MALASPINA UNIVERSITY COLLEGE
FINAL EXAMINATION DECEMBER 2000
COMPUTER SCIENCE CSCI 265

NAME ______________________________

REG. NUMBER ______________________________

LEGAL SIGNATURE ______________________________

Instructor: Dr. P Walsh **Duration:** 3 Hours.

TOTAL VALUE	100						
QUESTION	1	2	3	4	5	6	7
VALUE	20	15	10	15	10	20	10
MARK							

Instructions

- Students must count the number of pages in this examination paper before beginning to write, and report any discrepancy immediately to the invigilator.
- This examination paper consists of 16 pages not including this cover page.
- Print your name and reg. number on the top of this examination paper. Sign the examination paper in the space provided. Answer all questions in the space provided. All questions relate to material covered in class or in the lab.
- This is an OPEN BOOK - OPEN NOTES examination.

QUESTION 1. (20 marks)
Consider the following interactive session on `csciun1` (with nin-GNU make).

```
csciun1% ls -R
Makefile    Message     RCS         include     typescript

./Message:
Makefile  RCS

./Message/RCS:
Message.c,v

./RCS:
greet.c,v

./include:
RCS

./include/RCS:
Message.h,v
csciun1%
csciun1% cat Makefile
INC = include
MODULES = Message

RMFLAGS = -f
CFLAGS = -g
XCFLAGS = -I$(INC)
COFLAGS = -q

RM =rm
MAKE = make
CO = co

IMPINC = $(INC)/Message.h
IMPOBJ = Message/Message.o

.DEFAULT:
$(CO)  $(COFLAGS) $<

greet: greet.o impMod
$(CC) $(CFLAGS) $(XCFLAGS) -o greet greet.o $(IMPOBJ)

greet.o: greet.c $(INC)/Message.h
$(CC) $(CFLAGS) $(XCFLAGS) -c greet.c

impMod:
for m in $(MODULES); do \
(cd $$m;\
 $(MAKE) $(MFLAGS) 'CFLAGS=$(CFLAGS)' export) \
```

```
done

clean:
@ for m in $(MODULES); do \
(cd $$m; \
$(MAKE) $(MFLAGS) clean) \
done
@ $(RM) $(RMFLAGS) *.o core greet

cleanSrc:
@ for m in $(MODULES); do \
(cd $$m; \
$(MAKE) $(MFLAGS) cleanSrc) \
done
@ $(RM) $(RMFLAGS) *.c $(INC)/Message.h

csciun1%
csciun1% cat Message/Makefile
INC = ../include

XCFLAGS = -I$(INC)
COFLAGS = -q
RMFLAGS = -f

RM = rm
MAKE = make
CO = co

.DEFAULT:
$(CO)  $(COFLAGS) $<

export: Message.o

Message.o: Message.c $(INC)/Message.h
$(CC) $(CFLAGS) $(XCFLAGS) -c Message.c

clean:
@ $(RM) $(RMFLAGS) *.o

cleanSrc:
@ $(RM) $(RMFLAGS) *.c $(INC)/Message.h
```

PTO...

Assume the interactive session continues and the following commands are executed in the order in which they are presented. Your task is to fill-in what is printed when the commands are executed. Note, you may assume that all compiler invocations complete successively.

```
make
```

```
ls -R .. ; co include/RCS/Message.h,v
```

```
make clean; ls -R ..
```

QUESTION 2. (15 marks)
Answer the following questions based on the Dynamic Simple Set Module and the ShiftSort Module from KWIC. Interface Specification are given on the following pages.
Assume that:

- The file of noiseWords in WordTable contains the word "and".
- LSInit, SSInit, and SShiftSort have been called, in that order.
- LineStorage line 0 contains the three words: `"Peter", "Paul", "Mary"`
- LineStorage line 1 contains the three words: `"Tom", "Dick", "Harry"`

What value will be returned by the following calls:

1. SSNumLines()
2. SSNumWords(3)
3. SSGetWord(2,2)
4. SSGetShiftNum(3)
5. SSGetShiftNum(5)

Assume that DSSInit has been called. What value will be returned by the following calls:

1. DSSAdd(5)
2. DSSDel(7)
3. DSSMem(6)
4. DSSAdd(5)

```
/********** ShiftSort module---interface specification **********/

/***** overview *****
The ShiftSort module computes a shifted and sorted version of the lines
stored in the LineStorage module.  Lines and words are made available
by line number and word number within line, as for the LineStorage
module.  However, ShiftSort provides access to the shifted and sorted
version.

The ShiftSort implementation stores line and word numbers but no
character strings; these are accessed when needed from the LineStorage
module.

No shifts are generated for the noise words specified in the WordTable
module interface specification.  That is, no line has a noise word in
the first position.

ASSUMPTIONS:

SSInit is called before any other ShiftSort function.

LSInit is called before any ShiftSort function other than SSInit.

SSShiftSort is called before any ShiftSort function except SSInit.

For any ShiftSort function except SSInit and SSShiftSort:  if a change
is made to the contents of the LineStorage module, SSShiftSort must be
called again.
*/

/***** constants*****/

/***** types*****/

/***** exported functions *****/

/*
* Initialize the module to have 0 lines
*/
void SSInit(void);
```

```
/*
* Remove all lines and words, freeing any dynamically allocated storage
*/
void SSReset(void);

/*
* attempt to compute the shifted and sorted lines
* if not enough memory is available then
* return KWMEMORYERROR
* else
* return KWSUCCESS
*/
KWStatus SSShiftSort(void);

/*
* if lineNum in [0..SSNumLines()-1] and wordNum in [0..SSNumWords()-1] then
* return word wordNum from line lineNum
* else
* return 0
*/
const char* SSGetWord(int lineNum,int wordNum);

/*
* if lineNum in [0..SSNumLines()-1] then
* return the number of words in line lineNum
* else
* return KWRANGEERROR
*/
int SSNumWords(int lineNum);

/*
* if lineNum in [0..SSNumLines()-1] then
* return the number of left circular shifts that were applied to the
* original line to generate line lineNum
* else
* return 0
*/
int SSGetShiftNum(int lineNum);

/*
* return the total number of lines currently stored
```

```
*/
int SSNumLines(void);

/*
* Display the internal module state on stdout.
* NOTE: for testing use only.
*/
void SSPrintState(void);

/*****usage example*****

If the LineStorage module contains the lines:
"The C Programming Language"
"The Cat in the Hat"

And the following calls are invoked:
SSInit();
SSShiftSort();

ShiftSort will provide access to the following lines:
        "C", "Programming", "Language", "The"
        "Cat", "in", "the", "Hat", "The"
        "Hat", "The", "Cat", "in", "the"
        "Language", "The", "C", "Programming"
        "Programming", "Language", "The", "C"

As illustrated by the following calls:
SSNumLines() will return 5
SSNumWords(0) will return 4
SSGetWord(0,0) will return "C"
SSGetShiftNum(0) will return 3
*/
```

```
/********** Dynamic Simple Set module---interface specification **********/

/***** overview *****
This module maintains a set of integers in a dynamic data structure.
The capacity of the set is limited by the amount of available memory.
*/

/***** constants *****/

/***** types *****/

typedef enum {
        DSSELEMENTERR=-3, DSSMEMORYERR=-2,
        DSSNONELEMENTERR=-1, DSSSUCCESS=0
} DSSStatus;

/***** exported functions *****/

/*
* Initialize the module.
* Free any any dynamically allocated storage.
*/
void DSSInit(void);

/*
* If the element  exists in the set then
*       return DSSELEMENTERR
* else If not enough memory available to add a new element then
*       return DSSMEMORYERR
* else
*       add the element to the set
*       return DSSSUCCESS
*/
DSSStatus DSSAdd(int x);

/*
* If the element does not exist in the set then
*       return DSSNONELEMENTERR
* else
*       delete  element from  set
*       free up the element's dynamically allocated storage
```

```
*       return DSSSUCCESS
*/
DSSStatus DSSDel(int x);

/*
* If the element exists in the set then
*       return true
* else
*       return false
*/
int DSSMem(int x);

/*
* Display the elements in the set.
*/
void DSSDump(void); /*FOR TESTING PURPOSES ONLY*/
```

QUESTION 3. (10 marks)
Give the exact output generated by executing the following Perl script.

```
@a = ('(bb)+',         '\+ *',
      '^a*',            'bb+',
      '^b*'
);

$s0 = 'aa* Peter aa bbbbb?\+XX12XY';

foreach $s (@a) {
    if ($s0 =~ /$s/) {
        print "$s: $`!$&!$'\n";
    }
}
```

QUESTION 4. (15 marks)
Below is the contents of file `Makefile` and six *csh* commands divided into five groups. Assume that the current directory is empty except for `Makefile` and that the six commands are executed in the order shown. Write the *exact output* produced by each group of commands.

```
.SUFFIXES: .z .x
.z.x:
        touch $<
X = f0.y
Y = f1.y

f0.x: $(Y) $(X)
        touch f0.x
f0.y:
        touch f0.y
f1.y:
        touch f1.y
f2.y:
        touch f2.y
f2.z:
        @touch f2.z
```

```
make f0.x
```

```
touch f1.y; make
```

```
make -n f2.z
```

```
make Y=f2.y X=f2.z
```

```
make Y=f2.x
```

QUESTION 5. (10 marks)
Consider the C version of *cew* developed in Pex2. If there is a C syntax error in `bats.c` (generated from the script in `bats.script`), the compiler reports the error and the number of the line in bats.c containing the error. What is required is a way of relating a line number in `bats.c` to the corresponding line in `bats.script`. To this end, develop the following filters:
(a) A perl script (called `lincount`) that inserts line numbers as comments on each line in `bats.script`. For example,

```
/* 1 */ Line 1 from bats.script
/* 2 */ Line 2 from bats.script
/* 3 */ Line 3 from bats.script
```

(b) A perl script (called `locate`) that takes one integer command line argument and returns an integer. The input is a line number in `bats.c` and the output is its corresponding line number in `bats.script`. For example, let `bats.script` and `bats.c` be identical for the first 20 lines. If line 20 in `bats.script` contains a `cew` construct that is replaced by (m4 macro substitution) 5 lines then line 21 in `bats.script` is on line 25 in `bats.c`. `locate 25` should return 21 and `locate 23` should return 20 (the filter should return 20 when passed any integer in the range 20 to 24 inclusive).

bats.script	bats.c
19	19
20	20
	21
	22
	23
	24
21	25

Assumptions/Restrictions: You may assume that a working solution to Pex2 is your starting point. You must use the its infastructure to advantage. Comments in bats.script can NOT extend over a single line.

QUESTION 6. (20 marks)
Write a C program that implements the command `date` to the following specification. You are not allowed to call the Unix command `date` in your implementation.

```
date(1)                                                              date(1)

NAME
  date - Displays the current date and time.

SYNOPSIS

  date [-X]

  The date command  writes the current date and time to standard output
  if called with no flag(s).  With the -X flag, the date command
  writes to standard out, the number of whole days between today (the
  day the command executes) and Christmas day 2000 (December 25).

FLAGS

  -X  Counts days before Christmas day 2000.

EXAMPLES

   1.  To display current date and time, enter:
            date
       The output is formatted as shown below:
            Fri Dec 11 19:32:38 PST 2000

   2.  To display the number of days before Christmas day 2000, enter:
            date -X
       The output is formatted as shown below:
            13

RESTRICTIONS/ASSUMPTIONS

  Valid switches are always a single character prefixed with a hyphen
  (e.g., -X).  The error message "invalid switch" is displayed if date
  is passed a switch other than -X.
```

QUESTION 7. (10 marks)
Give the exact output generated by executing the following C program (using gcc on csciun1).

```
#include <stdio.h>
#define NLINES 4

int i, pid, status;

int main(void) {
   pid = fork();
   if (pid == 0) {
      for (i=1; i<=NLINES; i++)
         printf("%d Michael\n",i);
   } else {
      for (i=NLINES; i>0; i--)
         printf("%d Peter\n",i);
      wait(&status);
   }
}
```