Question 1: Short Answer Questions (7 x 5 marks each = 30 marks)

Answer the following short answer questions in your exam script book. Please do not start each question in this section on a new page as they are all rather short answers.

a) What will the following program print? Please indicate clearly any newline or other whitespace characters. If they are not indicated, they are not part of the output you intended:

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

int main(void)
{
        char * strings[] =
        {
                "Power", "corrupts", "and", "absolute", "power",
                "corrupts", "absolutely"
        };
        printf("%s\n", strings[3]);
        puts(*(strings+6));
        puts(*(strings+4)+3);
        putchar(*(*(strings+3)+4));
        putchar('\n');
        putchar(*(*(strings+1)+3)+4);
        putchar('\n');
        return EXIT_SUCCESS;
}
```

```
answer:
absolute\n
absolutely\n
er\n
l\n
v\n
```

deduct a mark for each incorrect line and half a mark for each almost correct line

b) given the follow data structures:

```
struct list_node
{
        int data;
        struct list_node * next;
};
struct list
{
        struct list_node * head;
        size_t num_nodes;
};
typedef enum
{
        FALSE, TRUE
} BOOLEAN;
```

c) Why is the contents the list header not properly initialised with the following code:

```
BOOLEAN list_init(struct list * list)
{
        list = malloc(sizeof(struct list));
        if(list == NULL)
        {
                return FALSE;
        }
        list->head = NULL;
        list->num_nodes=0;
        return TRUE;
}
```

Answer:

As the list pointer is assigned a memory address (via the return from malloc) the memory assigned in this function is not accessible and hence leaks away.

Full marks for a correct answer, half marks (2.5) for a partially correct answer.

d) Why does the following C macro behave incorrectly and how do we solve this?

```
#define MULT(X) X*X
```

There are insufficient brackets around the components of the macro and so it is not evaluated correctly for example if a sum is passed in as X, for example MULT(1+2) becomes 1+2*1+2 which is 5 and not the correct answer of 9. This can be solved with the introduction of brackets, such as:

#define MULT(X) ((X)*(X))

e) Why does gcc issue a compiler warning when compiling this program? Also, what is the output of the program?

```
int main(void)
{
    enum {
        FIRST, SECOND, THIRD, FOURTH, FIFTH
    } num_times;

    for( num_times = FIRST; num_times <= FIFTH;
           ++num_times)
    {
        switch(num_times)
        {
                case FIRST:
                   printf(" ## just once ##\n");
                        break;
                case SECOND:
                    printf(" ## So here we are again"
                           " ##\n");
                        break;
                case FOURTH:
                    printf(" ## Here for the fourth "
                           "time ##\n");
                case FIFTH:
                    printf(" ## All done now ##\n");
            }
        }
    return EXIT_SUCCESS;
}
```

Answer:

A compiler warning is issued because enumeration member “THIRD” is not used in this switch statement.

Output:

```
## just once ##
 ## So here we are again ##
 ## Here for the fourth time ##
 ## All done now ##
 ## All done now ##
```

2 marks for highlighting the correct compiler warning. 3 marks for the correct output – deduct one mark for each line of incorrect output.

f) The following is output from valgrind. Please provide as much detail as you can on the cause of this error message.

```
==21416== Use of uninitialised value of size 8
==21416==    at 0x400572: main (1e.c:7)
==21416==
==21416== Invalid read of size 4
==21416==    at 0x400572: main (1e.c:7)
==21416==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
```

Answer:
I would accept here an explanation that indicated that this is a case of use of an uninitialized pointer or a NULL pointer. Those answers would get 5 marks. I will allow half marks for an answer that is not correct but shows some correct understanding.

g) Consider the following Makefile:

```
TARGETS=first.c second.c third.c
OBJECTS=first.o second.o third.o
HEADERS=first.h second.h third.h shared.h
CFLAGS=-ansi -Wall -pedantic
LFLAGS=

all: project

project: $(OBJECTS)
      $(CC) $(LFLAGS) -o project

%.c:%o $(HEADERS)
      $(CC) $(CFLAGS) -c $<
```

Write a "clean" target for this Makefile that deletes the executable and any object files created by this Makefile.

```
clean:
      rm –f *.o project
```

Full marks for a clean target that would work for this program. Half marks for a mostly correct target. Note: some students might also include a .PHONY target ant that's fine.

Question 2: Edge Finding in an ascii image (15 + 25 = 35 marks)

2a) Your first task here is to write a function to find the edges in a piece of ascii art. An edge is defined as a non-space character which has space characters for neighbours. You should copy a character to the destination only if it is an edge character so defined. You may assume that the destination array starts off as totally containing space characters (as output by the space bar) but the contents of this array is NOT strings but it is an array of arrays of char - they are not nul terminated.

```
Note: is_edge() not required as a separate function.
BOOLEAN is_edge(char picture[][PICTURE_WIDTH], int x, int y)
{
        /* do bounds checking - 3 marks */

        if(x == 0 || y == 0 || x == PICTURE_WIDTH - 1 ||
           y == PICTURE_HEIGHT - 1)
        {
                return TRUE;
        }
        /* test if this is an actual edge - 5 marks */
        if( picture[y][x] != SPACE)
        {
                if( picture[y-1][x] == SPACE ||
                    picture[y+1][x] == SPACE ||
                    picture[y][x-1] == SPACE ||
                    picture[y][x+1] == SPACE)
                {
                        return TRUE;
                }
        }
        /* identify that this is not an edge - 2 marks */
        return FALSE;
}

void find_edges(char source_image[PICTURE_HEIGHT][PICTURE_WIDTH],
                char dest_image[PICTURE_HEIGHT][PICTURE_WIDTH])
{
        int y,x;
        /* iterate over the 2d array - 5 marks */
        for(y = 0; y < PICTURE_HEIGHT; ++y)
        {
                for(x=0; x < PICTURE_WIDTH; ++x)
                {
                        if(is_edge(source_image, x, y))
                                dest_image[y][x]
                                        = source_image[y][x];
                        else
                                dest_image[y][x] = SPACE;
                }
        }
}
```

2b) Write a COMPLETE PROGRAM that makes use of the function you have designed above.

```c
/* correct header files - 2 marks */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Note the these constants are not required as they were provided
 * as part of the question */
#define PICTURE_HEIGHT 80
#define PICTURE_WIDTH 80
#define EXTRA_CHARS 2
#define SPACE ' '
#define SOURCE_ARG 1
#define DEST_ARG 2
#define NUM_ARGS 3

int main(int argc, char** argv)
{
        /* correct variable declarations - 2 marks */
        FILE * source, *destination;
        char source_image[PICTURE_HEIGHT][PICTURE_WIDTH],
                dest_image[PICTURE_HEIGHT][PICTURE_WIDTH];
        char line [PICTURE_WIDTH + EXTRA_CHARS];
        unsigned curline = 0;
        unsigned x,y;
        /* check for correct command line arguments - 1 mark */
        if(NUM_ARGS != argc)
        {
                fprintf(stderr, "Error: incorrect arguments.\n");
                return EXIT_FAILURE;

        }
        /* open the file for reading and check it opened -
         * 2 marks
         */
        source = fopen(argv[SOURCE_ARG], "r");
        if(!source)
        {
                perror("failed to open file");
                return EXIT_FAILURE;
        }
        /* initialize the pictures - 2 marks */
        memset(source_image, SPACE,
               sizeof(char) * PICTURE_HEIGHT * PICTURE_WIDTH);
        memset(dest_image, SPACE,
               sizeof(char) * PICTURE_WIDTH * PICTURE_HEIGHT);
        /* read in each line from the file 3 marks */
        while(fgets(line, PICTURE_WIDTH + EXTRA_CHARS, source))
        {
                if(line[strlen(line)-1] != '\n')
                {
                        fprintf(stderr, "Error: line too long.\n");
                        return EXIT_FAILURE;
                }
                line[strlen(line)-1]=0;
                /* copy into the array without the null terminator
```

```
                 * - 2 marks
                 */
                if(strlen(line) > 0)
                {
                        memcpy(source_image[curline], line,
                               strlen(line));
                }
                ++curline;
        }
        fclose(source);
        /* call find_edges() correctly - 2 marks */
        find_edges(source_image, dest_image);
        /* open destination file and validate it opened correctly
         * - 2 marks
         */
        destination = fopen(argv[DEST_ARG], "w");
        if(!destination)
        {
                perror("failed to open file for writing");
                return EXIT_FAILURE;
        }
        /* write out data to the destination file (including
         * newlines)- 2 marks */
        for(y = 0; y < PICTURE_HEIGHT; ++y)
        {
                for(x =0; x < PICTURE_WIDTH; ++x)
                        putc(dest_image[y][x], destination);
                putc('\n', destination);
        }
        return EXIT_SUCCESS;
}
```

Question 3: Tokenization and String Processing (15 + 25 = 40 marks)

3a) For this function you will write a function to accept a string as a parameter along with an array of char pointers in which to store tokens received and a string containing the delimiters.

```
BOOLEAN tokenize(const char * line, char * tokens[MAXWORDS],
      char * delimiters)
{
        /* variable declarations - 2 marks */
        char * linecpy = mystrdup(line);
        char * tok = strtok(linecpy, " ");
        unsigned index = 0;
        /* initialise pointers to NULL - 2 marks */
        memset(tokens, 0, MAXWORDS * sizeof(char*));
        /* while loop for tokenization - 3 marks */
        while(tok)
        {
                /* copy the pointer for each token to the tokens
                 * array - 5 marks
                 */
                char * tokcpy = mystrdup(tok);
                if(tokcpy)
                {
                        tokens[index++] = tokcpy;
                }
                else
                {
                        return FALSE;
                }
                /* call strtok to get the next token - 3 marks */
                tok = strtok(NULL, " ");
        }
        free(linecpy);
        return TRUE;
}
```

3b) Write a COMPLETE PROGRAM that allows the user to specify a file to find the longest word in.

```
/* correct include of headers - 2 marks */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char** argv)
{
        /* correct variable declarations - 3 marks */
        FILE * fp;
        char line[LINE_LEN + EXTRA_CHARS];
        char * tokens[MAXWORDS];
        int max = 0;
        char * maxword = NULL;
        /* validate command line arguments - 2 marks */
        if(argc != NUM_ARGS)
        {
                fprintf(stderr, "Error: invalid arguments.\n");
                return EXIT_FAILURE;
        }
        /* open the file and validate that it opened correctly
         * - 3 marks
         */
        fp = fopen(argv[FILE_ARG], "r");
        if(!fp)
        {
                perror("failed to open file");
                return EXIT_FAILURE;
        }
        /* read in each line each line of the file - 3 marks */
        while(fgets(line, LINE_LEN + EXTRA_CHARS, fp))
        {
                int word_count;
                /* correctly call the tokenize function (2 marks)*/
                if(!tokenize(line, tokens, " "))
                {
                        fprintf(stderr, "Error: memory problem "
                                "with tokenization.\n");
                        return EXIT_FAILURE;
                }
                /* iterate over the words - 2 marks*/
for(word_count = 0; word_count < MAXWORDS &&
                        tokens[word_count]; ++word_count)
                {
                        /* check if the word is the longest
                         * - 2 marks */
                        if(strlen(tokens[word_count]) > max)
                        {
                                if(maxword)
                                {
                                        free(maxword);
                                }
                                /* if it is, store the new word
                                 * and get rid of the old word
                                 * - 2 marks
                                 */
                                maxword = mystrdup(
```

```
                                        tokens[word_count]);
                                max = strlen(maxword);
                        }
                        free(tokens[word_count]);
                }
        }
        printf("The longest word was %s which was %u characters "
                "long.\n", maxword, max);
        /* free all memory and files you allocated - 4 marks */
        free(maxword);
        fclose(fp);
        return EXIT_SUCCESS;
}
```

Question 4: 2d Arrays, random numbers and pointers (20 marks)

For this question, you may find the following data structures and constants from assignment 1 will come in handy:

We have decided to ask you to implement an extension of assignment 1. You are to implement the following function:

```
BOOLEAN swap_token( enum cell board[BOARDHEIGHT][BOARDWIDTH],
                    const unsigned oldx, const unsigned oldy,
                    enum cell expected,
                    unsigned * newx, unsigned *newy)
{
        /* appropriate variable declarations - 1 marks */
        BOOLEAN success = FALSE;
        unsigned lnewx,lnewy;
        /* validate that the token at the coordinates specified is
         * what was expected - 1 marks */
        if(board[oldy][oldx] != expected ||
                (expected != C_NOUGHT &&
                expected != C_CROSS))
        {
                return FALSE;
        }

        /* while loop for random generation - 3 marks */
        while(!success)
        {
                /* generate random x and y coordinates within the
                 * allowed range - 3 marks
                 */
                lnewx = rand() % BOARDWIDTH;
                lnewy = rand() % BOARDHEIGHT;
                /* test if the coordinates selected refer to an
                 * empty location - 2 marks
                 */
                if(board[lnewy][lnewx] == C_EMPTY)
                {
                        success = TRUE;
                }
        }
        /* assign expected to new location - 2 marks */
        board[lnewy][lnewx] = expected;
        /* assign empty to old location - 2  marks */
        board[oldy][oldx]=C_EMPTY;
        /* 6 marks - store the new coordinates in locations
         * referred to by the pointers
         */
        *newy = lnewy;
        *newx = lnewx;
        return TRUE;

}
```

Question 5: Linked Lists, Assignment 3 Extension (5+20+25 = 55 marks)

5a) Software Design:

```
struct word
{
        char * word;
        size_t count;
        /* new variable - cost - 2.5 marks*/
        int cost;
};

struct word_list
{
        struct list_node * head;
        size_t num_words;
        /* new variable - overall cost - 2.5 marks */
        int overall_cost;
};
```

5b) Calculate Costs:

```
void list_update(struct word_list * list)
{
        /* correct variable declarations - 2 marks */
        struct list_node * current;
        current = list->head;
        /* initialise overall cost to 0 - 2 marks */
        list->overall_cost = 0;
        /* correct iteration over the word list - 5 marks */
        while(current != NULL)
        {
                struct word * curdata = current->data;
                /* grab the lenght of the word using strlen()
                 * - 2 marks
                 */
                size_t len = strlen(curdata->word);
                /* correctly calculate cost - 5 marks */
                if(len <= 5)
                {
                        curdata->cost = curdata->count * 5;
                }
                else if(len <= 10)
                {
                        curdata->cost = curdata->count * 10;
                }
                else
                {
                        curdata->cost = curdata->count * 20;
                }
                /* update overall cost of the words in the list
                 * so far - 4 marks
                 */
                list->overall_cost += curdata->cost;
        }
}
```

5c) Update to list deletion:

You now need to rewrite the delete function for this list. In this case the delete is pretty much the same as what you implemented in your assignment except that you need to deduct the total cost of the word to be deleted from current overall cost. You should make every effort to keep all other variables up to date, including freeing of memory and returning an appropriate success or failure value. The function prototype for this question is:

```
BOOLEAN list_delete(struct word_list * list, const char* word)
{
	/* correct variable declarations - 2 marks */
	struct list_node * current, *prev = NULL;
	current = list->head;
	/* correct list iteration to find the word we want to
	 * delete - 3 marks
	 */
	while(current != NULL && word_cmp(current->data, word) < 0)
	{
		current = current->next;
	}
	/* handle the word not existing in the list - 3 marks */
	if(!current || word_cmp(current->data, word) != 0)
	{
		return FALSE;
	}
	/* handle deleting the word at the beginning of the list
	 * - 4 marks
	 */
	if(!prev)
	{
		list->head = current->next;
	}
	/* handle deleting the word from elsewhere in the list -
	 * 4 marks
	 */
	else
	{
		prev->next = current->next;
	}
	/* decrement the word list - 3 marks */
	--list->num_words;
	/* deduct from the overall_cost  - 3 marks */
	list->overall_cost -= current->data->cost;
	/* free memory - 3 marks */
	free(current->data->word);
	free(current->data);
	free(current);
	return TRUE;

}
```