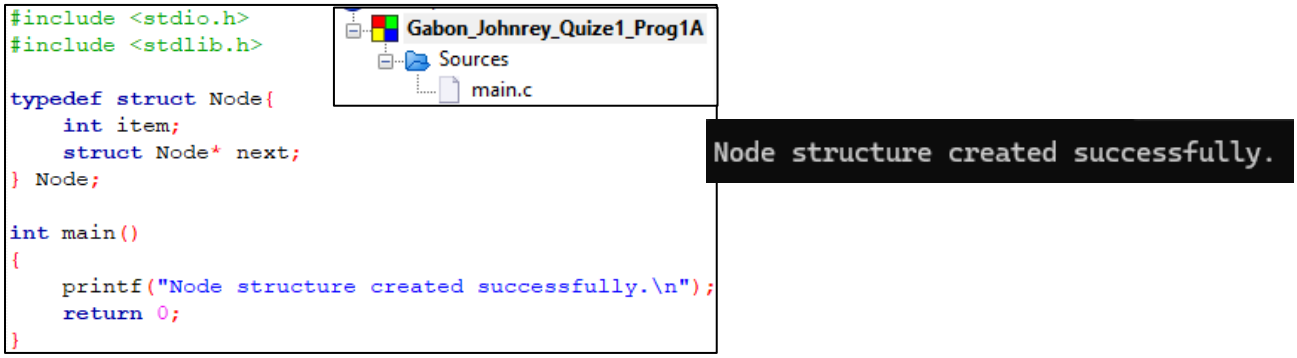



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct NodeEmployer {
    int id;
    char name[20];
    int age;
    char address[20];
    struct NodeEmployer *next;
} NodeEmployer;

NodeEmployer *head = NULL;
void AddEmployer(int id, char *name, int age, char *address) {
    NodeEmployer *newNode = (NodeEmployer *)malloc(sizeof(NodeEmployer));
    newNode->id = id;
    strcpy(newNode->name, name);
    newNode->age = age;
    strcpy(newNode->address, address);
    newNode->next = head;
    head = newNode;
}
int SizeofTheList() {
    int count = 0;
    NodeEmployer *curr = head;

    while (curr != NULL) {
        count++;
        curr = curr->next;
    }
    return count;
}

void DisplayNames() {
    NodeEmployer *curr = head;

    printf("Employers with age > 60:\n");
    while (curr != NULL) {
        if (curr->age > 60) {
            printf("%s\n", curr->name);
        }
        curr = curr->next;
    }
}

int main() {

    AddEmployer(1, "Michael", 62, "Bignay");
    AddEmployer(2, "Gabriel", 99, "Karuhatan");
    AddEmployer(3, "Andrew", 18, "Malinta");

    printf("Total number of employers: %d\n", SizeofTheList());
    DisplayNames();

    return 0;
}
```

```
Total number of employers: 3
Employers with age > 60:
Gabriel
Michael
```

QUIZE 1

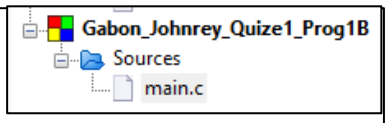
PROBLEM 1A:




```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.
```

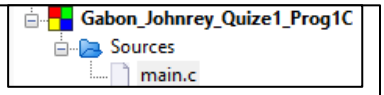
PROBLEM 1B:




```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:

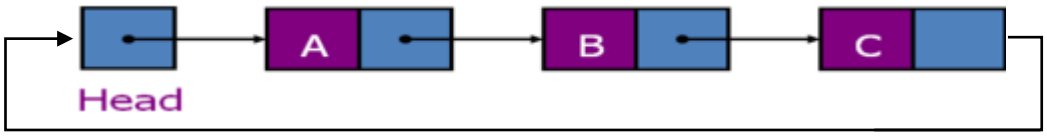
Gabon_Johnrey_Quize1_Prog5B
Sources
main.c

```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;
}
```

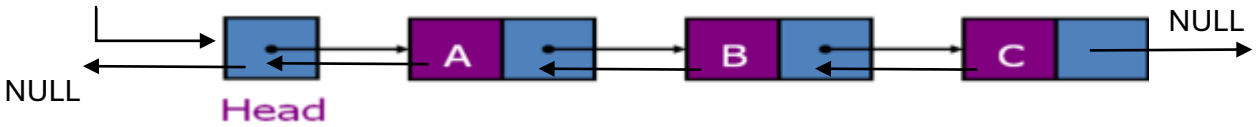
```
Before removal:
5 -> 10 -> 100 -> NULL
After removing 100:
5 -> 10 -> NULL
```

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.

QUIZE 2

Question 1: Evaluate the following postfix expression using stack data structure.

Postfix Expression:

2 + ((5 + 4) * 3) + 1

Let's break it down step by step. We'll use a stack to evaluate the expression.

Push 2 onto the stack:
Stack = [2]
Push 5 onto the stack:
Stack = [2, 5]
Push 4 onto the stack:
Stack = [2, 5, 4]
Encounter "+" (addition):
Pop 4 and 5 from the stack, add them:
5 + 4 = 9
Push the result (9) back onto the stack:
Stack = [2, 9]
Push 3 onto the stack:
Stack = [2, 9, 3]
Encounter "*" (multiplication):
Pop 3 and 9 from the stack, multiply them:
9 * 3 = 27
Push the result (27) back onto the stack:
Stack = [2, 27]
Encounter "+" (addition):
Pop 27 and 2 from the stack, add them:
2 + 27 = 29
Push the result (29) back onto the stack:
Stack = [29]
Push 1 onto the stack:
Stack = [29, 1]
Encounter "+" (addition):
Pop 1 and 29 from the stack, add them:
29 + 1 = 30
Push the result (30) back onto the stack:
Stack = [30]

Answer: The result is 30.

Question 2: Consider a stack A. What will be in the stack A after the following operations?

Operations:

Push(5):
Stack = [5]
Push(10):
Stack = [5, 10]
Pop():
Pop the top element (10):
Stack = [5]
Push(2):
Stack = [5, 2]
Push(5):
Stack = [5, 2, 5]

Push(Pop() + Pop()):

Pop 5 and 2, then add them:

5 + 2 = 7

Push the result (7) back onto the stack:

Stack = [5, 7]

Pop():

Pop the top element (7):

Stack = [5]

Push(9):

Stack = [5, 9]

Push(1):

Stack = [5, 9, 1]

Push(Pop()):

Pop 1 and push it back onto the stack:

Stack = [5, 9, 1]

Answer: The final stack is [5, 9, 1].

Question 3: Using stack, check if the following expression is correct or not.

Expression:

a + (b + 3) - ((c * d) + e

To check if this expression has balanced parentheses, we use a stack.

a: No parentheses, ignore.

+: No parentheses, ignore.

(: Push onto stack: Stack = ['(']

b: No parentheses, ignore.

+: No parentheses, ignore.

3: No parentheses, ignore.

): Pop from stack to match the opening parenthesis: Stack = []

-: No parentheses, ignore.

(: Push onto stack: Stack = ['(']

(: Push onto stack: Stack = ['(', '(']

c: No parentheses, ignore.

*: No parentheses, ignore.

d: No parentheses, ignore.

): Pop from stack to match the opening parenthesis: Stack = ['(']

+: No parentheses, ignore.

e: No parentheses, ignore.

At the end, there is still one unmatched opening parenthesis in the stack, which means the expression is not balanced.

Answer: The expression is incorrect.

Question 4: Convert the following infix expression to postfix expression using stack.

a. Infix: 8 * 2 - 3

8: Operand, add directly to the postfix expression: Postfix = 8

*: Operator, push onto the stack: Stack = [*]

2: Operand, add directly to the postfix expression: Postfix = 8 2

-: Operator, pop from the stack until encountering an operator with lower precedence (pop * and add to the postfix expression):

Postfix = 8 2 *

Then push - onto the stack: Stack = [-]

3: Operand, add directly to the postfix expression: Postfix = 8 2 * 3

Pop - from the stack and add to the postfix expression: Postfix = 8 2 * 3 -

Converted Postfix Expression: 8 2 * 3 -

b. Infix: 1 * 3 * 3 + 5 + 1

1: Operand, add directly to the postfix expression: Postfix = 1

*: Operator, push onto the stack: Stack = [*]

3: Operand, add directly to the postfix expression: Postfix = 1 3

*: Operator, pop from the stack (pop * and add to the postfix expression):

Postfix = 1 3 *

Then push * onto the stack: Stack = [*]

3: Operand, add directly to the postfix expression: Postfix = 1 3 * 3

+: Operator, pop from the stack (pop * and add to the postfix expression):

Postfix = 1 3 * 3 *

Then push + onto the stack: Stack = [+]

5: Operand, add directly to the postfix expression: Postfix = 1 3 * 3 * 5

+: Operator, pop from the stack (pop + and add to the postfix expression):

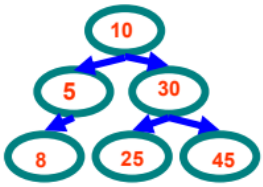
Postfix = 1 3 * 3 * 5 +

Then push + onto the stack: Stack = [+]
1: Operand, add directly to the postfix expression: Postfix = 1 3 * 3 * 5 + 1
Pop + from the stack and add to the postfix expression: Postfix = 1 3 * 3 * 5 + 1 +

Converted Postfix Expression: 1 3 * 3 * 5 + 1 +

QUIZE 2

PROBLEM 1. Consider the following tree and answer the following questions.



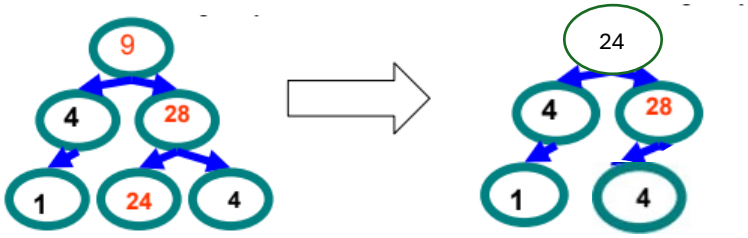
a. Is it a Binary Search Tree. (Yes or No)

b. Is it a Complete Binary Tree (Yes or No)

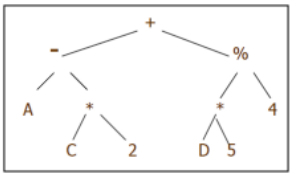
c. Is it a Balance Tree (Yes or No)

A. No – The left child of node 5 (8) violates the Binary Search Tree rule.
B. Yes – All levels are filled except the last, which is filled from left to right.
C. Yes – The height difference between left and right subtrees of every node is at most 1.

PROBLEM 2: Consider the following binary tree. What is the result if you delete 9 from the tree.

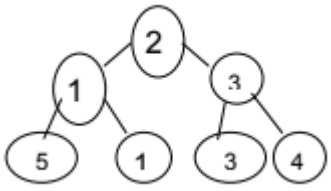


PROBLEM 3: Consider the following parse tree. What is the output after inorder traversal.



Answer: A-C*2 + D * 5 % 4

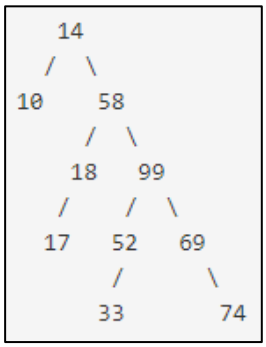
PROBLEM 4: Consider the following tree and answer the following questions.



MIN HEAP
1
/ \
1 2
/\ /\
5 33 4

MAX HEAP
5
/ \
4 3
/\ /\
1 12 3

PROBLEM 5: Draw the binary search tree that would result from the insertion of the following integer keys:



PROBLEM 6:

A. The node with the value 5 is a parent of the node with the value 10 (True/False). **False. The node with the value 10 is the parent of the node with the value 5.**

B. The node with the value 30 is a child of the node with the value 25 (True/False). **False. The node with the value 25 is a child of the node with the value 30.**

C. The tree is a complete tree (True/False). **False.** A complete tree is a binary tree in which all levels are filled except possibly the last level, and the last level has all keys as left as possible. This tree does not meet those criteria.

D. What is the depth of the tree? **Depth of the tree: 2**

F. What is the order of nodes visited using a pre-order traversal?

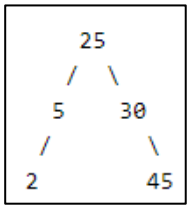
- Pre-order traversal: 10, 5, 2, 30, 25, 45

g. What is the order of nodes visited using a post-order traversal?

- Post-order traversal: 2, 5, 25, 45, 30, 10

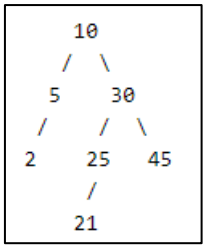
H. We remove the root, what will be the new tree?

- Here, the in-order successor is 25.
- New tree after removing 10 and replacing with 25:



I. We add the element 21, what will be the new tree?

- 21 is less than 30 but greater than 25, so it becomes the left child of 25.
- New tree after adding 21:



REVIEW QUESTIONS FOR DATA STRUCTURES

Give two reasons why dynamic memory allocation is a valuable device.

- Dynamic memory allocation is valuable because it allows programs to utilize memory more efficiently by allocating memory as needed during runtime.
- It provides flexibility, as it adapts to the changing memory requirements of a program and avoids fixed allocation limitations in static memory allocation.

Is it easier to insert a new node before or after a specified node in a linked list? Why?

- It is generally easier to insert a node after a specified node_in a linked list because:
- To insert after, only the pointer of the specified node needs to be updated to point to the new node, and the new node's pointer is set to the next node. Inserting before requires traversing the list to find the previous node, which adds complexity and time unless the previous node is already available in special cases.

What are the differences between a linked list and an array? Why would you choose one or the other? Your answer must include the following points:

- **a. Insertion performance**
- **Array**: Insertion can be inefficient as it may require shifting elements, especially for insertion at the beginning or middle.
- **Linked List**: Insertion is efficient as you only need to update pointers, regardless of the position.
- **b. Memory allocation**
- **Array**: Requires contiguous memory allocation, which might lead to memory shortages.
- **Linked List**: Uses dynamic memory allocation and is stored non-contiguously, which consumes memory for the pointers but provides flexibility.
- **c. Impact of removing elements (beginning/middle/end)**
- **Array:** Removal at the beginning or middle shifts all subsequent elements, making it slower.
- **Linked List:** Removal is efficient, as only the pointers need to be updated. No element shifting is required.

Use **array** when faster direct access (indexing) is needed and the size is predefined.

Use **linked list** when frequent insertions and deletions occur or the size is dynamic.

If the items in a list are floats taking 4 words each, compare the amount of space required altogether if

- **a.) the list is kept contiguously in an array 90 percent full**:
  Space = Array size × Word size = 1.11 × Actual items × 4 words.
- **b.) the list is kept contiguously in an array 60 percent full:**
  Space = 1.66 × Actual items × 4 words.
- **c.) the list is kept as a linked list (where the pointers take one word each)**
  Space = Items × (Data size + Pointer size) = Actual items × (4 + 1) = Actual items × 5 words.

If the items in a list are structures or records taking 200 words each, compare the amount of space required altogether if

- **(a) the list is kept contiguously in an array 85 percent full:** Space = Array size × Word size = 1.18 × Actual items × 200 words.
- **(b) the list is kept contiguously in an array 50 percent full:** Space = 2 × Actual items × 200 words.
- **(c) the list is kept as a linked list (where the pointers take one word each):**
  Space = Items × (Data size + Pointer size) = Actual items × (200 + 1) = Actual items × 201 words.

When it is appropriate to use the data structure Array?

- Arrays are appropriate when you need to store a fixed-size collection of elements of the same type and when you need fast access to elements by index.

What are the major disadvantages of the Array?

- Fixed size (can't resize after creation), inefficient for insertions or deletions in the middle or beginning.

Complete the following sentences.

i. **Front** refers to the first item in a queue.

ii. In **Stack**, insertion and deletion is performed at only one end called **the top**.

iii. Time complexity of linear search in the average case for n data items is **O(n).**

Dynamic memory allocation has overcome the drawbacks of static memory allocation. How?

- Dynamic memory allocation allows memory to be allocated at runtime, enabling flexible and efficient memory usage. Unlike static memory allocation, where the size of the data structure is fixed, dynamic memory allocation can grow or shrink as needed, preventing wastage or overflow.

What is the major disadvantage of linked lists in comparison with contiguous lists?

- Linked lists use extra memory for storing pointers, and they require more time for access because you need to traverse the list node by node. In contrast, contiguous lists (like arrays) provide direct access to elements.

How to overcome the false overflow problem? When this problem occurs?

- False overflow occurs in data structures like stacks or queues when they are implemented with static memory, but the structure is not actually full, just incorrectly marked as full due to limitations in how the memory is managed (like with circular buffers). To overcome this, ensure proper management of memory and use dynamic allocation or implement circular data structures correctly to prevent false overflow.

How a node can be inserted in the beginning, middle, and end of a linked list?

- **Beginning:** Create a new node, point it next to the current head, and update the head to this new node.
- **Middle:** Traverse to the desired position and adjust the pointers of the previous node and the new node so the new node points to the next node.
- **End:** Traverse to the last node, create a new node, and set the last node's next to point to this new node. If doubly linked, also update the new node's previous pointer.

Using stack, evaluate the postfix expression A B C D - * +, where A = 25, B = 2, C = 18 and D=13?

Final result: 35

To evaluate a postfix expression, you follow these steps:
Start from the left, and push operands (A, B, C, D) onto the stack.
When you encounter an operator, pop the required operands from the stack, perform the operation, and push the result back onto the stack.
Postfix expression: A B C D - * +

Push A = 25 onto the stack.
Push B = 2 onto the stack.
Push C = 18 onto the stack.
Push D = 13 onto the stack.
Encounter -: Pop C = 18 and D = 13, calculate 18 - 13 = 5, and push the result (5) onto the stack.
Encounter *: Pop B = 2 and the result of the subtraction (5), calculate 2 * 5 = 10, and push the result (10) onto the stack.
Encounter +: Pop A = 25 and the result of the multiplication (10), calculate 25 + 10 = 35.

Is it easier to insert a new node before or after a specified node in a linked list? Why? It is easier to insert a new node after a specified node in a linked list because you only need to update the pointer of the previous node to point to the new node, and the new node will point to the next node. For insertion before a node, you need to traverse to the previous node, which requires more effort.

List three operations that are possible in Array and Linked list but are not allowed in Stack.

- **Random Access (Array):** Arrays allow direct access to any element by its index, whereas stacks do not.
- **Insertion at Specific Positions (Array, Linked List):** Inserting at any position is allowed in arrays and linked lists but not in stacks (which only allows insertion at the top).
- **Traversal (Array, Linked List):** Both arrays and linked lists allow traversal through all elements, while stacks restrict access to only the top element.

What is a queue? Show two possible techniques of memory allocations for a queue. Discuss their advantages and disadvantages.

A **queue** is a linear data structure that follows the FIFO (First In, First Out) principle. Elements are added at the rear and removed from the front.

Two techniques for memory allocation in a queue:

Static Array Allocation:

Advantages: Simple to implement and provides fast access.

Disadvantages: Fixed size; if the queue is full, no more elements can be added. Wastage of memory may occur if the queue size is too large.

Dynamic (Circular) Array Allocation:

Advantages: Allows efficient use of memory, and the array size can be adjusted as needed. Circular allocation avoids memory wastage by reusing empty spaces when the front elements are dequeued.

Disadvantages: More complex to implement and resizing or managing the wrap-around behavior adds extra overhead.

When linear search algorithm is better than binary search algorithm? Why?

Linear search is better when:

The data is unsorted. Binary search requires sorted data, whereas linear search works with both sorted and unsorted data.

You have a small dataset, where the overhead of sorting or maintaining the order for binary search isn't worth it.

Compare linear search algorithm with Binary search algorithm (Refer to their big O notations)

Linear Search:

Time Complexity: O(n), where n is the number of elements.

It checks each element one by one, so it may need to scan the entire list.

Binary Search:

Time Complexity: O(log n), where n is the number of elements.

It works by repeatedly dividing the list into half and checking only a specific range of elements. This makes it much faster than linear search, but the list must be sorted beforehand.

What is the difference between pointer p = nil and p is undefined?

pointer p = nil: This means that the pointer p is explicitly set to nil (or null), meaning it is intentionally pointing to nothing. It is a known state.
p is undefined: This means the pointer p has not been assigned any value yet. It is in an undefined state and its value is unknown, leading to potential issues like segmentation faults or unpredictable behavior.

What are the impacts of storing huge number of integers in array?

Memory consumption: Storing a large number of integers will require a significant amount of memory. Each integer typically consumes 4 bytes of memory, so large arrays can use a substantial amount of system memory.
Performance issues: If the array becomes too large, it could lead to slower performance due to memory access times, especially if the system is running out of available memory (paging or swapping).
Cache inefficiency: Large arrays may not fit in CPU cache, causing slower access times.

Which data structure you should use if a program keeps track of patients as they check into a medical clinic, assigning patients to doctors on a first-come, first-served basis. The **queue** data structure should be used because it follows the FIFO (First-In, First-Out) principle, which perfectly fits the requirement of handling patients in the order they check in.

Suppose we have an integer-valued stack S and a queue Q. What are final values in the stack S and in the Q after the following operations. Show contents of both S and Q at each step indicated by the line.

```
Stack S;
Queue Q;
int x =10, y=20;
```

S.push(x); S.push(y); S.push(S.pop()+S.pop()); Q.enqueue(x); Q.enqueue(y);
Q.enqueue(S.pop());
S.push(Q.dequeue()+Q.dequeue());

Initial State:

Stack S: (empty)
Queue Q: (empty)
S.push(x): Push 10 onto stack S.
Stack S: 10
Queue Q: (empty)
S.push(y): Push 20 onto stack S.
Stack S: 10, 20
Queue Q: (empty)
S.push(S.pop() + S.pop()):

Pop 20 and 10 from **the stack, perform 10 + 20 = 30, then push the result (30) back onto the stack.**

Stack S: 30
Queue Q: (empty)
Q.enqueue(x): Enqueue 10 onto queue Q.
Stack S: 30

Queue Q: 10
Q.enqueue(y): Enqueue 20 onto queue Q.
Stack S: 30
Queue Q: 10, 20
Q.enqueue(S.pop()): Pop 30 from the stack, then enqueue 30 onto queue Q.
Stack S: (empty)
Queue Q: 10, 20, 30
S.push(Q.dequeue() + Q.dequeue()):
Dequeue 10 and 20 from the queue, perform 10 + 20 = 30, then push 30 onto the stack.
Stack S: 30
Queue Q: 30

Sort the given values using Bubble Sort, indicating the number of passes and the number of comparisons.

70	75	85	60	55	50

Initial Values:
70, 75, 85.60, 55.50

Pass 1:

Compare 70 and 75: no swap needed (70 < 75).
Compare 75 and 85.60: no swap needed (75 < 85.60).

Compare 85.60 and 55.50: swap (85.60 > 55.50).
New array: 70, 75, 55.50, 85.60
After Pass 1, we have bubbled the largest value (85.60) to the end.

Pass 2:

Compare 70 and 75: no swap needed (70 < 75).
Compare 75 and 55.50: swap (75 > 55.50).
New array: 70, 55.50, 75, 85.60
After Pass 2, 75 is in its final position.

Pass 3:

Compare 70 and 55.50: swap (70 > 55.50).
New array: 55.50, 70, 75, 85.60
After Pass 3, the array is fully sorted.

Summary:
Total Passes: 3
Total Comparisons: 6 (3 comparisons per pass for 3 passes)

Sorted Array:

55.50, 70, 75, 85.60

Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix and Postfix notations.

Prefix : - * + A B C ^ - D E + F G **Postfix:** A B + C * D E - F G + ^ -

Consider the following stack of characters, where STACK allocates memory size for 8 characters. STACK: A, C, D, F, K, ___, ___, ___ where "___" denotes an empty stack cell. Describe the stack as the following operations take place:

Initial Stack:

STACK: A, C, D, F, K, ___, ___, ___

There are 5 elements in the stack initially (A, C, D, F, K), and 3 empty spaces.

(a) POP():
This operation removes the top element from the stack, which is K.
STACK: A, C, D, F, ___, ___, ___, ___

(b) POP():
This operation removes the next top element, which is F.
STACK: A, C, D, ___, ___, ___, ___, ___

(c) PUSH(L):
This operation pushes L onto the top of the stack.
STACK: A, C, D, L, ___, ___, ___, ___

(d) PUSH(P):
This operation pushes P onto the top of the stack.
STACK: A, C, D, L, P, ___, ___, ___

(e) POP():
This operation removes the top element, which is P.
STACK: A, C, D, L, ___, ___, ___, ___

(f) PUSH(R):
This operation pushes R onto the top of the stack.
STACK: A, C, D, L, R, ___, ___, ___

(g) PUSH(S):
This operation pushes S onto the top of the stack.
STACK: A, C, D, L, R, S, ___, ___

(h) POP():
This operation removes the top element, which is S.
STACK: A, C, D, L, R, ___, ___, ___

Final Stack:
STACK: A, C, D, L, R, ___, ___, ___

Suppose STACK is allocated memory space for 6 integers and initially its top = -1. Show stack behavior and find the output of the following pseudo code:

```
1. a := 2; b := 5;
2. push (a);
3. push (b+2);
4. push (9);
5. while (top <> -1)
6. { pop ( item);
7. print item; }
```

Answer:

```
1. a := 2; b := 5;  6. Top: 0          11. Stack: [2, 7, 9]
2. a = 2            7. push(b + 2);    12. Top: 2
3. b = 5            8. Stack: [2, 7]   13. while (top <> -1) { pop(item); print item; }
4. push(a);         9. Top: 1
5. Stack: [2]       10. push(9);
```

While Loop Execution:

```
1. First Iteration:      6. Second Iteration:     11. Third Iteration:
2. Pop: 9                7. Pop: 7                12. Pop: 2
3. Print: 9              8. Print: 7              13. Print: 2
4. Stack: [2, 7]         9. Stack: [2]            14. Stack: []
5. Top: 1                10. Top: 0               15. Top: -1
```

Output: 9 7 2

Assume that you have a stack S, a queue Q, and the standard stack - queue operations: push, pop, enqueue and dequeue. Assume that print is a function that prints the value of its argument. Execute the operations below to show only the output of each print function.

```
push(S, 'T');
enqueue(Q, 'I');
push(S,dequeue(Q));
enqueue(Q, 'I');
enqueue(Q, 'G');
print(dequeue(Q));
enqueue(Q, T);
push(S, 'I');
push(S, dequeue(Q));
print(pop(S));
enqueue(Q, pop(S));
push(S, 'O');
print(pop(S));
enqueue(Q, 'O');
print(dequeue(Q));
enqueue(Q, pop(S));
push(S, dequeue(Q));
print(pop(S));
print(pop(S));
```

```
print(dequeue(Q));      // Output: I
print(pop(S));          // Output: T
print(pop(S));          // Output: I
print(pop(S));          // Output: O
print(dequeue(Q));      // Output: G
print(pop(S));          // Output: T
```

Consider the following queue QUEUE is allocated 6 memory cells:

FRONT= 1, REAR = 4 and

QUEUE: ___, London, Berlin, Rome, Paris, ___.

Describe queue's behavior, including FRONT and REAR, as the following operations take place:

(a) "Athens" is added (b) Two cities are deleted

(c) "Moscow" is added (d) "Madrid" is added

(a) "Athens" is added (Enqueue operation)

Operation: Add "Athens" to the queue.
Update: The value is added to the position after Paris (i.e., at index 5).
FRONT = 1, REAR = 5
Result: QUEUE: ___, London, Berlin, Rome, Paris, Athens

(b) Two cities are deleted (Dequeue operation)

Operation: Remove two cities from the front of the queue (starting from FRONT = 1).
First dequeue: Remove London from the queue.
Second dequeue: Remove Berlin from the queue.
FRONT = 3, REAR = 5
Result: QUEUE: ___, ___, Rome, Paris, Athens, ___

(c) "Moscow" is added (Enqueue operation)

Operation: Add "Moscow" to the queue.
FRONT = 3, REAR = 6
Result: QUEUE: ___, ___, Rome, Paris, Athens, Moscow

(d) "Madrid" is added (Enqueue operation)

Enqueue Operation: Add "Madrid" to the queue. However, since the queue is already full (6 memory cells), this operation will fail. A queue overflow occurs in this case.

Give some examples where the data structures Stack and Queue are used in computer when executing a program.

Examples where Stack is used:

- **Function Call Stack:** When a function is called in a program, the system stores its information (parameters, return address, etc.) on the call stack. When the function returns, this information is popped off the stack.
- **Undo Operations in Text Editors**: Text editors like Word use a stack to store actions (such as typing or deleting text), allowing users to undo or redo those actions.

Examples where Queue is used:

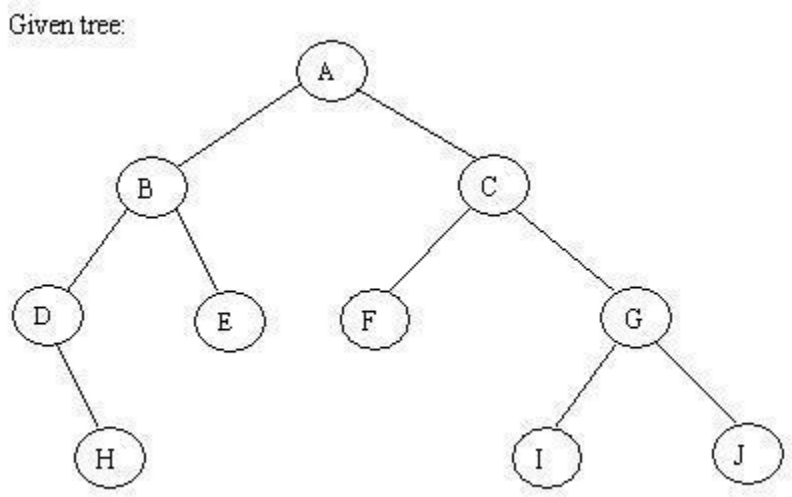
- **Task Scheduling:** Operating systems use queues to manage tasks in a multi-tasking environment. Processes are scheduled to run in a FIFO (First In, First Out) manner.
- **Print Spooling:** In a printer queue, print jobs are processed in the order they arrive, ensuring that the first job is printed first.

How many different trees are possible with 3 nodes ?

$$C(3) = \frac{1}{3+1}\left(\frac{6}{3}\right) = \frac{1}{4} * 20 = 5$$

There are **5 different binary trees** possible with 3 nodes.

Traverse the given following tree using Inorder, Preorder and Postorder traversals.



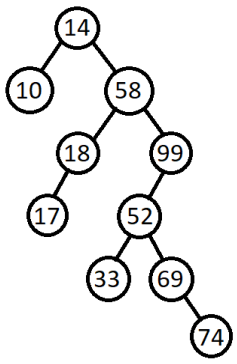
Inorder Traversal: **D, H, B, E, A, F, C, I, G, J**

Preorder Traversal: **A, B, D, H, E, C, F, G, I, J**

Postorder Traversal: **H, D, E, B, F, I, J, G, C, A**

Draw the binary search tree that would result from the insertion of the following integer keys:

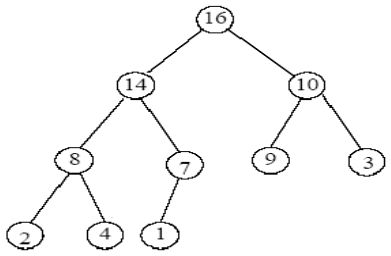
14, 58, 18, 10, 99, 52, 33, 69, 74, 17



What is the maximum number of nodes of a binary tree with height 3? 15

Maximum number of nodes = 2^ (3+1) - 1 = 2^4 - 1 = 16 - 1 = 15

Given the following tree T, answer the following questions and give a reason for your answer in case of (Yes/ No)



1. Is T a binary tree?

Answer: **Yes**, T is a binary tree. A binary tree is a tree in which each node has at most two children, and T follows this property

2. Is T a binary search tree?

Answer: **No**, T is not a binary search tree. In a BST, for each node, the value of its left subtree nodes must be smaller, and the value of its right subtree nodes must be greater. In T, the left and right subtrees of some nodes violate this condition, so it is not a BST.

3. Is T a max-heap?

Answer: **No**, T is not a max-heap. In a max-heap, for every node, the value of the node must be greater than or equal to the values of its children. In T, the parent nodes (16, 14, 10, etc.) are not always greater than their children, so it doesn't satisfy the max-heap property.

4. Is T full?

Answer: **No**, T is not a full binary tree. A full binary tree is one where every node has either 0 or 2 children. In T, the node with value 9 has only one child (3), so it is not full.

5. Is T complete?

Answer: **No**, T is not a complete binary tree. A complete binary tree is one where all levels are filled except possibly for the last level, which should be filled from left to right. The tree does not satisfy this condition.

6. Represent data nodes of T using an array data structure.

Answer: `int T[10] = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1];`

Consider the following contiguous implementation of a tree. Answer the following questions:

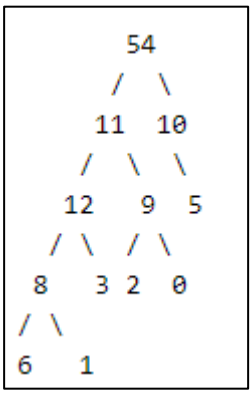
a. What is the left child of 2? **Left Child of 2 is 3**

b. What is the parent of 2? **Parent of 2 is 8**

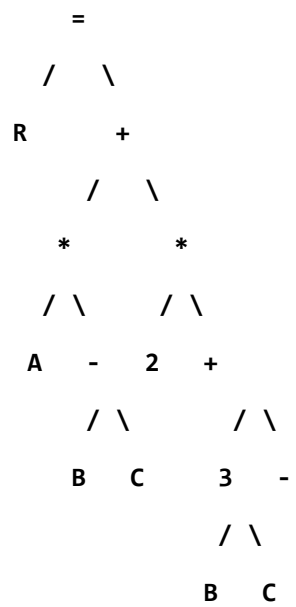
15	8	20	2	11	30	27	13	6	18	12

Build the heap from the following items. Show should be done to keep the heapness (structure and order) properties of the tree.

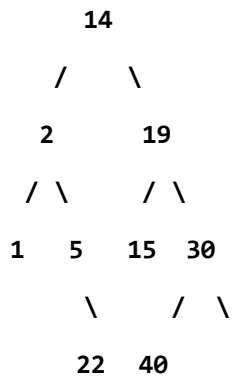
11 54 10 2 9 5 8 3 12 0 6 1



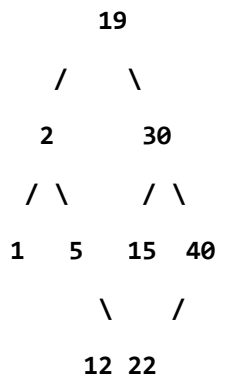
Draw the parse tree of the expression R = A * (B – C) + 2 * (3 + B - C)



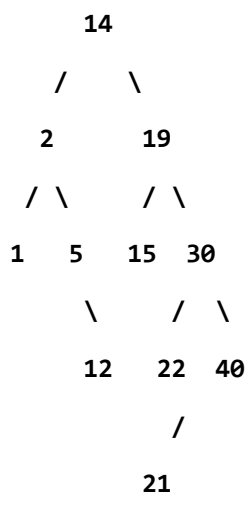
Here is a small binary tree of integers that is needed for the next seven questions.



a. The node with the value 12 is an parent of the node with the value 2 (True/False). False
b. The node with the value 40 is a child of the node with the value 15 (True/False). True
c. The tree is a complete tree (True/False). False
d. The tree is a complete tree (True/False). False
e. What is the depth of the tree? 4
f. What is the order of nodes visited using a pre-order traversal? 14, 2, 1, 5, 12, 19, 15, 30, 22, 40
g. What is the order of nodes visited using a post-order traversal? 1, 12, 5, 2, 15, 22, 40, 30, 19, 14
h. We remove the root, what will be the new tree?



i. We add the element 21, what will be the new tree?



What is the complexity of binary search algorithm? Justify.

O ($\log_2 n$)

Binary search divides the input array into half at every step and discards one half of the elements, leaving only the relevant half to search in.The number of steps required to reduce n elements to 1 is logarithmic to the base 2, i.e., $\log_2 n$.This makes the time complexity **O($\log_2 n$)**, where n is the size of the input.

Compute the time complexity of the following code:

```
sum := 0
for (int i = 0 ; i < n ; i++)
      for (int j = 0 ; j < m ; j++)
            sum := sum + i + j
```

Answer: O(n * m)

The outer loop runs n times. For each iteration of the outer loop, the inner loop runs m times. Inside the inner loop, there is a constant time operation (sum := sum + i + j). Total operations = n * m. Thus, the time complexity is O(n * m).

- **Arrange the following functions, often used to represent complexity of algorithms in order from slowest to fastest**

$O(1)$, $O(n)$, $O(n*\log_2 n)$, $O(\log_2 n)$, $O(n^2)$, $O(2^n)$

$O(2^n)$, $O(n^2)$, $O(n * \log_2 n)$,$O(n)$,$O(\log_2 n)$,$O(1)$

How the performance of an algorithm is measured?

The performance of an algorithm is typically measured based on:

- **Time Complexity**: The time an algorithm takes to complete relative to the size of the input (n)
- **Space Complexity:** The amount of memory or storage an algorithm uses during execution. Both are expressed using **Big-O notation**, which gives an upper-bound performance estimate.

Why time and space are the most important considerations in computer operation?

- **Time:** Efficient algorithms reduce execution time, improving responsiveness and productivity, especially in large-scale or real-time applications.
- **Space:** Memory is limited, and inefficient algorithms may exhaust memory or require more resources, leading to failures.

Why is the complexity of an algorithm generally more important than the speed of the processor?

- **Processor speed is fixed**: **Processor speed is fixed:** Even the fastest processor cannot compensate for an algorithm with poor complexity. For example, an $O(2^n)$ algorithm will remain slow for large inputs, regardless of processor speed.

What are the factors you will consider to select the proper data structures and memory allocation mechanism when building an algorithm?

- **Nature of the data:** Whether the data is static or dynamic, structured, or unstructured.
- **Operations required:** The type of operations needed, like insertion, deletion, searching, sorting, or traversal.
- **Time constraints:** How quickly the algorithm must execute for acceptable performance.
- **Space constraints:** The amount of memory available for the data structure and algorithm.
- **Complexity of access:** Whether random access (e.g., arrays) or sequential access (e.g., linked lists) is needed.
- **Scalability:** The ability to handle increasing input sizes without significant performance degradation.

Sample for MCQ

C **1.Two main measures for the efficiency of an algorithm are_**

A. Processor and memory

B. Complexity and capacity

C. Time and space

D. Data and space

B 2. The time factor when determining the efficiency of algorithm is measured by

A. Counting microseconds

B. Counting the number of key operations

C. Counting the number of statements

D. Counting the kilobytes of algorithm

A 3. The space factor when determining the efficiency of algorithm is measured by

A. Counting the maximum memory needed by the algorithm

B. Counting the minimum memory needed by the algorithm

C. Counting the average memory needed by the algorithm

D. Counting the maximum disk space needed by the algorithm

D 4. Which of the following case does not exist in complexity theory

A. Best case

B. Worst case

C. Average case

D. Null case

D 5. The Worst case occur in linear search algorithm when_

A. Item is somewhere in the middle of the array

B. Item is not in the array at all

C. Item is the last element in the array

D. Item is the last element in the array or is not there at all

A 6. The Average case occur in linear search algorithm_

A. When Item is somewhere in the middle of the array

B. When Item is not in the array at all

C. When Item is the last element in the array

D. When Item is the last element in the array or is not there at all

A 7. The complexity of the average case of an algorithm is

A. Much more complicated to analyze than that of worst case

B. Much more simpler to analyze than that of worst case

C. Sometimes more complicated and some other times simpler than that of worst case

D. None or above

A 8. The complexity of linear search algorithm is

A. O(n)

B. O(log n)

C. O(n2)

D. O(n log n)

B 9. The complexity of Binary search algorithm is

A. O(n)

B. O(log n)

C. O(n2)

D. O(n log n)

C 10. The complexity of Bubble sort algorithm is

A. O(n)

B. O(log n)

C. O(n2)

D. O(n log n)

D 11. The complexity of merge sort algorithm is

A. O(n)

B. O(log n)

C. O(n2)

D. O(n log n)

D 12.Which of the following data structure is not linear data structure?

A. Arrays

B. Linked lists

C. Both of above

D. None of above

C 13.Which of the following data structure is linear data structure?

A. Trees

B. Graphs

C. Arrays

D. None of above

D **14.The operation of processing each element in the list is known as**

A. Sorting

B. Merging

C. Inserting

D. Traversal

B **15.Finding the location of the element with a given value is:**

A. Traversal

B. Search

C. Sort

D. None of above

A **16.Arrays are best data structures**

A. for relatively permanent collections of data

B. for the size of the structure and the data in the structure are constantly changing

C. for both of above situation

D. for none of above situation

B **17.Linked lists are best suited**

A. for relatively permanent collections of data

B. for the size of the structure and the data in the structure are constantly changing

C. for both of above situation

D. for none of above situation

C 18.Each array declaration need not give, implicitly or explicitly, the information about

A. the name of array

B. the data type of array

C. the first data from the set to be stored

D. the index set of the array

A 19.The elements of an array are stored successively in memory cells because_

A. by this way computer can keep track only the address of the first element and the addresses of other elements can be calculated

B. the architecture of computer memory does not allow arrays to store other than serially

C. both of above

D. none of above

A 20. Inserting an item into the stack when stack is not full is called ……. Operation and deletion of an item form the stack, when stack is not empty is called …….operation.

A) push, pop

B) pop, push

C) insert, delete

D) delete, insert

B 21. Is a pile in which items are added at one end and removed from the other.

A) Stack

B) Queue

C) List

D) None of the above

A 22. is very useful in situation when data have to stored and then retrieved in reverse order.

A) Stack

B) Queue

C) List

D) Link list

A 23. Stack is also called as

A) Last in first out

B) First in last out

C) Last in last out

D) First in first out

D 24.The five items: A, B, C, D and E are pushed in a stack, one after the other starting from A. The stack is popped four times, and each element is inserted in a queue. Then two elements are deleted from the queue and pushed back on the stack. Now one item is popped from the stack.

The popped item is.

A) A

B) B

C) C

D) D

A 25. If post order traversal generates sequence xy-zw*+, then label of nodes 1,2,3,4,5,6,7 will be

A. +, -, *, x, y, z, w

B. x, -, y, +, z, *, w

C. x, y, z, w, -, *, +

D. -, x, y, +, *, z, w

PROBLEM 7 (PROGRAM 1)

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

#define MAX_PRODUCTS 5

struct Product {
    char name[30];
    float price;
    int quantity;
};
struct Store {
    struct Product items[MAX_PRODUCTS];
    int count;
};
// initialized features methods
void addProduct(struct Store *store);
void showProducts(struct Store *store);
void updateStock(struct Store *store);
float calculateTotalValue(struct Store *store);

int main() {
    struct Store store = {.count = 0};
    int choice;

    do {
        printf("\n1. Add Product\n");
        printf("2. Show Products\n");
        printf("3. Update Stock\n");
        printf("4. Calculate Total Value\n");
        printf("0. Exit\n");
        printf("Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: addProduct(&store); break;
            case 2: showProducts(&store); break;
            case 3: updateStock(&store); break;
            case 4: printf("Total inventory value: $%.2f\n",
                           calculateTotalValue(&store)); break;
            case 0: printf("Goodbye!\n"); break;
            default: printf("Invalid choice!\n");
        }
    } while(choice != 0);

    return 0;
}
void addProduct(struct Store *store) {
    if(store->count >= MAX_PRODUCTS) {
        printf("Store is full!\n");
        return;
    }

    printf("Enter product name: ");
    scanf(" %[^\n]s", store->items[store->count].name);
    printf("Enter price: ");
    scanf("%f", &store->items[store->count].price);
    printf("Enter quantity: ");
    scanf("%d", &store->items[store->count].quantity);

    store->count++;
}
void showProducts(struct Store *store) {
    printf("\nProduct List:\n");
    printf("Name\t\tPrice\tQuantity\n");
    for(int i = 0; i < store->count; i++) {
        printf("%-15s$%.2f\t%d\n",
            store->items[i].name,
            store->items[i].price,
            store->items[i].quantity);
    }
}
void updateStock(struct Store *store) {
    char searchName[30];
    int newQuantity;

    printf("Enter product name: ");
    scanf(" %[^\n]s", searchName);

    for(int i = 0; i < store->count; i++) {
        if(strcmp(store->items[i].name, searchName) == 0) {
            printf("Enter new quantity: ");
            scanf("%d", &newQuantity);
            store->items[i].quantity = newQuantity;
            printf("Stock updated successfully!\n");
            return;
        }
    }
    printf("Product not found!\n");
}
float calculateTotalValue(struct Store *store) {
    float total = 0;
    for(int i = 0; i < store->count; i++) {
        total += store->items[i].price * store->items[i].quantity;
    }
    return total;
}
```

Gabon_Johnrey_Problem7_Prog1
Sources
main.c

```
=====Store Management Program=====
1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
Choice: 1

Enter product name: Milk
Enter price: 150.99
Enter quantity: 50

1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
Choice: 1

Enter product name: Bread
Enter price: 50.00
Enter quantity: 30

1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
```

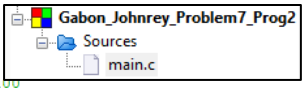
```
1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
Choice: 1

Enter product name: Eggs
Enter price: 15.99
Enter quantity: 40

1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
Choice: 1

Enter product name: Bananas
Enter price: 20.50
Enter quantity: 100
```

```
1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
Choice: 4

Total inventory value: $10984.15

1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
Choice: 0

Goodbye!
```

```
1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
Choice: 3

Enter product name: Milk
Enter new quantity: 45
Stock updated successfully!
```

```
1. Add Product
2. Show Products
3. Update Stock
4. Calculate Total Value
0. Exit
Choice: 2


Product List:
Name            Price   Quantity
Milk            P150.99  50
Bread           P50.00   30
Eggs            P15.99   40
Bananas         P20.50   100
```

PROBLEM 7 (PROGRAM 2)




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

#define MAX_ORDERS 100
typedef enum {
    PENDING,
    IN_PROCESS,
    READY,
    DELIVERED
} Status;
typedef enum {
    WASH = 1,
    DRY_CLEAN,
    IRON
} ServiceType;
struct Order {
    int id;
    char customerName[50];
    ServiceType service;
    float weight;
    float price;
    Status status;
    char dateReceived[20];
};
struct LaundryShop {
    struct Order orders[MAX_ORDERS];
    int orderCount;
    float rates[3];  // rates for wash, dry clean, iron
};
void initializeLaundryShop(struct LaundryShop *shop) {
    shop->orderCount = 0;
    shop->rates[0] = 5.0 * 25;  // wash rate per kg
    shop->rates[1] = 8.0 * 30;  // dry clean rate per kg
    shop->rates[2] = 3.0 * 20;  // iron rate per kg
}
void addOrder(struct LaundryShop *shop) {
    if(shop->orderCount >= MAX_ORDERS) {
        printf("Orders full!\n");
        return;
    }
    struct Order newOrder;
    newOrder.id = shop->orderCount + 1001;
    printf("Enter customer name: ");
    scanf(" %[^\n]s", newOrder.customerName);
    printf("Select service (1-Wash, 2-Dry Clean, 3-Iron): ");
    scanf("%d", (int*)&newOrder.service);
    printf("Enter weight in kg: ");
    scanf("%f", &newOrder.weight);
    printf("Enter date (DD/MM/YYYY): ");
    scanf(" %[^\n]s", newOrder.dateReceived);
    newOrder.price = shop->rates[newOrder.service - 1] * newOrder.weight;
    newOrder.status = PENDING;
    shop->orders[shop->orderCount] = newOrder;
    shop->orderCount++;
    printf("Order added! Total price: P%.2f\n", newOrder.price);
}
void displayOrders(struct LaundryShop *shop) {
    printf("\nCurrent Orders:\n");
    printf("ID\tCustomer\t\tService\t\tStatus\t\tPrice\n");
    for(int i = 0; i < shop->orderCount; i++) {
        char *service[] = {"Wash", "Dry Clean", "Iron"};
        char *status[] = {"Pending", "Processing", "Ready", "Delivered"};
        printf("%d\t%-20s\t%-10s\t%-10s\tP%.2f\n",
            shop->orders[i].id,
            shop->orders[i].customerName,
            service[shop->orders[i].service - 1],
            status[shop->orders[i].status],
            shop->orders[i].price);
    }
    // Display current rates
    printf("\nCurrent Rates (per kg):\n");
    printf("Wash: P%.2f\n", shop->rates[0]);
    printf("Dry Clean: P%.2f\n", shop->rates[1]);
    printf("Iron: P%.2f\n", shop->rates[2]);
}
void updateOrderStatus(struct LaundryShop *shop) {
    int id, newStatus;
    printf("Enter order ID: ");
    scanf("%d", &id);
    for(int i = 0; i < shop->orderCount; i++) {
        if(shop->orders[i].id == id) {
            printf("Enter new status (0-Pending, 1-Processing, 2-Ready, 3-Delivered): ");
            scanf("%d", &newStatus);
            shop->orders[i].status = newStatus;
            printf("Status updated successfully!\n");
            return;
        }
    }
    printf("Order not found!\n");
}
int main() {
    struct LaundryShop shop;
    initializeLaundryShop(&shop);
    int choice;
    do {
        printf("\n==========Laundry Shop Management Program==========\n");
        printf("1. Add New Order\n");
        printf("2. Display Orders\n");
        printf("3. Update Order Status\n");
        printf("0. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        printf("\n");

        switch(choice) {
            case 1: addOrder(&shop); break;
            case 2: displayOrders(&shop); break;
            case 3: updateOrderStatus(&shop); break;
            case 0: printf("Goodbye!\n"); break;
            default: printf("Invalid choice!\n");
        }
    } while(choice != 0);
    return 0;
}
```

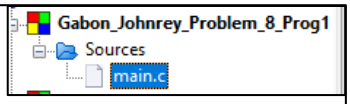
```
==========Laundry Shop Management Program==========
1. Add New Order
2. Display Orders
3. Update Order Status
0. Exit
Enter choice: 1

Enter customer name: Elliot Anderson
Select service (1-Wash, 2-Dry Clean, 3-Iron): 1
Enter weight in kg: 3.5
Enter date (DD/MM/YYYY): 11/12/2024
Order added! Total price: P437.50

==========Laundry Shop Management Program==========
1. Add New Order
2. Display Orders
3. Update Order Status
0. Exit
Enter choice: 1

Enter customer name: Mary Johnson
Select service (1-Wash, 2-Dry Clean, 3-Iron): 2
Enter weight in kg: 2
Enter date (DD/MM/YYYY): 10/12/2024
Order added! Total price: P480.00
```

```
==========Laundry Shop Management Program==========
1. Add New Order
2. Display Orders
3. Update Order Status
0. Exit
Enter choice: 2


Current Orders:
ID      Customer                Service         Status          Price
1001    Elliot Anderson         Wash            Pending         P437.50
1002    Mary Johnson            Dry Clean       Pending         P480.00

Current Rates (per kg):
Wash: P125.00
Dry Clean: P240.00
Iron: P60.00
```

```
==========Laundry Shop Management Program==========
1. Add New Order
2. Display Orders
3. Update Order Status
0. Exit
Enter choice: 3

Enter order ID: 1001
Enter new status (0-Pending, 1-Processing, 2-Ready, 3-Delivered): 1
Status updated successfully!
```

```
==========Laundry Shop Management Program==========
1. Add New Order
2. Display Orders
3. Update Order Status
0. Exit
Enter choice: 0

Goodbye!

Process returned 0 (0x0)   execution time : 76.669 s
Press any key to continue.
```

PROBLEM 8 (PROGRAM 1)




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

#define MAX_PARTS 50

typedef enum {
    CPU,
    GPU,
    RAM,
    STORAGE,
    MOTHERBOARD
} Category;
struct Part {
    char name[50];
    Category category;
    float price;
    int quantity;
};
struct Shop {
    struct Part inventory[MAX_PARTS];
    int partCount;
};
void addPart(struct Shop *shop) {
    if(shop->partCount >= MAX_PARTS) {
        printf("Inventory full!\n");
        return;
    }

    struct Part newPart;
    printf("Enter part name: ");
    scanf(" %[^\n]s", newPart.name);

    printf("Select category (0-CPU, 1-GPU, 2-RAM, 3-Storage, 4-Motherboard): ");
    scanf("%d", (int*)&newPart.category);

    printf("Enter price: $");
    scanf("%f", &newPart.price);

    printf("Enter quantity: ");
    scanf("%d", &newPart.quantity);
```

```c
    shop->inventory[shop->partCount] = newPart;
    shop->partCount++;
    printf("Part added successfully!\n");
}
void displayInventory(struct Shop *shop) {
    char *categories[] = {"CPU", "GPU", "RAM", "Storage", "Motherboard"};

    printf("\nCurrent Inventory:\n");
    printf("%Name\t\t\tCategory\tPrice\t\tQuantity\n");
    printf("-------------------------------------------------------\n");

    for(int i = 0; i < shop->partCount; i++) {
        printf("%-20s\t%-10s\t$%-10.2f\t%d\n",
            shop->inventory[i].name,
            categories[shop->inventory[i].category],
            shop->inventory[i].price,
            shop->inventory[i].quantity);
    }
}
void updateStock(struct Shop *shop) {
    char searchName[50];
    printf("Enter part name: ");
    scanf(" %[^\n]s", searchName);

    for(int i = 0; i < shop->partCount; i++) {
        if(strcmp(shop->inventory[i].name, searchName) == 0) {
            printf("Enter new quantity: ");
            scanf("%d", &shop->inventory[i].quantity);
            printf("Stock updated successfully!\n");
            return;
        }
    }
    printf("Part not found!\n");
}
int main() {
    struct Shop shop = {.partCount = 0};
    int choice;

    do {
```

```c
        printf("\n=====Computer Parts Shop Program=====\n");
        printf("1. Add New Part\n");
        printf("2. Display Inventory\n");
        printf("3. Update Stock\n");
        printf("0. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: addPart(&shop); break;
            case 2: displayInventory(&shop); break;
            case 3: updateStock(&shop); break;
            case 0: printf("Goodbye!\n"); break;
            default: printf("Invalid choice!\n");
        }
    } while(choice != 0);
    return 0;
}
```

```
=====Computer Parts Shop Program=====
1. Add New Part
2. Display Inventory
3. Update Stock
0. Exit
Enter choice: 1

Enter part name: AMD Ryzen 7 5800X
Select category (0-CPU, 1-GPU, 2-RAM, 3-Storage, 4-Motherboard): 0
Enter price: $299.99
Enter quantity: 15
Part added successfully!
```

```
=====Computer Parts Shop Program=====
1. Add New Part
2. Display Inventory
3. Update Stock
0. Exit
Enter choice: 1

Enter part name: NVIDIA RTX 3080
Select category (0-CPU, 1-GPU, 2-RAM, 3-Storage, 4-Motherboard): 1
Enter price: $699.99
Enter quantity: 8
Part added successfully!
```

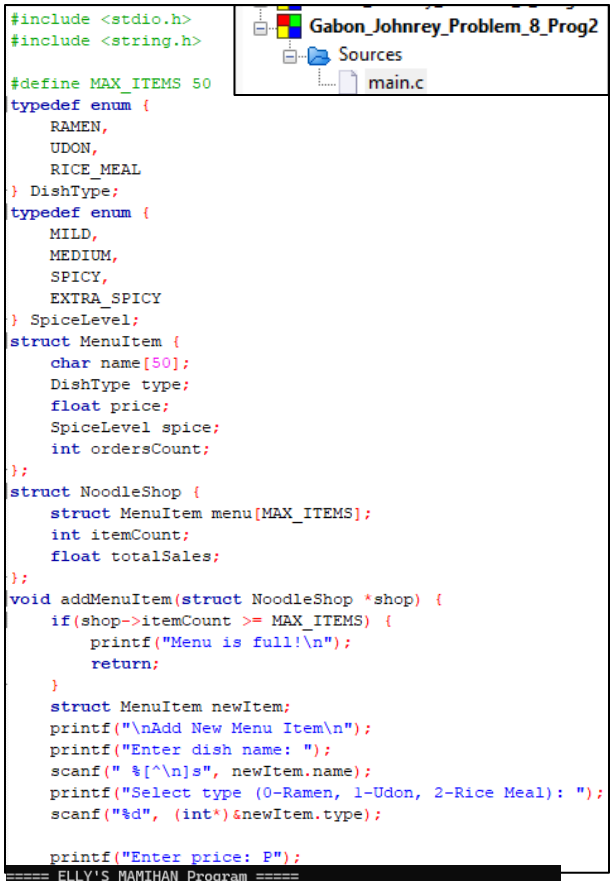
```
=====Computer Parts Shop Program=====
1. Add New Part
2. Display Inventory
3. Update Stock
0. Exit
Enter choice: 2


Current Inventory:
Name                    Category        Price           Quantity
-------------------------------------------------------
AMD Ryzen 7 5800X       CPU             $299.99         15
NVIDIA RTX 3080         GPU             $699.99         8
```

```
=====Computer Parts Shop Program=====
1. Add New Part
2. Display Inventory
3. Update Stock
0. Exit
Enter choice: 3

Enter part name: NVIDIA RTX 3080
Enter new quantity: 5
Stock updated successfully!
```

```
=====Computer Parts Shop Program=====
1. Add New Part
2. Display Inventory
3. Update Stock
0. Exit
Enter choice: 2


Current Inventory:
Name                    Category        Price           Quantity
-------------------------------------------------------
AMD Ryzen 7 5800X       CPU             $299.99         15
NVIDIA RTX 3080         GPU             $699.99         5
```

```
=====Computer Parts Shop Program=====
1. Add New Part
2. Display Inventory
3. Update Stock
0. Exit
Enter choice: 0

Goodbye!

Process returned 0 (0x0)   execution time : 97.276 s
Press any key to continue.
```

PROBLEM 8 (PROGRAM 2)




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

#define MAX_ITEMS 50
typedef enum {
    RAMEN,
    UDON,
    RICE_MEAL
} DishType;
typedef enum {
    MILD,
    MEDIUM,
    SPICY,
    EXTRA_SPICY
} SpiceLevel;
struct MenuItem {
    char name[50];
    DishType type;
    float price;
    SpiceLevel spice;
    int ordersCount;
};
struct NoodleShop {
    struct MenuItem menu[MAX_ITEMS];
    int itemCount;
    float totalSales;
};
void addMenuItem(struct NoodleShop *shop) {
    if(shop->itemCount >= MAX_ITEMS) {
        printf("Menu is full!\n");
        return;
    }
    struct MenuItem newItem;
    printf("\nAdd New Menu Item\n");
    printf("Enter dish name: ");
    scanf(" %[^\n]s", newItem.name);
    printf("Select type (0-Ramen, 1-Udon, 2-Rice Meal): ");
    scanf("%d", (int*)&newItem.type);

    printf("Enter price: P");
    scanf("%f", &newItem.price);

    printf("Select spice level (0-Mild, 1-Medium, 2-Spicy, 3-Extra Spicy): ");
    scanf("%d", (int*)&newItem.spice);

    newItem.ordersCount = 0;

    shop->menu[shop->itemCount] = newItem;
    shop->itemCount++;
    printf("Menu item added successfully!\n");
}

void displayMenu(struct NoodleShop *shop) {
    char *types[] = {"Ramen", "Udon", "Rice Meal"};
    char *spiceLevels[] = {"Mild", "Medium", "Spicy", "Extra Spicy"};

    printf("\n===== ELLY'S MAMIHAN MENU =====\n");
    printf("Name\t\t\tType\t\tSpice Level\tPrice\n");
    printf("-----------------------------------------------------\n");
    for(int i = 0; i < shop->itemCount; i++) {
        printf("%-20s\t%-10s\t%-10s\tP%.2f\n",
            shop->menu[i].name,
            types[shop->menu[i].type],
            spiceLevels[shop->menu[i].spice],
            shop->menu[i].price);
    }
}
void placeOrder(struct NoodleShop *shop) {
    char dishName[50];
    printf("Enter dish name: ");
    scanf(" %[^\n]s", dishName);

    for(int i = 0; i < shop->itemCount; i++) {
        if(strcmp(shop->menu[i].name, dishName) == 0) {
            shop->menu[i].ordersCount++;
            shop->totalSales += shop->menu[i].price;
            printf("Order placed successfully!\n");
            printf("Total sales: P%.2f\n", shop->totalSales);
            return;
        }
    }
    printf("Dish not found!\n");
}
int main() {
    struct NoodleShop shop = {.itemCount = 0, .totalSales = 0};
    int choice;
    do {
        printf("\nELLY'S MAMIHAN Management System\n");
        printf("1. Add Menu Item\n");
        printf("2. Display Menu\n");
        printf("3. Place Order\n");
        printf("0. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1: addMenuItem(&shop); break;
            case 2: displayMenu(&shop); break;
            case 3: placeOrder(&shop); break;
            case 0: printf("Goodbye!\n"); break;
            default: printf("Invalid choice!\n");
        }
    } while(choice != 0);
    return 0;
}
```

```
===== ELLY'S MAMIHAN Program =====
1. Add Menu Item
2. Display Menu
3. Place Order
0. Exit
Enter choice: 1


Add New Menu Item
Enter dish name: Mami with Ramen
Select type (0-Ramen, 1-Udon, 2-Rice Meal): 0
Enter price: P150.00
Select spice level (0-Mild, 1-Medium, 2-Spicy, 3-Extra Spicy): 1
Menu item added successfully!
```

```
===== ELLY'S MAMIHAN Program =====
1. Add Menu Item
2. Display Menu
3. Place Order
0. Exit
Enter choice: 1


Add New Menu Item
Enter dish name: Overload Mami
Select type (0-Ramen, 1-Udon, 2-Rice Meal): 2
Enter price: P280.00
Select spice level (0-Mild, 1-Medium, 2-Spicy, 3-Extra Spicy): 2
Menu item added successfully!
```

```
===== ELLY'S MAMIHAN Program =====
1. Add Menu Item
2. Display Menu
3. Place Order
0. Exit
Enter choice: 2


===== ELLY'S MAMIHAN MENU =====
Name                    Type            Spice Level     Price
-----------------------------------------------------
Mami with Ramen         Ramen           Medium          P150.00
Overload Mami           Rice Meal       Spicy           P280.00
```

```
===== ELLY'S MAMIHAN Program =====
1. Add Menu Item
2. Display Menu
3. Place Order
0. Exit
Enter choice: 3

Enter dish name: Overload Mami
Order placed successfully!
Total sales: P280.00
```

```
===== ELLY'S MAMIHAN Program =====
1. Add Menu Item
2. Display Menu
3. Place Order
0. Exit
Enter choice: 0

Goodbye!

Process returned 0 (0x0)   execution
Press any key to continue.
```