Indian Institute of Technology Mandi

IC150: Computation for Engineers

Tutorial 4 File IO

1) Fill in the blanks

   (a) The Linux OS provides a stream as an _____abstraction__ of different I/O devices.

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

   (c) The standard I/O streams for normal I/O are _stdin__ and _stdout__.

   (d) The standard output stream for error messages is _stderr_.

   (e) Opening a file in _"w" or "w+"_ mode destroys the existing contents of the file.

2) The file `marks.list` contains the marks of students in a batch. The information for each student is on two lines: the first line contains his/her name, the 2nd line contains his/her marks. Eg:

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

Write a C 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`

```
#include <stdio.h>
#include <string.h>

int main()
{
char name[1000];          // Large enough to handle any name
int i, mark;
int isFirst = 0;          // Print ':' before all except first
FILE *inFile,*outName,*outMark;
                          // For clarity, error-checking of fopen() is not shown
inFile  = fopen("mark.list","r");
outName = fopen("name.only","w");
outMark = fopen("marks.only","w");

while(!feof(inFile))
// Note maximum width specifier in %6s in case of invalid input
{          // For clarity, code to avoid trailing ":" at end of line is not shown
  if (fscanf(inFile,"%6s, %d", name, &mark) == 2)
    {
       fprintf(outName,"%s%s", (isFirst)?"":":", name);
       fprintf(outMark,"%s%d", (isFirst)?"":":", mark);
    }
  isFirst = 0;
}
```

```
 fprintf(outRoll, "\n");
 fprintf(outMark, "\n");
 fclose(inFile);
 fclose(outRoll);
 fclose(outMark);
}
```

3) 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 of `gdb` to directly modify the variable `n` in memory while `myprog` is running. This avoids the need to modify `myprog.c`

4) Write a program `tcalc.c` that implements a trignometric calculator. The first command line argument is the name of the function to be computed: `sin`, `cos`, `tan`. The next argument is the angle as a floating point number. The last argument is either '`r`' or '`d`' indicating that the angle is in radians or degrees respectively. In case the program is run with the wrong number of arguments it should print an error message. The computed number should be written to the display.

```
#include <stdio.h>
int main(int argc, char* argv[])
{
 float oper, res;
 char func[100];
 char unit;

 if (argc==4)
 {
   sscanf(argv[2],"%f",&oper);
   unit = argv[3][0];
   if (unit == 'd') oper = oper * 22.0/7/180;
   if (strcmp(argv[1], "sin") == 0)
       res = sin(oper);
   else if (strcmp(argv[1], "cos") == 0)
       res = cos(oper);
   else if (strcmp(argv[1], "tan") == 0)
```

```
            res = tan(oper);
        else
          {
            fprintf(stderr, “Unknown function %s\n”, argv[1]);
            return 1;
          }
        else
          {
            printf("Wrong number of command-line arguments,\n");
            printf(“expecting 3 arguments, found %d\n”, argc-1);
            return 1;
          }
        printf(“%f\n”, res);
        return 0;
    }
```

5) Design an algorithm to find the size of one or more files. The file names are given on the command line. To find the size of the file, open it in “r” mode and read the contents of the file character by character. Write pseudo-code for your algorithm. Convert this into C code.

Strategy: Open each file, read char by char until the end, counting the number of chars read.

Pseudo-code:

```
1    Let argv be the vector of cmd-line arguments
2    for each fname in argv do
2.1      f <- open fname
2.2      size <- 0
2.3      while more chars in f do
2.3.1      read a char
2.3.2      increment size
         end while
2.4      close f
2.5      print fname and size
     end for
```

C Code:

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   int ch, i, size;
  FILE *f;

  for (i=1; i < argc; i++)
    {
      size = 0;
      f    = fopen(argv[i], "r");

      while ((ch = fgetc(f)) != EOF) size++;

       printf("Size of %s is %d\n", argv[i], size);
      fclose(f);
    } //  for (..i < argc..)

  return 0;
 }
```