|  |  |
| --- | --- |
|  | Elimentary Datastructure |
|  |  |
|  | Pragal naatha V  Assignment  4/10/24 |

A) Part Delivery and Arm

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 6

// Stack and Queue Implementation

char\* parts[] = {"Drill", "Wheel", "Camera", "Arm", "Battery", "Sonar"};

char\* queue[SIZE];

char\* stack[SIZE];

int front = 0, rear = 0, top = -1;

void enqueue(char\* part) {

queue[rear++] = part;

}

char\* dequeue() {

return queue[front++];

}

void push(char\* part) {

stack[++top] = part;

}

char\* pop() {

return stack[top--];

}

int main() {

printf("Part Delivery and Repair Order:\n");

// Enqueue parts

for (int i = 0; i < SIZE; i++) enqueue(parts[i]);

// Dequeue and push to stack

while (front < rear) push(dequeue());

// Pop from stack to show repair order

while (top >= 0) printf("%s\n", pop());

return 0;

}

B) Assembly Storage Unit

#include <stdio.h>

#include <string.h>

#define STORAGE 5

int main() {

char\* storage[STORAGE];

char\* rovers[] = {"Rov1", "Rov2", "Rov3", "Rov4", "Rov5", "Rov6", "Rov7"};

// Insert first 5 rovers

for (int i = 0; i < STORAGE; i++) storage[i] = rovers[i];

// Handle overflow

for (int i = STORAGE; i < 7; i++) {

// Deploy oldest

for (int j = 0; j < STORAGE - 1; j++) storage[j] = storage[j + 1];

storage[STORAGE - 1] = rovers[i];

}

printf("Final Rovers in Storage:\n");

for (int i = 0; i < STORAGE; i++) printf("%s\n", storage[i]);

return 0;

}

C) Faulty Rover Tracker

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Singly Linked List Node

typedef struct SNode {

char data[10];

struct SNode\* next;

} SNode;

// Doubly Linked List Node

typedef struct DNode {

char data[10];

struct DNode\* next;

struct DNode\* prev;

} DNode;

int main() {

// Create singly linked list

SNode\* faulty = malloc(sizeof(SNode));

strcpy(faulty->data, "Rov3");

faulty->next = malloc(sizeof(SNode));

strcpy(faulty->next->data, "Rov6");

faulty->next->next = NULL;

// Remove Rov3 and add to doubly linked list

DNode\* repaired = malloc(sizeof(DNode));

strcpy(repaired->data, "Rov3");

repaired->next = NULL;

repaired->prev = NULL;

printf("Repaired Rovers (Forward): %s\n", repaired->data);

printf("Repaired Rovers (Backward): %s\n", repaired->data);

// Free memory

free(faulty->next);

free(faulty);

free(repaired);

return 0;

}

D) Priority Upgrades

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct CNode {

char data[10];

struct CNode\* next;

} CNode;

int main() {

CNode\* rov1 = malloc(sizeof(CNode));

CNode\* rov4 = malloc(sizeof(CNode));

strcpy(rov1->data, "Rov1");

strcpy(rov4->data, "Rov4");

// Circular link

rov1->next = rov4;

rov4->next = rov1;

// Traverse twice

CNode\* current = rov1;

for (int i = 0; i < 4; i++) {

printf("%s\n", current->data);

current = current->next;

}

// Free memory

free(rov1);

free(rov4);

return 0;

}