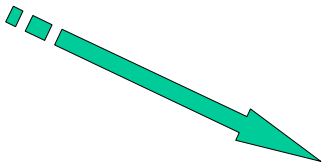
Lecture 14



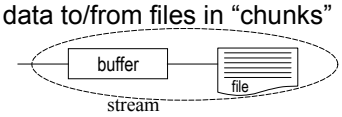
1. Introduction
2. Binary Representation
3. Hardware and Software
4. High Level Languages
5. Standard input and output
6. Operators, expression and statements
7. Making Decisions
8. Looping
9. Arrays
10. Basics of pointers
11. Strings
12. Basics of functions
13. More about functions
14. Files
14. Data Structures
16. Case study: lottery number generator

What is a file?

- A file is data stored in secondary storage
- This is usual a disk (hard/floppy/optical etc) connected to the computer bus by means of some interface
- Accessing data in files is much slower than from memory, but files are more permanent and can be larger

Streams in C

- In C each file is associated with a buffer to form a stream
- A buffer is simply an area of memory that is used for temporary storage
- This is because it is more efficient to move data to/from files in "chunks"



Streams in C

- To use a file we must first open a stream, when we are finished the stream must be closed
- When <stdio.h> is included in a program, 3 streams are automatically opened
  - stdin, stdout, stderr
- They are special in that they are normally connected to the keyboard and screen rather than a disk file however as weve already seen they can be redirected
- When the program has finished running these streams are automatically closed

Directories, paths and filenames

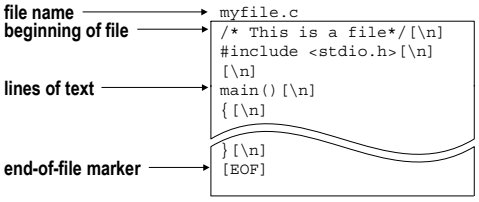
- Most operating systems organise secondary storage in a tree structure using directories.
- A directory is a file containing the names of other files and where they are stored
- Each file is identified by a filename
- Its location is given by a path
- e.g. `/usr/students/1996/A.Person/C/myfile.c`

root of directory tree — director — filename

- If "C" is the current directory we can refer to the file simply as "myfile.c" or we can give the full path

File Structure

- There are 2 sorts of file
  - text files which use ASCII codes to store text data
  - binary files which are used to store raw binary data
- In this course we will only deal with text files



Files in C

- In C, a stream is identified with a pointer-to-FILE datatype
- Here FILE is a special data type defined in `<stdio.h>`

```
FILE *pfile; /* pointer-to-FILE, or stream */
```

- We then have to point it at a particular file

```
pfile=fopen("myfile.c","r"); /* r stands for read */
```

- Function `fopen` is declared in `<stdio.h>` by the prototype

```
FILE* fopen(cons char* filename, const char* mode);
```

Pointer to file — A String — A String

- There are several other functions for file operations see notes pg 73

file1.c

```
/* Example: simple file operations */
/* No error checking is done, so the program may fail
All practical programs should check file operations. */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  FILE *pf;  /* file pointer, or stream */
  char str[40];
  int n;
  char c;

  pf = fopen("myfile.old", "w");      /* open a file for writing */
  fputs("Hello world!\n", pf);        /* write to the file */
  fclose(pf);                         /* close the file */

  rename("myfile.old", "myfile.new"); /* rename the file */

  pf = fopen("myfile.new", "a");      /* open file for appending */
  fprintf(pf, "Hello again\n");       /* write a string */
  fprintf(pf, "%d %f\n", 123, 3.1);   /* write numbers */
  fputc('A', pf); fputc('\n', pf);    /* write characters */
  fclose(pf);                         /* close the file */

  pf = fopen("myfile.new", "r");      /* open file for reading */
  fgets(str, 40, pf);                 /* read a line */
  printf(str);                        /* print it to the screen */

  rewind(pf);                         /* rewind file to start */
  fgets(str, 40, pf);                 /* read a line */
  printf(str);                        /* print it to the screen */

  fscanf(pf, "%s", str);              /* read a word */
  printf(str);
  putchar('\n');
  fscanf(pf, "%s", str);              /* ...another word */
  printf("%s\n", str);

  fscanf(pf, "%i", &n);               /* ...a number */
  printf("%i\n", n);

  c = fgetc(pf); putchar(c);          /* ...some characters */
  c = fgetc(pf); putchar(c);
  c = fgetc(pf); putchar(c);

  fclose(pf);                         /* close the file */
  remove("myfile.new");               /* delete the file */

  return EXIT_SUCCESS;
}
```

"Bomb-Proofing" file Operations

- Unfortunately there are many things that can go wrong.
  - Suppose we try to open a file for writing (which generally creates a new file) but the file already exists with read-only permission.
  - In this case the `fopen` will return `NULL`
  - We cant write to a `NULL` stream
  - To avoid the program crashing we should always check the result of file operations

file2.c

```
/* Example: standard file operations, with error checking */
/* All practical programs should check file operations. */
#include <stdio.h>
#include <stdlib.h>
#define STR_SIZE 40
int main(void)
{
  FILE *pf;  /* file pointer, or stream */
  char str[STR_SIZE], name[] = "myfile.tmp";

  /* open a file for writing: */

  pf = fopen(name, "w");  /* pf is NULL if an error occurs */
  if (pf == NULL)
  {
    fprintf(stderr, "Error opening file %s for writing\n", name);
    exit(EXIT_FAILURE);
  }

  fputs("Hello world!\n", pf);      /* write to the file */
  fclose(pf);                       /* close the file */

  /* open a file for reading: */

  pf = fopen(name, "r");  /* pf is NULL if an error occurs */
  if (pf == NULL)
  {
    fprintf(stderr, "Error opening file %s for reading\n", name);
    exit(EXIT_FAILURE);
  }

  fgets(str, STR_SIZE, pf);         /* read a line */
  printf(str);                      /* print it to the screen */
  fclose(pf);                       /* close the file */

  return EXIT_SUCCESS;
}
```

It is useful to send error messages to stderr rather than stdout
If stdout is redirected, stderr still connects to the screen

Copying a file

- This is like the unix `cp` command. We want to be able to enter and copy myfile.c to another
- Poor program with no bomb proofing. So lets improve on it

copy1.c

```
/* Example: copying a file */

/* Like Unix's cp command. When compiled, rename a.out to copy,
then use as: copy filename newname */

/* Warning: THIS SKELETON VERSION CONTAINS NO BOMB-PROOFING! */

#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *poldfile, *pnewfile;
  int byte;

  poldfile = fopen(argv[1], "r");  /* open existing file */
  pnewfile = fopen(argv[2], "w");  /* open new file */

  while ((byte = fgetc(poldfile)) != EOF)
    fputc(byte, pnewfile);
}
```

copy2.c

```
/* Example: copying a file */

/* Like Unix's cp command. When compiled, rename a.out to copy,
then use as: copy filename newname */

/* This version has pretty good bomb-proofing */

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

int main(int argc, char *argv[])
{
  FILE *poldfile, *pnewfile;

  /* check the arguments: */

  if (argc != 3) /* argv[0] is the program's name */
  {
    fprintf(stderr, "Error: 2 arguments expected\n");
    exit(EXIT_FAILURE);
  }

  /* open the files: */

  poldfile = fopen(argv[1], "r");  /* open existing file */
  if (poldfile == NULL)
  {
    fprintf(stderr, "Error: couldn't open file %s (read)\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  pnewfile = fopen(argv[2], "w");  /* open new file */
  if (poldfile == NULL)
  {
    fprintf(stderr, "Error: couldn't open file %s (write)\n", argv[2]);
    exit(EXIT_FAILURE);
  }

  /* do the copying: */

  while (1)
  {
    int check;

    byte = fgetc(poldfile);
    if (byte == EOF) /* copying completed */
      break;

    check = fputc(byte, pnewfile);
    if (check == EOF)  /* write error */
    {
      fprintf(stderr, "Error: file %s (write) \n
      exit(EXIT_FAILURE);
    }
  }

  /* notify the copy: */

  fclose(pnewfile);
  pnewfile = fopen(argv[2], "r");
  if (pnewfile == NULL)
  {
    fprintf(stderr, "Error: couldn't open file %s (read)\n", argv[2]);
    exit(EXIT_FAILURE);
  }

  while (1)
  {
    byte = fgetc(pnewfile);
    if (byte == EOF) /* verification completed */
      break;

    if (byte != fgetc(poldfile)) /* copy error */
    {
      fprintf(stderr, "Error: file %s is corrupt\n", argv[2]);
      exit(EXIT_FAILURE);
    }
  }

  return EXIT_SUCCESS;
}
```

Example: Student Marks File

- First well create a file of student names and marks

```
SMITH, Jane, mark = 47
JONES, Blodwyn, mark = 73
BLOGGS, Frederick, mark = 27
GASCOIGNE, Paul, mark = 65
BLAIR, Anthony, mark = 68
MAJOR, John, mark = 70
ASHDOWN, Patrick, mark = 54
```

marks1.c

```
/* Example: writing data to a file */

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

#define DATA "class.list"

int main(void)
{
  FILE *pf;
  char surname[7][20] = {"SMITH", "JONES", "BLOGGS", "GASCOIGNE",
                         "BLAIR", "MAJOR", "ASHDOWN"};
  char forename[7][20] = {"Jane", "Blodwyn", "Frederick", "Paul",
                          "Anthony", "John", "Patrick"};
  int mark[7] = {47, 73, 27, 65, 68, 70, 54};
  int s;

  pf = fopen(DATA, "w");  /* open new file */
  if (pf == NULL)
  {
    fprintf(stderr, "Error: couldn't open file %s\n", DATA);
    exit(EXIT_FAILURE);
  }

  /* write the data to file: */

  for (s = 0; s < 7; s++)
  {
    fprintf(pf, "%s, ", surname[s]);
    fprintf(pf, "%s, ", forename[s]);
    fprintf(pf, "mark = %i\n", mark[s]);
  }

  fclose(pf);
  fputs("Done.\n", stderr);
  return EXIT_SUCCESS;
}
```

marks2.c

```
/* Example: reading data from a file */
/* We know the number of lines in the file */

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

#define DATA "class.list"
#define CLASS 7

int main(void)
{
  FILE *pf;
  char surname[CLASS][20];
  char forename[CLASS][20];
  int mark[CLASS];
  int s;

  pf = fopen(DATA, "r");  /* open file to read */
  if (pf == NULL)
  {
    fprintf(stderr, "Error: couldn't open file %s\n", DATA);
    exit(EXIT_FAILURE);
  }

  /* read data from the file: */

  for (s = 0; s < 7; s++)
    fscanf(pf, "%s %s %i", &surname[s], &forename[s], &mark[s]);

  fclose(pf);

  /* print data to stdout: */

  for (s = 0; s < 7; s++)
  {
    printf("%s ", forename[s]);
    printf("%s ", surname[s]);
    printf("got %i%%\n", mark[s]);
  }

  return EXIT_SUCCESS;
}
```

marks3.c

```
/* Example: reading data from a file */
/* The number of lines in the file is unknown */

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

#define DATA "class.list"
#define MAX_CLASS 10

int main(void)
{
  FILE *pf;
  char surname[MAX_CLASS][40];
  char forename[MAX_CLASS][40];
  int mark[MAX_CLASS];
  int s, lines;

  pf = fopen(DATA, "r");  /* open file to read */
  if (pf == NULL)
  {
    fprintf(stderr, "Error: couldn't open file %s\n", DATA);
    exit(EXIT_FAILURE);
  }

  /* read data from the file: */

  for (s = 0; !feof(pf); s++) /* stop at end of file */
  {
    if (s >= MAX_CLASS)
    {
      fputs("Error: too many lines in file\n", stderr);
      exit(EXIT_FAILURE);
    }
    fscanf(pf, "%s %s %i", &surname[s], &forename[s], &mark[s]);
  }
  lines = s - 1;
  fclose(pf);

  /* print data to stdout: */

  for (s = 0; s < lines; s++)
    printf("%s %s got %i%%\n", forename[s], surname[s], mark[s]);

  return EXIT_SUCCESS;
}
```