Question 1. [14 MARKS]

For each code fragment below, indicate whether it WORKS GREAT, WORKS BUT HAS ISSUES, or NEVER WORKS. Then, explain the reasoning behind your answer. Answers without explanations will receive no marks.

For the category WORKS BUT HAS ISSUES, it might be the case that the code works sometimes and not others, or it might be that the code runs but doesn't do what the user is expecting as described in the comments.

(' is a single quote, and ` is a back quote.)

Part (a) [1 MARK]

```
//change the behaviour for SIGSEGV to be ignored
if (sigaction(SIGSEGV, SIG_IGN, NULL) != 0) {
  perror("sigaction");
  exit(1);
}
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: Works fine to ignore sigsegv, but it is a really bad idea.

Part (b) [1 MARK]

```
// print the length of a line of the file.
// assume that the FILE *fp is already open for reading
char line[256];
if((fread(line, 64, 1, fp)) > 0) {
    printf("%d\n", strlen(line));
}
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: May work, but there is no guarantee that there will be a null termination character in line.

Part (c) [1 MARK]

```
// replaces f with r in the string s
char * s = "fun";
s[0] = 'r';
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: Doesn't work because s is a pointer to a string literal and you can't change a string literal.

Part (d) [1 MARK]

```
// replaces f with r in the string s
char s[4] = "fun";
s[0] = 'r';
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: Works fine because s is an array of characters that has been statically initialized with the character "fun".

Part (e) [2 MARKS]

```
// prints "1 1 1 1" without error
int a[] = {1, 3, 5};
int *p = a;

printf("%d %d %d %d\n", p==a[0],
                        p==&a[0],
                        *p==a[0],
                        p[0]==a[0]);
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: First conditional is wrong. Everything else is fine.

Part (f) [2 MARKS]

```
// The function returns a pointer to a copy of str.
char *duplicate(char *str) {
    if(str == NULL)
        return NULL;
    char *line = malloc(strlen(str) + 1);
    strncpy(line, str, strlen(str) + 1);
    rerturn str;
}
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: Just fine.

Part (g) [2 MARKS]

```
// Returns an array of size ints where each element is initialized to 0
int *init_zeros(int size) {
    int counters[size];
    int i;
    for(i = 0; i < size; i++) {
        counters[i] = 0;
    }
    return counters;
}
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: Returning a pointer on the stack is a bad idea.

Part (h) [2 MARKS]

```
// initialize an array of strings to hold the empty string.
int i;
char **files = malloc(5 * 256 * sizeof(char *));
for(i = 0; i < 5; i++) {
    files[i][0] = '\0';
}
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation:

It doesn't work to allocated 2d arrays like this. This is an error.

Part (i) [1 MARK]

```
# argument to the script is a filename in the current directory
# print this filename and the size of the file
# reminder: ls -l format is:
# -rw-r--r--  1 joeowner  somegrp  740 28 Apr 04:44 somefile.txt
set `ls -l $1`
echo $1 $5
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation:

Doesn't work correctly since filename in `$1` is replaced by set command.

Part (j) [1 MARK]

```
#!/bin/sh
# If the current working directory contains 4 files a.c, b.h, c.c, and d.h
# and each file contains 10 words the output is "wc -w a.c c.c"
sizes='wc -w *.c'
echo "$sizes"
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation:

It don't quite work because the wild card expansion is not performed in the double quotes.

Question 2. [8 MARKS]

Write a shell program that prints to standard output the name of each regular file in the current working directory and a number that represents the maximum number of words that appear on any line of the file.

Full marks will be awarded to programs that are correct, and create a minimal number of processes.

Warning: `set` with an empty string as a argument will print the list of environment variables. If you use `set` you should handle this case.

```
#!/bin/sh

for file in *
do

    max=0
    if [ -f $file ]
        then

        while read line
        do
            if [ -z "$line" ]; then
                continue
            fi
            set $line
            if [ $# -gt $max ]; then
                 max=$#
            fi

        done < $file
        echo $file $max
    fi
done
```

Question 3. [8 MARKS]

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

int main() {
    int status;

    printf("%s\n", "Hello");
    int pid = fork();

    if(pid < 0) {
        exit(1);
    } else if(pid > 0) {
        printf("1\n");
    } else {
        printf("0\n");
    }

    if(wait(&status) == -1) {
        printf("No child\n");
    } else {
        printf("%d\n", WEXITSTATUS(status));
    }

    printf("%s\n", "Bye");
    exit(2);
}
```

Part (a) [4 MARKS] Write down all the lines that the parent process prints.

```
Hello
1
2
Bye
```

Part (b) [3 MARKS] Write down all the lines that the child process prints.

```
0
No child
Bye
```

Part (c) [1 MARK] The output may appear in different orders, but one or more lines must always appear in a fixed order. For example, "Hello" must always be the first line of output. Which line or lines must always appear at the end of the output?

2 and bye must always be the last two lines.

Question 4. [7 MARKS]

In some applications, a struct may belong to more than one list. In these cases, a reference count is often added to the struct so that each instance of the struct can keep track of the number of lists that have a reference to it.

Given the types defined below, complete the function `freelist` so that it frees all the memory used by the list that `head` points to. Free `Record` structs only if they have a `refcount` value of 1.

```
typedef struct {
    char *data;
    int refcount;
} Record;

struct node {
    Record *rec;
    struct node *next;
};
typedef struct *node LList;



void freelist(LList *head) {




    LList cur = head;
    while(cur != NULL) {
        LList temp = cur;
        cur = cur->next;
        if(temp->rec->refcount == 1) {
            free(temp->rec);
        }
        free(temp);
    }
}
```

Question 5. [14 MARKS]

Part (a) [10 MARKS]

Write a C program that mimics part of the operation of a shell. It reads one line from standard input, parses the line of input using the `mkarray` function described below, and executes the command. The program will treat the line of input as a shell command line. It assumes that the first word on the command line is the command to run. It will check for **one** of two I/O redirection operators: '`<`' or '`>`' and will set up the appropriate redirection and run the command with its arguments (if any).

You may assume the existence of a function `char **mkarray(char *line)` that takes a null-terminated string as an argument, and returns an array of strings where each element of the string is one word from the line. You may assume that there will be a single space on either side of the redirection operator, so it will be identified by `mkarray` as a separate word. (In other words, you may assumed that the return value of `mkarray` is well-formed.)

You do not need to write any error checking code for this part.

```
int main(int argc, char **argv) {

    // shell reads from stdin a line containing the command, arguments to the command
    // and possible I/O redirection
    int fd;
    char line[256];
    fgets(line, 256, stdin);

    char *ptr = strchr(line, '\n');
    *ptr = '\0';
    // Step 1: parse command line into an array of strings.
    char **args = mkarray(line);
    // Step 2: Identify whether any I/O redirection is necessary
    int i;
    for(i = 0; i < 10; i++) {
        if(args[i][0] == '>') {
            fd = open(args[i+1], O_WRONLY | O_CREAT);
            if((dup2(fd, fileno(stdout))) == -1) {
                perror("dup2:");
                exit(1);
            }
            args[i] = NULL;
            close(fd);
            break;
        } else if (args[i][0] == '<'){
            fd = open(args[i+1], O_RDONLY);
            if((dup2(fd, fileno(stdin))) == -1) {
                perror("dup2:");
                exit(1);
            }
            args[i] = NULL;
            close(fd);
            break;
         }
    }

    execvp(args[0], args);
    return 0;
}
```

Part (b) [2 MARKS]

In this course, we have emphasized the importance of error checking. Choose **one** of the system calls that you used in part (a) and write a snippet of C code that tests for an error and prints a useful message to the user. (A useful error message contains information about why the error occurred.)

Should use perror correctly.

Part (c) [2 MARKS]

For **two** different system calls that you used in part (a), describe one way in which each system call could fail.

1) open could fail if the file does not exist

2) exec could fail if the user does not have permission to execute the program.

Question 6. [10 MARKS]

Study the following program that installs two signal handlers.

```
pid_t pid;
int counter = 0;

void handler1(int sig) {
    counter++;
    fprintf(stderr, "Handler 1 counter = %d\n", counter);
    // E
}

void handler2(int sig) {
    counter += 3;
    fprintf(stderr, "Handler 2 counter = %d\n", counter);
    exit(0);
}

int main() {
    struct sigaction sa1, sa2;
    sa1.sa_handler = handler1;  sa1.sa_flags = 0; sigemptyset(&sa1.sa_mask);
    sa2.sa_handler = handler2;  sa2.sa_flags = 0; sigemptyset(&sa2.sa_mask);

    // A
    sigaction(SIGINT, &sa1, NULL);
    // B
    if ((pid = fork()) == 0) {
        // C
        sigaction(SIGINT, &sa2, NULL);
        while(1) {
            sleep(1);
        }

    } else {
        // D
        int status;
        if ((wait(&status)) > 0) {
            counter += 2;
            fprintf(stderr, "counter = %d\n", counter);
        } else {
            perror("wait");
        }
    }
    return 0;
}
```

Question 6. (CONTINUED)

Describe what happens if events in each row occur in the order presented. Each event is described as a signal that arrives *just before* the process executes the line of code following the specified comment line. Give the total output of the program in each case. If no output occurs, write NONE.

Events (In order)	What happens? (Be specific but brief)	Output (Write NONE if no output)	Still running? Parent/Child/Neither
SIGINT arrives at A	Program terminates	None	Parent
SIGINT arrives at B	call handler1	Handler 1 counter=1	
SIGINT arrives at C			
SIGINT arrives at D			
SIGINT arrives at E			

Question 7. [6 MARKS]

The following program fragment is taken from a server program that expects to read a string and then a number from each client that connects. The sequence of reading a string and then a number from a client may be repeated many times.

Identify **three** significant errors with the following program fragment. Explain why each one is a problem. Assume all the code that precedes this code fragment is correct. In other words, the socket is correctly set up, and all the variables are appropriately declared and initialized.

```
maxfd = listenfd;
for (i = 0; i < MAXCLIENTS; i++)
    client[i] = -1;
FD_ZERO(&rset);
FD_SET(listenfd, &rset);
while(1) {
    nready = Select(maxfd+1, &rset, NULL, NULL, NULL);

    if (FD_ISSET(listenfd, &rset)) {
        connfd = Accept(listenfd, (struct sockaddr *) &cliaddr, &clilen);

        for (i = 0; i < MAXCLIENTS; i++)
            if (client[i] < 0) {
                client[i] = connfd;
                break;
            }
        FD_SET(connfd, &rset);
    }

    for (i = 0; i < 3; i++) {
        sockfd = client[i];
        if((sockfd > 0) && FD_ISSET(sockfd, &rset)) {
            n = Readline(sockfd, line, MAXLINE);
            printf ("Client name = %s\n", line);

            Writen(sockfd, "Enter a number", 15);

            n = Readline(sockfd, line, MAXLINE);
            printf ("Number = %d\n", atoi(line));
        }
    }
}
```

- rset will not have the correct values the second time through the loop
- Second Readline may block
- maxfd is not reset as new clients are added.

Question 8. [10 MARKS]

A time server waits for a connection from a client. When a connection is made the server immediately sends two 32-bit unsigned integers to the client. The first integer is the time in seconds since January 1, 1900, and the second is the fractional seconds. (Note that the numbers are sent as ints and not strings.)

Write a client that queries several time servers. It reads the names of the time servers from a text file that is passed in as a command line argument. The file contains one server name on each line of the file.

Your client will print to standard output the following messages depending on the success of connecting to the server where SERVER is replace by the actual name of the server:

- "No such server: SERVER" – when the server does not exist
- "Connection refused: SERVER" – when the connection failed
- "Error reading from SERVER" – when an error occurs
- "SERVER: NUM1 NUM2" – when the client successfully reads the message from the server. NUM1 and NUM2 are the two numbers sent to the client.

Some initial code to get you started is provided below:

```
#define PORT 32000

int main(int argc, char **argv) {
    struct hostent *hp;
    int sock;

    struct sockaddr_in serv;
    serv.sin_family = PF_INET;
    serv.port = htons(PORT);

    sock = socket(PF_INET, SOCK_STREAM, 0);

    FILE *fp = fopen(argv[1], "r");

    while((fgets(line, 256, fp)) != NULL) {
        line[strlen(line) - 1] = '\0';
        hp = gethostbyname(line);
        if(hp == NULL) {
             printf("No such server: %s\n", line);
        }
        serv.sin_addr = *((struct in_addr *)hp->h_addr);
        if((connect(sock, serv, sizeof(serv))) == -1) {
             printf("Could not connect to %s\n", line);
        } else {
            //write
            // read
            printf("%s: %s", line, buf);
            close(sock);
        }
    }

    return 0;
}
```