ESC101N
Fundamentals of Computing

Arnab Bhattacharya
arnabb@iitk.ac.in

Indian Institute of Technology, Kanpur
http://www.iitk.ac.in/esc101/

1st semester, 2010-11

Tue, Wed, Fri 0800-0900 at L7

Files

- *Files* are used for specifying input and storing output
- A file is accessed using a pointer to a file: `FILE *`

```
FILE *fp;
```

Files

- *Files* are used for specifying input and storing output
- A file is accessed using a pointer to a file: `FILE *`

```
FILE *fp;
```

- To read or write a file, it must be first opened using `fopen`
  - `fopen` returns a file pointer
  - Two strings as parameters
  - First is the name (path) of the file
  - Second is the *mode* of operation
    - `r` for reading: error if file does not exist
    - `w` for writing: file created if does not exist, overwritten if exists
    - `a` for appending: file created if does not exist, contents preserved if exists

```
fp = fopen(``fin.txt'', ``r'');
```

Files

- *Files* are used for specifying input and storing output
- A file is accessed using a pointer to a file: `FILE *`

```
FILE *fp;
```

- To read or write a file, it must be first opened using `fopen`
  - `fopen` returns a file pointer
  - Two strings as parameters
  - First is the name (path) of the file
  - Second is the *mode* of operation
    - `r` for reading: error if file does not exist
    - `w` for writing: file created if does not exist, overwritten if exists
    - `a` for appending: file created if does not exist, contents preserved if exists

```
fp = fopen(``fin.txt'', ``r'');
```

- After finishing operations on a file, it must be closed using `fclose`

```
fclose(fp);
```

Input/output using files

- Analogous to `scanf` and `printf`, for file operations, there are `fscanf` and `fprintf`
- Parameters are same except an extra parameter in the beginning
- The first parameter is the file pointer

```
fscanf(fp, ``%d'', &n);
fprintf(fp, ``%d'', n);
```

Input/output using files

- Analogous to `scanf` and `printf`, for file operations, there are `fscanf` and `fprintf`
- Parameters are same except an extra parameter in the beginning
- The first parameter is the file pointer

```
fscanf(fp, ``%d'', &n);
fprintf(fp, ``%d'', n);
```

- `getc` and `putc` are analogous to `getchar` and `putchar` but require an extra file pointer argument

```
char c = getc(fp);
putc(c, fp);
```

Input/output using files

- Analogous to `scanf` and `printf`, for file operations, there are `fscanf` and `fprintf`
- Parameters are same except an extra parameter in the beginning
- The first parameter is the file pointer

```
fscanf(fp, ``%d'', &n);
fprintf(fp, ``%d'', n);
```

- `getc` and `putc` are analogous to `getchar` and `putchar` but require an extra file pointer argument

```
char c = getc(fp);
putc(c, fp);
```

- To check whether the file is finished (while reading input), the function `feof` is used
- The following code keeps on storing integers in the array `a` till the file is finished

```
i = 0;
while (!feof(fp))
  fscanf(fp, "%d", &a[i++]);
```

Input file

```
3
4    -7   5
2    5    1
1    6    -8
First    Last
19.34
char12
```

File input/output I

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

int main()
{
    FILE *ifp, *ofp;
    int n;
    int a[3][3];
    char str[2][10];
    double d;
    char c[10];
    int i, j;
    int count;

    ifp = fopen("fin.txt",  "r");
    ofp = fopen("fout.txt", "w");

    fscanf(ifp, "%d", &n);
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            fscanf(ifp, "%d", &a[i][j]);
    for (i = 0; i < 2; i++)
        fscanf(ifp, "%s", str[i]);
    fscanf(ifp, "%lf", &d);
    i = 0;
    while (!feof(ifp))
    {
        fscanf(ifp, "%c", &c[i]);
        i++;
    }
```

File input/output II

```
    count = i - 1;

    printf("Number of characters read = %d\n", count);

    fprintf(ofp, "%d\n", (n + 2));
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
            fprintf(ofp, "%d\t", a[j][i]);
        fprintf(ofp, "\n");
    }
    for (i = 0; i < 2; i++)
        fprintf(ofp, "%s ", strcat(str[i], "."));
    fprintf(ofp, "\n");
    fprintf(ofp, "%lf\n", -d);
    i = 0;
    while (i < count)
    {
        fprintf(ofp, "%c", c[i]);
        i++;
    }
    fprintf(ofp, "\n");

    fclose(ifp);
    fclose(ofp);
}
```

Output file

```
5
4    2    1
-7   5    6
5    1    -8
First. Last.
-19.340000

char12
```

Command line arguments

- Arguments can be passed to a program when executing it
- Arguments are supplied as strings *after* the program name
- These are specified as parameters of the `main` function

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

- `argc` denotes the number of arguments passed
- `argv` is the array that stores all the arguments as *strings*
- Only `char *` or strings can be passed
- If numbers are needed, they must be appropriately converted
  - Use `atoi` and `atof` functions from `stdlib.h`

Command line arguments

- Arguments can be passed to a program when executing it
- Arguments are supplied as strings *after* the program name
- These are specified as parameters of the `main` function

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

- `argc` denotes the number of arguments passed
- `argv` is the array that stores all the arguments as *strings*
- Only `char *` or strings can be passed
- If numbers are needed, they must be appropriately converted
  - Use `atoi` and `atof` functions from `stdlib.h`
- The first argument `argv[0]` is the program name (e.g., `a.out`)
- Command line arguments are useful for
  - Passing filenames (think of `gcc`)
  - Running multiple programs in a batch
  - Specifying a set of parameters or options
  - Error checking on correct number (and type) of parameters

Command line arguments

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i;
    int n;
    float f;
    double d;

    printf("The number of arguments passed is %d\n", argc);

    printf("The arguments passed are:\n");
    for (i = 0; i < argc; i++)
        printf("%s\n", argv[i]);

    n = atoi(argv[1]);
    f = atof(argv[2]);
    d = atof(argv[3]);

    printf("%d\t%f\t%lf\n", n, f, d);
}
```

Copying a file: simulating cp

```c
#include <stdio.h>

void filecopy(FILE *fp1, FILE *fp2)
{
    char c;
    while ((c = getc(fp1)) != EOF)  // when file ends, EOF is read
        putc(c, fp2);
}

int main(int argc, char *argv[])
{
    FILE *fp1, *fp2;

    if (argc != 3)  // error handling
    {
        printf("Program requires two file arguments\n");
        return -1;  // if error, terminate
    }

    fp1 = fopen(argv[1], "r");  // directly using
    fp2 = fopen(argv[2], "w");  // arguments as filenames

    filecopy(fp1, fp2);

    fclose(fp1);
    fclose(fp2);
}
```