REPORT OF ASSIGMENT 2

1-)QUESTION DESCRIPTION

In the given question, the C program should be coded as a scheduler to decide processes terminated or non terminated. Every process are in a Struct form with its' feature variables.

At first, Program can generate initial process struct array that sized 5 between 10 has random assigned contents according to given form like;

```
typedef struct{
        int process_id; 1,2,3,4..        //the id number of processes in sequential form
        int ttl;            5 between 10  //TimetoLive of process
        int lifeCycle;   intial value 1  //LifeCycle of process
        int priority;     1 or 2 or 3      //Priority of process
}PROCESS;
```

At each iteration randomly, process should be created or not and if created; new randomized process placed into the waiting_queue. If not created process;Scheduler selects only one process from waiting queue for processing.

After that, it controls lifecycle>ttl condition to find non terminated process and move it to non terminated list. If scheduler doesn't find non terminated it takes only one process according to priority,lifecycle (if priority is same then select highest life cycle one) to process and move it to terminated list at each step.

The program would display on the screen the waiting_queue, the terminated list, and the non_terminated_list at each step.

At last, if there is no process in the waiting_queue, the program give the list of the terminated processes and the list of non_terminated processes, and the average lifeCycle of the terminated processes and then it must stop.

2-)SOLUTION DESCRIPTION

The program has a PROCESS struct and it creates initial arrays for terminated list, non terminated list and waiting queue(struct array has random sized member) by using dynamic memory allocation techniques. Then it assigns randomly or not values to variables of waiting queue array members according to given restricted values in question.

Randomly it decides add new process to waiting queue or not. After that, it calls the nonterm_assigner function to find non terminated process in waiting queue.Then it control it's return value if it is equal to '1' it reallocates non-terminated list and move process into it. Then it calls shifter function to fix waiting queue. Shifter function shifts the elements left from rigth it accepts deleted process' index to stop point. Then reallocates waiting queue. If the return value is '0' it means there is no non-terminated process. So it calls terminate_assigner function to pick the process has a highest priority if there has a same priority processes then it picks the process that has highest lifecycle value. Then it reallocates terminated list array and move the process in it. Then it calls shifter function to fix waiting queue again.

Each iteration the lifecycle values of processes in waiting queue incerements by calling incrementer fuction until there is no process in waiting queue. In addition the condition of lists is printed on screen at each iteration of scheduler.

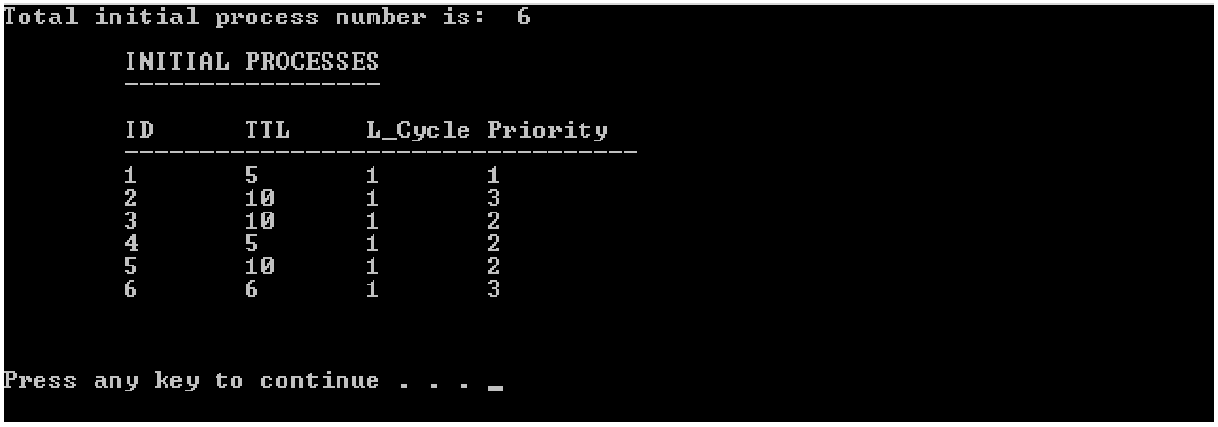
After iteration completed program calculate terminated processes average lifeCycle by calling function and prints it with the list of terminated processes and non terminated processes and sizes of lists.

HARDWARE SPECIFICATION OF COMPUTER THAT USED FOR THAT PROGRAM

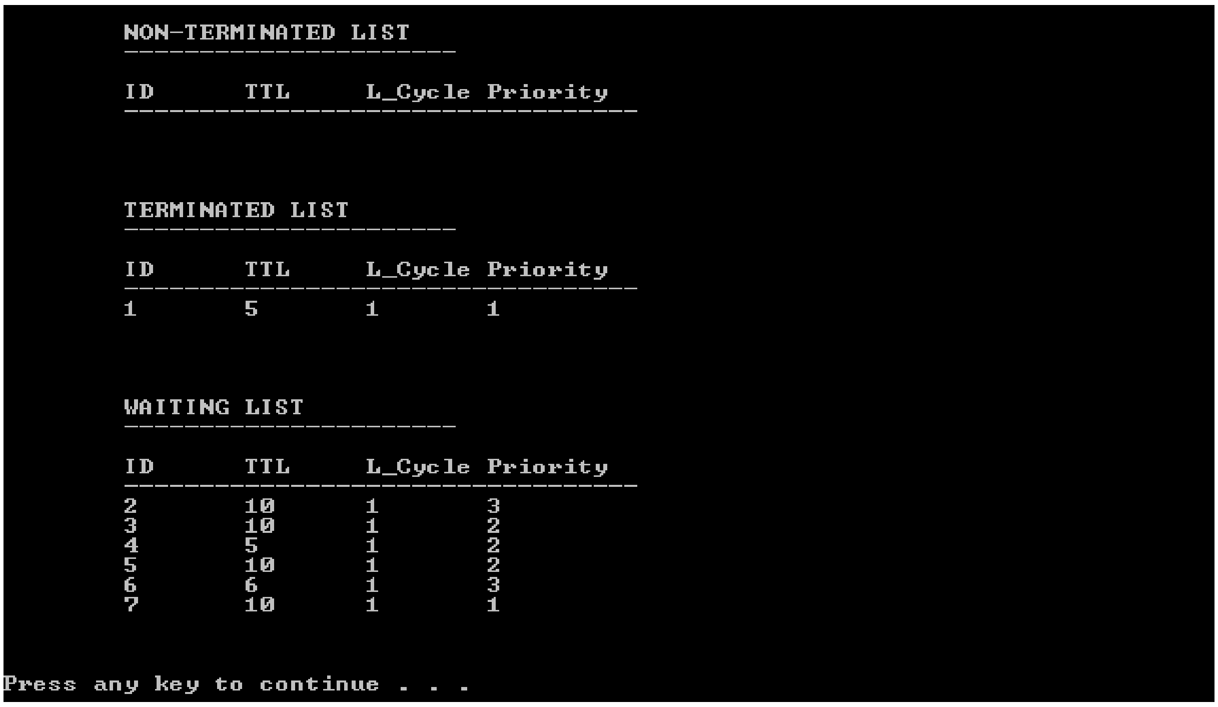
CPU	:	INTEL(R) CORE(TM) i5-2410M CPU @2.30 GHz
RAM	:	**8 GB**
Compiler Used	:	**GCC**
IDE	:	DEV-C++(Version 5.11)
Operating System	:	Windows Embedded 8.1 Industry Pro

3-)ANALYSIS

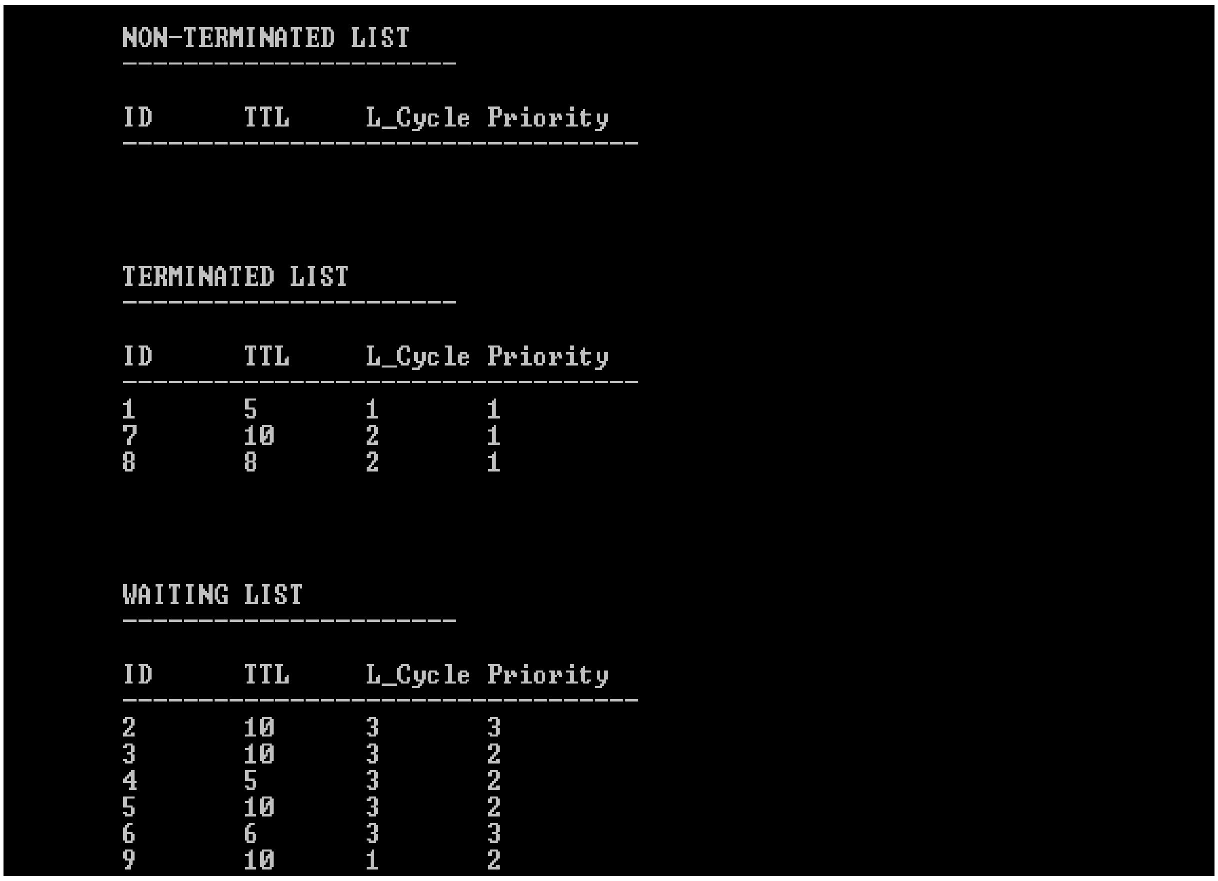
SCREENSHOTS OF PROGRAM FOR CORRECTNESS AND DETAILS

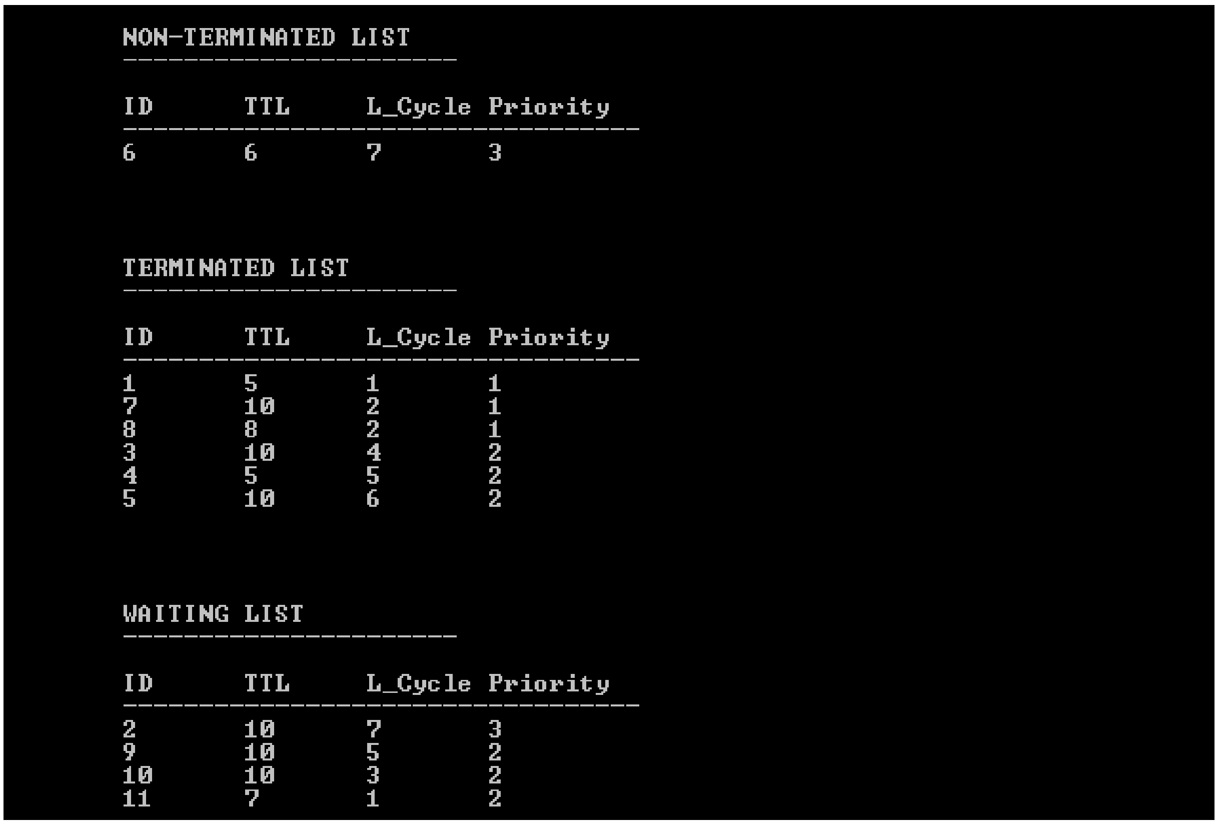


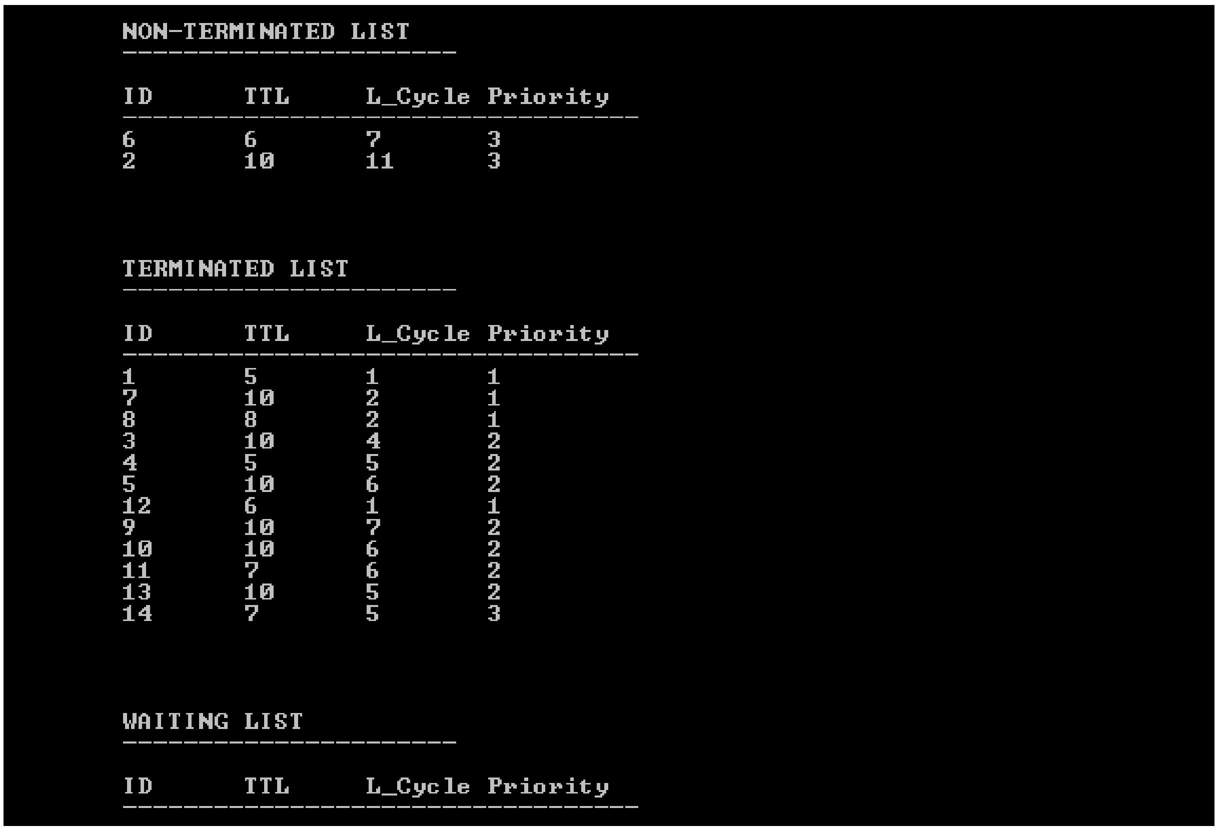
```
Total initial process number is:  6

        INITIAL PROCESSES
        -----------------

        ID      TTL     L_Cycle Priority
        ----------------------------------

        1       5       1       1
        2       10      1       3
        3       10      1       2
        4       5       1       2
        5       10      1       2
        6       6       1       3



Press any key to continue . . . _
```



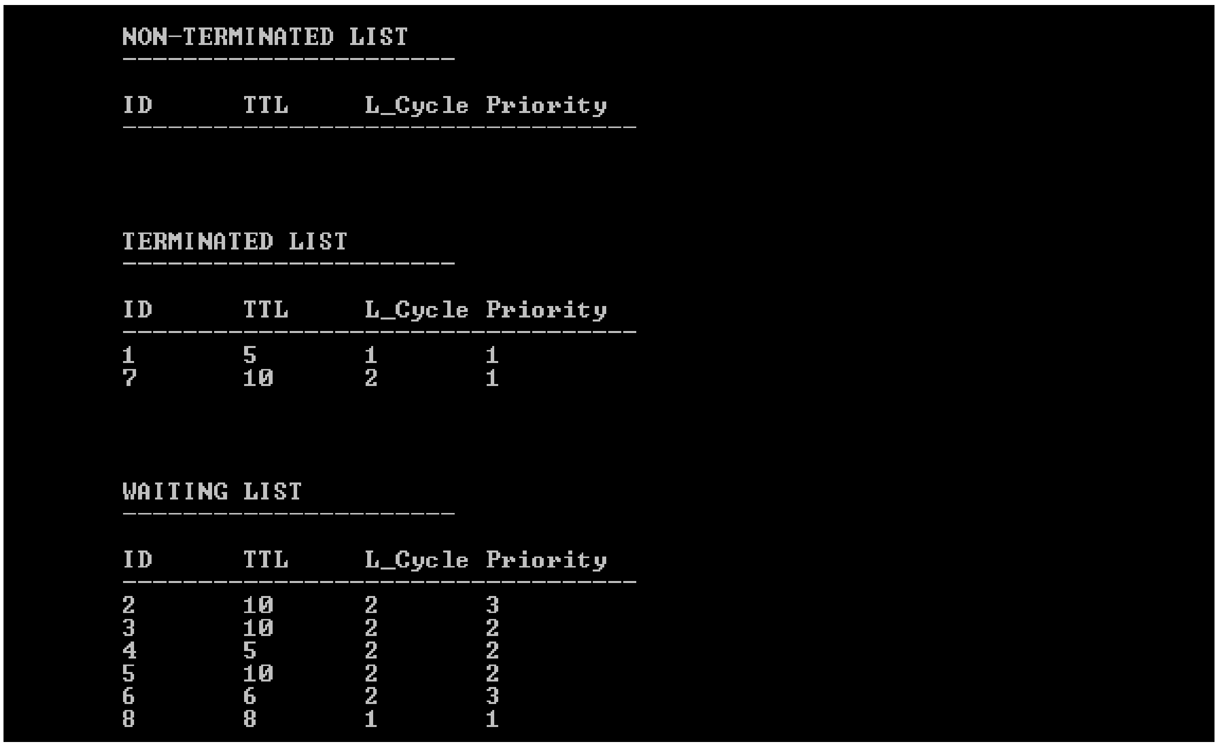
```
        NON-TERMINATED LIST
        ----------------------

        ID      TTL     L_Cycle Priority
        ----------------------------------



        TERMINATED LIST
        ----------------------

        ID      TTL     L_Cycle Priority
        ----------------------------------
        1       5       1       1



        WAITING LIST
        ----------------------

        ID      TTL     L_Cycle Priority
        ----------------------------------
        2       10      1       3
        3       10      1       2
        4       5       1       2
        5       10      1       2
        6       6       1       3
        7       10      1       1


Press any key to continue . . .
```


```
NON-TERMINATED LIST
----------------------

ID       TTL      L_Cycle Priority
-----------------------------------



TERMINATED LIST
----------------------

ID       TTL      L_Cycle Priority
-----------------------------------
1        5        1        1
7        10       2        1
8        8        2        1



WAITING LIST
----------------------

ID       TTL      L_Cycle Priority
-----------------------------------
2        10       3        3
3        10       3        2
4        5        3        2
5        10       3        2
6        6        3        3
9        10       1        2
```

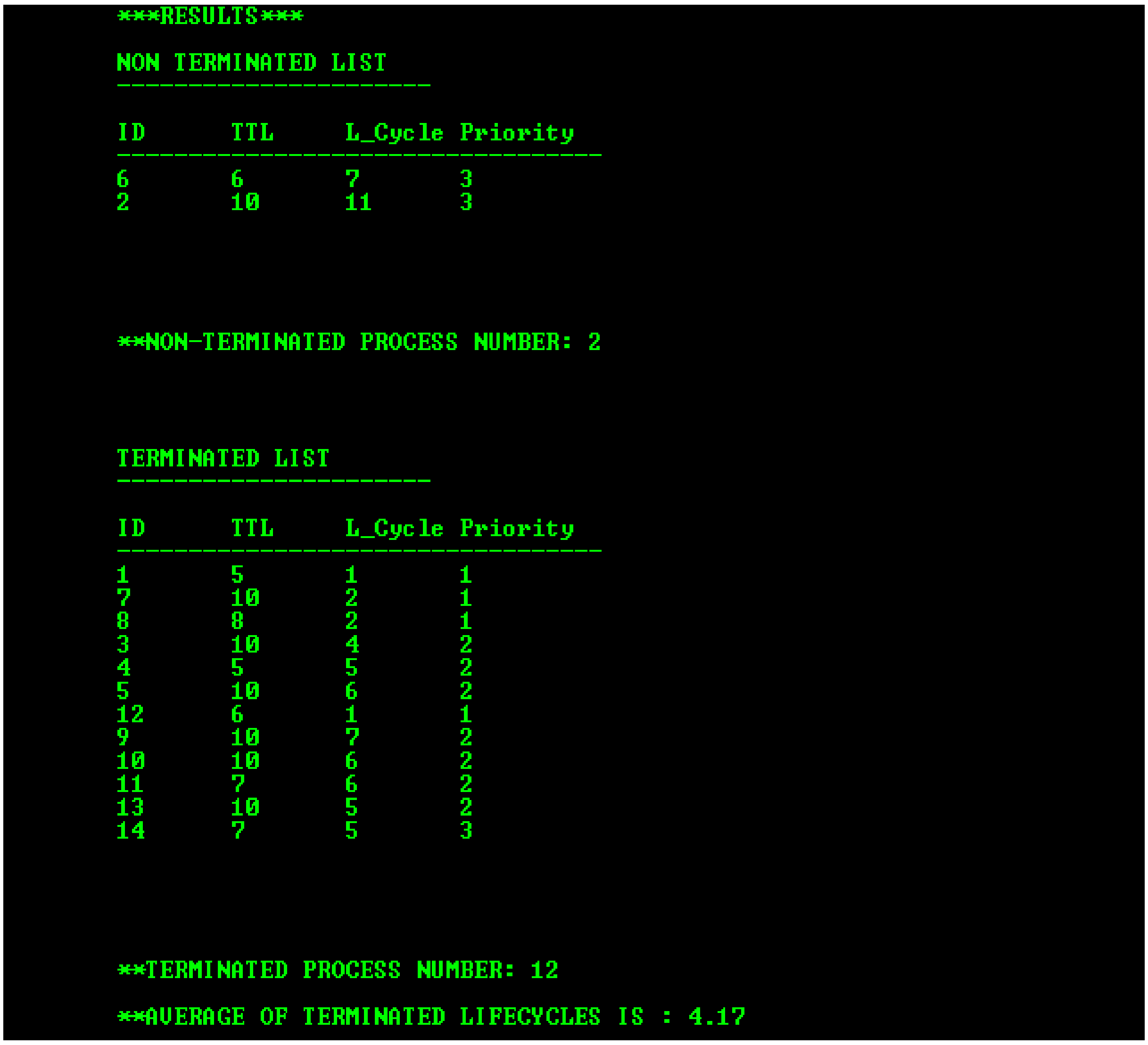

```
NON-TERMINATED LIST
----------------------

ID       TTL      L_Cycle Priority
-----------------------------------
6        6        7        3



TERMINATED LIST
----------------------

ID       TTL      L_Cycle Priority
-----------------------------------
1        5        1        1
7        10       2        1
8        8        2        1
3        10       4        2
4        5        5        2
5        10       6        2



WAITING LIST
----------------------

ID       TTL      L_Cycle Priority
-----------------------------------
2        10       7        3
9        10       5        2
10       10       3        2
11       7        1        2
```



```
NON-TERMINATED LIST
----------------------

ID      TTL     L_Cycle Priority
----------------------------------
6       6       7       3
2       10      11      3



TERMINATED LIST
----------------------

ID      TTL     L_Cycle Priority
----------------------------------
1       5       1       1
7       10      2       1
8       8       2       1
3       10      4       2
4       5       5       2
5       10      6       2
12      6       1       1
9       10      7       2
10      10      6       2
11      7       6       2
13      10      5       2
14      7       5       3



WAITING LIST
----------------------

ID      TTL     L_Cycle Priority
----------------------------------
```



```
NON-TERMINATED LIST
----------------------

ID      TTL     L_Cycle Priority
----------------------------------



TERMINATED LIST
----------------------

ID      TTL     L_Cycle Priority
----------------------------------
1       5       1       1
7       10      2       1



WAITING LIST
----------------------

ID      TTL     L_Cycle Priority
----------------------------------
2       10      2       3
3       10      2       2
4       5       2       2
5       10      2       2
6       6       2       3
8       8       1       1
```



```
***RESULTS***

NON TERMINATED LIST
----------------------

ID      TTL     L_Cycle Priority
----------------------------------
6       6       7       3
2       10      11      3




**NON-TERMINATED PROCESS NUMBER: 2




TERMINATED LIST
----------------------

ID      TTL     L_Cycle Priority
----------------------------------
1       5       1       1
7       10      2       1
8       8       2       1
3       10      4       2
4       5       5       2
5       10      6       2
12      6       1       1
9       10      7       2
10      10      6       2
11      7       6       2
13      10      5       2
14      7       5       3




**TERMINATED PROCESS NUMBER: 12

**AVERAGE OF TERMINATED LIFECYCLES IS : 4.17
```

4-)SOURCE CODE

```
/**
@file
BLM2541 spring2016 assignment 2.

A program behaves like scheduler as a virtual. It creates initial processes randomly 5 between 10 in waiting process queue. Then each iteratior
After that it decides only one process from waiting queue that will be non terminated or terminated according to process properties(Priority,Li
The processes of terminated list and non-terminated list printed on the screen at each iteration too.
As a conclusion, it prints the non-terminated processes, terminated processes and average lifecycle of terminated processes on the screen as ar
@author

Name             :      Muhammed Yasin SAĞLAM
Student No       :      15011804
Date             :      03/12/2016
E-Mail           :      myasinsaglam1907@gmail.com
Compiler Used    :      GCC
IDE              :      DEV-C++(Version 5.11)
Operating System :      Windows Embedded 8.1 Industry Pro
*/

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

//the structure that contains all properties of used processes in program.
 typedef struct{
    int process_id; //the id number of processes in sequential form
    int ttl;        //TimetoLive of process
    int lifeCycle;  //LifeCycle of process
    int priority;   //Priority of process
}PROCESS;

//Function procedures that used in this program
void terminate_assigner(PROCESS** wait,PROCESS** term,int *wait_size,int *term_size);
void incrementer(PROCESS *arr,int size);
void printer(PROCESS *arr,int n);
void strandomizer(PROCESS *arr,int first,int last);
void add_process(PROCESS** arr,int *size,int* id_index);
void re_allocator(PROCESS** arr,int *size);
void shifter(PROCESS** arr,int *size,int index);
int nonterm_assigner(PROCESS** wait,PROCESS** non_term,int *wait_size ,int *nterm_size);
float av_calculator(PROCESS* arr,int size);

/**
    In Main function, random initial process number (5 between 10) determined and waiting queue(struct array) created and randomized.
    Then the processes classified(terminated or non-terminated) by calling functions (at each iteration only one process).
    At last, average lifecycle of terminated processes are calculated and number of elements of terminated and non-terminated arrays printed or
*/

int main(){

srand(time(NULL));
//initial process number randomizing and printing on the screen.
int process_num=rand()%6+5;
printf("Total initial process number is:  %d\n\n",process_num);
//configurable waiting queue size,terminated list initial size,non-terminated list initial size defined and assigned.
int term_size=0;
int nterm_size=0;
int wait_size=process_num;

//Struct arrays created and allocated based on initial values.
PROCESS *waiting,*terminated,*non_terminated;
waiting=(PROCESS*)malloc(sizeof(PROCESS)*process_num);
terminated=(PROCESS*)malloc(sizeof(PROCESS));
non_terminated=(PROCESS*)malloc(sizeof(PROCESS));
    if(!waiting || !terminated || !non_terminated){
        printf("Initial Processes' Lists are not created.Quitting....");
        exit(0);
    }

//initial waiting queue elements randomizing according to given rules of project by calling function.
strandomizer(waiting,0,process_num);
//initial waiting queue elements printed on the screen by calling function.
printf("\tINITIAL PROCESSES\n\t-----------------\n\n");
printer(waiting,process_num);
system("PAUSE");
system("CLS");

    //iteration started until no elements in waiting queue.
    while(wait_size!=0){
        //if it is equal to 1 the new process will added to waiting queue.
        if(rand()%2==1){
            //Process created and added by calling function.
            add_process(&waiting,&wait_size,&process_num);
        }
        //move one element from waiting queue to nonterminated or terminated list by calling functions.
        //firstly looks for non terminated. if not it takes one of them to terminate according to priority and lifecylcle
        if(nonterm_assigner(&waiting,&non_terminated,&wait_size,&nterm_size)==0){
            terminate_assigner(&waiting,&terminated,&wait_size,&term_size);
        }

        //printing all lists for each iteration.
        printf("\n\tNON-TERMINATED LIST\n\t----------------------\n\n");
        printer(non_terminated,nterm_size);
        printf("\n\tTERMINATED LIST\n\t----------------------\n\n");
        printer(terminated,term_size);
        printf("\n\tWAITING LIST\n\t----------------------\n\n");
        printer(waiting,wait_size);
```

```c
        //incrementing lifeCycle values of remaining waiting queue elements.
        incrementer(waiting,wait_size);

        system("PAUSE");
        system("CLS");
    }

//printing the results part.
system("COLOR a");
printf("\t***RESULTS***\n");
printf("\n\tNON TERMINATED LIST\n\t----------------------\n\n");
printer(non_terminated,nterm_size);
printf("\n\n\t**NON-TERMINATED PROCESS NUMBER: %d\n\n\n\n",nterm_size);
printf("\n\tTERMINATED LIST\n\t----------------------\n\n");
printer(terminated,term_size);
printf("\n\n\t**TERMINATED PROCESS NUMBER: %d",term_size);
printf("\n\n\t**AVERAGE OF TERMINATED LIFECYCLES IS : %.2f\n\n",av_calculator(terminated,term_size));

// free the allocated struct arrays
free(non_terminated);
free(terminated);
free(waiting);
system("PAUSE");
return 0;
}

/**
function that calculates average lifeCycles of given process array and return as a float
@param *arr     array that calculated average according to elements(lifeCycle Values)
@param size     array size calculated average according to elements
@return average average value of given process array lifeCycles
*/
float av_calculator(PROCESS* arr,int size){
    int i;
    float sum=0.0;
    for(i=0;i<size;i++)
        sum+=(float)arr[i].lifeCycle;
    float average=(sum/(float)(size));
    return average;
}

/**
function that finds term process according to given assigment conditions and if found makes required reallocation and shifting in main.
@param  **wait      waiting struct array in main
@param  **term      terminated struct array in main
@param  *wait_size  configurable wait_size variable in main
@param  *term_size  configurable term_size variable in main
*/
void terminate_assigner(PROCESS** wait,PROCESS** term,int *wait_size,int *term_size){
    int i,index;
    index=0;
    //finding terminated element highest priority
    for(i=1;i<(*wait_size);i++){
        if(((*wait)[index].priority)>=((*wait)[i].priority)){
            //it is not necessary because new process will be added has 1 lifecycle value. if you add or swap any process has different lifeCyc
            if(((*wait)[index].priority)==((*wait)[i].priority)){
                if(((*wait)[index].lifeCycle)<((*wait)[i].lifeCycle)){
                    index=i;
                }
            }
            else{
                index=i;
            }
        }
    }
    //term size incrementing
    (*term_size)++;
    //reallocating terminated array
    re_allocator(term,term_size);
    //assigment of terminated process from waiting list
    (*term)[*term_size-1]=(*wait)[index];
    //shifting  for waiting list and reallocating waiting size inside shifter function
    shifter(wait,wait_size,index);
}

/**
function that prints given all members of struct array in a list form
@param  *arr    the process array will be printed
@param  size    the size of array
*/
void printer(PROCESS *arr,int size){
    int i=0;
    printf("\tID\tTTL\tL_Cycle\tPriority\n");
    printf("\t----------------------------------\n");
    for(i=0;i<size;i++)
        printf("\t%d\t%d\t%d\t%d\n",arr[i].process_id,arr[i].ttl,arr[i].lifeCycle,arr[i].priority);
    printf("\n\n\n");
}

/**
function that randomizes struct elements in given reference of struct array according to determined form of question
@param  *arr    struct array that aimed to randomize
@param  first   starting point process that will be randomize
@param  last    ending point procees that will be randomize
*/
```

```c
void strandomizer(PROCESS *arr,int first,int last){
    int i=0;
    for(i=first;i<last;i++){
        arr[i].lifeCycle=1;
        arr[i].priority=rand()%3+1;
        arr[i].process_id=i+1;
        arr[i].ttl=rand()%6+5;
    }
}

/**
function that increments lifeCycle variables all of struct array
@param  *arr    struct array
@param  size    struct array size
*/
void incrementer(PROCESS *arr,int size){
    int i=0;
    for(i=0;i<size;i++)
        arr[i].lifeCycle++;
}

/**
function that delete elements in array by shifting. Then it reallocates the array
@param  **arr   array that has shifting element inside.
@param  *size   size of array that will be shifted
@param  index   the index of element in array that will be shifted over
*/
void shifter(PROCESS** arr,int *size,int index){
    int i;
    //shifting
    for(i=index;i<(*size)-1;i++){
        (*arr)[i]=(*arr)[i+1];
    }
    //decrementing waiting size in main
    (*size)--;
    //reallaocation of waiting size
    if(*size>1)
        re_allocator(arr,size);
}

/**
function that finds nonterm process and if found makes required reallocation and shifting in main.
@param  **wait          waiting struct array in main
@param  **non_term      nor_terminated struct array in main
@param  *wait_size      configurable wait_size variable in main
@param  *nterm_size     configurable nterm_size variable in main
@return found           non_terminated found condition(0 is false,1 is true)
*/
int nonterm_assigner(PROCESS** wait,PROCESS** non_term,int *wait_size ,int *nterm_size){
    int i=0;
    int found=0;
    //searching in waiting queue for non-terminated processes according to ttl and lifeCycle values
    while( (i<(*wait_size)) && (found==0)){
        if((*wait)[i].lifeCycle>(*wait)[i].ttl){
            //if found incrementing nonterminated list size, reallocate, assign value from waiting queue to nonterminated list and shitfting wa
            (*nterm_size)++;
            re_allocator(non_term,nterm_size);
            (*non_term)[(*nterm_size)-1]=(*wait)[i];
            found=1;
            shifter(wait,wait_size,i);
        }
    i++;
    }
    //return found value control to take process as a terminate or not in main(1 is true, 0 is false)
    return found;
}

/**
function that makes reallocation and controls if its done or not
@param *arr     the array that will be reallocated
@param size     the new size of array that will be reallocated
*/
void re_allocator(PROCESS** arr,int *size){
    *arr=(PROCESS*)realloc(*arr,(sizeof(PROCESS)*(*size)));
        if(!(*arr)){
            printf("Not re-allocated... Quitting...");
            exit(0);
        }
}

/**
function that add random procces to given array by controlling sequential id number
@param  **arr       the array that will be added new random process
@param  *size       the size of array
@param  *id_count   the pointer of id_count in main
*/
void add_process(PROCESS** arr,int *size,int *id_count){
    //incrementing process number for assigning id values correctly.
    (*id_count)++;
    //incrementing configurable wait_size.
    (*size)++;
    re_allocator(arr,size);
    (*arr)[*size-1].lifeCycle=1;
    (*arr)[*size-1].priority=rand()%3+1;
    (*arr)[*size-1].process_id=(*id_count);
    (*arr)[*size-1].ttl=rand()%6+5;
}
```