[1] Pointer, dynamic memory assignment/free and structure

Structure

```
struct TokenizerT_
{
  member list
};

typedef struct TokenizerT_  TokenizerT;
```

Memory allocation and de-allocation: check the number

#include <stdlib.h> for malloc() and free()		
malloc()	Allocate requested size of bytes and returns a pointer first byte of allocated space	ptr = (cast-type*)malloc(byte-size) ptr = (int*)malloc(100*sizeof(int))
free()	De-allocate the previously allocated space	free(ptr)

[2] clarification for PA1

a.Define decimal:

Decimal integer token is a digit (**1**-9) followed by any number of digits.

b. Remain the quotation mark " " in input, remove it from output

c. Error message requirement:

It's not clear. We suggest to print out ***invalid error_token***

"The@sign is valid"
word "The"
error "[0x40] (The Hex of @ in ASCII is 40
word "sign"
word "valid"

How to separate a string into tokens: use FSM diagram

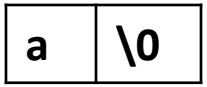
```
//consider type,  length(size_t),  content, order in the string
struct TokenizerT_ {

};
typedef struct TokenizerT_ TokenizerT;
```

How to separate a string into tokens: use FSM diagram

```
//Intention: to form tokens
//transform a token in string type to TokenizerT type:  ts is a token, like "0xF5"
//copy content information
TokenizerT* TKCreate( char * ts )
 {
   return NULL;
}

 //Free all dynamic memory inside TokenizerT.  Don't free(TokenizerT)
 void TKDestroy( TokenizerT * tk )
  {   }


  //Intention: to output tokens in string form (delimited by '\0'), in order.
  char *TKGetNextToken( TokenizerT * tk )
   {
    return NULL;
  }
```

[3] char and string

```
char c1;
c1 = 'a'      //c1 is a char

char* c2;  //you don't need to assign the length when declaring a char*
c2 = "a";   //c2 is a string
           //in memory, it takes 2 bytes:
           //But its length is 1 using strlen()
```



[4] Compile the code to a debug-able executable named tokenizer

-g **turn on debugging**

-Wall turn on most warnings

-O turn on optimizations

```
$ gcc -Wall -g -o tokenizer tokenizer.c
```

[5]tar the folder

```
bash-4.1$  cd  /ilab/users/gq19/Documents/TA_211ComputerArchitecture/Rec2
bash-4.1$  ls
PA1
bash-4.1$  tar cfz  pa1.tgz  PA1
```

[6] Use GDB to debug C program

http://www.thegeekstuff.com/2010/03/debug-c-program-using-gdb/

- (1) Compile the C program with debugging option
  gcc –Wall –g –o GDB_factorial GDB_factorial.c

- (2) Lanuch GDB
  gdb GDB_factorial

- (3) Set a break point (need to run the program to stop at this point)
  break linenumber
  break [file_name]:line_number
  break [file_name]:func_name
  break 12

[6] Use GDB to debug C program

http://www.thegeekstuff.com/2010/03/debug-c-program-using-gdb/

- (4) Run the program

  run

  run [args]

- (5) Continue, Stepping over and Stepping in

  c Debugger will continue executing until the next break point

  n Debugger will execute the next line as single instruction (step out)

  s Same as n, but does not treats function as a single instruction (step in)

- (6) Print variables

  print {variable}

  print i

  print j

  print num

- (7) Quit GDB

  quit