CS110: Computational Engineering

August – December 2010 Semester

Lab Manual

Department of Computer Science and Engineering
Indian Institute of Technology Madras

August 2010
Department of Computer Science and Engineering
Indian Institute of Technology Madras

CS110: Computational Engineering
August – December 2010 Semester

Course Instructor: Dr. Sukhendu Das
Dr. V. Kamakoti

Lab Coordinator: Dr. B. Ravindran

Faculty-in-charge of Lab Session:

Day	Faculty
Monday	N Sadagopan / G Ramakrishnan
Tuesday	Ragesh A
Wednesday	John Jose
Thursday	Uma Devi V
Friday	T Veena

Batches of students:

DAY	BRANCH AND ROLL NUMBERS								
	AE	CE	CH	CS	ME	EE	EP	MM	PH
MONDAY (12+1+19+11+24+6+10+2=85)	AE10B001 TO AE10B012	CE08B041	CH10B001 TO CH10B019	CS10B001 TO CS10B011		EE10B001 TO EE10B024	EP10B001 TO EP10B006	MM10B001 TO MM10B010	PH10B001 TO PH10B002
TUESDAY (12+19+11+1+2+4+6+10+2=85)	AE10B013 TO AE10B024		CH10B020 TO CH10B038	CS10B012 TO CS10B022	ME07B035	EE10B025 TO EE10B048	EP10B007 TO EP10B012	MM10B011 TO MM10B020	PH10B003 TO PH10B004
WEDNESDAY (12+19+11+25+6+10+2=85)	AE10B025 TO AE10B036		CH10B039 TO CH10B057	CS10B023 TO CH10B033		EE10B049 TO EE10B072 & EE08B083	EP10B013 TO EP10B018	MM10B021 TO MM10B030	PH10B005 TO PH10B006
THURSDAY (12+19+11+24+7+10+2=85)	AE10B037 TO AE10B048		CH10B058 TO CH10B076	CS10B034 TO CS10B044		EE10B073 TO EE10B096	EP10B019 TO EP10B025	MM10B031 TO MM10B039	PH10B007 TO PH10B009
FRIDAY (12+18+11+24+4+12=80)	AE10B049 TO AE10B060		CH10B077 TO CH10B094	CS10B045 TO CS10B055		EE10B097 TO EE10B119	EP10B026 TO EP10B029	MM10B040 TO MM10B051	

Classroom: RENUKA PATNAIK, YENDA RAMESH
Course Website: KVS DILEEP, PRAVEEN ARVIND BABU T, AMIT ARORA ARORA
Compilation of Marks: RAJESH BARUA

List of Teaching Assistants

Day	Teaching Assistants
Monday	CHIRANJOY CHATTOPADHYAY
	SIRRA RAMESH
	DAMARLA KRANTHIKUMAR
	RAJEEV RAJAN
	SIVA KUMAR MANDRAGUTHI
	ADARSH R
	AKSHAY GOYAL
	ARVIND EKKA
	BHARATH S
	MUTHU KUMAR R
	PAWAR PANKAJ SURESH
	SAKETHA M R
	SAMPATH M
	SRIHARI CHALAMCHARLA
	SUBHASH N
	VENKATA RAMANA MAHATHI KARNAM
	VIVEK PRAKASH SHRIBATRI
Tuesday	WILLIAM KUMAR MOSES, Jr.
	SHASHANK JAIN
	BHARATKUMAR BAGANA
	D.PRAVEEN RAJA
	GOPISETTI PRANEEL RAJA
	HARIKRISHNA PATNALA
	JAIN SACHIN SUNIL
	JIM CHACKO P
	KALPANA R
	SARADINDU KAR
	PATIL GANESH RAMESH
	KURMAN SANGEETA
	RAJESH J
	KARTHICK S
	DILLESWARA RAMESH NAIDU CH
	JOSNA V R
	RAHUL DEV BURMAN
Wednesday	ANAND SHARMA
	JYOTHI KRISHNA V S
	KINTHALI VENKATESH
	KONNA RAMAN KUMAR
	KRANTI RAJ AKKABATHULA
	MAHESH KUMAR S
	MOMLE SUMIT DILIP
	MUDDANA SAIPRANEETH
	CHIRANJOY CHATTOPADHYAY
	SIRRA RAMESH
	DAMARLA KRANTHIKUMAR
	RAJEEV RAJAN
	SIVA KUMAR MANDRAGUTHI
	ADARSH R
	AKSHAY GOYAL
	ARVIND EKKA
	BHARATH S

Thursday	MUTHU KUMAR R
	PAWAR PANKAJ SURESH
	SAKETHA M R
	SAMPATH M
	SRIHARI CHALAMCHARLA
	SUBHASH N
	VENKATA RAMANA MAHATHI KARNAM
	VIVEK PRAKASH SHRIBATRI
	WILLIAM KUMAR MOSES, Jr.
	SHASHANK JAIN
	BHARATKUMAR BAGANA
	D.PRAVEEN RAJA
	GOPISETTI PRANEEL RAJA
	HARIKRISHNA PATNALA
	JAIN SACHIN SUNIL
	JIM CHACKO P
	KALPANA R
Friday	SARADINDU KAR
	PATIL GANESH RAMESH
	KURMAN SANGEETA
	RAJESH J
	KARTHICK S
	DILLESWARA RAMESH NAIDU CH
	JOSNA V R
	RAHUL DEV BURMAN
	ANAND SHARMA
	JYOTHI KRISHNA V S
	KINTHALI VENKATESH
	KONNA RAMAN KUMAR
	KRANTI RAJ AKKABATHULA
	MAHESH KUMAR S
	MOMLE SUMIT DILIP
	MUDDANA SAIPRANEETH

Assignment Schedule

Week	Programming Assignment
9-13 August 2010	0. Linux, Editor
15-20 August 2010	1. Compiler, Debugger
23-27 August 2010	2. Assignment statements
30 Aug-3 Sept 2010	3. Control statements
6-10 September 2010 (Quiz I Week)	No Lab Session
13-17 September 2010	4. Loop statements
20-24 September 2010	5. Arrays
27 Sept-10 Oct 2010	Buffer week for Assignments 2 to 5
4-8 October 2010	Shashtra No Lab
11-15 October 2010	6. Functions
18-22 October 2010 (Quiz II Week)	No Lab Session
25-29 October 2010	7. Recursions
1-5 November 2010	8. Numerical Methods
8-12 November 2010	9. Spreadsheets and Search
15-19 November 2010	Buffer week for Assignments 6 to 9

General Instructions

1. **Time:** 7.30 PM - 9.30PM

2. **Venue:** DCF and Systems Lab

3. **Account Access:** The students should carry their Institute ID. Each student should login using the User ID assigned to him/her. Backup of files stored in the accounts are not assured.

4. **Systems:** The students should use the systems identified by the TAs.

5. **Teaching Assistants:** One TA has been identified for every 6 or 7 students in a particular lab session. Each TA will be associated with the same set of students through out the semester. The TAs will help the students in doing the assignments and evaluate the assignments.

6. **Evaluation:**

a) Total weightage for the Lab assignments is 25%. Each assignment will be evaluated for 15 marks.

b) The student should enter the code for all the programs of an assignment in the lab during the session. The TAs should ensure that the student has understood the code thoroughly and verify whether the programs are working correctly.

c) The assignments shall be evaluated only in order. If a student does not complete an assignment in a particular session, he/she may complete the same during the week and get it evaluated during the following week.

d) All disputes regarding the marks obtained by a student should be settled between the TA and the student on the same day of evaluation. No further revision of marks is possible.

7. **Change of slots for students:** The change of slot by the students is not allowed. The students having any difficulty in attending the lab session in the assigned slot may contact the Coordinator.

8. **Holidays:** In case a weekday is declared as an Institute Holiday, the lab session for that batch will be conducted on the same day or on Saturday during the same week. The Faculty-in-charge will decide the time, and inform the TAs and students.

List of Essential Linux, Emacs and Debugger Commands

Getting Started:

To login, type your username (Your roll number in lowercase letters) at the **Login:** prompt, and your password (same as your username) at the **Password:** prompt. Open a **shell** window from **Applications -> Accessories -> Terminal** menu.

Linux Shell Commands:

mkdir *dirname* Make a directory *dirname*

rmdir *dirname* Remove the directory *dirname*

cd *dirname* Change the current working directory to *dirname*

cd .. Change the current working directory to the parent directory

cd ~ Change the current working directory to your home directory

pwd Show your current working directory

mv *srcfile destfile* Rename the *srcfile* as *destfile*

cp *srcfile destfile* Copy one file, *srcfile* to *destfile*

cp *srcfile(s) destDir* Copy many file, *srcfile(s)* to *destDir*

rm -i *file(s)* Delete *file(s)*

ls -l List files in the directory with their details (size, time of creation)

gcc –o *prog prog.c* Compile the C program in the file *prog.c* and create the executable file *prog*

gcc –g –o *prog prog.c* Compile the C program in the file *prog.c* and create the executable file *prog* that can be used for debugging with **gdb**

./*prog* Run the program *prog*

Special characters in file and directory names:

***** - wildcard matches any string; **?** - matches any single character; **~** - your home directory

Emacs Editor:

Start the **Emacs** editor from **Applications -> Accessories -> GNU Emacs** menu

Commands in Emacs menu:

save as *filename*	Save the content in the file filename
save	Save the content in the current file
cut	Cut the marked block of text
copy	Copy the marked block of text
paste	Paste the marked block
close	Close the current file
exit	Save the current file and exit Emacs
help->tutorial	An online tutorial on the basic commands of Emacs

Debugger:

To debug *prog*, start the **gdb** debugger in the Shell window using: **gdb** *prog*

gdb Commands at the (gdb) prompt:

list	List 10 lines of the C source of *prog*
break *nnn*	Set a break-point: Program execution stops when it reaches line *nnn*
run	Start program execution
continue	Continue execution from a break-point
next	Execute the next line in program and then break. Does not break in functions.
step	Execute the next line, stepping into functions, and then break
<ENTER>	Pressing the <ENTER> key repeats the previous command
print x	Display the value of the the variable x
set x=10	Set the variable x to 10
quit	Quit the debugger

gdb commands can be abbreviated, *e.g.* **l** for list, **b** for break, **n** for next, **p** for print, etc.

A Standard for C Code

Prof.T.A. Gonsalves

6th August 2010

Following a coding standard is part of professional programming. This enhances the readability of your code, it improves the quality and makes it easier for other programmers to read and modify your code.

Names

To make the code self-documenting, choose meaningful names for variables. Abbreviations may be used so long as they are widely accepted. A good test of names is: *can you read your code to a fellow programmer over the phone?*

For names that consist of multiple words, capitalize the first letter of each word.

Distinguish classes of names as follows:

Functions, Macros, Types, Classes: First letter uppercase (eg. GetInput(), LengthType, Compute()).

Constants**:** All uppercase, separate words with '_' (eg. MAX_LINE_LEN, PI, VOTING_AGE)

Variables**:** First letter lowercase (eg. roomMessDistance, inBuf, myId, windowHt, wallWidth)

Names should differ in more than one character, especially if they are of the same type. E.g., for the transmit and receiver buffers, *txBuf* and *rxBuf* differ in only the first character which occurs on adjacent keys on the keyboard. *txBuf* and *rcvBuf* is a better choice.

Use the following abbreviations to identify particular names:

Type	Defined type (e.g. typedef struct {...} MsgType;)
Ptr	Pointer (e.g. bufPtr, msgPtr, pktPtr)
Fl	Boolean (e.g. moreFl)
Str	String (e.g. promptStr)
Chr	Character (e.g. inChr, outChr)
Tab	Table (e.g. relayTab, relayTabPtr)
Num	Number (e.g. numCourses) ["No" could be confused with the negative]
Ctrl	Control (e.g. CTRL_C)
Cmd	Command (e.g. LastCmd)
Cnt	Count (e.g. wordCnt)
Que	Queue (e.g. inBufQuePtr)
Len	Length (e.g. roadLen)

Internal Documentation

Apart from external documentation such as pseudo-code, flow-charts, state transition diagrams, function-call hierarchies, and prose, the program files should contain documentation. Begin **each file** with a comment including the following fields:

```
/*******************************************************************
 * sort.c – for sorting integers           filename with one-line description
 * Purpose: uses bubble-sort algorithm...  purpose in detail
 * Compilation: use the supplied makefile  Instructions for compiling
 * Revision history:                       Chronological list of changes/bug-
fixes
 *    A. Programmer, 7/7/77
 *    released version 1.0
 *    C. Debugger, 8/8/88
 *    fixed stack overflow with null input
 *    Eager B. Eaver, 9/9/99
 *    added ANewProc() to support 3-D
 * Bugs:                                   Known bugs/limitation/testing to be
done
 *    The program occasionally crashes when two users
 *    access the database simultaneously during the new moon.
 *************************************************************/
```

Declare **each variable** on a separate line, followed by an inline comment explaining the purpose of the variable. Use

```
     char *inBuf; // buffer for received keystrokes
     char *outBuf;      // buffer for text going to the printer
```

rather than

```
     char *inBuf,  *outBuf;  // input and output buffers
```

If there are a large number of variables, group them in blocks by function, and alphabetically within each block. Note: temporary variables such as loop indices need not follow some of these rules.

Preceding **each function**, include a comment block as follows:

```
/*******************************************************************
 * GetInput - get input from the keyboard.
 * Args:    Stores the string in the buffer buf, max size is bufSize
 * Returns: number of characters stored in buf
 *           or -1 on error.
 * Method:  a brief description if necessary.
 * Bugs:    list known bugs and limitations
 * To be done: if anything
 *******************************************************************/
int GetString(char *buf, int bufSize)
  {
    ...
  }               /*  End of GetString()  */
```

Within the body of the function, on separate lines at the start of **each major block** (if, while for, switch), describe briefly the purpose and peculiarities of the block. For obscure statements, include an inline comment.

Avoid obvious comments such as:

```
   i++;          /* increment i  */
```

Layout

Indent the code according to the following scheme and use blank lines to indicate breaks in the flow of control. This improves the readability.

```
while (moreFl)              /* The main loop, terminates when done */
  {
    if (i == 2)
      DoSomethingAppropriate();
    else
      DoSomethingElse();

    for (j = 0; j < maxFile; j++)   /* Mumbo-jumbo for each file */
         {
            total += table[i].wordCnt;
            i      = j + k;
            cnt    = j – 1;
         }
  } /* while (moreFl) */
```

Useful Features

Some C language features that will enhance the quality of your code:

Header files: collect macro, type, constant and global variable declarations and prototypes for public functions in one or more .h include files. Never include code in .h files. Group logically related functions into separate .c files. Use a utility such as *make* to automate rebuilding the program.

Information hiding: declaring a function static makes it private to he module (i.e., file) in which it is declared. Likewise for data. In a header file, define *#define PRIVATE static* and use it for private functions and data:

PRIVATE int myCount;
PRIVATE void LocalFunc();

Function prototypes: use these to enable the compiler to check for consistency of arguments. In a header file, include function prototypes for all public functions.   Remember to use void for functions that do not return any value.

Enumerated types: use *enum* rather than a sequence of *#defines*. This is less error-prone and enables the compiler to check type consistency.

Type casts: use explicit typecasts to avoid warning messages from the compiler about operands of different types.

Week: 9-13 August 2010

Programming Assignment 0: Linux Commands and Emacs

1. Login
2. EMACS Editor
3. Linux commands:
   a. Creation of directory
   b. Copy a file
   c. Rename a file
   d. Delete a file

Problem 0.1 (Emacs editor):

- Use the editor to type a letter to your friend describing your first semester experiences at IIT Madras
- It must be at least two paragraphs with six or more sentences each.
- Delete the third and fourth sentences of the first paragraph
- Move the fifth sentence of second paragraph as the third sentence of first paragraph - you should not retype
- Copy the fourth sentence of first paragraph as the last sentence of second paragraph.
- Now read the letter and edit (delete and insert) necessary words/sentences so that it sounds sensible.

Problem 0.2 (Linux Commands):

• Save the letter of Problem-1 as a "file".
• You want to send the same letter to four more friends
– Make four copies of the same (use the cp command)
– Open the copies and change the names of your friends
– Delete the file containing the letter for your third friend
(use "rm" command)
– You wanted to store these files in a separate place that you could remember. So, create a directory called "FriendsLetters" and move these files to it.

Week: 15-20 August 2010

Programming Assignment 1: Compilation and Debugging

1. Compilation of program using *gcc*
2. Execution of programs
3. Debugging of programs using *gdb*

Problem 1.1 (Hello World):

```
 #include<stdio.h>
main()
{
  printf(“Hello World\n”);
}
```

Problem 1.2 (Area of a Circle):

```
#include<stdio.h> /* Library File Access */
/* Program to calculate area of a circle */
main() /* Function Heading */
{
   float radius, area; /* Variable Declarations */
   printf(“Radius = ?”); /* Output Statement (Prompt) */
   scanf(“%f”, &radius); /* Input Statement */
   area = 22/7*radius*radius;
    /* Assignment Statement */
  printf(“Radius of the circle = %f    ,   Area of the circle = %f”,
               radius, area);
   /* Output Statement */
}
```

Week: 23-27 August 2010

Programming Assignment – 2: Assignment Statements

1. Data types: Integer and Float
2. Arithmetic operators
3. Arithmetic expressions
4. Precedence of operators

Problem 2.1:

Evaluate the expressions given below, and print their values for the following two sets of values of A, B, C, D **(Create two separate files for each set of values)**:

1. A = 5, B = -3, C = -6, D = 4
2. A = -2.5, B = 4.25, C = -6.0, D = 1.75

X1 = AB+CD
X2 = A(B+C)D
X3 = (A+B)(C-D)
X4 = (A+B)/(C-D)
X5 = A+B/C-D
X6 = AB/C-D
X7 = (A%D) + (C%B) {For integer data only}
X8 = (A%B) + (C%D) {For integer data only}
X9 = $A^3 – B^3 + 2A^2B – 5AB^2 + 6A – 9B + 4$
X10 = $(2 A^2 + 3B + 4C – D^2) / [(6 A+B) (3C-D)]$

Note: The inputs are to be hard-coded.

Week: 30 August-3 September 2010

Programming Assignment – 3: Control Statements

1. *if .. else* statement
2. *switch* statement
3. Use of Math library

Problem 3.1:

Write a program to determine the grade for a student based on the attendance and marks obtained by the student as follows:

If the attendance for the student is less than 75%, then the grade is W.

If the attendance for the student is equal to or more than 75%, then the grade is determined using the marks as in the table given below.

Marks	Grade
90 – 100	S
80 – 89	A
70 – 79	B
60 – 69	C
50 – 59	D
40 – 49	E
30 – 39	F
< 30	U

(a) Write the program using only *if .. else* statements
(b) Write the program using an *if .. else* statement and a *switch* statement
(Hint: Use *Marks/10* as the *expression* for *switch* statement)

Problem 3.2:

Write a program to find the roots of a quadratic equation

$$AX^2 + BX + C = 0$$

The roots can be real or complex.

Print the values of roots for the following values of coefficients:

1. A = 2, B = 5, C = 3
2. A = 4, B = 5, C = 3

Week: 13-17 September 2010

Programming Assignment – 4: Loop Statements

1. *for* statement
2. *while* statement
3. *do .. while* statement

Problem 4.1:

Write a program to compute the factorial of a positive integer n ($n \geq 1$), using the *for* loop statement.

Test your program for (i) n = 5, (ii) n = 10, and (iii) n = -3

Problem 4.2:

Write a program to determine whether a given number n ($n \geq 2$) is a prime number, using the *while* loop statement.

Test your program for (i) n = 79, (ii) n = 8, and (iii) n = 49

Problem 4.3:

Write a program to compute the greatest common divisor (GCD) of two non-zero positive integers using the remainder method. The program is to be written using the *do .. while* loop statement.

Test your program for the following pairs of numbers: (i) 24, 38 (ii) 21, 32 (iii) 75, 45

Week: 20-24 September 2010

Programming Assignment – 5: Arrays

Problem 5.1

Write a program to do the following:

(a) Read the elements of an array, *arrayA*, of *n* integers.

(b) Count the number of positive integers, negative integers and zeros in *arrayA.*

(c) Create another array, *arrayB,* such that the order of its elements are in the reverse order of the elements in the *arrayA* (*i.e.*, B[i] = A[n-i]).

Test your program for the following data:

***arrayA* = { -1, 2, 5, 0, -6, 3, 0, -2, 4}**

Problem 5.2

Write a program to verify whether a given matrix is a magic square. A magic square of order n has the elements 1 to n^2, appearing once only, and has the property that each of the rows, columns and the two main diagonals all sum to the same value. Note that you have to check both uniqueness of the elements and the sums. Your program should take as input the order of the matrix and the elements of the matrix row-wise.

Test your program for the following data:

Matrix *A* =
$$\begin{matrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{matrix}$$

Matrix B =
$$\begin{matrix} 7 & 12 & 1 & 14 \\ 2 & 13 & 8 & 11 \\ 16 & 3 & 10 & 5 \\ 9 & 6 & 15 & 4 \end{matrix}$$

Week: 11-15 October 2010

Programming Assignment – 6: Functions

In the following exercises, you will learn how to write functions and procedures. You will also understand the difference between passing parameters by value and by references.

You are not allowed to use global variables in both the following exercises. Data must be passed from one function to the other through function's interfaces. Write all the functions in one file and the **main** program in a **separate** file.

Problem 6.1: Sorting Numbers
Write a function **SortNumbers( )** that will take an array of integers and the number of elements in the array as arguments and sort the array in increasing order using bubble sort. Modularize your program by writing two other functions : **ReadInputs( )** must read the array and the count from the user; **PrintResults( )** must print the sorted results.

Test your program for the following data:

1. 20, -5, 4, 17, 9
2. 10, 9, 8, 7, 6, 1, 2, 3, 4, 5

Problem 6.2: Sorting Words
In this exercise, you will sort words according to their length. As in the previous exercise, you will have to write three functions:
ReadInputs( ), **SortWordsByLength( )** and **PrintResults( )**.
The ordering of words must be done based on the length of the names. Write another function **LengthOfWord( )** that will take a word as input and return the length of the word. This function must be called by SortWordsByLength( ) for comparing lengths of words.

Test your program for the following inputs

* How do you know C programmers
* They count from zero
* Do you know Cray supercomputer
* It is so fast it can run an infinite loop in six seconds

Week: 25-29 October 2010

Programming Assignment – 7: Recursive functions

Problem 7.1:

Write a recursive function to compute the factorial of a number N.

Test your program for the following numbers:
(i) N = 4 (ii) N=6

Problem 7.2:

Write a recursive function to compute the greatest common divisor (GCD) of two non-zero positive integers using the remainder method.

Test your program for the following pairs of numbers:
(i) 24, 38 (ii) 21, 32 (iii) 75, 45

Problem 7.3:

Write a recursive function to compute the binomial coefficient ${}^{N}C_{r}$ using the Pascal's rule, ${}^{N}C_{r} = {}^{(N-1)}C_{(r-1)} + {}^{(N-1)}C_{r}$

Test your program for the following:
(i) ${}^{4}C_{2}$ (ii) ${}^{5}C_{3}$

Week: 1-5 November 2010

Programming Assignment – 8: Numerical Methods

Problem 8.1

Write a program to find the root of a polynomial using the Newton-Raphson method. The program must take the following inputs:

- Degree of the polynmial - *n*.
- The set of coefficients – a_0, a_1, ..., a_n
- The initial value for the root - x_0
- Maximum number of iterations - *N*

The program must find a root of the polynomial and the value must be correct to 4 decimal places.

Problem 8.2

Write a program to calculate the value of π using Taylor's series.

In this program, you must use the Taylor's series of expansion of $\arctan(x)$ to calculate p/4.

The Taylor series for $\arctan(x)$ is arctan(x) = $x-x^3/3+x^5/5-x^7/7+$.. If you substitute x = 1, then

PI/4 = arctan(1) = 1-1/3+1/5-1/7+...

You must take the desired digits of accuracy *d* as input and print the result of p.

Week: 8-12 November 2010

Programming Assignment – 9: Spreadsheets and Search

Problem 9.1

A class of students are taking 3 courses, M, P and C. At the end of the semester, the final marks of all these courses are compiled into a spreadsheet. Every row in the spreadsheet has the name of a student, the roll number of the student, and the mark he/she obtained in each of the subjects in order (M, P, C). Your job is to assign grades to students for each of the three subjects based on the final marks in those subjects. You must use spreadsheets and built-in functions inside the spreadsheets for this exercise.

The grading scheme is as follows. For each subject, you must find the mean and the standard deviation. The letter grades can be one of S, A, B, C, D or E and is assigned based on the following criteria:

S	marks >= avg + 2 * stddev
A	avg + 2 * stddev > marks >= avg + 1 * stddev
B	avg + 1 * stddev > marks >= avg + 0 * stddev
C	avg + 0 * stddev > marks >= avg - 1 * stddev
D	avg - 1 * stddev > marks >= avg - 2 * stddev
E	avg - 2 * stddev > marks

The teachers and the dean are also interested in the following statistics for each course:

- Minimum and maximum marks scored
- A histogram of the number of students who attained each letter grade

The solutions used in the spreadsheet must be tested with at least 10 students' records but must be generic enough for any number of students.

Problem 9.2

You are given a lexicographically sorted list of words in an array. Given a letter of the alphabet you have to find the indices of all the words beginning with that letter. Use the **binary search** algorithm to find the indices. Note that you are not required to print all the words, but just the range of indices for the words beginning with that letter. Initialize a 2D character array at the beginning of your program with at least 20 entries in sorted order. Assume that all the words are in lower case.

Example: Given an array Names[ ][ ]={"apple" , "ball" , "bat", "eat", "fog", "rat"} and the letter "b", the program should output "1:2"; given the letter "e" it should output "3:3"; and given the letter "k" it should output "-1:-1".