Outline

- Review
- Standard Library
  - <stdio.h>
  - <ctype.h>
  - <stdlib.h>
  - <assert.h>
  - <stdarg.h>
  - <time.h>

6.087 Lecture 10 – January 25, 2010

- Review

- Standard Library
  - <stdio.h>
  - <ctype.h>
  - <stdlib.h>
  - <assert.h>
  - <stdarg.h>
  - <time.h>

Review: Libraries

- linking: binds symbols to addresses.
- static linkage: occurs at compile time (static libraries).
- dynamic linkage: occurs at run time (shared libraries).
- shared libraries:
  - ld.so - locates shared libraries
  - ldconfig - updates links seen by ld.so
  - `dlopen(),dlsym(),dlclose()` - load shared libraries on demand.
- compiling static libraries: gcc,ar
- compiling shared libraries: gcc,ldconfig

Review: BTree

- generalized search tree–multiple children.
- except for root, each node can have between t and $2t$ children.
- tree is **always** balanced.
- Used in file systems, databases etc.

Review: Priority Queue

- abstract data structure: many implementations
- common implementations: heaps,bst,linked list
- elements are queued and dequeued in order of priority.
- operations:
  `peek(),insert(),extract-max()/extract-min()`

6.087 Lecture 10 – January 25, 2010

- Review

- Standard Library
  - <stdio.h>
  - <ctype.h>
  - <stdlib.h>
  - <assert.h>
  - <stdarg.h>
  - <time.h>

<stdio.h>: Opening, closing files

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

- mode can be "r"(read),"w"(write),"a"(append).
- "b" can be appended for binary input/output (unnecessary in *nx)
- returns NULL on error.

```
FILE* freopen(const char* filename,const char* mode,FILE* stream)
```

- redirects the stream to the file.
- returns NULL on error.
- Where can this be used? (redirecting stdin,stdout,stderr)

```
int fflush (FILE* stream)
```

- flushes any unwritten data.
- if stream is NULL flushes all outputs streams.
- returns EOF on error.

<stdio.h>: File operations

```
int remove(const char* filename)
```

- removes the file from the file system.
- retrn non-zero on error.

```
int rename(const char* oldname,const char* newname)
```

- renames file
- returns non-zero on error (reasons?: permission, existence)

<stdio.h>:Temporary files

`FILE* tmpfile(void)`

- creates a temporary file with mode "wb+".
- the file is removed **automatically** when program terminates.

`char* tmpnam(char s[L_tmpnam])`

- creates a string that is not the name of an existing file.
- return reference to internal static array if s is NULL. Populate `s` otherwise.
- generates a new name every call.

<stdio.h>: Raw I/O

```
size_t fread(void* ptr,size_t size,size_t nobj,FILE* stream)
```

- reads at most `nobj` items of size `size` from stream into ptr.
- returns the number of items read.
- `feof` and `ferror` must be used to test end of file.

```
size_t fwrite(const void* ptr,size_t size,size_t nobj,FILE* stream)
```

- write at most `nobj` items of size `size` from `ptr` onto `stream`.
- returns number of objects written.

<stdio.h>: File position

`int fseek(FILE* stream, long offset, int origin)`

- sets file position in the stream. Subsequent read/write begins at this location
- origin can be `SEEK_SET, SEEK_CUR, SEEK_END`.
- returns non-zero on error.

`long ftell (FILE* stream)`

- returns the current position within the file. (limitation? long data type).
- returns -1L on error.

`int rewind(FILE* stream)`

- sets the file pointer at the beginning.
- equivalent to fseek(stream,0L,SEEK_SET);

<stdio.h>: File errors

`void clearerr(FILE* stream)`

- clears EOF and other error indicators on stream.

`int feof(FILE* stream)`

- return non-zero (TRUE) if end of file indicator is set for stream.
- only way to test end of file for functions such as `fwrite(),fread()`

`int ferror(FILE* stream)`

- returns non-zero (TRUE) if **any** error indicator is set for stream.

<ctype.h>: Testing characters

isalnum(c)	isalpha(c) || isdigit (c)
iscntrl (c)	control characters
isdigit (c)	0-9
islower(c)	'a'-'z'
isprint (c)	printable character (includes space)
ispunct(c)	punctuation
isspace(c)	space, tab or new line
isupper(c)	'A'-'Z'

<string.h>: Memory functions

`void* memcpy(void* dst,const void* src,size_t n)`

- copies `n` bytes from `src` to location `dst`
- returns a pointer to `dst`.
- `src` and `dst` **cannot overlap.**

`void* memmove(void* dst,const void* src,size_t n)`

- behaves same as `memcpy()` function.
- `src` and `dst` **can overlap.**

`int memcmp(const void* cs,const void* ct,int n)`

- compares first `n` bytes between `cs` and `ct`.

`void* memset(void* dst,int c,int n)`

- fills the first `n` bytes of `dst` with the value `c`.
- returns a pointer to `dst`

<stdlib.h>:Utility

```
double atof(const char* s)
int  atoi(const char* s)
long atol(const char* s)
```

- converts character to float,integer and long respectively.

```
int rand()
```

- returns a pseduo-random numbers between 0 and RAND_MAX

```
void srand(unsigned int seed)
```

- sets the seed for the pseudo-random generator!

<stdlib.h>: Exiting

void abort(**void**)

- causes the program to terminate abnormally.

void exit (**int** status)

- causes normal program termination. The value `status` is returned to the operating system.
- 0 `EXIT_SUCCESS` indicates successful termination. Any other value indicates failure (`EXIT_FAILURE`)

<stdlib.h>:Exiting

void atexit (**void** (∗fcn)(**void**))

- *registers* a function `fcn` to be called when the program terminates normally;
- returns non zero when registration cannot be made.
- After `exit()` is called, the functions are called in reverse order of registration.

int system(**const char**∗ cmd)

- executes the command in string `cmd`.
- if cmd is not null, the program executes the command and returns exit status returned by the command.

<stdlib.h>:Searchign and sorting

```
void* bsearch(const void* key,const void* base,
    size_t n,size_t size,
    int (*cmp)(const void* keyval,const void* datum));
```

- searches `base[0]` through `base[n-1]` for `*key`.
- function `cmp()` is used to perform comparison.
- returns a pointer to the matching item if it exists and NULL otherwise.

```
void qsort(void* base,size_t n,
           size_t sz,
           int (*cmp)(const void*,const void*))!
```

- sorts `base[0]` through `base[n-1]` in ascending/descending order.
- function `cmp()` is used to perform comparison.

<assert.h>:Diagnostics

```
void assert(int expression)
```

- used to check for invariants/code consistency during debugging.
- does nothing when expression is true.
- prints an error message indicating, expression, filename and line number.

Alternative ways to print filename and line number during execution is to use: `__FILE__`, `__LINE__` macros.

<stdarg.h>:Variable argument lists

Variable argument lists:

- functions can variable number of arguments.
- the data type of the argument can be different for each argument.
- atleast one mandatory argument is required.
- Declaration:

```
int printf(char* fmt ,...); /*fmt is last named argument*/
```

```
va_list ap
```

- `ap` defines an iterator that will point to the variable argument.
- before using, it has to be initialized using `va_start`.

<stdarg.h>:Variable argument list

$\boxed{\text{va\_start(va\_list ap, lastarg)}}$

- `ap` lastarg refers to the **name** of the last named argument.
- va_start is a macro.

$\boxed{\text{va\_arg(va\_list ap, type)}}$

- each call of `va_arg` points ap to the next argument.
- type has to be inferred from the fixed argument (e.g. printf) or determined based on previous argument(s).

$\boxed{\text{va\_end(va\_list ap)}}$

- must be called before the function is exited.

<stdarg.h>:Variable argument list(cont.)

```
int sum(int num,...)
{
    va_list ap;int total=0;
    va_start(ap,num);
    while(num>0)
    {
        total+=va_arg(ap,int);
        num--;
    }
    va_end(ap);
    return total;
}

int suma=sum(4,1,2,3,4);/*called with five args*/
int sumb=sum(2,1,2); /*called with three args*/
```

<time.h>

time_t, clock_t, **struct** tm data types associated with time.

struct tm:

int tm_sec	seconds
int tm_min	minutes
int tm_hour	hour since midnight (0,23)
int tm_mday	day of the month (1,31)
int tm_mon	month
int tm_year	years since 1900
int tm_wday	day since sunday (0,6)
int tm_yday	day since Jan 1 (0,365)
int tm_isdst	DST flag

<time.h>

```
clock_t clock()
```

- returns processor time used since beginning of program.
- divide by CLOCKS_PER_SEC to get time in seconds.

```
time_t time(time_t * tp)
```

- returns current time (seconds since Jan 1 1970).
- if tp is not NULL, also populates tp.

```
double difftime(time_t t1,time_t t2)
```

- returns difference in seconds.

```
time_t mktime(struct tm* tp)
```

- converts the structure to a time_t object.
- returns -1 if conversion is not possible.

<time.h>

`char* asctime(const struct tm* tp)`

- returns string representation of the form "Sun Jan 3 15:14:13 1988".
- returns static reference (can be overwritten by other calls).

`struct tm* localtime(const time_t * tp)`

- converts **calendar time** to local time".

`char* ctime(const time_t * tp)`

- converts **calendar time** to string representation of local time".
- equivalent to sctime(locltime(tp))!

<time.h>

size_t strftime (**char**∗ s,size_t smax,**const char**∗ fmt,**const struct** tm∗ tp)

- returns time in the desired format.
- does not write more than `smax` characters into the string `s`.

%a	abbreviated weekday name
%A	full weekday name
%b	abbreviated month name
%B	full month name
%d	day of the month
%H	hour (0-23)
%I	hour (0-12)
%m	month
%M	minute
%p	AM/PM
%S	second