Question 1. [8 MARKS]

Write a bash script that requires at least three arguments. The first argument is a pattern string `p`, the second is an integer `n` and the remainder are directories. The script must print to standard output the names of regular files in those directories that contain the string `p` and are fewer than `n` blocks. It should print a message and exit if called with fewer than 3 arguments. Note that calling the command `du` prints the size of the file in blocks and the filename separated by whitespace.

```
if [ $# -lt 3 ] ; then
  echo Usage: pattern n dir1 [dir2 ...]
  exit
fi
target=$1
lines=$2
shift
shift
for dir in "$@" ; do
  for file in $dir/* ; do
     if grep $target $file >> /dev/null ; then
       lines_here=‘wc -l <$file‘
       if [ $lines_here -lt $lines ] ; then
         echo $file
       fi
     fi
  done
done
```

Question 2. [8 MARKS]

Write a C program that takes at most two arguments. The first argument (which is required) is a string `p`. The second argument is a path to a directory `d`. It is optional. The program should list all the files in directory `d` whose names contain the string `p`. If the program is called with an invalid number of arguments, it should print a usage message to standard error and exit. If the optional directory is not provided, it should use the current working directory.

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

int main(int argc, char ** argv) {

  if (argc < 2 || argc > 3 ) {
     fprintf(stderr,"Usage: findfiles directory pattern\n");
     exit(1);
  }

  char * dirname;
  char def[2] = ".";

  if (argc == 2) {
    dirname = def;
  } else {
    dirname = argv[2];
  }

  DIR * dp;
  struct dirent * entry;

  if ((dp = opendir(dirname)) == NULL) {
    perror("opendir");
    exit(1);
  }
  while ((entry = readdir(dp)) != NULL) {
    if (strstr(entry->d_name,argv[1]) != NULL) {
      printf("%s\n",entry->d_name);
    }
  }
  return 0;
}

MARKING POSSIBLE GUIDELINE:

  1 for check of invalid num of args
  1 for Usage message to stderr
  1 for setting default to "." correctly
  1 for opening directory  (1/2 for checking return code)
  1 for reading entries until == NULL
  1 for using argv[1]  and argv[2] correctly
  1 for using dirent and entry->d_name
  1 for strstr != NULL or writing compare
```

Question 3. [6 MARKS]

Consider the following struct and type definition.

```
struct studentNode{
    char name[20];
    char university[30];
    struct studentNode *next;
};

typedef struct studentNode NN;
```

Complete the function `find_school` so that it creates and returns a string holding the university for the first student with the given name from the unsorted linked list. The function should allocate only enough space to hold the name of the school. The function should return `NULL` if there is no such student in the list.

```
/*
 * Create a new string set equal to the school name of the
 * first student with name st_name in list st_list.
 * If there is no such student, return NULL.
 * Do not change the list.
 *
 */
char * find_school(NN * st_list, char * st_name) {

  while ((st_list != NULL) && (strcmp(st_list->name,st_name) != 0)) {
    st_list = st_list->next;
  }
  if (st_list == NULL) {
    return NULL;
  } else {
    char * result;
    if ((result = malloc(strlen(st_list->university) + 1)) == NULL) {
      perror("malloc");
      exit(1);
    }
    strncpy(result,st_list->university,(strlen(st_list->university) + 1));
    return result;
  }
}
```

Question 4. [8 MARKS]

Part (a) [5 MARKS]

In the next question, there will be a lot of pipes to close. Write a wrapper function called `Close` that has the string `msg` as one of its formal parameters. Within your function, call the `close` system call and perform error checking. If you encounter an error, print a message to standard error that includes the string "Close:" appended with the string from `msg` along with the standard message associated with `errno`.

```
void Close(char * msg, int fd) {
  if (close(fd) != 0) {
    int len = strlen(msg);
    char fullmsg[len+8];
    strncpy(fullmsg,"close: ",7);
    strncpy(msg,fullmsg[7],len);
    perror(msg);
    exit(1);
  }
}

MARKING: 1 perror, 1 for close != 0, 1 for enough space for fullmsg
         1 for strncpy or strncat, 1 for max number to copy
```

Part (b) [1 MARK]

Explain why it is important to close the writing end of a pipe in a process which is going to use the pipe for reading. SOLUTION: If a pipe to which you are writing doesn't close then the process at the other end may end up blocking waiting for more input when no more is coming.

Part (c) [1 MARK]

In assignment three, some solutions waited for the map processes to complete before reading from any of the pipes in the reduce processes. This appeared to work on some test cases but failed on others. Explain why. SOLUTION: Pipes have a finite capacity. In the larger test cases the pipe was full and so the writes to the pipe were blocking waiting for the other process to read.

Question 5. [10 MARKS]

The C program below is intended to create 5 child processes, send each of them the message "hello" using a pipe and read an integer returned from each child. Fill in the missing parts of the program so that it works correctly according to the comments in the code. Assume that the necessary includes and headerfiles are already in place. Some of the code is formatted oddly to allow it to fit on the page. There should be lots of room for your code.

```
#define NUMKIDS 5
int main() {
  int pid, i;
  // declare the pipes needed for 2-way communication
  int tochild[NUMKIDS][2];
  int tomaster[NUMKIDS][2];

  for (i=0; i<NUMKIDS; i++) {
```

```
   // create the pipes needed to communicate with this next child
if (pipe(tochild[i]) != 0) { perror("pipe"); exit(1); }
if (pipe(tomaster[i]) != 0) { perror("pipe"); exit(1); }
```

```
    if ((pid = fork())<0) { perror("fork"); exit(1); }
    if (pid == 0) {
```

```
       // close the pipes not needed here            (use the Close function from Question 4)
        Close("tochild[i][1]",tochild[i][1]);
        Close("tomaster[i][0]",tomaster[i][0]);
      // close pipes for previously forked processes
      int j;
      for (j=0;j<i;j++) {
          Close("tochild[j][1]",tochild[j][1]);
          Close("tomaster[j][0]",tomaster[j][0]);
      }
```

```
       // now call the do_child method filling in the parameters to match
       // match the signature  do_child(int child_num,int infd,int outfd)
       child(i,tochild[i][0],tomaster[i][1]);
```

```
        exit(0);                          // LINE A MARKED FOR LATER QUESTION
   } // bracket matches if (pid == 0)
```

```
    else {

        //close pipes that aren't needed here
        Close("tochild[i][0]",tochild[i][0]);
        Close("tomaster[i][1]",tomaster[i][1]);

    } // matches else
  } // matches for
  // all five children created with full duplex communication


  // send "hello" to each child
  char buf[6] = "hello";
  for (i=0; i<NUMKIDS; i++) {
    printf("parent about to write to child %d\n",i);
    if (write(tochild[i][1],&buf,6) != 6) {
      perror("write");
      exit(1);
    }
  }

  int total=0;

  // read an integer from each child and add it to total
  int ret;
  for (i=0; i<NUMKIDS; i++) {
    if (read(tomaster[i][0],&ret,sizeof(int)) != sizeof(int) ) {
      perror("read");
      exit(1);
    }
    total += ret;
  }

  printf("total from my children\n",total);
  return 0;                        //LINE B MARKED FOR LATER QUESTION
} // matches main
```

Part (a) [1 MARK]

In the previous program, there is a line marked `// LINE A MARKED FOR LATER QUESTION` . What would happen if this line was removed from the program?

SOLUTION: The children would go on forking new processes when their code executed the next pass of the loop.

Part (b) [1 MARK]

In the previous question, there is a line marked `// LINE B MARKED FOR LATER QUESTION` . What would happen if this line was removed from the program?

SOLUTION: The code would compile with a warning but the program would run fine.

Question 6. [15 MARKS]

For each code fragment below, indicate whether it WORKS GREAT, WORKS BUT HAS ISSUES, or NEVER WORKS. Then, explain the reasoning behind your answer. Answers without explanations will receive no marks. For the category WORKS BUT HAS ISSUES, it might be the case that the code works sometimes and not others, or it might be that the code runs but doesn't do what the user is expecting as described in the comments. ' is a single quote, and ‘ is a back quote.

Part (a) [1 MARK]

```
//reset behaviour for SIGKILL to be ignored
if (sigaction(SIGKILL, SIG_IGN, NULL) != 0) {
  perror("sigaction");
  exit(1);
}
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☑

Explanation: Can't ignore sigkill but would work for other signals.

Part (b) [2 MARKS]

```
# argument to the script is a filename in the current directory
# print this filename and the owner of the file
# reminder ls -l format is:
# -rw-r--r--  1 joeowner  somegrp  740 28 Nov 04:44 somefile.txt
echo $1
set ‘ls -l $1‘
echo $3
```

WORKS GREAT ☑ WORKS BUT HAS ISSUES ☑ NEVER WORKS ☐

Explanation: Works although name and owner are on different lines but ok if they don't notice this

Part (c) [2 MARKS]

```
# argument to the script is a filename in the current directory
# print this filename and the owner of the file
# reminder ls -l format is:
# -rw-r--r--  1 joeowner  somegrp  740 28 Nov 04:44 somefile.txt
set ‘ls -l $1‘
echo $1 $3
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☑

Explanation: Doesn't work correctly since filename in `$1` is replaced by set command.

Part (d) [2 MARKS]

```
file=analysis.txt
words="wc -w $file"
echo There were $words words in $file
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☑

Explanation: Doesn't work because wc command isn't executed. Would print *There were wc-w analysis.txt words in analysis.txt*

Part (e) [2 MARKS]

```
char s[25] = "2001: A Space Odyssey";
char * n = malloc(strlen(s) + 1);
char * p = s;
int c = 0;
while ((*n = *p) != '\0') {
  p++; n++; c++;
}
n -= c;
printf("n now points to %s\n",n);
```

WORKS GREAT ☑ WORKS BUT HAS ISSUES ☑ NEVER WORKS ☐

Explanation: Works fine but why not simply use strncpy.

Part (f) [2 MARKS]

```
struct rec {
  char name[20]; int age;
};

int main() {
  struct rec *a, *p;
  a = malloc(10*sizeof(struct rec));

  ... assign values to the 10 struct rec names ...

  // print out all the names
  for (p=a; p < a+10; p++) {
     printf("%s\n",p->name);
  }
```

WORKS GREAT ☑ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: Works perfectly

Part (g) [1 MARK]

```
char * s = "hotdogs";
```

WORKS GREAT ☑ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☐

Explanation: works fine "hotdogs" is in global space

Part (h) [1 MARK]

```
char p[7] = "mustard";
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☑ NEVER WORKS ☐

Explanation: Doesn't always work because not enough room allocated for null terminator. May happen to work sometimes.

Part (i) [1 MARK]

```
#print date in format Mon dd, yyyy
#reminder format of date output is: Tue Mar 18 10:05:47 EDT 2008
set ‘date‘
shift; echo $1
shift; echo $1,
shift; shift; shift;
echo $1
```

WORKS GREAT ☑ WORKS BUT HAS ISSUES ☑ NEVER WORKS ☐

Explanation:

```
sort of works prints Mon
                     Mar 17
                     2008
on 3 separate lines
```

Part (j) [1 MARK]

```
// return pointer to the struct containing the longer run
struct run * longer(struct run r1, struct run r2) {
  if (r1.miles > r2.miles)
    return &r1;
  else
    return &r2;
}
```

WORKS GREAT ☐ WORKS BUT HAS ISSUES ☐ NEVER WORKS ☑

Explanation: Doesn't work because r1 and r2 are on stack and gone once function returns.

Question 7. [10 MARKS]

Write a C program that takes an integer argument `s` representing a number of seconds and a string representing an existing filename. The program repeatedly generates a random integer from 0 to 99 and reads an integer starting from that byte offset in the file. The program runs for `s` seconds (of real time not virtual time) and then stops and reports how many reads were completed. You may assume that the file is at least 104 bytes long.

Hint: We have provided an abridged manual page for `setitimer` at the back of the exam.

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>

int reads, seconds;

void timeout(int signal) {
  printf("%d reads were done in %d seconds.\n",reads,seconds);
  exit(0);
}

int main(int argc, char ** argv) {
  seconds = atoi(argv[1]);

  FILE * fp;
  if ((fp = fopen(argv[2],"r")) == NULL) {
    perror("fopen");
    exit(1);
  }

  // install signal handler to report number of transactions
  struct  sigaction sa;
  sa.sa_handler = timeout;
  sa.sa_flags = 0;
  sa.sa_mask = 0;
  sigaction(SIGALRM, &sa, NULL);

  // set up and start timer
  struct itimerval value;
  value.it_interval.tv_sec = 0;
  value.it_interval.tv_usec = 0;
  value.it_value.tv_sec = seconds;
  value.it_value.tv_usec = 0;
  setitimer(ITIMER_REAL, &value, NULL);

  // now start the reads and writes and do as many as you can before

  reads = 0;
  int buf;
  for (;;) {
    int offset = (random() % 100) +1;
    //printf("run for %d seconds look at %d\n",seconds,offset);
    if (fseek(fp,offset,SEEK_SET) != 0) { perror("seek"); exit(1);}
    if (fread(&buf,1,sizeof(int),fp) <= 0) {
```

```
      perror("read");
      exit(1);
    }
    reads++;
  }
  return 1; //something is wrong if we ever get here!
}
```

Question 8. [15 MARKS]

For each code fragment or statement below, a subsection is indicated with a box. Explain what the boxed code does and why it is important. It is insufficient to simply say that a variable is assigned a value or a function is called. Your explanation should be clear about why the code is included in the program.

Part (a) [1 MARK]

```
fd_set      rset, allset;

FD_ZERO(&allset);
FD_SET(listenfd, &allset);

for ( ; ; ) {
  rset = allset;
  nready = Select(maxfd+1,  &rset, NULL, NULL, NULL);
```

Creating a set of file descriptors that only includes listenfd.

Part (b) [2 MARKS]

```
fd_set      rset, allset;
FD_ZERO(&allset);
FD_SET(listenfd, &allset);
for ( ; ; ) {

  rset = allset;
  nready = Select(maxfd+1,  &rset, NULL, NULL, NULL);
```

Part (c) [1 MARK]

```
    for ( ; ; ) {
        rset = allset;
        nready = Select(maxfd+1, &rset, NULL, NULL, NULL);

        if (FD_ISSET(listenfd, &rset)) {        /* new client connection */
```

Part (d) [2 MARKS]

```
struct sigaction newact;
sigemptyset(&newact.sa_mask);
newact.sa_handler = sig_quit;
newact.sa_flags = 0;
if(sigaction(SIGQUIT, &newact, NULL) == -1)
     perror("Could not install handler for SIGQUIT");
```

A function pointer to function `sig_quit` which looks like something the programmer wrote themselves.

Part (e) [2 MARKS]

```
struct sigaction newact;
sigemptyset(&newact.sa_mask);
newact.sa_handler = sig_quit;
newact.sa_flags = 0;
if(sigaction(SIGQUIT, &newact, NULL) == -1)
```

`perror("Could not install handler for SIGQUIT");` Calls a predefined routine that looks up the error code in errno and prints associated message along with this provided string.

Part (f) [1 MARK]

```
struct sockaddr_in self;
```

`self.sin_port = htons(PORT);` Converts the PORT number from the byte order on the host machine to the standard network byte order.

Part (g) [2 MARKS]

```
while((ch = getopt(argc, argv, "bd:f")) != -1)
```

Part (h) [2 MARKS]

```
int status;
wait(&status);
if (WIFEXITED(status)) {
  printf("exit status %d\n", WEXITSTATUS(status));
```

WIFEXITSTATUS is a macro that returns the return status from the exiting child. It is only part of what is in `status`.

Part (i) [2 MARKS]

```
void printArray(int a[12], int size) {
  int i;
  for (i=0;i<size;i++) {
    printf("A[%d] is %d\n",i,a[i]);
  }
}
```

The 12 in the a[12] isn't used at all and the function needs to be passed the size of the array