Structure

Review Problems 1-9 with their solutions.
Buffer Overflows

```c
int main()
{
    char x[4];
    char y[4];
    gets(x);
    gets(y);
    printf("%s %s", x, y);
    return 0;
}
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Variable</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x108</td>
<td>y[0]</td>
<td>0x0</td>
</tr>
<tr>
<td>0x109</td>
<td>y[1]</td>
<td>0x0</td>
</tr>
<tr>
<td>0x10a</td>
<td>y[2]</td>
<td>0x0</td>
</tr>
<tr>
<td>0x10b</td>
<td>y[3]</td>
<td>0x0</td>
</tr>
<tr>
<td>0x10c</td>
<td>x[0]</td>
<td>0x0</td>
</tr>
<tr>
<td>0x10d</td>
<td>x[1]</td>
<td>0x0</td>
</tr>
<tr>
<td>0x10e</td>
<td>x[2]</td>
<td>0x0</td>
</tr>
<tr>
<td>0x10f</td>
<td>x[3]</td>
<td>0x0</td>
</tr>
</tbody>
</table>

Fill in the third column of the table for the following input -
i. abc  efgh
What is printed on the screen by the printf statement?
<table>
<thead>
<tr>
<th>Address</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x108</td>
<td>y[0]</td>
<td>‘e’</td>
</tr>
<tr>
<td>0x109</td>
<td>y[1]</td>
<td>‘f’</td>
</tr>
<tr>
<td>0x10a</td>
<td>y[2]</td>
<td>‘g’</td>
</tr>
<tr>
<td>0x10b</td>
<td>y[3]</td>
<td>‘h’</td>
</tr>
<tr>
<td>0x10c</td>
<td>x[0]</td>
<td>‘\0’</td>
</tr>
<tr>
<td>0x10d</td>
<td>x[1]</td>
<td>‘b’</td>
</tr>
<tr>
<td>0x10e</td>
<td>x[2]</td>
<td>‘c’</td>
</tr>
<tr>
<td>0x10f</td>
<td>x[3]</td>
<td>‘\0’</td>
</tr>
</tbody>
</table>

Input - abc efgh
Code Optimization - Machine Independent Techniques

Which code would you expect faster? Why?
How do we increase speed even further?

```
int dot_product_0(char *x, char*y, int size){
    int sum=0;
    for (int i=0; i<size; i++){
        sum = sum + x[i]*y[i];
    }
    return sum;
}
```

```
int dot_product_1(char *x, char*y, int size){
    int sum=0;
    int sum_1=0
    for (int i=0; i<size; i+=2){
        sum = sum + x[i]*y[i];
        sum_1 = sum_1 + x[i+1]*y[i+1];
    }
    return sum+sum_1;
}
```
Faster due to loop unrolling optimization.
Speed can be increased even further by unrolling the loop even further. Experimentally found that it was about 20% faster with 8-way unrolling.
/* foo5.c */
#include <stdio.h>

void f(void);

int x = 65536;
int y = 1024;

int main(){
    f();
    printf("x = 0x%x y = 0x%x\n", x, y);
    return 0;
}

/* bar5.c */
double x;

void f(){
    x = -0.0;
}

What is the expected result upon running this program on a 64 bit machine? Can you explain why?
The main cause of this unexpected result is that x is declared as both type ‘int’ and ‘double’. It is initialized as an int type in foo5.c, but it is updated by f(), which thinks it is a double (8 bytes). Therefore, 4 bytes of ‘y’ get overwritten.
Dynamic Memory Allocation - C memory usage mistakes

```c
void initialize(int *x, int s){
    x = (int *)malloc(s*sizeof(int));
    for (int i=0; i<s; i++) {x[i]=rand();}
    return ;
}

int main(){
    srand(time(0));
    int *x;
    initialize(x,10);
    for (int i=0; i<10; i++){
        printf("Initialized x[i] to %d\n", x[i]);
    }
    return 0;
}
```

What are the errors in this code and the possible fixes?
Dynamic Memory Allocation - C memory usage mistakes

```c
void initialize(int *x, int s){
    for (int i=0; i<s;i++) {x[i]=rand();}
    return ;
}

int main(){
    srand(time(0));
    x = (int *)malloc(s*sizeof(int));
    int *x;
    initialize(x,10);
    for (int i=0; i<10; i++){
        printf("Initialized x[i] to %d\n", x[i]);
    }
    free(x);
    return 0;
}

Solution 1
```

```c
void initialize(int **x, int s){
    *x = (int *)malloc(s*sizeof(int));
    for (int i=0; i<s;i++) {(*x)[i]=rand();}
    return ;
}

int main(){
    srand(time(0));
    int *x;
    initialize(&x,10);
    for (int i=0; i<10; i++){
        printf("Initialized x[i] to %d\n", x[i]);
    }
    return 0;
}

Solution 2
```
The cache lines are 1 byte in size.

What is the cache hit rate for the direct mapped and set associative cache, for each access pattern?
Direct Mapped

Set Associative (2-way)

Direct Mapped

Set Associative (2-way)

Direct Mapped

Set Associative (2-way)

Hit Rate = 100 ;  Hit Rate = 100

Hit Rate = 100 ;  Hit Rate = 100

Hit Rate = 0 ;  Hit Rate = 0
Write a C program to check if the number of bits in a char type is even or odd.

If it is even, then it should return 0, otherwise, it should return 1.
A Possible Solution :

```c
int oddEven(char x){
    int r=0;
    while (i < 8){
        r = r + (x >> 1) & 1;
    }
    return r & 1;
}
```
Virtual Memory -- Page Tables and TLBs

<table>
<thead>
<tr>
<th>Tag</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x15</td>
<td>0x2b</td>
</tr>
<tr>
<td>0x15</td>
<td>0x1b</td>
</tr>
</tbody>
</table>

**TLB**

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2a</td>
<td>0x17</td>
<td>Y</td>
</tr>
<tr>
<td>0x2f</td>
<td>0x26</td>
<td>Y</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>N</td>
</tr>
</tbody>
</table>

**Virtual address access List:**

- 0xabcd, 0xabba, 0xbeef, 0xdead

**16 bit address --**

- i. Page Offset (10 bits)
- ii. PPN (6 bits)

Please specify for each access if a TLB hit/miss occurred, and whether a Page Fault occurred, and the physical address. The initial contents of the TLB and the page table are shown on the left.
## Virtual Memory -- Page Tables and TLBs

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>VPN</th>
<th>Physical Address</th>
<th>TLB Hit</th>
<th>Page Fault</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xabcd</td>
<td>0x2a</td>
<td>0x5fcd</td>
<td>Y</td>
<td>N</td>
<td>0x2b</td>
</tr>
<tr>
<td>0xabba</td>
<td>0x2a</td>
<td>0x5fba</td>
<td>Y</td>
<td>N</td>
<td>0x2b</td>
</tr>
<tr>
<td>0xbeef</td>
<td>0x2f</td>
<td>0x9aef</td>
<td>N</td>
<td>N</td>
<td>0x26</td>
</tr>
<tr>
<td>0xdead</td>
<td>0x37</td>
<td>--</td>
<td>N</td>
<td>Y</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x17</td>
<td>0x26</td>
</tr>
<tr>
<td>0x15</td>
<td>0x1b</td>
</tr>
</tbody>
</table>

**TLB final state**

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>VPN</th>
<th>Offset</th>
<th>PPN</th>
<th>Offset</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xabcd</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>101010 1111001101</td>
</tr>
<tr>
<td>0xabba</td>
<td>1010</td>
<td>1011</td>
<td>1011</td>
<td>1010</td>
<td>101010 1110111010</td>
</tr>
<tr>
<td>0xbeef</td>
<td>1011</td>
<td>1110</td>
<td>1110</td>
<td>1111</td>
<td>101111 1011101111</td>
</tr>
<tr>
<td>0xdead</td>
<td>1101</td>
<td>1110</td>
<td>1010</td>
<td>1101</td>
<td>110111 1010101111</td>
</tr>
</tbody>
</table>
Design a logic circuit that finds the second smallest value among the set of 3 words, A, B, and C, using an HCL case expression.
A possible solution is as follows:

(A <= B && A >= C) || (A >= B && A <= C): A;

(B <= C && B >= A) || (B >= C && B <= A): B;

1: C;
CPU architecture -- Y86-64

Please write a Y86-64 program that implements a swap function, similar to the C code:

```c
void swap(int *x, int *y){
    int temp_1 = *x;
    int temp_2  = *y;
    *y = temp_1;
    *x = temp_2;
}
```
CPU architecture -- Y86-64

A possible solution is as follows:

swap:

- `mrmovq (%rdi), %rcx`
- `mrmovq (%rsi), %rax`
- `rmmovq %rdx, (%rsi)`
- `rmmovq %rcx, (%rdi)`
- `ret`
Thank You

Questions ?