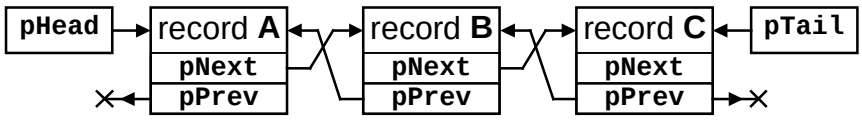
EE443 - Embedded Systems

Exercise - 6

Applications

1. Consider the following structure type definition for a double linked list.

```
typedef struct
{ unsigned char       WaveType;  // sample record data
  . . .               . . .      // other structure members
  void               *pNext;     // pointer to next record
  void               *pPrev;     // pointer to previous record
} WaveStruct;
WaveStruct   WaveArray[MaxNumberOfWave];  // reserved storage
WaveStruct  *pFree;  // pointer to list of unused records
WaveStruct  *pHead;  // pointer to first record of linked list
WaveStruct  *pTail;  // pointer to last record of linked list
```



1.a) Write a function that initializes the linked list pointers and builds a linked list for the unused records.

1.b) Complete the following function that deletes a record from the double linked list and puts the deleted record back to the list of unused records.

```
unsigned char DeleteWave(WaveStruct  *pWave)
// Deletes the linked list record pointed to by pWave. Returned value is:
// 0 => delete operation is successful, 1 => linked list is empty
```

1.c) Write a C function that plays a sequence of waveforms by calling the following function once for each element in the linked list.

```
void PlayWave(WaveStruct  *pWave);
```

1.d) Modify the function you wrote above, so that the waveform sequence is reversed starting with the last element in the linked list.

2.a) Write a C program that implements the finite state machine described by the truth table given below. **FSMin[1:0]** inputs are read from **bit-3** and **bit-2** of **PORTA**. **PORTA** is an 8-bit port and the status of the other port bits are unknown. **Out1** is a single bit output, and **Out2[15:0]** are 16-bit counter outputs.

FSMin[1:0]	State$_{(k)}$	State$_{(k+1)}$	Out1$_{(k+1)}$	Out2$_{(k+1)}$[15:0]
0 0	x x	0 0	0	0
0 1	0 0	0 1	1	Out2$_{(k)}$ + 1
0 1	0 1	0 1	0	
0 1	1 0	0 1	1	
0 1	1 1	0 1	1	
1 0	0 0	1 0	1	Out2$_{(k)}$ - 1
1 0	0 1	1 0	1	
1 0	1 0	1 0	0	
1 0	1 1	1 0	1	
1 1	x x	State$_{(k)}$	0	Out2$_{(k)}$

2.b) What can go wrong if the **PORTA** inputs are read more than once in your program? What can be done to solve this problem?

3. Implement a circular queue buffer where buffer elements have the **`ControlData`** structure type defined below.

```
typedef struct
{ signed short int     ADCout;  // input sample
  signed char          Error;   // calculated error
  signed short int     DACin;   // controller output
} ControlData;
```

3.a) Declare a storage array and the necessary global variables. The maximum buffer size is given by **`BUFFER_SIZE`**.

3.b) Write a function that initializes the buffer operation.

3.c) Clearly state the conditions that identify the buffer status

i. when the buffer is empty,

ii. when the buffer is full (no space is left to store data)

3.d) Write a program segment that performs an enqueue operation in an ISR. ISR should not write any new data when the buffer is full.

3.e) Write a program segment that performs a dequeue operation in the main program.

3.f) Modify the ISR program segment so that it will overwrite the oldest data when the buffer is full.

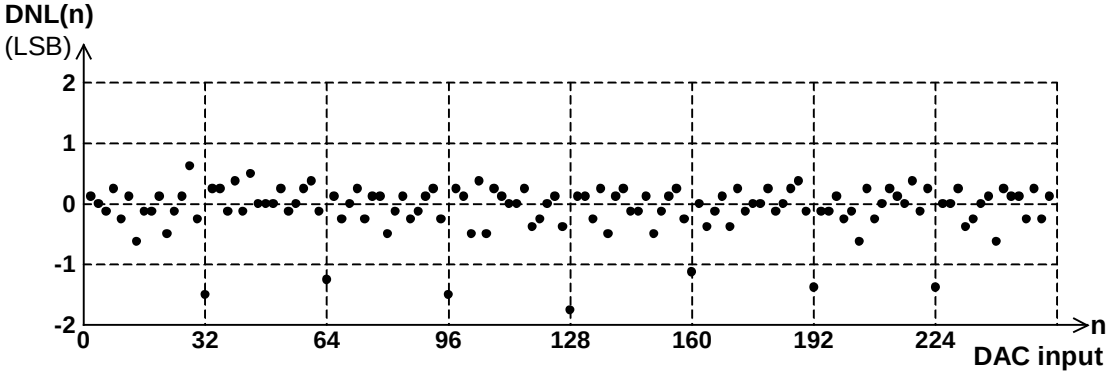
3.g) Check for race conditions that may occur in the main program. Make the required corrections for safe buffer operation.

4.a) Explain the reason why accuracy of feedback path is more important than accuracy of forward path in a closed-loop control system?

4.b) Why is it critical to have a monotonic response at the driver output and sensor input in a closed-loop control system?

4.c) How can you check if the input/output response of a DAC or an ADC is monotonic or not according to its specifications?

5. Write a program that makes the necessary corrections to obtain a monotonic (non-decreasing) response when using an 8-bit DAC with the differential nonlinearity given below.



DNL(n) = 0 LSB => $V_{out}(n) - V_{out}(n-1) = \Delta V_{LSB}$ (ideal response)

DNL(n) = -1 LSB => $V_{out}(n) - V_{out}(n-1) = 0V$ (no change at the output)

6. Starting with the following continuous time PID controller function definition,

$$F_{out}(t) = G_p\, E(t) + G_i \int_{t'=0}^{t} E(t)\, dt' + G_d \frac{dE(t)}{dt}$$

derive the discrete time PID controller function in the form of

Fout(k) = Fout(k-1) + . . .

7.a) Why is the saturation recovery time of a controller is critical in handling overflow conditions?

7.b) What can go wrong when an overflow occurs in the calculation of PID controller output, **Fout(k)**, implemented in the following form?

Acc(k) = Acc(k-1) + Err(k)

$$\textbf{Fout(k)} = K_p\, \textbf{Err(k)} + K_i\, \textbf{Acc(k)} + K_d \left[\textbf{Err(k) - Err(k-1)}\right]$$

7.c) Analyze the saturation recovery of the PID controller implementation given in the previous question (i.e. **Fout(k) = Fout(k-1) + . . .**). Compare the saturation recovery response of the two implementations.

8.a) Explain the difference between concurrency and parallelism in digital systems.

8.b) What is the main function of the scheduler in a RTOS?

8.c) Explain the constraints that should be considered when setting the frequency of scheduler operation.

8.d) Describe the common scheduling strategies that are used for distributing processor time.