Warning: These notes are not complete, it is a Skelton that will be modified/add-to in the class. If you want to us them for studying, either attend the class or get the completed notes from someone who did

CSE2301

File Access

Random Numbers

Testing and Debugging

These slides are based on slides by Prof. Wolfgang Stuerzlinger at York University

Introduction

- In this part, we introduce
  - Testing and debugging
  - Errors
  - I/O
  - Time and date
  - Random number generation

Streams and Files

- **Stream**: any source of input or any destination for output.
- Files, but could be also devices such as printers or network ports.
- Accessing streams is done via *file pointer* that is of type `FILE *`.
- Standard streams `stdin, stdout, stderr.`

Files

- You must open the file before you read or write to it (what about stdin, …).
- The system checks the file, and returns a small non-negative integer known as **file descriptor**, all reads and writes are through this file descriptor.
- 0,1,2 are reserved for stdin, stdout, and stderr.

Files

- `FILE *fp1;`
- `FILE *fopen(char *name, char *mode)`
- `fp1=fopen(name, mode);`
- **Do not assume file will open, always check for a null pointer.**
- Name is a character string containing the name of the file, mode is a character string to indicate how the file will be used
- Mode could be “r”, “w”, “a”, “r+”, ...

Files

- To read or write characters from a file
- `int fgetc(FILE *fp);`
- Returns a byte from a file, or EOF if it encountered the end of file
- `int fputc(int c, FILE *fp);`
- Writes the character c to the file (where to write it?)
- Be aware of “\” in the file name it might be treated as escape char. use “/”, or “\” ”\”

opening a file

```
FILE *fp
fp = fopen(“name”, “r”);
if(fp == NULL) {printf (…); exit }
```

- …..
- OR

```
if((fp=fopen(NAME,”r”) == NULL)
{..}
```

Character I/O

- putchar(ch) writes one char to stdout
- fputc(ch, fp) writes ch to fp (same for putc)
- putc is usually implemented as a macro or function, fputc is a function.
- putchasr is defined as
- #define putchar(c) putc((c, stdout)
- If error, return EOF

Character I/O

- int fgetc(FILE *);
- int getc(FILE *);
- int getchar(void); /* from stdin */
- int ungetc(int c, FILE *fp);
- Read char is unsigned char converted to int (must be int for EOF to work properly).

```
while((ch = getc(fp) ) != EOF {
      bla bla bla
}
```

Line I/O

- int fputs(const char * s, FILE *fp);
- int puts(const char * s);
- puts adds a newline char after s, fputs doesn’t.
- Both return EOF in case of error

Line I/O

```
char *fgets(char * s, int n, FILE *fp);
char *gets(char * s);
```

- gets reads character till a new line (discards)
- fgets reads characters till a newline or n-1 characters. if newline is read, it is added to the string.

Block I/O

```
size_t fread(void * ptr, size_t
size, size_t nmemb, FILE *fp);
size_t fwrite(void * ptr, size_t
size, size_t nmemb, FILE *fp);
```

- The function reads `nmemb` elements of data each is `size` bytes long from the stream pointed to by `fp` and returns the actual number of items successfully read.

Position in Files

- int fseek(FILE *stream, long offset, int whence);
- The fseek() function shall set the file-position indicator for the stream pointed to by stream. If a read or write error occurs, the error indicator for the stream shall be set and fseek() fails.
- The new position, measured in bytes from the beginning of the file, shall be obtained by adding offset to the position specified by whence. The specified point is the beginning of the file for SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for SEEK_END.

Position in File

- some problems when dealing with text files.
- See example in the lecture.

Formatted I/O

- we can use fprintf and fscanf with the first parameter a file pointer.
- Error?

Formatted I/O

- for scanf and fscanf, error may be
- *End-of-file* `feof(fp)` returns a non-zero value
- *Read error* `ferror(fp)` returns a non-zero value
- *A matching error*, neither of the above two indicators returns a non-zero.

I/O

- size_t fread(void *restrict ptr, size_t size, size_t ntimes, FILE *restrict stream)
- size_t fwrite(const void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);

Random Numbers

- #include <stdlib.h>
- Int rand(void)
- Returns a random number between 0 and RAND_MAX
- Each time the program runs the function returns the same sequence
- Important in debugging, but sometimes we want to return random numbers every time we run the program
- drand48() uses a much more elaborate random number generator use **srand(seed) first**

Random Numbers

- void srand(unsigned int seed)
- Seeds the random number generator
- For a truly random number that will not be repeated every time you run the program open and read from /dev/random or /dev/urandom

Random Numbers

```
FILE *fp1;
int c;
  fp1=fopen(“/dev/urandom”,”r”);
  c=getc(fp1);
  printf(“%d\n”, (int)c);
  fclose(fp1);
```

Time

- In time.h
- time_t time(time_t *x)
- Returns the number of seconds from jan. 1 1970
- If x is not NULL, the value is stored in the variable pointed to by x
- Could be used to seed the RNG
- srand((unsigned int) time(NULL));

Testing

- You wrote your program, compiled it (correcting syntax errors), ran it, and tested it, but it failed, what to do?
- There is a bug somewhere, to remove it we have to know where is it first.
- After finding it, remove/correct it changing as little code as possible to minimize introducing new bugs.

Testing/Debugging

- Testing to find out as many things about the problem as possible
- Try to isolate the bug, what caused it
- Correct it. After that
  - Test to see if the problem is solved
  - Every thing that worked before is working
  - No new errors are introduced

Instrument your code

- Insert debugging statements.

```
i=f(j);
 printf("f.in  %d  out  %d  \n",i,j);
```

- That shouldn't be in the final version
- You have to remove it after debugging, that may cause extra errors if you are not very careful in removing it

Instrument your code

- You can use conditional compilation

```
#define DEBUG 1
……….
 i=f(j);
 #if DEBUG
 printf("f:in  %d  out  %d\n",i,j);
 #endif
```

Debugging

```
 main( ) {
….
 printf(" before function 1………..");
function1( … )
 printf(" After function 1………..");

 printf(" before function 2………..");
function2 (….)
 printf(" After function 3………..");

 printf(" before function 3………..");
function3 (…..)
```

Crashes, where is the problem

Examples

- 2 examples on the use of gdb

Debugger

- **Next** Advance one statement, do not step into calls.
- **Step** Advance one statement in the program
- **List** show your program
- **Run** run
- **Break** line number, function name, fun:line

Debugger

- **Print**: variable
- **X**: contents of an address
- **Backtrace:**The "***backtrace***" command tells gdb to list all the function calls (that leads to the crash) in the stack frame.
- del [n] delete break point number n

Common Crash Causes

- Unaligned memory access (access to more than one byte must be aligned at a particular size), depends on the CPU.
- Using uninitialized pointers
- Going outside the array

Writing Good Code

- Debugging tools are no substitute to good programming practice.
- Be very careful with pointers and memory allocation and de-allocation
- Good modular design.
- Clear logic, boundary condition testing, assertions, limited global data, …
- Assigning responsibility

Core files

- When a program dumps core, it creates a dump files called “`core`” in the current directory.
- The file is an image of memory, registers including stack and data at time of crash.
- Can use gdb program core
- Some extra tools for memory dmalloc, electric fence, bcheck, valgrind

Errors

- #include <errno.h>
- Defines an int errno and some other constants
- For example, math function (in case of error) sets errno to ERANGE or EDOM
- An errno of 0, means no error
- errno is not reset by itself, you must reset it.

Example

```
#include <stdio.h>
#include <errno.h>
#include <math.h>
#define POSITIVE 25
#define NEGATIVE -25

int main()
{
double ret;

errno = 0;
ret = sqrt(NEGATIVE);
if (errno == EDOM) /*EDOM
    Signifies Domain Error*/
printf("Domain Error : Invalid Input
    To Function\n");
else
printf("Valid Input To Function\n");

errno = 0;
ret = sqrt(POSITIVE);

if (errno == EDOM)
printf("Domain Error : Invalid Input
    To Function\n");
else
printf("Valid Input To Function\n");

return 0;
}
```

Source : http://www.daniweb.com/code/snippet614.html

Example

```
• #include <stdio.h>
• #include <errno.h>
• #include <math.h>
• void test(double value) {
• double ret;
• errno = 0;
• ret = sqrt(value);
•  if ( errno )
•   {
•     perror("sqrt");
•   }
• else
•   {
•      printf("ret = %g\n", ret);
•   }
• }
• int main() {
•  test(-25);
•  test(25);
• }
```

sqrt: Numerical argument out of domain

Source : http://www.daniweb.com/code/snippet614.html