Problem 1. Layer programming

1. The program can be divided into the kernel layer (bottom), the logic layer (middle), and the GUI layer (top).
2. The kernel layer includes `linkedlist.h` and `linkedlist.c`, the logic layer includes `logic/h` and `logic.c`, and the GUI layer includes `gui.h` and `gui.c`.
3. The header file of linkedlist is defined as follows

```c
//
// Created by grave on 2020/10/26.
//

#ifndef L5_LINKEDLIST_H
#define L5_LINKEDLIST_H

#include <stdio.h>

typedef struct node {
    char *str;
    void *data;
    struct node *next;
} Node;

typedef struct list {
    struct node *first;
    size_t length;
} List;

Node *list_insert(List *list, char *str, void *data);
void list_init(List **list);
void list_sort(List *list, int (*cmp)(const void *, const void *));
void list_print(const List *list, FILE *file, void(*print)(FILE* file,
↪   const void *));
void list_free(List *list);

#endif //L5_LINKEDLIST_H
```

The header file of the logic part is defined as follows

```c
//
// Created by grave on 2020/10/26.
//

#ifndef L5_LOGIC_H
#define L5_LOGIC_H

#include "linkedlist.h"

const char *DATA_TYPES[3] = {"int.txt", "double.txt", "string.txt"};
const char *SORT_TYPES[3] = {"inc", "dec", "rand"};
void sort(int i, int j);

#endif // L5_LOGIC_H
```

4. The related code is included in `linkedlist.c` and `logic.c`, there is no function call across the layer.

5. Since we've originally created a main function providing command line interface, this function is now divided into `oldmain`

```c
oldmain(int argc, char *argv[]){
    if (argc < 3)
        return 0;

    srand(time(NULL));
    char filename[20];
    int i = 0;
    int j = 0;

    for (i = 0; i < 3; ++i) {
        strcpy(filename, "rand");
        int len = strlen(filename);
        if (len){
            filename[len] = '_';
            strcpy(filename + len + 1, DATA_TYPES[i]);
            if (strcmp(filename, argv[1]) == 0) {
                break;
            }
        }
    }

    for (j = 0; j < 3; ++j) {
        if (strcmp(SORT_TYPES[j], argv[2]) == 0) {
            break;
        }
    }
    sort(i, j);
    return 0;
}
```

Now we implement a `newmain`, which feature a menu interface with user-friendly prompts. Notice that now user could freely have their choice of options to run this script for as many times as they want, they could exit anytime too!

```
newmain(int argc, char *argv[]){
    if (argc < 3)
        return 0;

    srand(time(NULL));
    char filename[20];
    char sin[100];
    int flag = 1;
        printf("****************************************************");
        printf("*************Welcome to Menu Interface!************");
        printf("****************************************************");

    while (flag)
    {
        printf("***************** DATA OPTIONS *****************\n");
        printf("*** A. rand_int.txt (read random integers) ******\n");
        printf("*** B. rand_double.txt (read random doubles) ****\n");
        printf("*** C. rand_string.txt (read random strings) ****\n");
        printf("*** D. exit (exit the program) *****************\n");
        printf("**************** DATA OPTIONS *****************\n");

        int i = 3;
        for (int m = 0; m < 100; ++m)
            sin[m] = '\0';

        while (i == 3)
        {
            for (i = 0; i < 3; ++i) {
                printf("> ");
                fgets(sin, 999, stdin);
                sin[strlen(sin) - 1] = '\0';
                if (!strcmp(sin, "exit")) {
                    flag = 0;
                    break;
                }

                strcpy(filename, "rand");
                int len = strlen(filename);
                if (len){
                    filename[len] = '_';
                    strcpy(filename + len + 1, DATA_TYPES[i]);
                    if (strcmp(filename, sin) == 0) {
                        break;
                    }
                }
            }
        }
        if (!flag) break;

        printf("**************** SORT OPTIONS *****************\n");
        printf("*** A. inc (output in increasing order) *********\n");
        printf("*** B. dec (output in decreasing order) *********\n");
        printf("*** C. rand (output in random order) ***********\n");
        printf("*** D. exit (exit the program) *****************\n");
        printf("***************** SORT OPTIONS ******************\n");

        int j = 3;
        for (int n = 0; n < 100; ++n)
            sin[n] = '\0';

        while (j == 3)
        {
            for (j = 0; j < 3; ++j) {
                printf("> ");
                fgets(sin, 999, stdin);
                sin[strlen(sin) - 1] = '\0';
                if (!strcmp(sin, "exit")) {
                    flag = 0;
                    break;
                }

                if (strcmp(SORT_TYPES[j], sin) == 0) {
                    break;
                }
            }
        }
        if (!flag) break;
        sort(i, j);
        printf("The sorting is completed successfully! Want to try
         ↪  again?(Y/N)\n");
        for (int l = 0; l < 100; ++l)
            sin[l] = '\0';
        printf("> ");
        fgets(sin, 999, stdin);
        sin[strlen(sin) - 1] = '\0';
        if (sin == 'N') {
            flag = 0;
            break;
        }
        if (!flag) break;
```

```
    }
    return 0;
}
```

Problem 2. Libraries

1. Pre-processing, compiling and linking.
2. (a) Pre-processing is the initial phase of a source code transformation, and the data for further analysis is produced.
   (b) Compiling translates the source code into assembly instructions, which are later transformed into machine language.
   (c) Linking process establishes a connection to the operating system for primitives, **here is where magic happens when we introduce static library(.a .lib) and dynamic library(.dll .so)**.
3. (a) Static libraries, while reusable in multiple programs, are locked into a program at compile time.
   (b) Dynamic libraries exist as separate files outside of the executable file.
4. To create your own library, you need to do the following
   (a) Compile your Library code(.c files) into an object code

```
$ gcc -c *.c
```

   (b) Create a Static Library from all ".o" files using command `ar`

```
$ ar -rc libyourlibname.a *.o
```

   This tells ar to create an archive (option c) and to insert the objects, replacing older files where needed (option r)

   (c) Create index for the library using ranlib command, it makes a header in the library with the symbols of the object file contents.This helps the compiler to quickly reference symbols.

```
$ ranlib libyourlibname.
```

(d) We have now created a static library yourlibraryname.a. Now let us use the static library by invoking it as part of the compilation and linking process when creating a program executable. Incase of gcc we use following flags to create static library

```
gcc main.c -L. -lyourlibname -o main
```

(e) Finally, run the executable program `main`

```
$./main
```

5. To create two static libraries, we just need to combine the above recipe.

```
## Libarary for linkedlist
gcc -c linkedlist.c
ar -cr liblinkedlist.a linkedlist.o
ls # now we have liblinkedlist.a

## Libarary for logic
gcc -c logic.c
ar -cr liblogic.a logic.o
ls # now we have liblogic.a
```

6. To compile with the library we just created, we simply use

```
gcc main.c -L. -llinkedlist -o main
./main

gcc main.c -L. -llogic -o main
./main
```

Or, to combine two library, we can use

```
ar x liblinkedlist.a
ar x liblogic.a
ar c libcombine.a  *.o
```

7. To create the dynamic library, we need to use compiler option `-fpic` and linker option `-shared`

```
gcc -fPIC -c linkedlist.c
gcc -shared -o liblinkedlist.so linkedlist.o

## The above two command can be combined into the following command
gcc -fPIC -shared -o liblinkedlist.so linkedlist.c

# It is the same for logic to create its dynamic library
gcc -fPIC -c logic.c
gcc -shared -o liblogic.so logic.o

## Or do the short version
gcc -fPIC -shared -o liblogic.so logic.c
```

8. Compile the file using dynamic library is basically the same as to do with the static version. Only notice that we need to move the copy of the dynamic library to `/lib` or `/usr/lib`, otherwise we need to put the absolute path into the file `/etc/ld.so.cache`.

```
gcc main.c -L. -llinkedlist -o main
./main

gcc main.c -L. -llogic -o main
./main
```

9. To compile the menu program, we just need to combine two dynamic library

```
gcc -o main gui.c -L. -llinkedlist -llogic
./main
```

10. The main difference is that the library refers to the code itself, while API refers to the interface. Or we can say API represents what is inside the library, since library usually is a large chunk of code designed to be reused. Somehow API can be consisted of multiple libraries while library cannot be an API itself.