Data Structures

Date	@July 7, 2023
# R	8
# R (2)	
# R (3)	
# R(1)	
R1	
R2	
R3	

▼ Main points

- Queue
- Stack
- DMA ⇒ Dynamic memory allocation
- Linked List

▼ Notes

- Data structures is not used in embedded C
- ▼ What are the things that are forbidden in C?
    - go to
    - Rec
    - Inline
    - gets
    - DMA
- There is part of memory named stack because it works the same way the data structure stack works

▼ Queue

▼ How queue works?

- FIFO
- First in First out

▼ Queue

▼ code

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

#define size 5

/*
arr[size] => represents the queue
head => points to the first element in the queue
tail => points to the first place empty in the queue
*/
typedef struct
{
    int arr[size];
    int head;
    int tail;
}queue;

void initQueue(queue *);
void in(queue*, int);
void out(queue*, int*);
void display(queue *);

int main()
{
    int data;
    queue q;
    initQueue(&q);
    in(&q, 10);
    display(&q);
    in(&q, 20);
    display(&q);
    in(&q, 30);
    display(&q);
    in(&q, 40);
    display(&q);
    in(&q, 50);
    display(&q);
    in(&q, 60);

    out(&q,&data);
    display(&q);
    printf("%d\n\n", data);
    out(&q,&data);
    display(&q);
    printf("%d\n\n", data);
    out(&q,&data);
    display(&q);
    printf("%d\n\n", data);
    out(&q,&data);
    display(&q);
    printf("%d\n\n", data);
    out(&q,&data);
    display(&q);
    printf("%d\n\n", data);

}

/*To access any member use the ptr don't forget*/
void initQueue(queue * ptr)
{
    ptr->head = ptr->tail = 0;
}

/*Function for entering new data in the queue*/
void in(queue* ptr, int data)
{
    if(ptr->tail == size)
    {
        printf("Queue is full\n");
    }
    else
    {
        ptr->arr[ptr->tail] = data;
        ptr->tail++;
    }

}
/*Function for getting data out from queue and return it in a variable, that
void out(queue* ptr, int* data)
{
    if(ptr->head == ptr->tail)
    {
        printf("The queue is empty\n");
    }
    else
    {
        *data = ptr->arr[ptr->head];
        int i;
        for(i = 0; i < ptr->tail-1; i++)
        {
            ptr->arr[i] = ptr->arr[i + 1];
        }
        ptr->tail--;
    }
}

/*Function to display the elements of the queue*/
void display(queue*ptr)
{
    if(ptr->tail == ptr->head)
    {
        printf("Queue is empty\n");
    }
    else
    {
        int i;
        for(i = 0; i < ptr->tail; i++)
        {
            printf("%d\t", ptr->arr[i]);
        }
        printf("\n");
    }
}
```

▼ output

```
10
10      20
10      20      30
10      20      30      40
10      20      30      40      50
Queue is full
20      30      40      50
10

30      40      50
20

40      50
30

50
40

Queue is empty
50
```

▼ notes

- Note that this is simple queue implementation and not circular
- Data structures are not commonly used in embedded C but it is a bonus if you know them, after diploma increase you skills here

▼ Stack

▼ How stack works?

- LIFO
- Last in first out

▼ Stack

▼ code

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

#define max 5

typedef struct
{
    int arr[max];
    int top;
}stack;

/*
top points to the element of the data, so it must start with -1 and increase
*/
void initStack(stack*);
void push(stack*, int);
void pop(stack*, int*);
void display(stack*);

int main()
{
    stack s;
    int data;
    initStack(&s);
    push(&s,10);
    display(&s);
    push(&s,20);
    display(&s);
    push(&s,30);
    display(&s);
    push(&s,40);
    display(&s);
    push(&s,50);
    display(&s);
    push(&s,60);
    display(&s);
    printf("\n\n");
    pop(&s,&data);
    display(&s);
    pop(&s,&data);
    display(&s);
    pop(&s,&data);
    display(&s);
    pop(&s,&data);
    display(&s);
    pop(&s,&data);
    display(&s);

    return 0;
}

void initStack(stack* p)
```

```
{
    p->top = -1;
}

void push(stack* p, int data)
{
    if(p->top == max - 1)
    {
        printf("Stack is full\n");
    }
    else
    {
        p->top++;
        p->arr[p->top] = data;
    }
}

void pop(stack *p, int *datap)
{
    if(p->top == -1)
    {
        printf("Stack is empty\n");
    }
    else
    {
        *datap = p->arr[p->top];
        p->top--;
    }
}
void display(stack* p)
{
        if(p->top == -1)
    {
        printf("Stack is empty\n");
    }
    else
    {
        int i;
        for(i = p->top; i >=0; i--)
        {
            printf("%d\t", p->arr[i]);
        }
        printf("\n");
    }
}
```

▼ output

```
10
20      10
30      20      10
40      30      20      10
50      40      30      20      10
Stack is full
50      40      30      20      10
```

```
40      30      20      10
30      20      10
20      10
10
Stack is empty
```

- ▼ notes
  - This is also a simple implementation
  - For more information refer to the papers

▼ DMA

- ▼ When to use DMA?
  - When you want to allocate space but in runtime
- ▼ What is the problem of the heap memory?
  - Space is allocated randomly not in an organized way like stack
- ▼ What is the function use to allocate space in heap?
  - malloc (x) where x is the number of bytes you want
- ▼ What is the return of malloc function?
  - the address of the space allocated

▼ Single Linked List

- ▼ How single linked list works (stack)?
  - Look at papers
- ▼ Single Linked List
  - ▼ code

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

typedef struct
{
    int data;
    struct node* next;
}node;
```

```
/*
This head structure is equivalent to stackLinkedList in c++
*/
typedef struct
{
    node*top;
}head;

void init(head*);
void pushNode(head*, int );
void popNode(head*, int*);
void display(head*);

int main()
{
    head h1;
    node n1;
    int data;
    init(&h1);
    pushNode(&h1, 10);
    pushNode(&h1, 20);
    popNode(&h1, &data);
    printf("%d", data);
    return 0;
}
/*
Top points to the last node
*/


void init(head* h)
{
    h->top = NULL;
}
/*
Each time I make node I want this node to be the top node
and the next after it is the node that was top thus I need temp pointer
*/
void pushNode(head* h, int data)
{
    //allocate space in heap memory equal to node stucture size
    //return the address of the first byte of the space allocated
    //cast this address to node type (like internal casting in float y = 5)
    //make temp pointer that points to the node (used to hold  address of th
    node *p = (node *) malloc(sizeof(node));  //h->node is wrong
    //Through this pointer enter the data to the node
    p->data = data;
    //Through this pointer enter the address of the next node to the next po
    p->next = h->top;
    //Make top point to this node through temp beacuse it carries its addres
    h->top = p;
}

/*
Temp pointer is used to hold the value of the address of the node to be dele
before making it point to the next one in order to be able to delete it
*/
```

```
void popNode(head *h, int*datap)
{
    *datap = h->top->data;
    node *temp = h->top;
    h->top = h->top->next;
    free(temp);
}
```

▼ output

```
20
```

▼ notes

- Free is used to free the space that the pointer points to