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

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>67</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>85</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>85</strong></td>
<td></td>
</tr>
</tbody>
</table>

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
-dw-r-x-- 2 james rek 1024 Feb 1 13:18 .
-dw-r-x-- 3 james rek 80 Feb 1 09:31 ..
-rw-rw-rw 1 james rek 3 Feb 1 10:09 pork
-rw-r----- 1 james rek 11 Feb 1 10:09 pr2.c
-rw-rw-r-- 1 james rek 7 Feb 1 10:09 pr2.txt
-rw-r----- 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
-dw-rw-rw- 2 james rek 80 Feb 1 13:21 ping +2 points per correct line
-rw-rw-r- 1 james rek 3 Feb 1 10:09 pork -1 point per extra line
-rw-r----- 1 james rek 11 Feb 1 13:21 r2.c
-rw-r----- 1 james rek 5 Feb 1 10:09 rice
```
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, 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?

```c
#include <stdio.h>
define SIZE 10
define HSIZE SIZE/2
define dime 0.10
int wpi = 2017;
double stir (float y, int k)
{
    static float c = 2.0;
    int j = 8;
    wpi = wpi + (--c);
    j++;
    y--;
    printf("S%d:%6.2f%6.1f  %4dn", 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 = 6.30;
    d = a + dime;
    --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 %4dn", d,e,f);
    for(i=1; i<3; i++)
    {
        b = 8.75 + i-1 * dime;
        j = i+1;
        team[HSIZE-j]= team[HSIZE -(2*j)]- i;
        wpi++;
        a = stir(2.5,j);
        c = team[i+1] % 3;
        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**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>6.30</td>
<td>0.00</td>
<td>-4.95</td>
<td>3 points</td>
</tr>
<tr>
<td>M1</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>3 points</td>
</tr>
<tr>
<td>S2</td>
<td>1.50</td>
<td>1.0</td>
<td>9</td>
<td>4 points</td>
</tr>
<tr>
<td>M2</td>
<td>3.0</td>
<td>9.65</td>
<td>2.0</td>
<td>3 points</td>
</tr>
<tr>
<td>M3</td>
<td>class of 2019</td>
<td>1 point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>1.50</td>
<td>0.0</td>
<td>9</td>
<td>4 points</td>
</tr>
<tr>
<td>M2</td>
<td>3.0</td>
<td>10.65</td>
<td>0.0</td>
<td>3 points</td>
</tr>
<tr>
<td>M3</td>
<td>class of 2020</td>
<td>1 point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>4 points</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>10</td>
<td>???</td>
<td>9</td>
</tr>
<tr>
<td>T</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

for getting M and T’s printed 1 point

Total 32 points
What is the output from this program?

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

void prnt (int *rptr)
{
    int i;
    static counter = 1;
    printf("cnt  = %2d\n", counter++);
    for (i=0; i < SIZE; i++)
    {
        printf(\"%-3d\", *rptr);
        rptr++ ;
    }
    printf("\n");
}

int main ( )
{
    int i, j = 5, k = 3, 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 = 1
25 36 49 64
qptr = 64
cnt = 2
25 -57-47-37
25 0 3
```

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:

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

An example is:

```bash
./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:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {
    int process;
    int arrival;
    struct Node *link;
} Node;
typedef Node *Link;

void insert (Link *sptr, int proc, int arrtime)
{
    Link nptr;
    nptr = malloc(sizeof(Node));
    if (nptr != NULL)
    {
        // Build a new node for the list.
        nptr->process = proc;
        nptr->arrival = arrtime;
```

```c
insert fcn 10 points
```

```c
1.5 points
```

```c
0.5 point
```

```c
1 point
```

```c
0.5 point
```

```c
1 point
```

```c
1 point
```

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

int main (int argc, char *argv[])
{
    int i, processes, pid, atime;
    Link lptr =NULL;
    char *procstring;
    if(argc != 2)
        printf("Proper Usage is: com-arg samples time\n");
    else
    {
        procstring = argv[1];
        processes = atoi(procstring);
        printf("Please enter %d processes\n", processes);
        for (i=0; i < processes; i++)
        {
            scanf("%d %d", &pid, &atime);
            init(&lptr, pid, atime);
        }
    }
    return 0;
}