For each question in this exam, you may assume that a BOOLEAN type has been defined where FALSE is 0 and TRUE is 1. Any questions which ask for a 'main function' or a 'program' require you to also specify any header files that need to be included. You are not expected to do validation of input files however you are expected to check return values from all functions you call. I have provided a function reference at the end of this exam of most of the functions you will need to complete this exam.

1 Question 1: Short Answers

A) What is the output of this small program? Please be as exact as you can. Please note that I have checked this program by compiling and running it so there are no errors (10 marks):

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

int main(void)
{
        char * strings[] =
        {
                "Two",  "things",  "are","infinite:",  "the",
                "universe", "and",  "human",  "stupidity"
        };
        char * word = malloc(strlen(strings[8]) + 1);
        strcpy(word, strings[8]);
        word[6]=0;
        strcat(word, *(strings+7)+2);
        puts(*strings);
        printf("%s\n", strings[1]+3);
        puts(word);
        puts(*(strings+3)+4);
        putchar(*(strings[5]+3)+2);
        putchar('\n');
        free(word);
        return EXIT_SUCCESS;
}
```

Answer:
Two
ngs
stupidman
nite:
x

2 marks for each correct line of output.

B) The purpose of the function below is to allocate an array of ints and make that space available to the function that called this function. However, there is a memory allocation problem in this function. The "ints" array seems to have been unmodified when we exit this function. Identify what is causing this problem and a solution to this problem. Please note that problems other than memory will not get any marks (5 marks):

```
BOOLEAN allocate_ints(int * ints, int num_ints)
{
        ints = malloc(sizeof(int) * num_ints);
        if(ints)
        {
                return TRUE;
        }
        return FALSE;
}
```

Answer:
All variables passed into a function are passed by value including pointers. That means that when we assign to ints we assign to the local copy and not to the pointer in the function that was passed in here. There are two acceptable solutions: simply return a pointer instead of a BOOLEAN. Another acceptable solution is to make the ints argument a pointer to a pointer and then dereference it every time we use it within this function. (2 marks for identification of the problem), 3 marks for identification of a solution.

C) The following error has been output by valgrind. What kind of programming error would cause this output (5 marks)?

```
==26516== Conditional jump or move depends on uninitialised
value(s)
==26516==    at 0x4E7E79D: vfprintf (vfprintf.c:1636)
==26516==    by 0x4E861F8: printf (printf.c:33)
==26516==    by 0x1086CD: main (1c.c:8)
```

Answer:

a variable has been used before it has been initialised. This is usually caused either by declaring a variable and reading from it before writing to it or accessing memory outside of your valid allocation and therefore it has not been initialised yet.

D) The following is a macro defined in a C program. Identify the error in this macro and propose a fix for this error (5 marks):

```
#define MULTIPLY(A,B) A*B
```

Answer:
#define MULTIPLY(A,B) ((A)*(B))
If they only have one set of brackets deduct 2 marks.

E) The following code will generate a compiler warning when compiled with a C compiler. What is this compiler warning and why is it dangerous to ignore such warnings (5 marks)?

```
printf(3);
```

Answer:

compiler warning: conversion from int to pointer without a cast. This kind of warning is dangerous to ignore as the program will interpret 3 as a memory address which is not valid on a modern computer with multiple processes running. This means when this program runs it will crash immediately.

Question 2 – Memory Allocation (30 marks)

Write a function that accepts a string to tokenize, an array of char pointers and a variable that specifies the size of that array. You are to tokenize the string with strtok() and then malloc space for each string in the array of char pointers, ensuring that there are no more tokens than specified by the third argument. Each token must be malloced (don't just assign them). You must not modify the line passed in as it might be needed later in the program. You should ensure that any functions calls that may fail are tested (such as malloc()). Any memory that you allocate must also be freed unless it needs to be accessed outside this function. The prototype for the function you must implement is:

```
BOOLEAN tokenize(const char * line, char *tokens[MAX_TOKENS],
    char delimiter, int max_num_tokens);
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef enum { FALSE, TRUE } BOOLEAN;

#define MAX_TOKENS 40

BOOLEAN tokenize(const char * line, char * tokens[MAX_TOKENS],
        char delimiter, int max_num_tokens)
{
        /* correct variable declarations - 3 marks */
        char * copy;
        char delims[2];
        char * tok;
        int num_toks;

        /* allocate memory for copy and check it succeeded 5 marks */
        copy = malloc(sizeof(char) * (strlen(line) + 1));
        if(!copy)
        {
                perror("failed to allocate memory");
                return FALSE;
        }
        strcpy(copy, line);
        /* initialise the array to be empty - 2 marks */
        memset(tokens, 0, sizeof(char *) * MAX_TOKENS);

        /* store delimiter in a string 2 marks */
        sprintf(delims, "%c", delimiter);
        num_toks = 0;
        /* correct tokenization - 5 marks */
```

```
        tok = strtok(copy, delims);
        while(tok && num_toks < max_num_tokens)
        {
                /* malloc the string to be copied - 5 marks */
                char * copy_tok = malloc(strlen(tok) + 1);
                if(!copy_tok)
                {
                        perror("failed to allocate memory");
                        return FALSE;
                }
                /* copy the string - 3 marks */
                strcpy(copy_tok, tok);
                /* store the string in the array - 3 marks */
                tokens[num_toks++] = copy_tok;
                tok = strtok(NULL, delims);
        }
        /* return correct exit status - 2 marks */
        return TRUE;
}
```

Question 3 – FILE Input/Output (30 marks)

Write a program that accepts a filename as an argument. You should open that file as an ascii file and read it into memory. The file will only have one line of text which is a comma delimited set of integers. You are to tokenize this line and convert these to integers using strtol. You are then to sum these numbers together. Any resources allocated by your program must be freed before you exit the program. You are not required to validate the contents of this file but you should check the return value of any functions that might fail. Your program should output the sum of the integers read in from the file such as:

The total of the numbers read in was: 48

```
/* including appropriate headers - 3 marks */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <ctype.h>

#define EXTRA_CHARS 2
#define DECIMAL 10

int main(int argc, char *argv[])
{
        /* correct variable declarations - 3 marks */
        FILE * fp;
        char line[BUFSIZ + EXTRA_CHARS];
        char * tok;
        int sum = 0;

        /* check correct number of arguments - 2 marks */
        if(argc != 2)
        {
                fprintf(stderr, "Error: invalid number of arguments.\n");
                return EXIT_FAILURE;
        }
        /* correct call to fopen and check it was successful 2 marks */
        fp = fopen(argv[1], "r");
```

```
        if(!fp)
        {
                perror("failed to open file");
        }
        /* read line from file - 3 marks */
        if(!fgets(line, BUFSIZ + EXTRA_CHARS, fp))
        {
                fprintf(stderr, "Error: no data to read.\n");
                return EXIT_FAILURE;
        }
        /* string tokenization - 5 marks */
        tok = strtok(line, ", ");
        while(tok)
        {
                char * end;
                long lnum;
                /* extract number and validate it was correct - 3
                   marks */
                lnum = strtol(tok, &end, DECIMAL);
                while(isspace(*end))
                {
                        ++end;
                }
                if(*end)
                {
                        fprintf(stderr, "Error: %s is not a valid number."
                                "\n", tok);
                        return EXIT_FAILURE;
                }
                /* check number is in range - 3 marks */
                if(lnum < INT_MIN || lnum > INT_MAX)
                {
                        fprintf(stderr, "Error: %ld is out of range.\n",
                                lnum);
                        return EXIT_FAILURE;
                }
                /* add number to total - 2 marks */
                sum += lnum;
                tok = strtok(NULL, ", ");
        }
        /* print out the sum - 2 marks */
        printf("The sum is: %d\n", sum);
        /* close files and cleanup resources - 2 marks */
        fclose(fp);
        return EXIT_SUCCESS;
}
```

Question 4 – Linked List (30 marks)

Consider the following data structure definitions for a dynamic array of students, which each student has a linked list of the grades they have attained.

```
#define COURSE_CODE_LEN 8
#define NAME_LEN 30
struct grade
{
        char course_code[COURSE_CODE_LEN + 1];
        double result;
};
```

```
struct grade_list
{
        struct node * head;
        int num_grades;
};

struct grade_list
{
        struct node * head;
        int num_grades;
};

struct student
{
        char first_name[NAME_LEN + 1];
        char last_name [NAME_LEN + 1];
        struct grade_list grade_list;
};


struct student_vector
{
        struct student * students;
        int num_students;
};
```

A) We wish to store a “grade point average” for each student where the grade point average is the average of the “grade points” attained for each course. Insert into the above data structure variables to store the grade points for each course as well as an overall grade point average for each student. Both need to be floating point quantities. It is sufficient to list each struct that needs to be modified followed by the declaration that needs to be added to that struct. (5 marks).

Answer:

struct grade: double grade_points;

struct student / struct grade_list: double gpa; this would be better in student that grade_list though.

B) Write a function to calculate grade points as well as the grade point average. Grade points are calculated as follows:

For a high distinction (greater than or equal to 80), 4 grade points are awarded.

For a distinction (greater than or equal to 70 and less than 80), 3 grade points are awarded.

For a Credit (greater than or equal to 60 and less than 70), 2 grade points are awarded.

For a pass (greater than or equal to 50 and less than 60), 1 grade point is awarded.

For a fail (less than 50), 0 grade points are awarded.

Your task is as follows: iterate over the students in the student_vector passed in and calculate the grade points to be awarded for each course as well as the grade point average for the student. Assign these to the variables that you inserted in these structures in part A. The function prototype for the function to calculate the grade point average is (25 marks):

note: struct node not defined so be lenient with those 5 marks.

```
void calculate_gpas(struct student_vector * student_list);
```

```
void calculate_gpas(struct student_vector * student_list)
{
        /* all variables have been declared correctly - 2 marks */
        /* loop through the array of students - 5 marks */
        int count;
        for(count = 0; count < student_list->num_students; ++count)
        {
                /* iterate over the grade list  - 5 marks */
                struct grade_list * list =
                        & student_list->students[count].grade_list;
                struct node * current = list->head;
                double total = 0.0;
                while(current)
                {
                        /* set the grade points for each grade - 5
                           marks */
                        struct grade * data = current->data;
                        if(data->result >= 80.0)
                        {
                                data->grade_points = 4.0;
                        }
                        else if(data->result >= 70.0)
                        {
                                data->grade_points = 3.0;
                        }
                        else if(data->result >= 60.0)
                        {
                                data->grade_points = 2.0;
                        }
                        else if(data->result >= 50.0)
                        {
                                data->grade_points = 1.0;
                        }
                        else
                        {
                                data->grade_points = 0.0;
                        }
                        /* add to the total of grade points - 3 marks */
                        total += data->grade_points;
```

```
                        current = current->next;
                }
                /* calculate gpa for student - 5 marks */
                student_list->students[count].gpa =
                        total / list->num_grades;
        }
}
```

Appendix A—Some commonly used C library functions

The following list presents selected standard library functions available in the C language, with brief descriptions as they might appear in a range of available texts and online manuals. For your convenience, you may wish to detach this and following pages, for ease of reference during the examination.

stdio.h

`FILE * fopen(const char * filename, const char * mode)`

> `fopen` opens the named file, and returns a stream, or `NULL` if the attempt fails. Legal values for mode include "r" open a file for reading, "w" create a file for writing; discard previous content if any, etc.

`int fclose(FILE * stream)`

> `fclose` flushes any unwritten data for `stream`, discards any unread buffered input, frees any automatically allocated buffer, then closes the stream. It returns `EOF` if any errors occurred, and zero otherwise.

`int fprintf(FILE * stream, const char * format, …)`

> `fprintf` converts and writes output to stream under the control of `format`. The return value is the number of characters written, or negative if an error occurred. The `format` string contains two types of objects: ordinary characters, which are copied to the output stream, and conversion specifications. (eg. %c, %d, %f, %s, etc.)

`int fscanf(FILE * stream, const char * format, …)`

> `fscanf` reads from stream under control of `format`, and assigns converted values through subsequent arguments, *each of which must be a pointer*. It returns when `format` is exhausted. `fscanf` returns `EOF` if end of file or an error occurs before any conversion; otherwise it returns the number of input items converted and assigned.

`int printf(const char * format, …)`

> `printf(…)` is equivalent to `fprintf(stdout, …)`

`int scanf(const char * format, …)`

> `scanf(…)` is identical to `fscanf(stdin, …)`

`int fgetc(FILE * stream)`

> `fgetc` returns the next character of `stream` as an unsigned char (converted to an int), or `EOF` if end of file or error occurs.

`int fseek(FILE *stream, long offset, int whence);`

`fseek()` function sets the file-position indicator for the stream pointed to by `stream` and returns 0 on success and –1 on error.

The new position, measured in bytes, from the beginning of the file, is obtained by adding `offset` to the position specified by whence, whose values are defined in `<stdio.h>` as follows:

`SEEK_SET`
Set position equal to offset bytes.

`SEEK_CUR`
Set position to current location plus offset.

`SEEK_END`
Set position to `EOF` plus offset.

ctype.h

```
int isspace(int c)
```

returns non-zero (true) if the argument `c` is the character code for a space, form-feed, newline, carriage return, tab, or vertical tab. Returns zero (false) for any other value of `c`.

```
tolower(int c)
```

if `c` is an uppercase character it returns the corresponding lowercase character, otherwise it returns `c`

```
toupper(int c)
```

if `c` is an lower case character it returns the corresponding uppercase character, otherwise it returns `c`

string.h

```
char * strcat(char * s, const char * ct)
```

concatenate string `ct` to end of string `s`; returns `s`.

```
char * strtok(char * s, const char * ct)
```

`strtok` searches `s` for tokens delimited by characters from `ct`; see below. To use `strtok`, see example below:

```
#define SEPCHARS " \t\n"

int main(void)
{
     char * word;
     char * line = "Hello there        sir";
```

```
        word = strtok(line,SEPCHARS); /*Finds first token*/
        while (word != NULL)
        {
            printf("\"%s\"\n",word);
            word = strtok(NULL,SEPCHARS); /*Finds next token*/
        }
        return EXIT_SUCCESS;
    }
```

Output:

Hello
there
sir

```
int strcmp(const char * cs, const char * ct)
```

compare string `cs` to string `ct`; return < 0 if `cs<ct`, 0 if `cs==ct`, or >0 if `cs>ct`.

```
size_t strlen(const char * cs)
```

return length of `cs`.

stdlib.h

```
void * malloc(size_t size)
```

`malloc` returns a pointer to memory for an object of size `size`, or `NULL` if the request cannot be satisfied. The space is uninitialised.

```
void free(void *p)
```

`free` deallocates the memory pointed to by `p`; it does nothing if `p` is `NULL`. `p` must be a pointer to memory previously allocated by `calloc`, `malloc`, or `realloc`.

```
int atoi(const char * s)
```

converts `s` to `int`;

End of Appendix A