Quiz3 will be similar in style to the online Homework
Quiz3 will cover material from Chapters 1,2,3,4,5,6,7,8,10.
Good things to study are
the Homework questions and solutions
the lecture notes aLec1 to aLec32
It will be closed book with about 25 short answer questions
I will give you similar technical documents as I did with Quiz1
The numerical questions will be simple enough not to require a calculator.

10.10. *Dynamically allocated data structures
1) fixed allocation (global variable)
   fixed size
   allocated at assembly (design) time
   simple and fast access

   e.g.,
   
   org  $3800
   buffer rmb 20
   org  $4000
   funct ldx #buffer  pointer into buffer
   ldaa #20
   loop  clr 1,x+     access buffer
   dbne A,loop
   rts

2) stack allocation (local variable)
   allocated at the beginning of a function
   released at the end
   simple and fast access

   e.g.,
   
   buffer set 0           binding
   funct leas –20,s     allocation
   leax buffer,s
   ldaa #20
3) dynamic memory allocation
   allocated and released at run time
   size will be determined at run time
   allows for reuse

   org  $3800
   pt   rmb  2
   org  $4000
   funct1 ldd  #20       size
       jsr  allocate  create, return pt
       stx  pt
       ldaa #20
   loop  clr  1,x+       access buffer
       dbne A,loop
       rts
   funct2 ldx  pt
       jsr  release
       rts

With a dynamic allocation the size will be determined at run time. To implement dynamic allocation we will manage a heap. The heap is a chunk of RAM that is
   1) dynamically allocated when it creates the data structure
   2) used by the program to store information
   3) dynamically released by program when no longer needed

The heap manager provides the system two operations:

   pt=allocate(size);
   // returns pointer to block of size bytes
release(pt);
// deallocates the block at pt

The implementation of this general memory manager is beyond the scope of this book. Instead, we will develop a very useful, but simple heap manager with these two operations:

\[
\text{pt=Heap}_\text{Allocate}(); \\
// \text{returns pointer to block of fixed size}
\]

\[
\text{Heap}_\text{Release}(\text{pt}); \\
// \text{deallocates the block at pt}
\]

10.10.1. Fixed block memory manager

![Figure 10.15. The initial state of the heap has all of the free blocks linked in a list.](image)

SIZE equ 4
NUM equ 5
NULL equ 0
FreePt rmb 2
Heap rmb SIZE*NUM

Program 10.23. Global structures for the fixed-block memory manager.

Heap_Init ldx #Heap
  stx FreePt FreePt=&Heap[0];
  ldab SIZE
imLoop pshx
  puly RegY = pt;
  aby pt+SIZE
Program 10.24. Functions to initialize the heap.

* returns RegX points to new block
* RegX=NULL if no more available
Heap_Allocate
  ldx FreePt pt=FreePt;
  cpx #NULL
  beq aDone if (pt!=NULL)
  ldy 0,x
  sty FreePt FreePt=*pt;
aDone rts

* RegX => block being released
Heap_Release
  ldy FreePt oldFreePt=FreePt;
  stx FreePt FreePt=pt;
  sty 0,x *pt=oldFreePt;
  rts

Program 10.25. Functions to allocate and release memory blocks.

10.10.2. Linked list FIFO

Next  equ 0    next
Data   equ 2    16-bit data for node
GetPt  rmb 2    pointer to oldest node
PutPt  rmb 2    pointer to newest node
NULL   equ 0

Program 10.26. Definition of the linked list structure.
Figure 10.16. A linked list FIFO after putting 1,2,3.

Fifo_Init
    ldx  #NULL
    stx  GetPt          GetPt=NULL;
    stx  PutPt          PutPt=NULL;
    jsr  Heap_Init
    rts

* Inputs:  RegD data to put
* Outputs: V=0 if successful
*          V=1 if unsuccessful

Fifo_Put jsr  Heap_Allocate
    cpx  #NULL
    beq  Pful          skip if full
    std  Data,x        store data
    ldy  #NULL
    sty  Next,x        next=NULL
    ldy  PutPt
    cpy  #NULL         previously MT?
    beq  PMT
    stx  Next,y        link to previous
    bra  PCon
PMT  stx  GetPt      Now one entry
PCon  stx  PutPt      points to newest
    clv                 success
    bra  PDon
PFul  sev             failure, full
PDon  rts

* Inputs:  none
* Outputs: RegD data removed
*          V=0 if successful
*          V=1 if empty

Fifo_Get ldx GetPt
    cpx  #NULL
    beq  GMT          empty if NULL
    ldd  Data,x      read
ldy Next,x pointer to next
sty GetPt
cpy #NULL
bne GCon
sty PutPt Now empty
GCon sty GetPt points to oldest
jsr Heap_Release
clv success
bra GetDone
GMT sev failure, empty
GDon rts

Program 10.27. Implementation of the linked list FIFO.

Figure 10.17. Flowcharts of a linked list FIFO Put and Get operations.
**Checkpoint 10.20:** Draw a picture like Figure 10.16 of a doubly-linked list. How might this more complicated structure be more efficient than the single linked list?

I suggest you do tutorial 10 before next week’s lab!!!
```assembly
jsr Fifo_Init
FreePt
PutPt
GetPt

ldd #5
call Fifo_Put
FreePt
PutPt
GetPt

ldd #6
call Fifo_Put
FreePt
PutPt
GetPt

jsr Fifo_Get
FreePt
PutPt
GetPt

ldd #7
call Fifo_Put
FreePt
PutPt
GetPt
```
```plaintext
jsr Fifo_Get
FreePt
PutPt
GetPt

ldd #8
jsr Fifo_Put
FreePt
PutPt
GetPt

ldd #9
jsr Fifo_Put
FreePt
PutPt
GetPt

jsr Fifo_Get
FreePt
PutPt
GetPt

jsr Fifo_Get
FreePt
PutPt
GetPt
```