Question 1. [5 MARKS]

For each task below, give one line that you could type at the bash shell prompt to accomplish the task.

Part (a) [1 MARK]

Change the access permissions of the file `runTest` so that the owner can read, write and execute it, group members can read it and execute it, and everyone else can only read it.

SOLUTION:
```
chmod 754 runTest
```

Part (b) [1 MARK]

Send a SIGUSR1 signal to the process with PID 715.

SOLUTION:
`kill -SIGUSR1 21715` or `kill -30 21715`

Part (c) [1 MARK]

Change into the parent directory of your current working directory.

SOLUTION:
`cd ..` or `cd ../`

Part (d) [1 MARK]

Run the program `mark_em_all` found in the current directory so that it reads standard input from the file `classlist`, saves the standard output in a file `results`, and leaves standard error going to the console.

SOLUTION:
`./mark_em_all < classlist > results` or `mark_em_all < classlist > results`

Part (e) [1 MARK]

Execute the program `celebrate` from the directory `/usr/mcraig/bin` with the commandline arguments `presents` and `food`.

SOLUTION:
```
/usr/mcraig/bin/celebrate presents food
```

Question 2. [10 MARKS]

Each example below contains an independent code fragment. In each case there are variables `x` and `y` that are missing declaration statements. In the boxes to the right of the code write declaration statements so that the code fragment would compile and run without warnings or errors. The first is done for you as an example.

Code Fragment	**Declaration for** `x`	**Declaration for** `y`

`x = 10;`
`y = 'A';`
	`int x;`	`char y;`

`int age = 99;`
`x = &age;`
`y = &x;`
	`int *x;`	`int **y;`

`char *result[2];`
`x = result[0];`
`// some hidden code`
`result[0] = "read only";`
` y = x[0];`
	`char *x;`	`char y;`

`char *name = "John Tory";`
`x = &name;`
`y = *(name+3);`
	`char **x;`	`char y;`

`struct node {`
`    int value;`
`    struct node * next;`
`};`
`typedef struct node List;`
`List *head;`
`// some hidden code`
`x = head->next;`
`y.value = 14;`
`y.next = x3;`
	`List *x;`
	or
	`struct node * x;`
		`List y;`
		or
		`struct node y;`

`char fun(char *s, int n) {`
`    return s[n];`
`}`
 	
`x = fun;`
`y = fun("easy as pie",6);`
	`char (*x)(char *, int);`	`char y4;`

Question 3. [3 marks]

The following program attempts to assign values to elements of a struct. The boxed sections of code are two alternatives to the accomplish the same thing. For each set of boxes, cross off the box holding code that doesn't work. Below it, provide an explanation for why the code doesn't work. There are no marks for correctly indicating broken code without giving a correct explanation of the problem.

```
struct song {
    int track;
    char title[25];
    char *album;
    double length;
};

int main(int argc, char* argv[]) {

    struct song s1;

    s1.track = 2; | s1->track = 2;
```

```
    // set the title of the song to "Knee Deep"

    XXXX s1.title = "Knee Deep"; | strcpy(s1.title,"Knee Deep");
```

// Error because can't assign to char array like that in C

```
    // set the album to "You Get What you Give"

    s1.album = "You Get What You Give"; | XXXX strcpy(s1.album, "You Get What You Give");
```

using strcpy is an error because writes over memory that isn't yours Seg fault

```
}
```

Question 4. [7 MARKS]

Each of the code fragments below has a problem. It might be a compile error or it might lead to a run-time error. Explain what is wrong, and then fix the code by printing neatly on the code itself.

Part (a) [2 MARKS]

```
char s[32] = argv[2];
strncat(s, argv[1], strlen(s));
```

SOLUTION: s may not be large enough

```
char s[32] = "lions and";
strncat(s, argv[1], 31-strlen(s));
```

Part (b) [1 MARK]

```
char s[5] = "bears";
```

SOLUTION: not enough space for null terminator

```
char s[6] = "bears";
```

Part (c) [2 MARKS]

```
int sum_array(int *A) {
    int i;
    int sum = 0;
    for (i=0; i < sizeof(A); i++) {
        sum += A[i];
    }
    return sum;
}
```

SOLUTION: // can't use sizeof on A

```
int sum_array(int *A, int size) {
    int i;
    int sum = 0;
    for (i=0; i < size; i++) {
        sum += A[i];
    }
    return sum;
}
```

Part (d) [2 MARKS]

```
int *status_ptr;
if (wait(status_ptr) == -1) {
    perror("wait");
    exit(1);
}
```

SOLUTION: calling wait without allocation space for status

```
int status_ptr;
if (wait(&status_ptr) == -1) {
    perror("wait");
    exit(1);
}
```

Question 5. [7 marks]

All the subquestions below concern this `Makefile` and the programs it builds.

Makefile

```
FLAGS = -Wall -g

all : mktrans do_trans

do_trans : do_trans.o list.o
        gcc ${FLAGS} -o $@ $^

mktrans : mktrans.o list.o
        gcc ${FLAGS} -o $@ $^

%.o : %.c
        gcc ${FLAGS} -c $<

do_trans.o : list.h
mktrans.o : list.h
list.o : list.h

clean :
        rm *.o mktrans do_trans
```

Initial contents of the current working directory:

```
Makefile     list.c     mktrans.c
do_trans.c   list.h     short_trans
```

Part (a) [1 mark]

If you run `make mktrans`, what files will be created or modified?

Solution: list.o, mktrans.o and mktrans

Part (b) [1 mark]

If immediately after doing part a, you call `make` with no argument, what files will be created or modified?

Solution: `do_trans.o` and `do_trans`.

Part (c) [1 MARK]

If immediately after doing part b, you edit `list.h` and then call `make do_trans`, what files will be created or modified?

SOLUTION: `do_trans.o`, `list.o` and `do_trans`.

Part (d) [2 MARKS]

Write a shell command that will run `mktrans` in the background, with no arguments, and will redirect standard output to a file called `transactions` and standard error to a file called `err`.

SOLUTION: `mktrans >transactions 2>err &`

Part (e) [2 MARKS]

The executable `do_trans` takes one argument which is the name of a transaction file. `short_trans` is such a file. Write a rule for the makefile so that when `make test` is run, `do_trans` will be compiled if necessary, and then run using `short_trans` as the argument.

SOLUTION

```
test : do_trans
    do_trans short_trans
```

Question 6. [10 MARKS]

In this question you will complete a program that analyzes the characters of its command line arguments. It uses a function (that you will write) called `stats` that takes two parameters. The first is a string `str` and the second a character `ch`. The function returns an array of two integers. The first is the number of times `ch` occurs in `str` and the second is the number of spaces in `str`.

Part (a) [8 MARKS]

//Write the function here.

```
/*
 * given string str and char ch, return an array of 2 integers. The first is
 * number of times char ch occurs in str. The second is the number of spaces in str.
 */
int *stats(char *str, char ch) {
    int *result = malloc(2*sizeof(int));
    int i;
    result[0] = 0;
    result[1] = 0;

    for (i=0; i<strlen(str); i++) {
        if (str[i] == ch) {
            result[0]++;
        } else if (str[i] == ' ') {
            result[1]++;
        }
    }
    return result;
}
int main(int argc, char **argv) {
    int i, *result;
    for (i=1; i<argc; i++ ) {
        result = stats(argv[i], 'e');
        printf("arg %d: %d e's and %d spaces\n",i, result[0], result[1]);
    }
    return 0;
}
```

Part (b) [1 MARK]

This main function calls `stats` on each of the command line arguments and counts the number of `e`'s and the number of spaces. The program has a memory leak. Fix it by writing directly on this copy of the code.

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

    int i, *result;
    for (i=1; i<argc; i++ ) {
        result = stats(argv[i], 'e');
        printf("arg %d: %d e's and %d spaces\n",i, result[0], result[1]);
        free(result[0]);
        free(result[1]);
    }

    return 0;
}
```

Part (c) [1 MARK]

Assume that `stats` is written correctly and the executable is named `analyze_args`. When the program is run from the shell as `analyze_args one two three`, it prints

```
arg 1: 1 e's and 0 spaces
arg 2: 0 e's and 0 spaces
arg 3: 2 e's and 0 spaces
```

Give one example of how the program could be called from the shell so that it reports at least one non-zero number of spaces.

POSSIBLE SOLUTION:

```
analyze_args "anything in quotes with a space"
analyze_args 'single quotes are fine too'
```

Question 7. [10 MARKS]

For this question you will write a program that forks one child for each command line argument. The child computes the length of the command line argument and exits with that integer as the return value. The parent sums these return codes and reports the total length of all the command line arguments together. For example if your program is called `spread_the_work` and is called as `spread_the_work divide the load` it prints `The length of all the args is 13`.

We have provided some parts of the code and you must work within this framework and complete the missing pieces. You do **not** need to write `include` statements.

```
int main(int argc, char **argv) {
    //declare any variables you need
    int i;
    int result;

    // write the code to loop over the command line arguments (remember to skip the executable name)
    for (i=1; i < argc; i++) {
        // call fork
        result = fork();
        if (result < 0) {   // case: a system call error
            // handle the error
            perror("fork");
            exit(1);
        } else if (result == 0) {  // case: a child process
            // child does their work here
            int len = strlen(argv[i]);
            exit(len);
        }
    }

    // Finish the code to sum up the return values from the children

    int sum = 0;
    int status;
    for (i=1; i< argc; i++) {
        wait(&status);
    if(WIFEXITED( status ) ) {
            sum += WEXITSTATUS(status);
        }
    }

    printf("The length of all the args is %d\n",sum);
    return 0;
}
```

Question 8. [10 marks]

Tom has a **binary** file named `sales` where he keeps sales records for his customers. For each customer he has a struct with many fields. Among other things, it includes the total sales for this customer and the time and date of the most recent sale.

In this question, you will complete a program that will update this customer file. The program takes two command line arguments. The first is the customer's position in the data file and the second is the amount of today's sale. Your program will read the appropriate record for this customer from the file (without reading all the other records) and then updates the total sales and the current date. The updated record needs to replace the old record so that this customer remains in the same position in the file. We have completed much of the code for you and you must work with our existing code and comments. For this question, please do not check the return values of your system calls for errors.

```
struct cust{
    // lots of other fields not shown
    double total;
    time_t last_update;
};

int main(int argc, char **argv) {
    if (argc != 3) {
        fprintf(stderr, "Usage: update position sale_amt");
        return 1;
    }

    // compute the position in the file where we will find this customer
    int offset = atoi(argv[1]) * sizeof(struct sale);

    // open the file and read the struct for the correct customer
    FILE *fp = fopen("sales","r+");
    fseek(fp, offset, SEEK_SET);

    struct sale this_customer;
    fread(&this_customer, 1, sizeof(struct cust), fp);

    // not required -- only in to test my solution
    printf("This customer previously had %.2f in sales.\n",this_customer.total);
    printf("last sale was %s\n",ctime(&(this_customer.last_update)));

    // add to the total sales and change time_t to current time
    this_customer.total += atoi(argv[2]);
    time(&(this_customer.last_update));
    // could also do this_customer.last_update = time(NULL);
    // should put time() on API

    // write the updated record back to the file
    // NOTE: Do this without closing and reopening the file

    fseek(fp, -1 * sizeof(struct cust), SEEK_CUR);
    fwrite(&this_customer, 1, sizeof(struct cust), fp);

    // close the file
    fclose(fp);
    return 0;
}
```

Question 9. [3 MARKS]

Part (a) [1 MARK] From this list of system calls, check all the ones that do **NOT** block.

☐ accept [✓ fork] [✓ pipe] ☐ read ☐ select ☐ wait

Part (b) [1 MARK]

What are the values of x and y after the following C fragment has executed? Assume that it is properly within a main function and that x and y have been declared properly.

```
int a = 2;
int b = 8;
x = a | b;
y = a || b;
```

- ☐ Both `x` and `y` are `false`.
- ☐ `x` is 1 and `y` is 10.
- ☐ `x` is 2 and `y` is 8.
- [✓ `x` is 10 and `y` is 1.]
- ☐ Even if the code surrounding this is correct, the fourth line will give a syntax error and so the code can not compile.

Part (c) [1 MARK] What is `htons`?

- ☐ It is another size typedef for unsigned short used for networks.
- ☐ It is a function to change newlines between different machines.
- ☐ It is a field in `struct hostent` the stores the host byte order.
- [✓ It is a function to convert byte order for integers stored in binary.]

Question 10. [5 MARKS]

Consider the program below which compiles and runs without error on CDF.

```
void sneeze(int signal) {
  printf("Achoo!\n");
}

int main() {
  //kill(getpid(), SIGINT);        STATEMENT A   NOTICE IT IS CURRENTLY COMMENTED OUT

  struct  sigaction sa;
  sa.sa_handler = sneeze;
  sa.sa_flags = 0;  sigemptyset(&sa.sa_mask);
  sigaction(SIGINT, &sa, NULL);

  printf("morning\n");
  kill(getpid(), SIGINT);

  sigset_t sigset;
  sigemptyset(&sigset);
  sigaddset(&sigset, SIGINT);

  printf("class\n");
  kill(getpid(), SIGINT);

  sigprocmask(SIG_BLOCK, &sigset, NULL);
  printf("my presentation\n");
  kill(getpid(), SIGINT);
  printf("glad that's over\n");
  sigprocmask(SIG_UNBLOCK, &sigset, NULL);

  printf("party!\n");
}
```

Part (a) [4 MARKS] What is the output from the program?

```
morning
Achoo!
class
Achoo!
my presentation
glad that's over
Achoo!
party!
```

Part (b) [1 MARK]

What would happen if we uncommented STATEMENT A? (select one)

- ☐ The program would no longer compile.
- [✓] The program would run only until it reached that statement and then terminate before printing anything.
- ☐ The program would print one extra `Achoo` before the output shown in part a.
- ☐ The program would behave identically with or without that statement.

Question 11. [4 MARKS]

Suppose that we were writing a client program for our calendar server from A4. This program would set up the socket connection to the calendar server, ask the user for some identification such as a username and pass that information to the server. Eventually it would ask the user for a command, pass the command to the server and then echo back to the user the returned message. When the user finally issues the `quit` command, the client program would send this command, echo the reply and then shut down.

One of the operations that a user may perform is to add events to a calendar. But possibly other users are subscribed to this same calendar. Some of these other subscribed users might be logged in at the time this new event is added. We could design our calendar server to immediately notify these logged-in and subscribed users of the new event. It would also be possible to not notify the other users but only let them discover the new event the next time they called a function that used the events such as listing all events.

When the server is designed to notify all subscribed and logged-in users of any new events, will the client program **require** the use of the `select` statement?

[✓ YES] ☐ NO (Check one but no marks without a correct answer below.)

If you answered YES, list the different file descriptors that the client program will put into the readset. If you answered NO, explain why select is not needed.

SOLUTION: The client wouldn't know when to wait for an "unscheduled" notification message from the server. The file descriptors are stdin from the user and the socket to the server.

When the server is designed so that it does **not** notify all subscribed and logged-in users of any new events, will the client program **require** the use of the `select` statement?

☐ YES [✓ NO] (Check one but no marks without a correct answer below.)

If you answered YES, list the different file descriptors that the client program will put into the readset. If you answered NO, explain why select is not needed.

SOLUTION: Because the client can know exactly when a response is expected from the server, it can do the read after calling a command and not listen for other commands from the user until the response has been received.