MA 511: Computer Programming

Lecture 15: File & command line parameters

http://www.iitg.ernet.in/psm/indexing_ma511/y10/index.html

Partha Sarathi Mandal

psm@iitg.ernet.ac.in

Dept. of Mathematics, IIT Guwahati

Semester 1, 2010-2011

Example: Data files

```
typedef struct {
        int acct_no;
        char acct_type;
        char name[80];
        float balance;
}record;
record readData(record cust);
void writeToFile(record cust);
FILE *fp1;
```

```
main(){
      int i;
      record customer;
      fp1 = fopen("file.dat", "w");
      if(fp1==NULL)
                printf("Error for opening a file\n");
      else{
            while(1){
                printf("Type 0 (zero)  to stop ");
                scanf("%d", &i);
                if(i==0) break;
                customer= readData(customer);
                 writeToFile(customer);
             }
      }
    fclose(fp1);
}
```

```
record readData(record cust){
        scanf("  %[^\n]", cust.name);
        scanf("  %d",&cust.acct_no);
        scanf("  %c",cust.acct_type);
        scanf(" %f",&cust.balance);
   return(cust);
}
void writeToFile(record cust){
        fprintf(fp1, "%s\n", cust.name);
        fprintf(fp1, "%d\n", cust.acct_no);
        fprintf(fp1, "%c\n", cust.acct_type);
        fprintf(fp1, "%.2f\n", cust.balance);
}
```

Unformatted data files

- fread fwrite: are called unformatted read write functions follows:
  - Read an entire block from data file or write the entire block to a data file.
  - Function required four arguments:
    - A pointer to the data block
    - The size of the data block
    - No of the data block being transferred
    - Stream pointer (File pointer)

```
fwrite(&customer, sizeof(record), 1, fp1);
fread(&customer, sizeof(record), 1, fp1);
```

Unformatted data files

```
typedef struct {
        int acct_no;
        char acct_type;
        char name[80];
        float balance;
}record;
record readData(record cust);
void writeToFile(record cust);
FILE *fp1;
```

```
main(){
        int i;
        record customer;
        fp1 = fopen("file.dat", "w");
        if(fp1==NULL)
                printf("Error for opening a file\n");
        else{
                while(1){
                printf("Type 0 (zero)  to stop ");
                scanf("%d", &i);
                if(i==0) break;
                customer= readData(customer);
                // writeToFile(customer);
                fwrite(&customer, sizeof(record), 1, fp1);
                strset(customer.name, ' '); //erase strings
                strset(customer.acct_type, ' ');
                }
        }
    fclose(fp1);
}
```

```
record readData(record cust){
        scanf("  %[^\n]", cust.name);
        scanf("  %d",&cust.acct_no);
        scanf("  %c",cust.acct_type);
        scanf(" %f",&cust.balance);
   return(cust);
}
void writeToFile(record cust){
        fprintf(fp1, "%s\n", cust.name);
        fprintf(fp1, "%d\n", cust.acct_no);
        fprintf(fp1, "%c\n", cust.acct_type);
        fprintf(fp1, "%.2f\n", cust.balance);
}
```

Binary Files

- We learned how to handle text file in last class, which is a default mode.
- All machine language files are binary files
  Ex: .com, .exe, .obj, .dll etc.
- File mode has to be mentioned as “rb” & “wb” in fopen command for opening a binary file [“rt” & “wt” for text file]

  fptr=fopen(“file.dat”, “rb/wb”)
- Text files can also be stored and processed as binary files but not vice versa.
- Binary files differ from text files in two ways mainly:
  - The storage of newline characters (\n)
  - The eof character

the storage of newline characters

- in text files ‘ \n ’ is stored as a single characters by user, it takes 2 bytes of storages inside memory since it’s a collection of two characters.
- In binary file it takes 1 bytes of storages inside memory.
- If we count the number of characters of a text file, each newline character contributes by one.
- If we store 10 newline characters in a text file but try to count characters of this file by opening in binary mode.
- The count will be 20.

the **eof** character

- The **eof** corresponds to the character having ASCII code 26 for text file.
- In binary files there is no such explicit **eof** character, and do not store any special character at the end of the file and their file-end is verified by using their size itself.

Storage of number in binary format

- **fprintf** stores numbers as sequence of alphabets
  - storage of 1001 in a file (text and binary both) done as sequence of 4 alphabets ‘1’, ‘0’, ‘0’, ‘1’. 4-digit number will take 4 bytes.

```
for(i=10001; i<=10100; i++)
    fprintf(fp, “%d”, i);
```

- **fwrite** will store every integer value by taking 2 bytes (independent of the number of digits).

```
for(i=10001; i<=10100;i++)
    fwrite(&i, sizeof(int), 1, fp);
```

[200 bytes (2 bytes for each 100 integer)]

Storage number in binary format

```
main(){                                  //here sizeof(int) is 4 bytes
    FILE *fp1, *fp2;
    int i;
    fp1 = fopen("fp.dat", "wb");
    fp2 = fopen("fw.dat", "wb");
                                 //printf("Size of Integer = %d\n", sizeof(int));
    if(fp1==NULL) printf("Error for opening  file fp2\n");
    else  if (fp2==NULL) printf("Error for opening file fp2\n");
        else{
            for(i=10001; i<=10100; i++){
                    fprintf(fp1, "%d", i); //fprintf(fp1, "%d\n", i);
                    fwrite(&i,  sizeof(int),  1,  fp2);
            }
        }
    fcloseall();
}
```

Verify the results with different data types and different file types and check file size in each case

adv. and disadvantage in Binary file format

- File storing in binary form save a lot of space.
- Any editor / word processor cannot read the file in binary format.
- Special program is required to read file in binary format.

Command line parameters

- Without recompiling a program its possible to pass different starting values (special arguments) in an iterations through the empty parentheses of the **main** i.e., **main()**.
- Following two arguments are generally allow most of the C version.
- The parameters to be passed to **main** from OS.

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

}
```

- argc : an integer variable.
- argv : an array of pointer to characters i.e., an array of strings.
- Each string in this array will represent a parameter that is passes to **main**.

Command line parameters

mainArg.c

```
#include <stdio.h>

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

    int i;
    printf("argc = %d\n", argc);
    for(i = 0; i<argc; ++i)
        printf("argv[%d] = %s\n", i, argv[i]);
}
```

```
$ cc mainArg.c -o mainArg
$ ./mainArg my name is Rana
argc = 5
argv[0] = ./mainArg
argv[1] = my
argv[2] = name
argv[3] = is
argv[4] = Rana
```

Command line parameters

mainArg1.c

```
#include <stdio.h>
main(int argc, char *argv[ ]){
    int i, n;
    float sum=0.0, x, term = 1.0;
    sscanf(argv[1], "%f", &x);
    sscanf(argv[2], "%d", &n);
    for(i = 1; i<=n; ++i){
        term *= x/(float)i;
        sum = sum + term;
    }
printf("x = %f, n = %d, sum = %f\n", x,n,sum);
}
```

```
$ cc mainArg1.c -o mainArg1
$ ./mainArg1 0.3 10
x = 0.300000, n = 10, sum = 0.349859
```

Command line parameters

mainArg2.c

```
#include <stdio.h>
main(int argc, char *argv[ ]){
	FILE *fp;
	char ch;
	fp = fopen(argv[1], "r");
	if(fp == NULL) printf("Error for opening a file\n");
	else{
		while(!feof(fp)){
			fscanf(fp, "%c\n",  &ch);
			printf("%c\n", ch);
		}
	}
	fclose(fp1);
}
```

output.dat

```
A
B
C
D
E
F
```

```
$ cc mainArg2.c -o mainArg2
$ ./mainArg2 output.dat
A
B
C
D
E
F
```

Command line parameters

mainArg.c

```
#include <stdio.h>
main(int argc, char *argv[]){
	FILE *fp1;
	int n, i, j=0;
	float x, sum=0.0, term = 1.0;
	fp1 = fopen(argv[1], "r");
	if(fp1==NULL) printf("Error for opening a file\n");
	else{ while(j++!=7){
		fscanf(fp1, "%d %f\n", &n, &x);
		 for(i = 1; i<=n; ++i){
			term *= x/(float)i;
			sum = sum + term;
		 }
		 printf("n = %d, x = %f, sum = %f\n", n, x,sum);
		}
	}
	fclose(fp1);
}
```

inputFile.dat

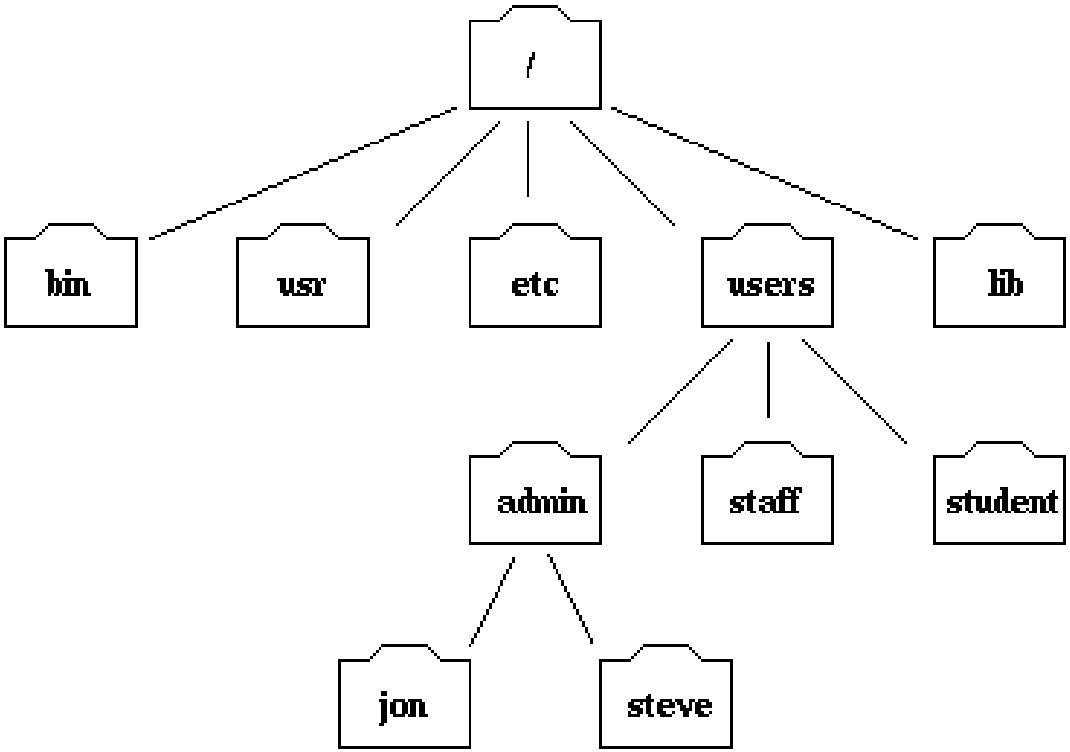
```
2 0.200000
1 0.800000
3 0.900000
2 0.100000
3 0.700000
4 0.100000
2 0.500000
```

```
$ cc mainArg.c -o mainArg
$ ./mainArg inputFile.dat
n = 2, x = 0.200000,  sum = 0.220000
n = 1, x = 0.800000,  sum = 0.236000
n = 3, x = 0.900000,  sum = 0.258824
n = 2, x = 0.100000,  sum = 0.259028
n = 3, x = 0.700000,  sum = 0.259038
n = 4, x = 0.100000,  sum = 0.259038
n = 2, x = 0.500000,  sum = 0.259038
```

File system

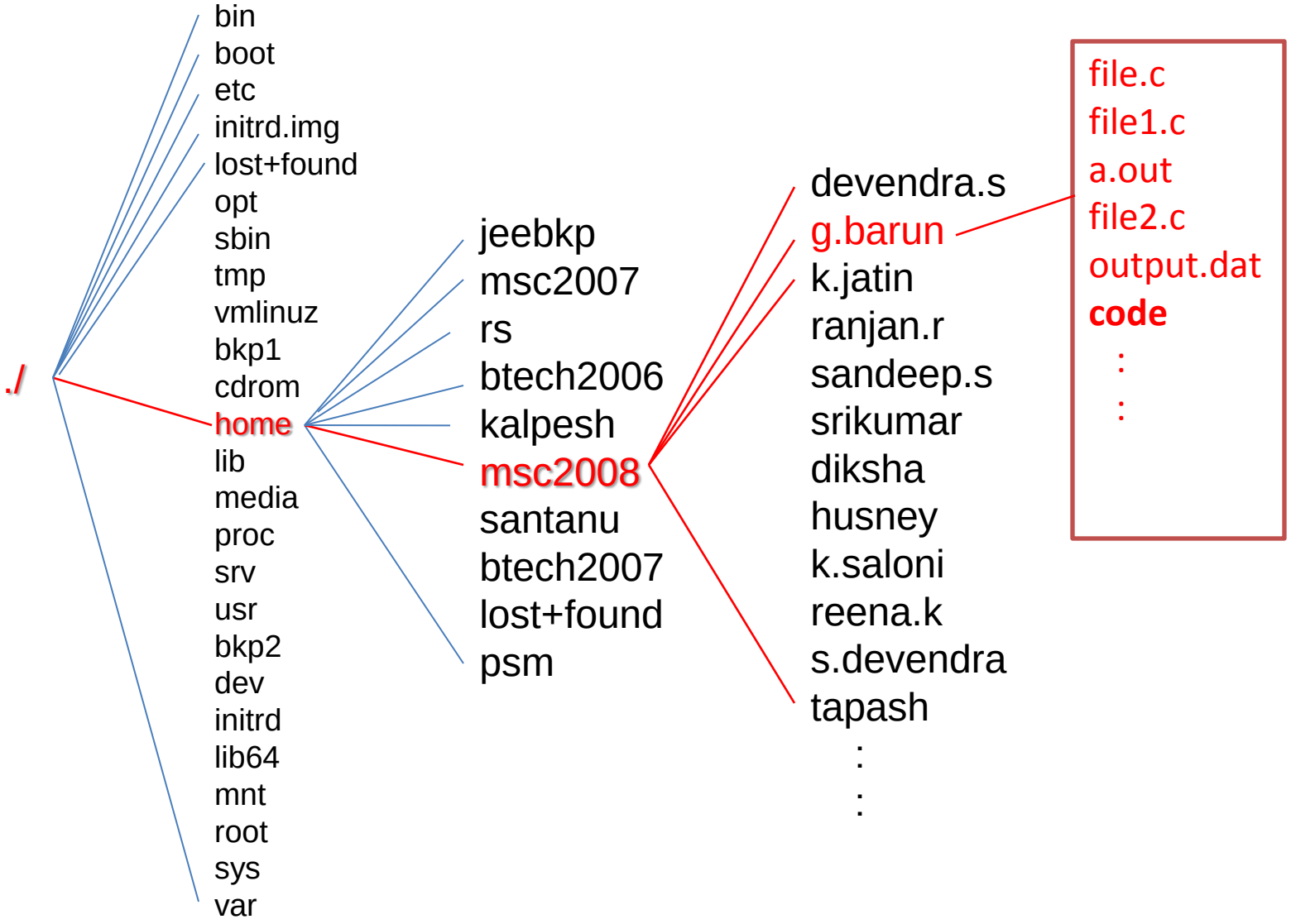
- A **file system** is a method for storing and organizing computer files.
- Make it easy to find and access them.
- Systems may use a data storage device such as a hard disk or CD-ROM.

Understanding the Root File System



Part of the filesystem tree

Understanding the Root File System



GNU/Linux Command-Line Tools Execute the command *ls -l*

To view the information on the system password database

$ ls -l /etc/passwd

The output should look similar to this:

$ -rw-r--r-- 1 root sys 41002 Apr 17 12:05 /etc/passwd

- The first 10 characters describe the access permissions.
- The first dash indicates the type of file (d for directory, s for special file, - for a regular file).
- The next three characters ("rw-") describe the permissions of the owner of the file: read and write, but no execute.
- The next three characters ("r--") describe the permissions for those in the same group as the owner: read, no write, no execute.
- The next three characters describe the permissions for all others: read, no write, no execute.

Frequent used Unix Commends

```
mandal@mandal-PC ~
$ pwd
/home/mandal
mandal@mandal-PC ~
$ ls
code
mandal@mandal-PC ~
$ ls -al
total 36
drwxrwxrwx+ 4 mandal None  4096 Sep 11 01:12 .
drwxrwxrwx+ 3 mandal None     0 Jun 23 16:56 ..
-rw-------  1 mandal None  2157 Nov 12 20:41 .bash_history
-rwxr-xr-x  1 mandal None  1150 Jun 23 10:30 .bash_profile
-rwxr-xr-x  1 mandal None  3116 Jun 23 10:30 .bashrc
-rwxr-xr-x  1 mandal None  1461 Jun 23 10:30 .inputrc
drwx------+ 2 mandal None     0 Jul  1 16:07 .ssh
drwxrwxrwx+ 3 mandal None 16384 Nov 14 14:20 code
mandal@mandal-PC ~
$ cd code
mandal@mandal-PC ~/code
$ pwd
/home/mandal/code
mandal@mandal-PC ~/code
```

$pwd
Show path to current directory
$ls
Show list of files and folder in the current directory
$ls –l
Show specification of all files and folders with permission.
$cd folder
Change directory
$cd ..
Return to previous directory
$ mkdir newfolder
Create a folder
$ rm filename
Delete file
$ cp file1 file2
Copy file1 to file2
$cp file1 folder\
Copy file1 to the folder

Combining C progs in different files

- If **a large C program** is developed by different programmer in a team then its would be preferable to store different modules (function) in <u>different files</u>.
- Compile them and test them separately and then combine these files.
- Suppose there are two files file1.c and file2.c
- cc file1.c file2.c is the UNIX command for compile

Example

file1.c

```
#include<stdio.h>


main(){
    output();
    printf(“File 1 ");
}
```

file2.c

```
void output(void){


    printf(“File 2 ");
    return;
}
```

```
$ cc fine1.c file2.c
$./a.out
$ File 2 File 1
```