Indian Institute of Technology Mandi
IC150: Computation for Engineers
Tutorial 3 Arrays, File IO

1) Fill in the blanks

   (a) Input/output in C occurs as a stream or sequence of bytes.

   (b) Most C programs should include the stdio.h header file that contains basic information required for all I/O operations.

   (c) Opening a file in “w” and _″w+″______ modes destroys the existing contents of the file.

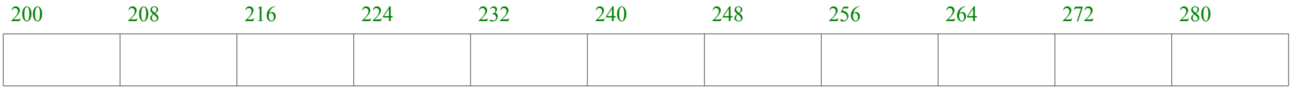
   (d) The best-case time complexity of Insertion Sort is O (n) while that of Selection Sort is_ O(n^2).

   (e) Name an unstable sorting algorithm: Selection sort

   (f) An array is declared: `NewType mda[M][N][P]`. Assume that `M`, `N` and `P` are constants and the base of the array is at address `base`. The address of element `mda[i][j][k]` is given by `adr = base + size(New Type)*[(P*N)i + (P*j) + k)]`.

2) An array is declared: `double d[5][2];` The base of the array is memory location 200. Assume that a double occupies 8 bytes. Draw a neat memory diagram of the array showing the addresses of the elements `d[i][0]` for i in the range [0..4].

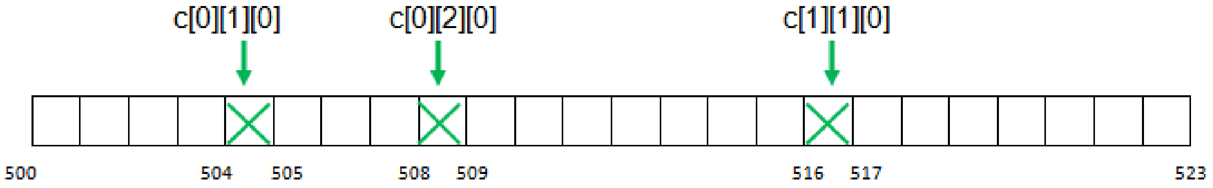
Sol:



Addresses: d[0][0]: 200-207, d[1][0]: 216-223, d[2][0]: 232-239, d[3][0]: 248-255, d[4][0]: 264-271

3) An array is declared: `char c[2][3][4];` The base of the array is memory location 500. Draw a neat memory diagram of the array showing the addresses of the elements `c[0][1][0]`, `c[0][2][0]`, `c[1][1][0]`.

Sol:



4) An array is declared: `struct {int a; char c[4]; } s[2][3];` The base of the array is memory location 1000. Draw a neat memory diagram of the array. What is the total space occupied by the array?

Sol:

Total space = (4+4x1)x2x3 = 48 bytes

1000	1004	1008		1016		1024		1032		1040	1044	1048
s[0][0].a	s[0][0].c	s[0][1].a		s[0][2].a		s[1][0].a		s[1][1].a		s[1][2].a	s[1][2].c	

5) The file `marks.list` contains the marks of students in a batch. The information for each student is on one lines: his/her name followed by his/her marks, separated by `':'`. Eg:

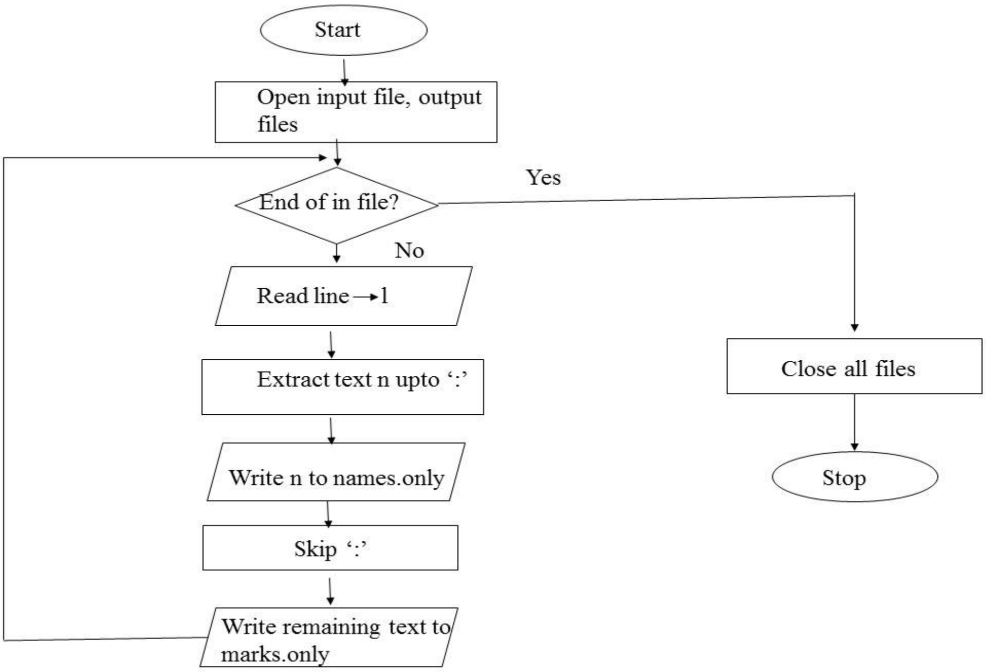
```
A.N. Aardvark:43
Eager Beaver:98
```

Design a program that read `marks.list` and creates two output files, `marks.only` and `name.only`. These contain only the `marks` and `name` respectively, all on one line separated by `':'`. There is no `':'` after the last entry on the line. Eg:

marks.only – `43:98`
name.only – `A.N. Aardvark:Eager Beaver`

(a) Draw a neat flow-chart for the program
(b) Write pseudo-code corresponding to the flow-chart. Write a serial number for each line.
(b) Convert the pseudo-code to C code. Indicate the serial numbers from (b) in comments.
(c) Modify the C code to avoid the trailing `':'` in the output files.

a)



b)

```
1.    Open file marks.list as inf
2.    Open files names.only and marks.only as nout and mout
3.    While not EOF inf
3.1       Read line → l
3.2       Extract text upto ':' → n
3.3       Write n to nout
3.3       Skip ':'
3.4       Write remaining text in l to mout
4.    Close all files.
```

c)

```
#include <stdio.h>
#include <string.h>
#define MAX_NAME 256            // Large enough to handle any name

int main()
{
  char name[MAX_NAME];
  int i, mark;
  FILE *inFile,*outName,*outMark;
                                // For clarity, error-checking of fopen() is not shown
  inFile  = fopen("marks.list", "r");
  outName = fopen("name.only",  "w");
  outMark = fopen("marks.only", "w");

  while(!feof(inFile))
  {
    if (fscanf(inFile,"%[^:]:%d", name, &mark) == 2)
    {
       fprintf(outName,"%s:", name);
       fprintf(outMark,"%d:", mark);
    }
  }
  fprintf(outRoll, “\n”);
  fprintf(outMark, “\n”);
  fclose(inFile);
  fclose(outRoll);
  fclose(outMark);
}
```

6) It is desired to read an integer from a file `input.data` into the variable `n`. C has several I/O mechanisms and functions that could be used for this purpose. Give 6 different methods (C code and/or shell command) that equivalently accomplish this purpose.

Assume the program name is `myprog.c` and the executeable is `myprog`.

1. `inf = fopen(“input.data”, ...) and fscanf(inf, “%d”, &n)`

2. `inf = fopen(“input.data”, ...)` ... `fgets(inf, s), n = atoi(s)`

3. `inf = fopen(“input.data”, ...)` ... `fgets(inf, s), sscanf(s, “%d”, &n)`

4. I/O redirection on command-line: `$ myprog < input.data` and `scanf(“%d”, &n)`

5. I/O redirection on command-line: `$ myprog < input.data` and `gets(inf, s), n = atoi(s)`

6. I/O redirection on command-line: `$ myprog < input.data` and `gets(s), sscanf(s, “%d”, &n)`

Some advanced techniques include:

7. Piping: `cat input.data | myprog` and any of 4-6

8. Use `gdb` to directly modify the variable `n` in memory while `myprog` is running. This avoids the need to modify `myprog.c`