Name ________KEYA_________

Section ______________

CS2303 C14
Systems Programming Concepts
Mid Term Exam (A)
February 7, 2014

Question	Points	Score
0	1	
1	8	8
2	4	12
3	3	15
4	32	47
5	20	67
6	18	85
Total	85	

Trivia Question (1 extra credit point)

0. (a) What city hosted this year's Superbowl ? **1 point**

East Rutherford, New Jersey

-OR-

(b) Name one country that borders the Black Sea.

Bulgaria, Romania, Ukraine, Russia, Georgia and Turkey

(8 pts)1. Given the following screen output from a ccc computer and assuming you are james:

```
$ pwd
home/james
$chdir china
$ls -la
total 48
drwxr-x--- 2 james rek 1024 Feb  1 13:18 .
drwxr-x--- 3 james rek   80 Feb  1 09:31 ..
-rwxrwxrwx 1 james rek    3 Feb  1 10:09 pork
-rwxr----- 1 james rek   11 Feb  1 10:09 pr2.c
-rwxrwx--- 1 james rek    7 Feb  1 10:09 pr2.txt
-rwxr----x 1 james rek    5 Feb  1 10:09 rice
```

james types in the following **five** command lines:

```
$ mkdir  ping
$ cp pr2.c r2.c
$ chmod 774 p*
$ mv pr* ping/
$ ls -l
```

Complete the EXACT printout to the screen after the **ls -l** command is executed.

```
total 24
drwxrwxr-- 2 james rek 80 Feb  1 13:21 ping        +2 points per correct line
-rwxrwxr-- 1 james rek  3 Feb  1 10:09 pork        -1 point per extra line
-rwxr----- 1 james rek 11 Feb  1 13:21 r2.c
-rwxr----x 1 james rek  5 Feb  1 10:09 rice
```

(4 pts) 2. Explain the difference between **call by reference** and **call by value** in C. Use the passing of an array vs passing an array element to explain the difference.

When you pass an array name (e.g., field), C treats an array name as a pointer which means you are using call-by-reference. If the called function modifies any element of the array, then upon returning to the calling function, those elements are seen as changed in the calling function. **2 points**

When you pass an array element (e.g., field [5][6]), C uses call-by-value. In this case, the compiler passes a copy of the value stored in the array element field [5][6]. If the called routine changes the value stored in the element and it stays changed in the called function. However, when the program returns to the calling function, the original value of field [5][6] has not been changed. **2 points**

(3 pts) 3. Name the **three** standard methods used by computer scientists to study **computer/network performance**.

Empirical (or run actual experiments on the real system) **1 point each**
Analytic modeling
Simulation modeling

(32 pts) 4. What is the output from this program?

```
#include <stdio.h>
#define SIZE 10
#define HSIZE SIZE/2
#define quarter 0.25
int wpi = 2017;
double stir (float y, int k)
{
  static float c = 2.0;
  int j = 7;
  wpi = wpi -(--c);
  j++;
  y++;
  printf("S%d:%6.2f%6.1f  %4d\n", k,y,c,j);
  return  2*y;
}

int main ( )
{
  float a,b,c;
  int d, e = 6, f = 2;
  int team[SIZE] = {0, 10, 20, 30, 40};
  size_t i,j ;
  a = 3.30;
  d = a + quarter;
  --e;
  b = f/d;
  c = - (float) e / 1 + 0.05;
  printf("M :%6.2f%6.2f%6.2f\n", a,b,c);
  printf("M1:  %4d %5d %-4d\n", d,e,f);
  for(i=1; i<3; i++)
  {
    b = 8.75 + i-1 * quarter;
    j = i+1;
    team[HSIZE-i] = team[HSIZE -(2*i)- j;
    wpi++;
    a = stir(2.5,j);
    c = team[i+1] % 4;
    printf("M2:%6.1f%6.2f%6.1f\n", a,b,c);
    printf("M3: class of %4d\n", wpi);
  }

  for (j=0; j<=7; j++)
    if (j % 4)
      printf("%6d", team[j]);
    else
      printf ("\nT :%6d", team[j]);
  return 0;
```

OUTPUT Box

```
M :  3.30  0.00 -4.95          3 points
M1:     3     5 2              3 points
S2:  3.50   1.0     8          4 points
M2:   7.0  9.50   0.0          3 points
M3: class of 2017              1 point
S3:  3.50   0.0     8          4 points
M2:   7.0 10.50   3.0          3 points
M3: class of 2018              1 point
                               1 point
T :     0    10    20     7    4 points
T :    28     0     0     0    4 points

 for getting M and T's printed    1 point

                       Total   32 points
```

```
}
```
(20 pts) 5. What is the output from this program?

```
#define SIZE 4
#include <stdio.h>
#include <stdlib.h>

void prnt (int *rptr)
{
  int i;
  static counter = 0;
  printf("cnt  = %2d\n", counter++);
  for (i=0; i < SIZE; i++)
  {
    printf("%-3d", *rptr);
    rptr++ ;
  }
  printf("\n");
}

int main ( )
{
  int i, j = 3, k = 5, room[SIZE]= {0};
  int *pptr, *qptr, *rptr;
  pptr = &j;
  rptr = &k;

  for (i=0; i < SIZE; i++)

  {
    room[i] = j*(*pptr);
    j++;
  }

  prnt (room);
  pptr = malloc(sizeof(j));
  qptr = &room[SIZE-1];
  *pptr = *rptr;
  printf("qptr = %2d\n", *qptr);

  for (i=1; i < SIZE; i++)
  {
    (*pptr)++;
    *(qptr--) = k - 10*(*pptr);
  }

  prnt (room);
  free(pptr);
  printf ("%3d%3d%3d\n", *qptr, *pptr, *rptr);
  return 0;
}
```

OUTPUT Box

```
cnt  =  0                 1 point
9  16 25 36               6 points
qptr = 36                 1 point
cnt  =  1                 1 point
9  -75-65-55              8 points
  9  0  5                 3 points
                  Total   20 points
```

(18 pts.) 6. Given the partial program below, write code for the function **insert** and the **main** function for a program **create-list**. Assume **create-list** has one command line argument to determine the number of process lines (**processes)** of input to read from standard-input.

The execution of the command line looks like:

```
./create-list   processes
```

An example is:

```
./create-list   3
```

create-list reads **processes** lines of input. Each input line contains a process id and an arrival time. For each line read in, **create-list** calls **insert** to create a new Node that holds the process id and arrival time as integers. **insert** adds the Node to the **front** of a linked-list such that it is accessible from **lptr** in main. Assume the input data lines are input in **decreasing** arrival time order. **Please** indicate any assumptions you need to make to increase your chances for partial credit.

Three sample input lines are:

```
1260   9
2308   2
1017   0
```

where process **1017** arrives at time **0.**

The partial program is:

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    typedef struct {
        int process;
        int arrival;
        struct Node *link;
    } Node;
    typedef Node *Link;
                                                            insert fcn 10 points
    void insert (Link *sptr, int proc, int arrtime)                  1.5 points
    {
     Link nptr;                                                      0.5 point
      nptr = malloc(sizeof(Node));                                     1 point

      if (nptr != NULL)                                              0.5 point
      {
       // Build a new node for the list.
       nptr->process = proc;                                           1 point
       nptr->arrival = arrtime;                                        1 point
```

```
   nptr->link = *sptr;                                                2 points
   *sptr = nptr;                                                      2 points
  }
  else
    printf ("Out of memory\n");                                       0.5 point
  return ;
}

int main (int argc, char *argv[])                                main fcn 8 points
{
   int i, processes, pid, atime;
   Link lptr =NULL;
   char *procstring;                                                  1 point

  if(argc != 2)                                                       1 point
    printf("Proper Usage is: com-arg samples time\n");
  else
  {
    procstring = argv[1];                                             1 point
    processes = atoi(procstring);                                     1 point
    printf("Please enter %d processes\n", processes);                 0.5 point
    for (i=0; i < processes; i++)                                for loop 3.5 points
    {
      scanf("%d %d", &pid, &atime);      1 point
      init(&lptr, pid, atime);           1.5 points
    }
  }
  return 0;
}
```