CMPE-231 DATA STRUCTURES(2008-09 Spring)

••• Laboratory Work #2

Duration – 100 minutes

This laboratory work (Tutorial) covers introduction to Microsoft Visual C++ 6 IDE ,Arrays, Array of structures and related laboratory experiments.

This week three experiments will be done. (Experiment 2 and 3 will be prepared before lab hour by the student)

Experiment 1– Writing programs(prepared by the Assistant in the lab and will be explained in details)

One and two dimensional array application

Write a separate C function for each of the following sub problems.

1.1) Using Random number generator example given below,
Generate 150 unique(<u>do not add duplicated integer numbers into array of variable</u>) numbers
1.2) List them in descending order(<u>**write a Bubble sort program**</u>).

Maximum generated number will be in between 1 and 600.

Example program :
Following portion of program generates integer numbers(maximum number) in between 1 and 100 using random number generator function rand().

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
      int number;
      do
      {
      number = rand()%100+1;
      printf("%d \n",number);
      }while(number !=100 );
}
```

Experiment 2– Writing programs(Will be written by the student during lab hour) **(One and two dimensional arrays application.)**

1. Initialize the following real number values into two dimensional 3x4 matrix(**x**) and copy all the elements into one dimensional array(**y**) by

   . Column wise
   . Row wise

   3x4 Matrix **x** has the following data values in it.

$$\begin{vmatrix} 13.5 & 24.88 & 66.28 & 30.0 \\ 2.84 & 412.15 & 13.2 & 9.9 \\ 15.5 & 38.89 & 12.5 & 16.18 \end{vmatrix}$$

2. Display content of two dimensional array x and
   Display one dimensional array **y** which is obtained Row wise using x

Experiment 3. After reading **Using Data Files** section.

```
struct persalary
{
int     month; /* number 1 corresponds to January, */
long salary;   /* 2 corresponds to February, and so on */
};
struct employee
{
int empnr;                        /* Employee number */
char name[12];                    /* Employee name */
struct persalary salaries[3]; /*  three months salaries */
long total;                /* Sum of 3 months salaries */
float average;                   /* Average of 3 months salaries*/
};
```

A) Using the above structure **define an array of structure employee for 4 person** that will be initialized using a data file(named **emp.txt**) which has the following data in it. Using *Notepad* create the following file

```
121  Ahmet    2  300000  4  400000   12   750000
234  Ayse     3  250000  6  800000    7   475000
456  Mehmet  4  750000  5  880000   10   660000
875  Sevgi     2  950000  4  660000    9   775000
```

B) Write a function that displays the initialized values as shown below.
Note that the integer values corresponding to months are converted to the proper **month names.** This can be achieved by declaring an **array of the month names.**

Employee Nr	**Name**	**Month**	**Salary**
121	**Ahmet**	**February**	**300000**
		April	**400000**
		December	**750000**
234	**Ayse**	**March**	**250000**
		June	**800000**
		July	**475000**
456	**Mehmet**	**April**	**750000**
		May	**880000**
		October	**660000**
875	**Sevgi**	**February**	**950000**
		April	**660000**
		September	**775000**

C) Write a function that computes for each employee the **Average** and the **Sum** of the three months' salaries and sets the values of the corresponding variables in each record and display the output as follows.

```
Employee Nr         Name            Average         Sum
     121            Ahmet           xxxx.xx         xxxxxx
     234            Ayse            ..              ..
     456            Mehmet          ..              ..
     875            Sevgi           ..              ..
```

D) What will be the total memory size of this array? Explain Your answer,

E).Assume that you declared allEmployee for 3 employees as follows

struc employee allEmployee[3];

- If the base address of this structure is **2001**, what the starting address of **allEmployee[2].salaries[1].salary**
- If the base address of this structure is 5001, what is the address of the salary of **880000** which is the salary of the **Mehmet**

Using Data Files
Instead of typing data at the keyboard, we can store information in files in our own disk areas and use the data as input to our programs. Also, instead of writing our results directly to a monitor screen, we may write information to a data file that could be printed later or used as input to another program.

When creating data files to be used as input to a program(for example , by using editor), we have to specify a file name according to the conventions of the computer's operating system. Similarly, when we write to a data file from within a program we must specify the name of the file to be created and written to.

To specify the input and output files used by a program, we declare special identifiers called file pointers.

```
FILE *file_Identifier;
```

Example

```
FILE *student_file;
```

The type is **FILE** and should always be written using capital letters. The **file identifier** that follows is always preceded by * to indicate that it is a pointer. Within the program we always use the file pointer to refer to a particular file.

To associate the file pointer within the file name as used by the operating system, we use the ***fopen*** function.

<u>Opening a file</u>
General form of an opening a file is

```
File_identifier = fopen(arg1, arg2);
```

The first argument(arg1) is a character string constant containing the name of the file as known to the operating system. The second argument(arg2) is also a string constant known as the ***mode***. The mode may be **"r"** to show that the file is to be **read**, "**w**" to create a file for **writing**, or some others are as given as;

If we want to read from the file called **student.txt** and write to a new file to be called **result.txt** , we would write the following.
"r" open an existing text file for reading.

“w” create a text file for writing.
“a” append to an existing or create a new text file for writing
“r+” open an existing text file for reading or writing.
“w+” create a text file for reading or writing.
“a+” append to an existing or create a new text file for reading and writing

```
student_file = fopen(“c:student.txt”, “r”);
student_result = fopen(“c:result.txt”,”w”);
```

The ***fopen*** function is normally placed at the beginning of the executable part of the program.

Also, for convenience, it may be combined with the declaration of the file pointer.

```
FILE *student_file=fopen(“c:student.txt”, “r”);
FILE *student_result = fopen(“c:result.txt”,”w”);
```

Input/Output Functions

To read from or write to data files, we use the standard functions **fscanf()** and **fprintf()** for formatted input and output files and **fgets()** and **fputs()** for string input and output files**.**

General form of **fscanf()** and **fprintf()** are as follows.

```
fprintf(file_identifier, format_descriptor, Variabla_list);

fscanf(file_identifier, format_descriptor, Variabla_list);
```

General form of **fputs()** and **fgets()** are as follows.

```
fputs (s1 , file_identifier);

fgets (s1 , size , file_identifier);
```

```
fputs(line, student_file);
```

statement writes one record of data onto **student_file.**

```
fgets(line, 60, student_file);
```

statement reads one record of information from file **student_file** into string variable **line** which must be defined as **character[60].**

Example

```
fscanf(student_file,"%d %s %d", &st_number, st_name, &st-grade);
fprintf(student_file,"%d %s %d" st_number, st_name, st_grade);
```

First example reads student number , name and grade information from file and assign the values into the variables **st_number**, **st_name** and **st_grade** respectively.
Second example, writes student information into a file.

After completing all the operations we must close them using ***fclose*** function.

Closing a file

General form of ***fclose*** function is as follows.

```
fclose (file_identifier);
```

Example

```
fclose (student_file);
```

End Of File(*feof*)

The end-of-file value is an implementation defined constant called EOF(normally having the value –1). We may use it after calling the function **fputc, fputs, fgetc** and **fgets** described above.

For example,

```
letter=fgetc(alphabet_file);
if (letter == EOF)      printf("No more characters in Alphabet file \n");
```

Another way of detecting the end of an input file is invoking the function ***feof*** which takes the file pointer as its argument and returns a non-zero value if the end of the file has been reached, otherwise it returns zero.

```
do
{
  ..
  ..
}
while ( feof(stud_file) == 0);
```

Example1 :
Following program reads inputs of student information from the file **student.txt** and finds average grade of student. Than program opens existing file as a second time and reading each student record and list the student name whose grades are greater than average grade until **feof** is reached.

```
#include <stdio.h>
int main(){
int stnumber , grade , cnt;
float avr;
char name[10];
FILE *stud_file;
stud_file = fopen("c:\\student.txt","w");
cnt=0 , tgrade = 0;
fclose(stud_file);
stud_file = fopen("ogrenci.dat" , "r");
do
{
 fscanf(stud_file ,"%d %s %d" , &stnumber , name , &grade);
   tgrade+=grade;
   cnt++;
 }while ( feof(stud_file) == 0);
avr = tgrade / cnt;
fclose(stud_file);

stud_file = fopen("ogrenci.dat" , "r");
do
{
 fscanf(stud_file ,"%d %s %d" , &stnumber , name , &grade);
   if (grade > avr )
          printf("Student Info = %d %s %d\n" , stnumber ,
          name , grade);
 }
while ( feof(stud_file) == 0);
fclose(stud_file);
}
```