

School of Computer Science and Software Engineering
Clayton Campus, Monash University

CSE1303 Part A
Semester II, 2001

Practical Session 1: Revision



Aims of this Practical Session

To revise the following concepts:

1. Reading from files and printing to files.
2. Using arrays and structs.
3. Sorting an array.
4. Searching for items in an array.

Project: To write a C program to search and solve word-find puzzles.

Background:

A word-find puzzle consists of a 2 dimensional array of characters and a list of words. An example is given below. The object of the puzzle is to find the words in the array of characters. The words can appear in the puzzle going up, down, across, backwards and diagonally. In the example below the words which have been found appear shaded.

(This is the 2d array stored as characters in the file array.txt)

S	B	U	Q	P	M	U	Q	B	P	M	R
S	E	L	B	A	T	E	G	E	V	C	T
M	A	E	R	C	E	C	I	S	A	U	E
H	S	I	F	K	B	Y	R	N	R	D	C
H	P	T	C	A	K	E	S	K	I	O	I
T	S	I	R	G	Z	S	E	B	E	O	U
S	E	R	E	E	B	Y	L	F	T	F	J
L	L	K	E	S	S	E	S	O	Y	A	D
A	F	R	R	N	F	S	W	T	Z	Q	R
E	F	N	J	A	N	N	E	Z	O	R	F
M	A	T	A	E	M	I	I	D	E	R	M
I	W	P	T	G	U	P	D	L	O	C	E

List of words found so far *(in the file words.txt):*

CAKES MARKET PACKAGES FISH WAFFLES



Note: All example input files mention below can be found can be found at:

<http://www.csse.monash.edu.au/courseware/cse1303s/parta/prac01/>

Preparation (3 marks if completed before class)

Before this prac:

1. Find at least 5 more words in the above puzzle.
2. Write a C declaration to store a word, and what the position of it is within the puzzle.

⊕ **You may assume all words have at most 15 letters, and you need to store information like where the first letter is (remember 2D arrays have both rows and columns), in what direction it is written (down, right, diagonal-up-left, diagonal-down-left, etc), and the length of the word.**

3. Write a C program which asks for a file name, reads an array of characters contained in the file, and prints out the array to the screen.

⊕ **You may also like to print coordinates around the array to help a user identify the position of the word and may assume that the puzzle is 12 by 12.**

	1	2	3
1	S	B	U
2	S	E	L
3	M	A	E

⊕ **You may assume that the array is stored in the input file in the format of 12 characters per line. (See *array.txt*)**

4. Write a C program which asks for two file names, the input file and the output file. The program should open both files and read the words contained in the input file. The words should be stored in an array of structs (the same structs that you defined in the first part of the preparation). Then print the words to the output file. Don't worry about the position information at the moment, you do this later.

⊕ **You may assume that there will be at most 24 words in the input.**

⊕ **You may assume the input file contains a word per line. (See *words.txt*)**

Question 1: (2 marks)

Extend the C program you wrote in the second part of the preparation so that the program performs the following steps:

1. Ask for input file name and open the file.
2. Read the array of characters in the input file.
3. Print out the array of characters.
4. Ask the user for the location of a word. (*position, direction and length*)
5. Either print out the word found in this location, or print an error message if it is not valid location.
6. Ask the user whether they want to quit.
7. If the user wants to quit, then halt.
8. Else goto step 3.

Question 2: (2 marks)

Extend the C program you wrote in the third part of the preparation so that the program performs the following steps:

1. Ask for an input and an output file name, and open both files.
2. Read the words stored in the input file and store them in an array of structs.
3. Asks the user for a word.
4. Search the array of structs to find whether the word is contained in the array of structs, then notify the user whether it was or not.
5. Ask the user whether they want to quit.
6. If the user wants to quit, then the program should then print all the words to the output file in alphabetical order and halt.
7. Else goto step 3.

Question 3: (3 marks)

Using the programs you wrote for question 1 and question 2, write a new C program to perform the following steps:

1. Ask for an input and an output file name, and open both files.
2. Read the array of characters and words stored in the input file.
3. Store the words in an array of structs.
4. Print out the array of characters and the list of words not found.
5. Ask the user for the location of a word.
6. If the location is valid, then get the word in the puzzle found in this location, else print an error message.
7. Search the array of structs to find whether the word is contained in the array of structs.
8. If the word is in the array of structs, then store its location in the corresponding struct, else print an error message.
9. Ask the user whether they want to quit.
10. If the user wants to quit, then the program should then print to the output file all the words it has found, in alphabetical order, and where they were found, and then halt. (*word, position, direction, length*)
11. Else goto step 4.

⊕ **You may assume that the array of characters (12 characters per line) is stored in the input file followed by the list of words (one per line). (See puzzle.txt)**

Advanced Questions: (2 marks)

Consider the following command in UNIX

```
sort -r example.txt
```

The strings **-r** and **example.txt** are known as command-line arguments. In this example **-r** is an option which makes the program sort the lines in the file **example.txt** in reverse order. (**Note:** the name of the program, in this case **sort**, is also a command-line argument.)

To pass command-line arguments to a program call main as follows:

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

The first argument (called by convention **argc**, for argument count) contains the number of command-line arguments the program was invoked with; the second argument (called by convention **argv**, for argument vector) is a pointer to an array of character strings that contain the arguments.

For the example of **sort** given above **argc** is 3, and **argv[0]**, **argv[1]**, and **argv[2]** are the character strings, "**sort**", "**-r**" and "**example.txt**", respectively.

Questions:

- Change the program you wrote for Question 3, so that instead of asking the user for two file names, it reads the names of the input file and output file as command-line arguments.
- Extend the program you wrote above, so that when it is run with the option **-s** it solves the word-find puzzle and prints the solution to the output file.