1. In three sentences or less explain the difference between a Single-Instruction Multiple-Data (SIMD) parallel system and a Multiple-Instruction Multiple-Data (MIMD) parallel system. [2 points]

In a SIMD system there’s a single control unit and multiple datapaths; so at any given time all of the datapaths execute the same instruction on their own data (or they are idle). In an MIMD system, each processor has its own control unit and its own datapath; so at any given time all of the processors execute their own instructions on their own data.

2. In two sentences or less explain what it means to pass arguments to a function by value. [2]

When an argument is passed by value, the contents of the actual arguments — the arguments passed by the caller — are copied into the formal arguments — the arguments stored in the called function. This is different from “pass-by-reference” where the addresses of the actual arguments are copied to the formal arguments.

Figure 1: Variables in Problem 3
3. Find the output of the following C program. [4]

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

void Shmoo(int a, int *a_p, int* b_p, int* c_p);

int main(void) {
    int x = 8, *x_p = &x, y = 3, *y_p = &y;
    printf("In main 1: x = %d, *x_p = %d, y = %d, *y_p = %d\n", x, *x_p, y, *y_p);
    Shmoo(x, x_p, &y, y_p);
    printf("In main 2: x = %d, *x_p = %d, y = %d, *y_p = %d\n", x, *x_p, y, *y_p);
    return 0;
} /* main */

void Shmoo(int a, int *a_p, int* b_p, int* c_p) {
    int p = 11, *p_p = &p;
    printf("In Shmoo 1: a = %d, *a_p = %d, *b_p = %d, *c_p = %d\n", a, *a_p, *b_p, *c_p);
    printf("p = %d, *p_p = %d\n", p, *p_p);
    a = 6;
    *a_p = 2;
    *b_p = a;
    c_p = a_p;
    *p_p = 82;
    printf("In Shmoo 2: a = %d, *a_p = %d, *b_p = %d, *c_p = %d\n", a, *a_p, *b_p, *c_p);
    printf("p = %d, *p_p = %d\n", p, *p_p);
} /* Shmoo */
```

Output:
In main 1:  x = 8, *x_p = 8, y = 3, *y_p = 3
In Shmoo 1:  a = 8, *a_p = 8, *b_p = 3, *c_p = 3
             p = 11, *p_p = 11
In Shmoo 2:  a = 6, *a_p = 2, *b_p = 6, *c_p = 2
             p = 82, *p_p = 82
In main 2:  x = 2, *x_p = 2, y = 6, *y_p = 6

There's a diagram showing memory on page 1.
4. The last page of the exam shows the definition of a struct and a linked list that was built using that struct. Show the list after the function \( F \) is run with the following arguments. [4]

(a) \( \text{head}_p \) is the head of the linked list shown on the last page. \( v_1 \) is 1, and \( v_2 \) is 4.
(b) \( \text{head}_p \) is the head of the linked list shown on the last page. \( v_1 \) is 3, and \( v_2 \) is 5.

```
void F(struct list_node_s* head_p, int v1, int v2) {
    struct list_node_s* curr_p = head_p;
    struct list_node_s* temp_p;

    while (curr_p != NULL) {
        if (curr_p->data == v1) {
            temp_p = malloc(sizeof(struct list_node_s));
            temp_p->data = v2;
            temp_p->next_p = curr_p->next_p;
            curr_p->next_p = temp_p;
            break;
        } else
            curr_p = curr_p->next_p;
    }
}
```

\( F \) finds the first occurrence of \( v_1 \) in the list and inserts \( v_2 \) immediately following it. If \( v_1 \) is not in the list, \( F \) will return without making any changes to the list.

(a)  
```
    +----+    +------+-    +------+-    +------+-    +------+-    no change  
 head_p| -+----| 3 | -+----| 1 | -+----| 4 | -+----| 2 | -+----| from here on  
    +----+    +------+-    +------+-    +------+-    +------+-
```

(b)  
```
    +----+    +------+-    +------+-    +------+-    +------+-    no change  
 head_p| -+----| 3 | -+----| 5 | -+----| 1 | -+----| 2 | -+----| from here on  
    +----+    +------+-    +------+-    +------+-    +------+-
```
5. Find the output of the following MPI program if it’s run with
   (a) One process [1]
   (b) Four processes [4]

```c
#include <stdio.h>
#include <mpi.h>

int Assign_vals(int my_rank) {
    switch (my_rank) {
    case 0:
        return 2;
    case 1:
        return 5;
    case 2:
        return 3;
    case 3:
        return 4;
    }
} /* Assign_vals */

int main(void) {
    int p, my_rank, x, m, ml, q, t;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    x = Assign_vals(my_rank);

    /* Every process can print to stdout */
    printf("Proc %d > x = %d\n", my_rank, x);
    if (my_rank == 0) {
        m = x; ml = 0;
        for (q = 1; q < p; q++) {
            MPI_Recv(&t, 1, MPI_INT, q, 0, MPI_COMM_WORLD,
                      MPI_STATUS_IGNORE);
            if (t > m) {
                m = t;
                ml = q;
            }
        }
        printf("m = %d, ml = %d\n", m, ml);
    } else {
        MPI_Send(&x, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
} /* main */
```

Continue on the following page.
Continue with problem 5.

(a) Output with one process
Proc 0 > x = 2
m = 2, ml = 0

(b) Output with four processes. (Note that the order of the output from the different processes may be random.)
Proc 0 > x = 2
Proc 1 > x = 5
Proc 2 > x = 3
Proc 3 > x = 4
m = 5, ml = 1

Memory for one process:

<table>
<thead>
<tr>
<th>my_rank</th>
<th>x</th>
<th>m</th>
<th>ml</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Memory for four processes

<table>
<thead>
<tr>
<th>my_rank</th>
<th>x</th>
<th>m</th>
<th>ml</th>
<th>q</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2,5</td>
<td>0,1</td>
<td>1,2,3</td>
<td>5,3,4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. Bob is using the struct on the last page in his linked list program. He has written a function `Delete_all` (see below). Its purpose is to eliminate all occurrences of a value from the list. It proceeds by searching the list for the user-specified value. Each time it finds the user-specified value, it attempts to delete it from the list. If there are no occurrences, it simply traverses the list. The function usually performs as advertised. However, it occasionally crashes with a segmentation fault.

(a) In two sentences or less describe what is causing the segmentation faults. [2]

(b) Modify the code so that it correctly eliminates all occurrences of `value` and doesn’t seg fault. [2]

```c
struct list_node_s* Delete_all(struct list_node_s* head_p, int value) {
    struct list_node_s* curr_p = head_p;
    struct list_node_s* pred_p = NULL;
    struct list_node_s* temp_p;

    while (curr_p != NULL) {
        if (curr_p->data == value) {
            pred_p->next_p = curr_p->next_p;
            temp_p = curr_p;
            curr_p = curr_p->next_p;
            free(temp_p);
        } else {
            pred_p = curr_p;
            curr_p = curr_p->next_p;
        }
    } /* while */

    return head_p;
} /* Delete_all */
```

(a) The program will crash if it attempts to delete the first node in the list. When the first node is deleted, `pred_p` is `NULL` and the assignment

```c
pred_p->next_p = curr_p->next_p;
```

causes a seg fault.

(b) Before attempting to update `pred_p->next_p`, the function should check whether `pred_p` is `NULL`. If it is, it should update `head_p` instead of `pred_p->next_p`:

```c
if (pred_p == NULL)
    head_p = curr_p->next_p;
else
    pred_p->next_p = curr_p->next_p;
```

The rest of the code is OK.
7. Write a C function that reads input text one word at a time and prints the word that is alphabetically last. For example, if the input text were

for all avered I had killed the bird
that made the breeze to blow
ah wretch said they the bird to slay
that made the breeze to blow

then the function should print the word “wretch”. A word is any sequence of characters separated by whitespace (space, tab, newline) from the rest of the text. You may assume that no word will consist of more than 99 characters and that the input text will consist only of lower case characters and whitespace. You may also assume that there is at least one word in the input text. You can use any of the functions in the C string library.

Here’s a function prototype:

```c
void Find_last_word(void);
```

```c
void Find_last_word(void) {
    char word[100];
    char last_word[100];
    int rv;

    /* Don’t need to worry about EOF: at least one word of input */
    scanf("%s", last_word);

    rv = scanf("%s", word);
    while (rv != EOF) {
        if (strcmp(word, last_word) > 0)
            strcpy(last_word, word);
        rv = scanf("%s", word);
    }

    printf("The last word in an alphabetic ordering of the input is %s\n",
           last_word);
} /* Find_last_word */
```
A C program contains the following definition.

```c
struct list_node_s {
    int data;
    struct list_node_s* next_p;
};
```

The program builds the following unsorted linked list.

```
+--+ +---+--+ +---+--+ +---+--+ +---+--+ +---+--+ +---+-+
head_p| -+-->| 3 | -+-->| 1 | -+-->| 2 | -+-->| 3 | -+-->| 2 | -+-->| 4 |/|
+--+ +---+--+ +---+--+ +---+--+ +---+--+ +---+--+ +---+-+
```