Please Note that for all questions on this exam you may assume that the BOOLEAN type is defined and has valid values of `TRUE` and `FALSE`.

Question 1 – Short answer Questions (5+5+5+5+5+5)

a) what is the output of the following program:

```
int i=4;
int doit(int i)
{
      i=5;
      return i*2;
}

int main(void)
{
      i=doit(i);
}
```

b) what is the output of the following program:

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

int main(void)
{
    char * message[] =
    {
        "darkness", "cannot", "drive", "out", "darkness"
    };
    char * word = *(message + 3);
    puts(message[4]);
    putchar(*(*(message+4)+3)+6);
    puts(*(message+2)+3);
    puts("darkness"+2);
    puts(word);
    return EXIT_SUCCESS;
}
```

c) The following is an error given by gcc – what could be wrong? :

```
/tmp/ccMmCq0c.o: In function `main':
1c.c:(.text+0x5): undefined reference to `doit'
collect2: error: ld returned 1 exit status
```

d) the following is the definition of a macro to implement division of two numbers. There are a range of ways this macro could be used incorrectly. Provide a correct definition that does not have these problems:

```
#define DIV(A,B) A/B
```

e) There is a problem with the **output** of this function (there is also a problem with the input but we will look at that in the next question). What is the problem and how could we fix this?

```
#include <stdio.h>
#include <stdlib.h>
#define LEN 80
#define EXTRACHARS 2
/* converts metres to kilometres */
#define M2KM 1000
int main(void)
{
    int metres, kilometres;
    char line[LEN + EXTRACHARS];
    printf("How many metres have you run this week? ");
    fgets(line, LEN + EXTRACHARS, stdin);
    metres = atoi(line);
    kilometres = metres/M2KM;
    printf("You have run %d kilometres this week\n", kilometres);
    return EXIT_SUCCESS;
}
```

f) There were also several problems in the previous function with how **input** was **validated** (or in fact not validated). Identify at least one of those problems and provide a solution to that problem.

Question 2 – Generics (10+5+15)

a) Write a function called iterate() to iterate over any type of array. The function should use the passed in function pointer to perform some operation on each element of the array.

```
void iterate(void * array, size_t element_size,
          size_t element_count, void alter(void*));
```

b) Write two functions to be passed into the function you defined in the previous question. One should add 5 onto an integer, the other should add 5 onto a double. You should perform any necessary casts to achieve this task. The function prototypes are:

```
void add_5_int(void* num);
void add_5_dbl(void* num);
```

c) Write a main function where you create two arrays – one of type double and one of type int and initialise each one with ten values (it doesn't matter what they are). Use the functions above to process these arrays – so call iterate() and pass in the appropriate function pointer. Next, print out the altered values in these arrays.

Question 3 – Dynamic Memory Allocation(5 + 25 + 10)

Consider a data structure to represent a pixel in an image as a number between 0 and 255 for red, green and blue. That will be defined by the struct pixel:

```
struct pixel
{
      char red;
      char green;
      char blue;
};
```

We will have a dynamic array of pixels for an image as follows:

```
struct image
{
      struct pixel ** pixels;
      unsigned width;
      unsigned height;
};
```

a) Write a function that creates a new image of the width and height specified (including allocating a 2d array pixels using malloc()) and initializes the red, green and blue elements of each pixel to 0. If there are any problems allocating memory, you should return `NULL` from this function. The function prototype for this function is:

```
struct image * create_image(unsigned width, unsigned height);
```

b) We want to be able to load in an 'image' from file. Now, instead of using a binary format that you are not familiar with, we will assume the following format for a text file that represents an image. The first line of the text file will have two comma separated values: the width and the height of the image. Each other line will have three comma separated values to represent the red, green and blue of each pixel respectively.

Write a function to load in such an image file. You should validate that there are two comma separated numbers on the first line and three comma separated numbers on every other line. You should also validate that all numbers to represent colors are between 0 and 255 and there are not more lines to represent pixels than was specified via the width and height at the start of the image. The pixel values should be assigned going from left to right across a row and then moving to the next row.

As part of this function you should call create_image() defined above to allocate space for the image to be loaded. You may assume that the infile parameter is valid and refers to a file that has just been opened prior to being passed in here. Your function should return the struct image pointer created and assigned. The function prototype is:

c) Write a main function that takes in as a command line argument the name of a file to open that is assumed to be in the format specified above. You should print out each row of red, green and blue values making it clear when a new row begins. You should also free any memory allocated and close the file that was opened by this function.

Question 4 – Software Design (10+10+5+5)

Consider a program to manage the collection of research papers that a student might read in preparation for writing a thesis. Each paper needs to store the following details (we'll stick to just academic papers as books and other source materials might need other data):

- Article title:
- Year published:
- Journal Name:
- Volume no:
- Issue no:
- Pages used:

We want to be able to load a text file with entries for each paper, edit the details about a paper, add papers, delete papers and save these papers back to disk.

a) make a list of at least 5 modules (.c and .h pairs) that you would create for this program. for each module, write down a sentence or two explaining its purpose and what would be contained in that module.

b) One of the modules created in the previous question would have been for managing a paper. Write down all the contents of the header file for that module.

c) How can we limit the visibility of a function so that it can only be run by other functions in the same file? Provide an example and explanation of what you are doing.

d) What is the purpose of the auto storage class? Does it make sense to ever explicitly use the auto keyword? Explain your answer.

Question 5 – Linked Lists (10+15+30)

Perry's Pretty Pastries is the name of a pie maker that sells pies direct to the public in the Central Business District of Melbourne.

Perry has asked you to construct a program for his company that will allow users to enter the products they would like to purchase from a preloaded list of products. Perry does not know much about these new-fangled computer programs but he has been told that a linked list would be a good way to store this data. Perry also wants to be able to store the total number of each type of pie that has been sold recently as well as the overall number of pies sold.

a) Provide a set of struct definitions that will allow Perry's requirements to be satisfied. You should pay attention in particular to good software design and the development of a modular solution. The only restriction is that the handle that refers to the list should be called pie_list.

b) Periodically, Perry wants the ability to reset the number of each type of pie sold to 0 as well as the global total of pies sold. Please write this function. The function prototype is:
```
void reset_number_sold(pie_list * list);
```

c) For reasons of reporting to the shareholders, Perry may want to sort the pie list according to the number sold with the highest selling pie at the top. Please implement this function. The function prototype is:
```
void sort_by_number_sold(pie_list * list);
```

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