Outline

- Review

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

- Standard Library
  - `<stdio.h>`
  - `<ctype.h>`
  - `<stdlib.h>`
  - `<assert.h>`
  - `<stdarg.h>`
  - `<time.h>`
  - `<stdarg.h>`
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.
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)
FILE* tmpfile(\textbf{void})

- creates a temporary file with mode "wb+".
- the file is removed \textbf{automatically} when program terminates.

\textbf{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.
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

```c
void clearerr (FILE* stream)
```
- clears EOF and other error indicators on stream.

```c
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()

```c
int ferror (FILE* stream)
```
- returns non-zero (TRUE) if any error indicator is set for stream.
### `ctype.h`: Testing characters

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum(c)</td>
<td>`isalpha(c)</td>
</tr>
<tr>
<td>iscntrl (c)</td>
<td>Control characters</td>
</tr>
<tr>
<td>isdigit (c)</td>
<td>0-9</td>
</tr>
<tr>
<td>islower(c)</td>
<td>'a'-'z'</td>
</tr>
<tr>
<td>isprint (c)</td>
<td>Printable character (includes space)</td>
</tr>
<tr>
<td>ispunct(c)</td>
<td>Punctuation</td>
</tr>
<tr>
<td>isspace(c)</td>
<td>Space, tab or new line</td>
</tr>
<tr>
<td>isupper(c)</td>
<td>'A'-'Z'</td>
</tr>
</tbody>
</table>
<string.h>: Memory functions

```c
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.

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

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

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

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

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

- fills the first `n` bytes of `dst` with the value `c`.
- returns a pointer to `dst`
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 pseudo-random numbers between 0 and RAND_MAX

void srand(unsigned int seed)

• sets the seed for the pseudo-random generator!
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)
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>`: Searching and sorting

```c
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.

```c
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.
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: \texttt{\_\_FILE\_\_, \_\_LINE\_\_} macros.
Variable argument lists:

- functions can variable number of arguments.
- the data type of the argument can be different for each argument.
- at least one mandatory argument is required.
- Declaration:
  
  ```c
  int printf (char* fmt ,...); /*fmt is last named argument*/
  ```

  ```c
  va_list ap
  ```

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

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

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).

va_end(va_list ap)

- must be called before the function is exited.
<stdarg.h>: Variable argument list (cont.)

```c
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;
}
```

```c
int suma = sum(4, 1, 2, 3, 4); /* called with five args */
int sumb = sum(2, 1, 2); /* called with three args */
```
time_t, clock_t, **struct** tm data types associated with time.

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>tm_sec</td>
<td>seconds</td>
</tr>
<tr>
<td>int</td>
<td>tm_min</td>
<td>minutes</td>
</tr>
<tr>
<td>int</td>
<td>tm_hour</td>
<td>hour since midnight (0, 23)</td>
</tr>
<tr>
<td>int</td>
<td>tm_mday</td>
<td>day of the month (1, 31)</td>
</tr>
<tr>
<td>int</td>
<td>tm_mon</td>
<td>month</td>
</tr>
<tr>
<td>int</td>
<td>tm_year</td>
<td>years since 1900</td>
</tr>
<tr>
<td>int</td>
<td>tm_wday</td>
<td>day since Sunday (0, 6)</td>
</tr>
<tr>
<td>int</td>
<td>tm_yday</td>
<td>day since Jan 1 (0, 365)</td>
</tr>
<tr>
<td>int</td>
<td>tm_isdst</td>
<td>DST flag</td>
</tr>
</tbody>
</table>
<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.
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(localtime(tp))!
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 $s_{max}$ characters into the string $s$.

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