SEIF MWITA MGENI
BCS/18/23/024/TZ

a.) The best data structure in C Programming is queue with a linked list.

1. For insertion:
    - Traverse the list to find the correct position based on priority.
    - Insert the new request into the list.
2. For servicing:
    - Remove the request from the front of the list (highest priority).

b.)

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

struct Node *top = NULL;

void Push(int value) {
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   if (newNode == NULL) {
      printf("Stack Overflow!\n");
      return;
   }
   newNode->data = value;
   newNode->next = top;
   top = newNode;
   printf("Pushed %d onto the stack.\n", value);
}
```

c.)

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#define size 100

struct stack {
   int top;
   int s[size];
}st;

struct charStack {
   int top;
   char s[size];
}opSt;

void push(int item) {
   if (st.top >= size-1) {
      printf("stack overflow\n");
      return;
   }
```

```
    st.s[++st.top] = item;
}

int pop() {
    if (st.top == -1) {
        printf("stack underflow\n");
        return -1;
    }
    int item = st.s[st.top];
    st.top--;
    return item;
}

void pushChar(char item) {
    if (opSt.top >= size-1) {
        printf("stack overflow\n");
        return;
    }
    opSt.s[++opSt.top] = item;
}

char popChar() {
    if (opSt.top == -1) {
        return '\0';
    }
    char item = opSt.s[opSt.top];
    opSt.top--;
    return item;
}

int precedence(char op) {
    switch(op) {
        case '^': return 3;
        case '*':
        case '/': return 2;
        case '+':
        case '-': return 1;
        default: return 0;
    }
}

int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

int evaluate(int a, int b, char op) {
    switch(op) {
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': return a / b;
        case '^': return (int)pow(a, b);
```

```
        default: return 0;
    }
}

int main() {
    int a = 2, b = 6, c = 3, d = 2, e = -2;
    char infix[] = "((a+b)^((c/d)*e))";
    char postfix[100] = "";
    int postfix_idx = 0;

    st.top = -1;
    opSt.top = -1;

    printf("Converting infix expression: %s\n", infix);

    for(int i = 0; i < strlen(infix); i++) {
        char current = infix[i];

        if (current >= 'a' && current <= 'z') {
            postfix[postfix_idx++] = current;
        }
        else if (current == '(') {
            pushChar(current);
        }
        else if (current == ')') {
            while(opSt.top != -1 && opSt.s[opSt.top] != '(') {
                postfix[postfix_idx++] = popChar();
            }
            if(opSt.top != -1) {
                popChar();
            }
        }
        else if (isOperator(current)) {
            while(opSt.top != -1 && opSt.s[opSt.top] != '(' &&
                  precedence(opSt.s[opSt.top]) >= precedence(current)) {
                postfix[postfix_idx++] = popChar();
            }
            pushChar(current);
        }
    }

    while(opSt.top != -1) {
        postfix[postfix_idx++] = popChar();
    }
    postfix[postfix_idx] = '\0';

    printf("Postfix expression: %s\n", postfix);

    for(int i = 0; i < strlen(postfix); i++) {
        char current = postfix[i];

        if (current >= 'a' && current <= 'z') {
```

```
            switch(current) {
                case 'a': push(a); break;
                case 'b': push(b); break;
                case 'c': push(c); break;
                case 'd': push(d); break;
                case 'e': push(e); break;
            }
        }
        else if (isOperator(current)) {
            int val2 = pop();
            int val1 = pop();
            int result = evaluate(val1, val2, current);
            push(result);
        }
    }

    int result = pop();
    printf("Result after evaluation: %d\n", result);

    return 0;
}
```

d.)

```
#include <stdio.h>

float balance = 2000;

void deposit(float amount) {
    balance += amount;
    printf("%.2f si deposited to your account ", amount);
}

void withdraw(float amount) {
    if (amount > balance) {
        printf("Insufficient Balance\n");
    } else {
        balance -= amount;
        printf(“%.2f is withdrawn to your account", amount);
    }
}

int main() {
    int choice;
    float amount;
    int choice=1;

    while(choice=1)
        printf(“Enter Your choice”);
        printf("1. Deposit”)
        Printf(“2.Withdraw”)
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter amount to deposit: ");
                scanf("%f", &amount);
                deposit(amount);
                break;
            case 2:
                printf("Enter amount to withdraw: ");
                scanf("%f", &amount);
                withdraw(amount);
                break;
            default:
                printf("Invalid choice!\n");
            printf(“Do you wish to continue (0/1);
            scanf(“%d,choice);
        }
    }

    return 0;
}
```

e.)
Insertion at the Biggining:

```
void front_insertion(){
 struct Node *newnode;
 newnode =(struct Node*)malloc(sizeof(struct Node));
 printf("Enter the Element to be inserted");
 scanf("%d",&newnode->data);
 newnode->next = NULL;
 if(head==NULL){
    head==newnode;
 }else{
   newnode->next=head;
   head = newnode;
 }
 return;
}
```

Insertion at the End:

```
void back_insertion()
{
            struct node *newnode, *temp;
            if(head ==NULL)
                        front_insertion();
            else
            {
```

```
                newnode = (struct node*)malloc(sizeof(struct node));
                printf("enter the element to be inserted in back \n");
                scanf("%d",&newnode->data);
                newnode->next = NULL;
                temp = head;

                while(temp->next!=NULL)
                {
                        temp = temp->next;
                }
                temp->next = newnode;
        }
                return;
}
```