a. CPU Request Handling with Priority

In a system where multiple devices (keyboard, mouse, scanner, printer) send requests to the CPU, we can use a **priority queue** to manage these requests. Each device has a priority level, with lower numbers indicating higher priority:

- **Keyboard**: 1 (highest priority)
- **Mouse**: 2
- **Scanner**: 3
- **Printer**: 4 (lowest priority)

Data Structure

A suitable data structure for this scenario is a **min-heap** or a **priority queue** implemented using a linked list. Each node in the linked list can represent a request from a device, containing the device type and its priority.

Handling Requests

1. **Insert Request**: When a new request arrives, compare its priority with existing requests. Insert it in the correct position in the linked list based on its priority.
2. **Process Request**: The CPU processes the request at the head of the list (the highest priority).
3. **Remove Processed Request**: After processing, remove the request from the list.

b.

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

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

struct Stack {
    struct Node* top;
};

// Function to push an element onto the stack
void push(struct Stack* stack, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = stack->top;
    stack->top = newNode;
}

// Example usage
int main() {
    struct Stack stack;
    stack.top = NULL; // Initialize stack
    push(&stack, 10);
    push(&stack, 20);
    return 0;
}
```

c.

Diagram

```
Stack:
+-------+
| Top  |-> 20
+-------+
|      |-> 10
+-------+
```

c. Infix to Postfix Conversion

To convert the infix expression `((a+b)^((c/d)*e))` to postfix, we can use a stack. Here's how it works:

1. **Operands** are added directly to the output.
2. **Operators** are pushed onto the stack, but first pop from the stack to the output if the operator at the top of the stack has greater or equal precedence.
3. **Parentheses** are handled by pushing `(` onto the stack and popping until `)` is encountered.

Using the values `a=2`, `b=6`, `c=3`, `d=2`, and `e=−2`, the postfix expression becomes `ab+cde/*^`.

d.

```
#include <stdio.h>

struct BankAccount {
    float balance;
};

void deposit(struct BankAccount* account, float amount) {
    account->balance += amount;
    printf("Deposited: %.2f, New Balance: %.2f\n", amount, account->balance);
}

void withdraw(struct BankAccount* account, float amount) {
    if (amount > account->balance) {
        printf("Insufficient funds!\n");
    } else {
        account->balance -= amount;
        printf("Withdrawn: %.2f, New Balance: %.2f\n", amount, account-
>balance);
    }
}

int main() {
    struct BankAccount account = {1000.0}; // Initial balance
    deposit(&account, 500.0);
    withdraw(&account, 200.0);
    withdraw(&account, 1500.0); // Attempt to withdraw more than balance
    return 0;
}
```

e.

INSERTION AT THE BEGINNING:

```
void insertAtBeginning(struct Node** head, int newData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
    newNode->next = *head;
    *head = newNode;
}
```

INSERTION AT THE END:

```
void insertAtEnd(struct Node** head, int newData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head;
    newNode->data = newData;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}
```