/\*

The system consists of one server process and one client process. The client

process must be started by the server using an explicit call to fork()

followed by an appropriate form of exec. The server maintains a database

of student and course information. The client sends a sequence of commands

to the server, one command at a time. The server executes each command and sends

back a response to the client. (The client sends a command only after receiving

the response to the previous command.) The client stores the responses received

from the server in a log file. The only exception to this is the exit command.

By sending this command, the client informs the server that there are no more

commands and that the client will exit right after sending the command. So,

the server does not send a response to the exit command.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./p5b\_h\_define.h Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define S\_COMMAND\_ARG 5 //server arg

#define S\_INITDB\_FILE 1 //server

#define S\_FINALDB\_FILE 2 //server

#define S\_CMD\_FILE 3 //server

#define S\_LOG\_FILE 4 //server

#define C\_COMMAND\_ARG 3 //client arg

#define C\_CMD\_FILE 1 //client

#define C\_LOG\_FILE 2 //client

#define L\_SIZE 1000 //line size

#define C\_SIZE 10 //course size

#define CH\_SIZE 10

#define CMD\_SIZE 101 //read from command file

#define FIFO\_MODE 0666 //check

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./p5b\_h\_global.h Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node1 \*head1;

struct node2 \*head2;

struct node3 \*head3;

//create two pipes

char \*cfifo\_name = "COMMAND\_FIFO";

char \*rfifo\_name = "REPLY\_FIFO";

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./p5b\_h\_extern.h Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern struct node1 \*head1;

extern struct node2 \*head2;

extern struct node3 \*head3;

extern char \*cfifo\_name;

extern char \*rfifo\_name;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./p5b\_h\_prototypes.h Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void del\_newline(char[]);

void set\_null(char[]);

void line\_token(char[], struct database \*);

void insert\_list1(char \*, int, int[]);

void insert\_list2(int, int, char \*);

void print\_final(FILE \*);

void check\_cmd(char[], struct node3 \*);

void addc\_cmd(char[], struct node3 \*);

void drpc\_cmd(char[], struct node3 \*);

void wdrw\_cmd(char[], struct node3 \*);

void tcre\_cmd(char[], struct node3 \*);

void newc\_cmd(char[], struct node3 \*);

void csch\_cmd(char[], struct node3 \*);

void ccre\_cmd(char[], struct node3 \*);

void gsch\_cmd(char[], struct node3 \*);

void gcre\_cmd(char[], struct node3 \*);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./p5b\_h\_struct\_def.h Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Struct to store first part of data from init\_db.txt \*/

struct node1

{

char \*name;

int total\_course;

int course[C\_SIZE];

struct node1 \*next1;

};

/\* Struct to store 2nd part of data from init\_db.txt \*/

struct node2

{

int course2; //course name

int creadit;

char \*schedule;

struct node2 \*next2;

};

/\* For log.txt file \*/

struct node3

{

char cmd[10];

int test;

char error[1000];

};

/\* Keep database \*/

struct database

{

int flag1;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./p5b\_f\_server\_main.c Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

Main Method: Read initial database into a suitable data structure in memory.

than create and appropriately open the command and reply FIFOs. after that fork

the client proccess. Receive command from client while command does not equal to

exit. execute command wich will modify the data structre in memory. Last set is

to write modified database to the specified file and stop.

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include "p5b\_h\_define.h"

#include "p5b\_h\_struct\_def.h"

#include "p5b\_h\_global.h"

#include "p5b\_h\_prototype.h"

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

{

char line[L\_SIZE]; //read from init\_db\_file

/\* FIFO variables \*/

int fd1; //open pipe called COMMAND\_FIFO

int fd2; //open pipe called REPLY\_FIFO

int nread; //open pipe for reading

int nwrite; //open pipe for writeing

char cmd\_line[CMD\_SIZE]; //store commands

char temp\_cmd\_line[CMD\_SIZE]; //store temp commands

/\* Fork variables \*/

pid\_t child; //child pid return by fork

/\* Variable for opening files \*/

FILE \*finp1; //initial db file

FILE \*foutp2; //final db file

struct database buff1 = {0};

struct node3 s\_buff2 = {{'\0'}, 0 ,{'\0'}};

/\*\*\* Check command line \*\*\*/

if(argc != S\_COMMAND\_ARG)

{

fprintf(stderr, "Error - Enter 5 commands.\n");

exit(1);

}

/\*\*\* Open initial db file \*\*\*/

if((finp1 = fopen(argv[S\_INITDB\_FILE], "r")) == NULL)

{

fprintf(stderr, "Could not open file %s\n", argv[S\_INITDB\_FILE]);

exit(1);

}

/\*\*\* Open final db file \*\*\*/

if((foutp2 = fopen(argv[S\_FINALDB\_FILE], "w")) == NULL)

{

fprintf(stderr, "Could not open file %s\n", argv[S\_FINALDB\_FILE]);

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (1) create data structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while(fgets(line, L\_SIZE, finp1) != NULL) //read one line from initial db file

{

del\_newline(line); //remove \n

line\_token(line, &buff1); //create data struct

}

print\_list1(); //del later

print\_list2(); //del later

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (2.1) Create FIFO if it does not already exist \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(mkfifo(cfifo\_name, FIFO\_MODE) == -1) //create fifo

{

if(errno != EEXIST)

{

fprintf(stderr, "Server: Couldn't create FIFO.\n"); //error check

exit(1); //exit if fail

}

}

if(mkfifo(rfifo\_name, FIFO\_MODE) == -1) //create fifo

{

if(errno != EEXIST)

{

fprintf(stderr, "Server: Couldn't create FIFO.\n"); //error check

exit(1); //exit if fail

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (2.2) Open FIFO for read and write \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if((fd1 = open(cfifo\_name, O\_RDWR)) == -1) //open fifo

{

fprintf(stderr, "Server: FIFO open failed.\n"); //error check

exit(1); //exit if fail

}

/\*\*\* (2.2) Open FIFO for read and write \*\*\*/

if((fd2 = open(rfifo\_name, O\_RDWR)) == -1) //open fifo

{

fprintf(stderr, "Server: FIFO open failed.\n"); //error check

exit(1); //exit if fail

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (3) Fork the client process \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if((child = fork()) == 0) //for client

{

/\* child will now execute the client program \*/

execlp("p5b\_client", "p5b\_client", argv[S\_CMD\_FILE], argv[S\_LOG\_FILE], NULL);

/\* if the child proccess reaches this point, then execlp must failed \*/

fprintf(stderr, "Child process could not do execlp.\n");

exit(1);

}

else /\* Parent process \*/

{

if(child == (pid\_t)(-1))

{

fprintf(stderr, "Fork failed.\n"); //error

exit(1); //exit if fail

}

else

{

/\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* (4) \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*/

do

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (4.a) Receive command from client \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if((nread = read(fd1, cmd\_line, CMD\_SIZE)) == -1)

{

fprintf(stderr, "Server: Reading from fifo failed.\n");

exit(1);

}

else

{

printf("Received message: %s\n", cmd\_line);

strcpy(temp\_cmd\_line, cmd\_line);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (4.b) if(command != exit) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(strcmp(temp\_cmd\_line, "exit") != 0)

{

//check what cmd they enter

check\_cmd(temp\_cmd\_line, &s\_buff2);

if((nwrite = write(fd2, &s\_buff2, sizeof(struct node3))) == -1)

{

fprintf(stderr, "Server: write to fifo failed.\n");

exit(1);

}

}

}

}while(strcmp(cmd\_line, "exit") != 0);

}

}

/\*\* change data struct \*\*\*/

print\_list1();

print\_list2();

print\_list3();

print\_final(foutp2);

/\*\*\* Close final db file \*\*\*/

if(fclose(foutp2) == EOF)

{

fprintf(stderr, "Error in closing file %s", argv[S\_FINALDB\_FILE]);

}

/\*\*\* Close inital db file \*\*\*/

if(fclose(finp1) == EOF)

{

fprintf(stderr, "Error in closing file %s", argv[S\_INITDB\_FILE]);

//EXit program

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./p5b\_f\_client\_main.c Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

Main Mehod: 1st open the command file and log file. Than open command and reply

FIFO's appropriately. While there are commands in the command file. Get the next

command and send the command to the server. Recive reply from sserver and wrtie

an appropriate entry into the log files. Lastly send exit command to server and

stop

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/stat.h>

#include "p5b\_h\_define.h"

#include "p5b\_h\_struct\_def.h"

#include "p5b\_h\_global.h"

#include "p5b\_h\_prototype.h"

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

{

int counter = 0;

char cmd\_line[CMD\_SIZE];

/\* FIFO variables \*/

int fd1; //create pipe called COMMAND\_FIFO(write)

int fd2; //create pipe callied RESPONSE\_FIFO(read)

int nread; //read from command)fifo

int nwrite; //write from replay\_FIFO

/\* File variables \*/

FILE \*finp1; //command file

FILE \*foutp2; //log file

struct node3 c\_buff2 = {{'\0'}, 0, {'\0'}};

/\*\*\* Check command line \*\*\*/

if(argc != C\_COMMAND\_ARG)

{

fprintf(stderr, "Error - Enter three commands.\n");

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (1) Open the command file and the log file \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* Open command file \*\*\*/

if((finp1 = fopen(argv[C\_CMD\_FILE], "r")) == NULL)

{

fprintf(stderr, "Could not open file: %s\n", argv[C\_CMD\_FILE]);

exit(1);

}

/\*\*\* Open log file \*\*\*/

if((foutp2 = fopen(argv[C\_LOG\_FILE], "w")) == NULL)

{

fprintf(stderr, "Could not open file: %s\n.", argv[C\_LOG\_FILE]);

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (2) Open FIFO for read and write \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* Open FIFO for reading \*\*\*/

if((fd1 = open(cfifo\_name, O\_RDWR)) == -1)

{

fprintf(stderr, "Client: FIFO open failed.\n");

exit(1);

}

if((fd2 = open(rfifo\_name, O\_RDWR)) == -1)

{

fprintf(stderr, "Client: FIFO open failed.\n");

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (3) while (there are commands in the command file) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while(fgets(cmd\_line, CMD\_SIZE, finp1) != NULL)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (3.b) Send the command to the server \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if((nwrite = write(fd1, cmd\_line, CMD\_SIZE)) == -1)

{

fprintf(stderr, "Client: Write to fifo failed.\n");

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (3.c) Receive realy from server \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if((nread = read(fd2, &c\_buff2, sizeof(struct node3))) == -1)

{

fprintf(stderr, "client: read to fifo failed.\n");

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (3.d). Write an appropriate entry into the log files \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

fprintf(foutp2, "%d %s %d %s\n",

counter, c\_buff2.cmd, c\_buff2.test, c\_buff2.error);

counter++;

}//end of while loop

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* (4) Send exit command to server and stop \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

strcpy(cmd\_line,"exit");

if((nwrite = write(fd1, cmd\_line, CMD\_SIZE)) == -1)

{

fprintf(stderr, "Client: write to fifo failed.\n"); //error here

exit(1);

}

/\*\*\* Close log file \*\*\*/

if(fclose(foutp2) == EOF)

{

fprintf(stderr, "Could not close file %s\n", argv[C\_LOG\_FILE]);

}

/\*\*\* close Command file \*\*\*/

if(fclose(finp1) == EOF)

{

fprintf(stderr, "Could not close file %s\n", argv[C\_CMD\_FILE]);

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./p5b\_f\_funct\_main.c Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include "p5b\_h\_define.h"

#include "p5b\_h\_struct\_def.h"

#include "p5b\_h\_extern.h"

#include "p5b\_h\_prototype.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* replace '\n' with '\0' \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void del\_newline(char line[])

{

int i = 0;

for(i = 0; i <= L\_SIZE; i++)

{

if(line[i] == '\n')

{

line[i] = '\0';

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* set char array to '\0' \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void set\_null(char temp[])

{

int i = 0;

for(i = 0; i <= L\_SIZE; i++)

{

temp[i] = '\0';

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* set int array to 0 - int course[15]\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void set\_zero(int temp[])

{

int i = 0;

for(i = 0; i < C\_SIZE; i++)

{

temp[i] = 0;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SERVER \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void line\_token(char line[], struct database \*buff1)

{

char \*tptr; //token for linked list 1

char \*tptr2; //token2 for linked list 2

char \*error; //converting

int num = 0; //converting

int num2 = 0; //converting

char \*name; //struct 1

int total\_course; //struct 1

int course[C\_SIZE]; //struct 1

int course2; //struct 2

int creadit; //struct 2

char \*schedule; //struct 2

int i = 0; //index

if(buff1->flag1 == 1)

{

set\_zero(course); //put all zero in array

i = 0;

tptr = strtok(line, " \t\n"); //NAME

if(tptr != NULL)

{

if((strstr(tptr, "0") != 0) || (strstr(tptr, "1") != 0) ||

(strstr(tptr, "2") != 0) || (strstr(tptr, "3") != 0) ||

(strstr(tptr, "4") != 0) || (strstr(tptr, "5") != 0) ||

(strstr(tptr, "6") != 0) || (strstr(tptr, "7") != 0) ||

(strstr(tptr, "8") != 0) || (strstr(tptr, "9") != 0))

{

buff1->flag1 = 2;

return; //exit funtion

}

name = tptr; //copy value in to name

}

tptr = strtok(NULL, " \t\n"); //total number of courses

if(tptr != NULL)

{

num = strtod(tptr, &error);

total\_course = num; //copy value into total\_course

}

tptr = strtok(NULL, " \t\n"); //all courses

while(tptr != NULL)

{

num = strtod(tptr, &error);

course[i] = num;

i++;

tptr = strtok(NULL, " \t\n");

}

insert\_list1(name, total\_course, course);

}

else if(buff1->flag1 == 0) //skip 1st line

{

buff1->flag1 = 1;

}

else if(buff1->flag1 == 2) //insert in 2nd linked list

{

tptr2 = strtok(line, " \t\n");

if(tptr2 != NULL)

{

num = strtod(tptr2, &error);

course2 = num;

}

tptr2 = strtok(NULL, " \t\n");

if(tptr2 != NULL)

{

num2 = strtod(tptr2, &error);

creadit = num2;

}

tptr2 = strtok(NULL, " \t\n");

if(tptr2 != NULL)

{

schedule = tptr2;

}

insert\_list2(course2, creadit, schedule);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* Insert in linked list 1 \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void insert\_list1(char \*name, int total\_course, int course[])

{

struct node1 \*cur\_node1 = head1;

struct node1 \*new\_node1;

int i = 0;

if(cur\_node1 == NULL)

{

head1 = (struct node1 \*) malloc(sizeof(struct node1));

if(head1 == NULL)

{

printf("Node allocation failed.\n"); fflush(stdout);

exit(1);

}

head1->name = strdup(name);

head1->total\_course = total\_course;

for(i = 0; i < C\_SIZE; i++)

{

head1->course[i] = course[i];

}

head1->next1 = NULL;

}

else

{

while(cur\_node1->next1 != NULL)

{

cur\_node1 = cur\_node1->next1;

}

new\_node1 = (struct node1 \*)malloc(sizeof(struct node1));

if(new\_node1 == NULL)

{

printf("Node allocation failed.\n"); fflush(stdout);

exit(1);

}

new\_node1->name = strdup(name);

new\_node1->total\_course = total\_course;

for(i = 0; i < C\_SIZE; i++)

{

new\_node1->course[i] = course[i];

}

new\_node1->next1 = NULL;

cur\_node1->next1 = new\_node1;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* Insert in linked list 2 \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void insert\_list2(int course2, int creadit, char \*schedule)

{

struct node2 \*cur\_node2 = head2;

struct node2 \*new\_node2;

if(cur\_node2 == NULL)

{

head2 = (struct node2 \*) malloc(sizeof(struct node2));

if(head2 == NULL)

{

printf("Node allocation failed.\n"); fflush(stdout);

exit(1);

}

head2->course2 = course2;

head2->creadit = creadit;

head2->schedule = strdup(schedule);

head2->next2 = NULL;

}

else

{

while(cur\_node2->next2 != NULL)

{

cur\_node2 = cur\_node2->next2;

}

new\_node2 = (struct node2 \*)malloc(sizeof(struct node2));

if(new\_node2 == NULL)

{

printf("Node allocation failed.\n"); fflush(stdout);

exit(1);

}

new\_node2->course2 = course2;

new\_node2->creadit = creadit;

new\_node2->schedule = strdup(schedule);

new\_node2->next2 = NULL;

cur\_node2->next2 = new\_node2;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* print final part 1 \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_final(FILE \*foutp2)

{

struct node1 \*cur\_node1 = head1;

struct node2 \*cur\_node2 = head2;

int i = 0;

int count1 = 0;

int count2 = 0;

if(cur\_node1 == NULL)

{

}

else

{

while(cur\_node1 != NULL)

{

count1++;

cur\_node1 = cur\_node1->next1;

}

}

if(cur\_node2 == NULL)

{

}

else

{

while(cur\_node2 != NULL)

{

count2++;

cur\_node2 = cur\_node2->next2;

}

}

/\*\*\* print part 1 \*\*\*/

cur\_node1 = head1;

fprintf(foutp2, "%d\n", count1);

cur\_node1 = head1;

while(cur\_node1 != NULL)

{

fprintf(foutp2, "%s %d ",

cur\_node1->name, cur\_node1->total\_course);

for(i = 0; i < C\_SIZE; i++)

{

if(cur\_node1->course[i] == 0)

{

cur\_node1->course[i] = '\0';

}

fprintf(foutp2, "%d ",cur\_node1->course[i]);

}

fprintf(foutp2, "\n");fflush(stdout);

cur\_node1 = cur\_node1->next1;

}

/\*\*\* print part 2\*\*\*/

fprintf(foutp2, "%d\n", count2);

cur\_node2 = head2;

while(cur\_node2 != NULL)

{

fprintf(foutp2, "%d %d %s\n",

cur\_node2->course2, cur\_node2->creadit, cur\_node2->schedule);

cur\_node2 = cur\_node2->next2;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\* check command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void check\_cmd(char temp\_cmd\_line[], struct node3 \*s\_buff2)

{

char temp\_cmd\_line2[CMD\_SIZE];

char \*tptr;

char \*check\_er = "";

strcpy(temp\_cmd\_line2, temp\_cmd\_line);

tptr = strtok(temp\_cmd\_line, " \t\n");

if(tptr != NULL)

{

if(strcmp(tptr, "addc") == 0)

{

addc\_cmd(temp\_cmd\_line2, s\_buff2);

}

else if(strcmp(tptr, "drpc") == 0)

{

drpc\_cmd(temp\_cmd\_line2, s\_buff2);

}

else if(strcmp(tptr, "wdrw") == 0)

{

wdrw\_cmd(temp\_cmd\_line2, s\_buff2);

}

else if(strcmp(tptr, "tcre") == 0)

{

tcre\_cmd(temp\_cmd\_line2, s\_buff2);

}

else if(strcmp(tptr, "newc") == 0)

{

newc\_cmd(temp\_cmd\_line2, s\_buff2);

}

else if(strcmp(tptr, "csch") == 0)

{

csch\_cmd(temp\_cmd\_line2, s\_buff2);

}

else if(strcmp(tptr, "ccre") == 0)

{

ccre\_cmd(temp\_cmd\_line2, s\_buff2);

}

else if(strcmp(tptr, "gsch") == 0)

{

gsch\_cmd(temp\_cmd\_line2, s\_buff2);

}

else if(strcmp(tptr, "gcre") == 0)

{

gcre\_cmd(temp\_cmd\_line2, s\_buff2);

}

else

{

printf("\nT:%s\n",tptr);

strcpy(s\_buff2->cmd, tptr);

s\_buff2->test = 0;

strcpy(s\_buff2->error, check\_er);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* addc commands \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void addc\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

int i = 0;

int x = 0; //check for error

int t = 0; //cmd fail or succuss

char \*tptr;

char \*addc\_1st; //cmd

char \*addc\_2nd; //name

int addc\_3rd; //course number

char \*addc\_er = "";

char \*error;

int num = 0;

int in\_list = 0;

struct node1 \*cur\_node1 = head1;

struct node1 \*new\_node1;

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

addc\_1st = strdup(tptr);

}

tptr = strtok(NULL, " \t\n");

if(tptr != NULL)

{

addc\_2nd = strdup(tptr);

}

tptr = strtok(NULL, " \t\n");

if(tptr != NULL)

{

num = strtod(tptr, &error);

addc\_3rd = num;

}

/\*\*\*\*\* change data struct \*\*\*\*\*/

if(cur\_node1 == NULL)

{

}

else

{

while(cur\_node1 != NULL)

{

if(strcmp(cur\_node1->name, addc\_2nd) == 0)

{

cur\_node1->total\_course++;

if(cur\_node1->total\_course >= 11)

{

cur\_node1->total\_course--;

strcpy(s\_buff2->cmd, addc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, addc\_er);

in\_list = 1;

}

else //add in list

{

for(i = 0; i < 10; i++)

{

if(cur\_node1->course[i] == 0)

{

cur\_node1->course[i] = addc\_3rd;

in\_list = 1;

strcpy(s\_buff2->cmd, addc\_1st);

s\_buff2->test = 1;

strcpy(s\_buff2->error, addc\_er);

break;

}

if(cur\_node1->course[i] == addc\_3rd) //course already in list

{

cur\_node1->total\_course--;

in\_list = 1;

strcpy(s\_buff2->cmd, addc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, addc\_er);

break;

}

}//end of for loop

}

}

x++;

cur\_node1 = cur\_node1->next1;

}

}

if(in\_list == 0)

{

if(x < 100)

{

cur\_node1 = head1;

while(cur\_node1->next1 != NULL)

{

cur\_node1 = cur\_node1->next1;

}

new\_node1 = (struct node1 \*) malloc(sizeof(struct node1));

if(new\_node1 == NULL)

{

printf("Node allocation failed.\n"); fflush(stdout);

exit(1);

}

new\_node1->name = strdup(addc\_2nd);

new\_node1->total\_course = 1;

new\_node1->course[0] = addc\_3rd;

new\_node1->next1 = NULL;

cur\_node1->next1 = new\_node1;

strcpy(s\_buff2->cmd, addc\_1st);

s\_buff2->test = 1;

strcpy(s\_buff2->error, addc\_er);

insert\_list3(addc\_1st, t, addc\_er);

}

else

{

strcpy(s\_buff2->cmd, addc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, addc\_er);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* drpc command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void drpc\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

int i = 0;

char \*tptr;

int flag = 0;

int in\_list = 0;

char \*drpc\_1st;

char \*drpc\_2nd; //name

int drpc\_3rd; //course number

char \*drpc\_er = ""; //error

char \*error;

int num = 0;

struct node1 \*cur\_node1 = head1;

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

drpc\_1st = tptr;

}

tptr = strtok(NULL, " \t\n"); //2nd word

if(tptr != NULL)

{

drpc\_2nd = strdup(tptr);

}

tptr = strtok(NULL, " \t\n"); //3rd word

if(tptr != NULL)

{

num = strtod(tptr, &error);

drpc\_3rd = num;

}

/\*\*\*\*\* change data struct \*\*\*\*\*/

if(cur\_node1 == NULL)

{

}

else

{

while(cur\_node1 != NULL)

{

if(strcmp(cur\_node1->name, drpc\_2nd) == 0)

{

cur\_node1->total\_course--;

if(cur\_node1->total\_course <= 0)

{

cur\_node1->name = NULL;

cur\_node1->total\_course = 0;

cur\_node1->course[0] = 0;

//error

in\_list = 1;

strcpy(s\_buff2->cmd, drpc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, drpc\_er);

}

else

{

for(i = 0; i < 10; i++)

{

if(cur\_node1->course[i] == drpc\_3rd)

{

cur\_node1->course[i] = 0;

in\_list = 1;

flag = 1;

strcpy(s\_buff2->cmd, drpc\_1st);

s\_buff2->test = 1;

strcpy(s\_buff2->error, drpc\_er);

break;

}

}//end of for loop

if(flag == 0)

{

cur\_node1->total\_course++;

in\_list = 1;

strcpy(s\_buff2->cmd, drpc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, drpc\_er);

}

}

}

cur\_node1 = cur\_node1->next1;

}//end of while loop

if(in\_list == 0)

{

strcpy(s\_buff2->cmd, drpc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, drpc\_er);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* wdrw command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void wdrw\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

int i = 0;

char \*tptr;

char \*wdrw\_1st; //cmd

char \*wdrw\_2nd; //name

char \*wdrw\_er = "";

int in\_list = 0;

struct node1 \*cur\_node1 = head1;

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

wdrw\_1st = strdup(tptr);

}

tptr = strtok(NULL, " \t\n");

if(tptr != NULL)

{

wdrw\_2nd = strdup(tptr);

}

/\*\*\*\*\* change data struct \*\*\*\*\*/

if(cur\_node1 == NULL)

{

}

else

{

while(cur\_node1 != NULL)

{

if(strcmp(cur\_node1->name, wdrw\_2nd) == 0)

{

cur\_node1->total\_course = 0;

for(i = 0; i < 10; i++)

{

cur\_node1->course[i] = 0;

}//end of for loop

in\_list = 1;

strcpy(s\_buff2->cmd, wdrw\_1st);

s\_buff2->test = 1;

strcpy(s\_buff2->error, wdrw\_er);

}

cur\_node1 = cur\_node1->next1;

}//end of while loop

}

if(in\_list == 0)

{

strcpy(s\_buff2->cmd, wdrw\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, wdrw\_er);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* tcre command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void tcre\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

int i = 0;

char \*tptr;

char \*tcre\_1st; //cmd

char \*tcre\_2nd; //name

char \*tcre\_er;

char \*e = "-1";

int total = 0;

char \*ch\_total;

int flag = 0;

int flag3 = 0;

int in\_list = 0;

struct node1 \*cur\_node1 = head1;

struct node2 \*cur\_node2 = head2;

if((ch\_total = (char \*) malloc(10 \*sizeof(char))) == NULL)

{

printf("storage allocation failure.\n");

exit(1);

}

if((tcre\_er = (char \*) malloc(10 \*sizeof(char))) == NULL)

{

printf("storage allocation failure.\n");

exit(1);

}

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

tcre\_1st = strdup(tptr);

}

tptr = strtok(NULL, " \t\n");

if(tptr != NULL)

{

tcre\_2nd = strdup(tptr);

}

/\*\*\*\*\* change data struct \*\*\*\*\*/

if(cur\_node1 == NULL)

{

}

else

{

while(cur\_node1 != NULL)

{

if(strcmp(cur\_node1->name, tcre\_2nd) == 0)

{

if(cur\_node1->total\_course >= 0)

{

in\_list = 1;

break;

}

}

cur\_node1 = cur\_node1->next1;

}//end of while loop

}

if(in\_list == 0)

{

flag = 1;

strcpy(s\_buff2->cmd, tcre\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, e);

}

if(flag == 0)

{

if(cur\_node2 == NULL)

{

}

else

{

i = 0;

while(cur\_node2 != NULL)

{

if(cur\_node2->course2 == cur\_node1->course[i])

{

if(i < 10)

{

total = total + cur\_node2->creadit;

i++;

flag3 = 1;

}

}

flag3 = 0;

cur\_node2 = cur\_node2->next2;

if(flag3 == 1)

{

cur\_node2 = head2;

}

}//end of while loop

}

strcpy(s\_buff2->cmd, tcre\_1st);

s\_buff2->test = 1;

sprintf(ch\_total,"%d",total);

strcpy(s\_buff2->error, ch\_total);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* newc command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void newc\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

int x = 0;

char \*tptr;

int in\_list = 0;

char \*newc\_1st; //cmd

int newc\_2nd; //course number

int newc\_3rd; //credit number

char \*newc\_4th; //schedule

char \*newc\_er = ""; //error

char \*error;

struct node2 \*cur\_node2 = head2;

struct node2 \*new\_node2;

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

newc\_1st = strdup(tptr);

}

tptr = strtok(NULL, " \t\n"); //2nd word

if(tptr != NULL)

{

newc\_2nd = strtod(tptr, &error);

}

tptr = strtok(NULL, " \t\n"); //3rd word

if(tptr != NULL)

{

newc\_3rd = strtod(tptr, &error);

}

tptr = strtok(NULL, " \t\n"); //4th word

if(tptr != NULL)

{

newc\_4th = strdup(tptr);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\* change data struct \*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(cur\_node2 == NULL)

{

}

else

{

if(newc\_3rd > 0)

{

while(cur\_node2 != NULL)

{

if(cur\_node2->course2 == newc\_2nd)

{

in\_list = 1;

strcpy(s\_buff2->cmd, newc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, newc\_er);

}

x++;

cur\_node2 = cur\_node2->next2;

}//end of while loop

}

else

{

in\_list = 1;

strcpy(s\_buff2->cmd, newc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, newc\_er);

}

if(in\_list == 0)

{

if(x < 100)

{

cur\_node2 = head2;

while(cur\_node2->next2 != NULL)

{

cur\_node2 = cur\_node2->next2;

}

new\_node2 = (struct node2 \*) malloc(sizeof(struct node2));

if(new\_node2 == NULL)

{

printf("Node allocation failed.\n"); fflush(stdout);

exit(1);

}

new\_node2->course2 = newc\_2nd;

new\_node2->creadit = newc\_3rd;

new\_node2->schedule = strdup(newc\_4th);

new\_node2->next2 = NULL;

cur\_node2->next2 = new\_node2;

strcpy(s\_buff2->cmd, newc\_1st);

s\_buff2->test = 1;

strcpy(s\_buff2->error, newc\_er);

}

else

{

strcpy(s\_buff2->cmd, newc\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, newc\_er);

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* csch command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void csch\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

char \*tptr;

int flag = 0;

int in\_list = 0;

char \*csch\_1st; //cmd

int csch\_2nd; //course number

char \*csch\_3rd; //schedule

char \*csch\_er = ""; //error

char \*error;

struct node2 \*cur\_node2 = head2;

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

csch\_1st = strdup(tptr);

}

tptr = strtok(NULL, " \t\n"); //2nd word

if(tptr != NULL)

{

csch\_2nd = strtod(tptr, &error);

}

tptr = strtok(NULL, " \t\n"); //3rd word

if(tptr != NULL)

{

csch\_3rd = strdup(tptr);

}

/\*\*\*\*\* change data struct \*\*\*\*\*/

if(cur\_node2 == NULL)

{

}

else

{

while(cur\_node2 != NULL)

{

if(cur\_node2->course2 == csch\_2nd)

{

cur\_node2->schedule = strdup(csch\_3rd);

in\_list = 1;

flag = 1;

strcpy(s\_buff2->cmd, csch\_1st);

s\_buff2->test = 1;

strcpy(s\_buff2->error, csch\_er);

}

cur\_node2 = cur\_node2->next2;

}//end of while loop

if(flag == 0)

{

in\_list = 1;

strcpy(s\_buff2->cmd, csch\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, csch\_er);

}

if(in\_list == 0)

{

strcpy(s\_buff2->cmd, csch\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, csch\_er);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* ccre command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void ccre\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

char \*tptr;

int in\_list = 0;

char \*ccre\_1st; //cmd

int ccre\_2nd; //course number

int ccre\_3rd; //schedule

char \*ccre\_er = ""; //error

char \*error;

struct node2 \*cur\_node2 = head2;

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

ccre\_1st = strdup(tptr);

}

tptr = strtok(NULL, " \t\n"); //2nd word

if(tptr != NULL)

{

ccre\_2nd = strtod(tptr, &error);

}

tptr = strtok(NULL, " \t\n"); //3rd word

if(tptr != NULL)

{

ccre\_3rd = strtod(tptr, &error);

}

/\*\*\*\*\* change data struct \*\*\*\*\*/

if(cur\_node2 == NULL)

{

}

else

{

if(ccre\_3rd > 0)

{

while(cur\_node2 != NULL)

{

if(cur\_node2->course2 == ccre\_2nd)

{

cur\_node2->creadit = ccre\_3rd;

in\_list = 1;

strcpy(s\_buff2->cmd, ccre\_1st);

s\_buff2->test = 1;

strcpy(s\_buff2->error, ccre\_er);

}

cur\_node2 = cur\_node2->next2;

}//end of while loop

}

else

{

in\_list = 1;

strcpy(s\_buff2->cmd, ccre\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, ccre\_er);

}

if(in\_list == 0)

{

strcpy(s\_buff2->cmd, ccre\_1st);

s\_buff2->test = 0;

strcpy(s\_buff2->error, ccre\_er);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* gsch command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void gsch\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

char \*tptr;

int in\_list = 0;

char \*gsch\_1st; //cmd

int gsch\_2nd; //course number

char \*gsch\_er = ""; //error

char \*error;

struct node2 \*cur\_node2 = head2;

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

gsch\_1st = strdup(tptr);

}

tptr = strtok(NULL, " \t\n"); //2nd word

if(tptr != NULL)

{

gsch\_2nd = strtod(tptr, &error);

}

/\*\*\*\*\* change data struct \*\*\*\*\*/

if(cur\_node2 == NULL)

{

}

else

{

while(cur\_node2 != NULL)

{

if(cur\_node2->course2 == gsch\_2nd)

{

in\_list = 1;

strcpy(s\_buff2->cmd, gsch\_1st);

s\_buff2->test = 1;

gsch\_er = strdup(cur\_node2->schedule);

strcpy(s\_buff2->error, gsch\_er);

}

cur\_node2 = cur\_node2->next2;

}//end of while loop

if(in\_list == 0)

{

strcpy(s\_buff2->cmd, gsch\_1st);

s\_buff2->test = 0;

gsch\_er = "Error";

strcpy(s\_buff2->error, gsch\_er);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* gcre command \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void gcre\_cmd(char temp\_cmd\_line2[], struct node3 \*s\_buff2)

{

char \*tptr;

int in\_list = 0;

char \*gcre\_1st; //cmd

int gcre\_2nd; //course number

char \*gcre\_er; //error

int e = -1;

char \*error;

struct node2 \*cur\_node2 = head2;

if((gcre\_er = (char \*) malloc(10 \*sizeof(char))) == NULL)

{

printf("storage allocation failure.\n");

exit(1);

}

tptr = strtok(temp\_cmd\_line2, " \t\n"); //1st word

if(tptr != NULL)

{

gcre\_1st = strdup(tptr);

}

tptr = strtok(NULL, " \t\n"); //2nd word

if(tptr != NULL)

{

gcre\_2nd = strtod(tptr, &error);

}

/\*\*\*\*\* change data struct \*\*\*\*\*/

if(cur\_node2 == NULL)

{

}

else

{

while(cur\_node2 != NULL)

{

if(cur\_node2->course2 == gcre\_2nd)

{

in\_list = 1;

strcpy(s\_buff2->cmd, gcre\_1st);

s\_buff2->test = 1;

sprintf(gcre\_er, "%d", cur\_node2->creadit);

strcpy(s\_buff2->error, gcre\_er);

}

cur\_node2 = cur\_node2->next2;

}//end of while loop

}

if(in\_list == 0)

{

strcpy(s\_buff2->cmd, gcre\_1st);

s\_buff2->test = 0;

sprintf(gcre\_er, "%d", e);

strcpy(s\_buff2->error, gcre\_er);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File ./makefile Starts Here:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#The following rule tells make about possible suffixes

#(extensions) of file names.

.SUFFIXES: .c .o

#The following definition of CC ensures that

#gcc will be used to compile the C source files.

CC = gcc

#The following definition of CFLAGS ensures that

#the debugger can be used with the executable file (sample)

#created by running make.

CFLAGS = -g

#The following rule tells make how a ".o" file should

#be created from the corresponding ".c" file.

#Note that the "-c" option must be used here since we are

#compiling source files separately. (Note that the line

#following the ".c.o:" line begins with the "tab" character.)

.c.o:

$(CC) $(CFLAGS) -c $<

#Dependency rule for the default target and how the

#default target is to be created. (Note that the line

#following the dependency rule begins with the "tab"

#character.)

all: p5a p5b\_server p5b\_client

p5a: p5a\_f\_main.o

gcc p5a\_f\_main.o -o p5a

p5b\_server: p5b\_f\_server\_main.o p5b\_f\_funct.o

gcc p5b\_f\_server\_main.o p5b\_f\_funct.o -o p5b\_server

p5b\_client: p5b\_f\_client\_main.o p5b\_f\_funct.o

gcc p5b\_f\_client\_main.o p5b\_f\_funct.o -o p5b\_client

#Dependency rules for other targets. (We don't need to

#specify how these targets are created since we have already

#given a general rule for creating a ".o" file from the

#corresponding ".c" file.)

p5b\_f\_server\_main.o: p5b\_h\_define.h p5b\_h\_struct\_def.h p5b\_h\_global.h p5b\_h\_prototype.h

p5b\_f\_client\_main.o: p5b\_h\_define.h p5b\_h\_struct\_def.h p5b\_h\_global.h p5b\_h\_prototype.h

p5b\_f\_funct.o: p5b\_h\_define.h p5b\_h\_struct\_def.h p5b\_h\_extern.h p5b\_h\_prototype.h

#Target for removing unnecessary files.

clean:

rm -f \*.o core