The C Standard Library
C Programming and Software Tools
N.C. State Department of Computer Science
The Standard C Library

• A small set of highly useful functions, standardized across all platforms
• Definitions are captured in 24 header files

• (Color-coding in the table on next slides
  – green = we’ve already talked about
  – red = will discuss now
  – blue = will get too soon
  – grey = skipping)
The Standard Library

<assert.h> Testing for errors and printing helpful error messages
<ctype.h> Classify characters, convert between upper and lower case
<limits.h> Defined constants specifying the implementation-specific properties of the integer types
<stdio.h> Accessing a varying number of arguments passed to functions
<stdbool.h> Defining boolean data types
<string.h> Manipulating strings
<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;errno.h&gt;</code></td>
<td>Testing error codes reported by library functions</td>
</tr>
<tr>
<td><code>&lt;math.h&gt;</code></td>
<td>Computing common mathematical functions</td>
</tr>
<tr>
<td><code>&lt;stdlib.h&gt;</code></td>
<td>Various; including conversion, pseudo-random numbers, memory allocation, process control, environment, signalling, searching, and sorting</td>
</tr>
<tr>
<td><code>&lt;stdio.h&gt;</code></td>
<td>Core input and output capabilities of the C language</td>
</tr>
</tbody>
</table>
The Standard Library (cont’d)

`<stdio.h>`  Defining various integer types

`<time.h>`  Converting between time and date formats

`<complex.h>`  Functions for manipulating complex numbers.

`<fenv.h>`  Controlling the floating-point environment

`<float.h>`  Constants specifying the implementation-specific properties of the floating-point library

`<inttypes.h>`  Precise conversion between integer types

`<iso646.h>`  Programming in ISO 646 variant character sets

`<setjmp.h>`  setjmp/longjmp for non-local exits
The Standard Library (cont’d)

`<locale.h>` Choosing and customizing for a locale

`<stddef.h>` Defining several useful types and macros

`<signal.h>` Controlling asynchronous process interaction

`<wchar.h>` Manipulating strings using wide characters - key to supporting a range of languages

`<wctype.h>` Classifying wide characters
<math.h>: Using the Math Library

• Note: to use math functions, be sure you specify `\texttt{-lm}` to \texttt{gcc} (after the source file names)

\texttt{> gcc pgmx.c -lm -o pgmx}

• Some constants

<table>
<thead>
<tr>
<th>M_E</th>
<th>The base of natural logarithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_PI</td>
<td>Value of $\pi$</td>
</tr>
<tr>
<td>M_SQRT2</td>
<td>Square root of 2</td>
</tr>
</tbody>
</table>

• See King, Chapter 23.4 for additional resources
Trigonometry

- Input type is a `double`, returns type `double`.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cos()</code>, <code>sin()</code>, <code>tan()</code></td>
<td>Input in radians</td>
</tr>
<tr>
<td></td>
<td>Return value in [-1.0, 1.0]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acos()</code>, <code>asin()</code>, <code>atan()</code></td>
<td>Input in [-1.0, 1.0]</td>
</tr>
<tr>
<td></td>
<td>Return value in [-π, π] radians</td>
</tr>
</tbody>
</table>
<math.h>  Exponents, logs, and more

• Input types **double**, returns type **double**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp(x)</td>
<td>$e^x$</td>
</tr>
<tr>
<td>exp2(x)</td>
<td>$2^x$</td>
</tr>
<tr>
<td>exp10(x)</td>
<td>$10^x$</td>
</tr>
<tr>
<td>log(x)</td>
<td>$\log_e x$</td>
</tr>
<tr>
<td>log2(x)</td>
<td>$\log_2 x$</td>
</tr>
<tr>
<td>log10(x)</td>
<td>$\log_{10} x$</td>
</tr>
<tr>
<td>pow(x, y)</td>
<td>$x^y$</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>$\sqrt{x}$</td>
</tr>
<tr>
<td>fabs(x)</td>
<td>absolute value</td>
</tr>
<tr>
<td>floor(x)</td>
<td>largest integer $\leq x$</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>smallest integer $\geq x$</td>
</tr>
</tbody>
</table>
Exercise 21a

Trig tool

• Write a program that takes an integer number of degrees on the command line and prints the cosine of the value.
  – Recall that trig functions take parameters in floating point radians
  – Recall that \( \text{radians} = \frac{\pi}{180} \text{ degrees} \)
  – You can skip error checking

\[
\begin{array}{l}
gcc -lm cos.c -o cos \\
./cos 0 \\
1.000000 \\
./cos 45 \\
0.707107 \\
./cos 90 \\
0.000000
\end{array}
\]
<errno.h>: System Error Messages

```c
void perror(const char *s)
```

Prints string `s` + an implementation-defined error message corresponding to value of the integer `errno`

- `errno` set by a previous standard library call
- `perror` is a function in `stdlib.h`

- Always set `errno` to 0 before any function call that you want to test

See King Section 24.2
• Example

```c
if ((myfile = fopen("test.txt", "r")) == NULL {
    perror("test.txt");
    exit(-1);
}
```

Output

```
> a.out
test.txt: No such file or directory
```
Errors

• Domain Error: argument outside domain
  – EDOM is stored in errno
  – Function may return NaN, but return value is implementation defined

• Range Error: result is outside range of double
  – ERANGE is stored in errno if overflow (may be stored for underflow)
  – Function returns + or – HUGE_VAL if overflow
  – Function returns 0 for underflow
<stdlib.h>: Miscellaneous

- `void abort(void), void exit(int status)`
  - terminate execution, terminate with a non-zero return code

- `void * bsearch(void * key, void *base, size_t nelems, size_t size_elem, int (*compar) (void *, void *))`
  - binary search in a sorted array starting at `base`, with `nelems` elements each of size `size_elem`, looking for the item with value `key`, using the comparison function `compar` to determine equality
• `void qsort(void *base, size_t nelems, size_t size_elem, int (*compar) (void *, void *))`  
  – sort the array starting at `base`, with `nelems` elements each of size `size_elem`, using the comparison function `compar` to determine ordering
char strings[][20] = {
    "apples",
    "grapes",
    "strawberries",
    "bananas"};
char item[20] = "strawberries";

// sort the strings
qsort(strings, 4, 20, strcmp);

// search for "strawberries"
char *pos =
    (char *) bsearch(item, strings, 4, 20, strcmp);

if (pos)
    printf("The string \"%s\" was found.\n", pos);
### <stdlib.h>: Standard size types!

<table>
<thead>
<tr>
<th>signed type</th>
<th>unsigned type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intmax_t</td>
<td>uintmax_t</td>
<td>Integer type with the maximum width supported.</td>
</tr>
<tr>
<td>int8_t</td>
<td>uint8_t</td>
<td><strong>Integer type with a width of exactly 8, 16, 32, or 64 bits.</strong></td>
</tr>
<tr>
<td>int16_t</td>
<td>uint16_t</td>
<td>For signed types, negative values are represented using 2's complement.</td>
</tr>
<tr>
<td>int32_t</td>
<td>uint32_t</td>
<td>No padding bits.</td>
</tr>
<tr>
<td>int64_t</td>
<td>uint64_t</td>
<td>Optional: These typedefs are not defined if no types with such characteristics exist.</td>
</tr>
<tr>
<td>int_least8_t</td>
<td>uint_least8_t</td>
<td>Integer type with a minimum of 8, 16, 32, or 64 bits.</td>
</tr>
<tr>
<td>int_least16_t</td>
<td>uint_least16_t</td>
<td>No other integer type exists with lesser size and at least the specified width.</td>
</tr>
<tr>
<td>int_least32_t</td>
<td>uint_least32_t</td>
<td></td>
</tr>
<tr>
<td>int_least64_t</td>
<td>uint_least64_t</td>
<td></td>
</tr>
<tr>
<td>int_fast8_t</td>
<td>uint_fast8_t</td>
<td>Integer type with a minimum of 8, 16, 32, or 64 bits.</td>
</tr>
<tr>
<td>int_fast16_t</td>
<td>uint_fast16_t</td>
<td>At least as fast as any other integer type with at least the specified width.</td>
</tr>
<tr>
<td>int_fast32_t</td>
<td>uint_fast32_t</td>
<td></td>
</tr>
<tr>
<td>int_fast64_t</td>
<td>uint_fast64_t</td>
<td></td>
</tr>
<tr>
<td>intptr_t</td>
<td>uintptr_t</td>
<td>Integer type capable of holding a value converted from a void pointer and then be converted back to that type with a value that compares equal to the original pointer. Optional: These typedefs may not be defined in some library implementations.*</td>
</tr>
</tbody>
</table>
<time.h>: Time!

- UNIX Epoch Time: Number of seconds elapsed since **midnight on January 1st, 1970**.
- A plain integer – lets you do math!
  - Time elapsed: *now − then*
  - Time in the future: *now + delay*

UNIX Epoch clock:
<time.h>: Time!

```c
int main()
{
    time_t epoch_time = time(NULL);
    printf("Epoch time: %d\n", epoch_time);

    struct tm* now = localtime(&epoch_time); // in our time zone
    printf("Manual formatting: The date is %d/%d/%d.\n",
           now->tm_mon+1, now->tm_mday, now->tm_year+1900);

    char* now_str = asctime(now); // output includes newline
    printf("asctime formatting: %s", now_str);

    char buf[128];
    strftime(buf, sizeof(buf),
             "Date: %e %B %Y, Time: %H:%M %Z", now);
    printf("strftime custom formatting: %s\n", buf);
}
```

Epoch time: 1405894768
Manual formatting: The date is 7/20/2014.
asctime formatting: Sun Jul 20 18:19:28 2014
strftime custom formatting: Date: 20 July 2014, Time: 18:19 EDT
UNIX, POSIX, and you

• UNIX: The defining operating system of the 20th century, created by Kernigan and Ritchie (the C guys)
  – Many descendants still in use today (HP-UX, Solaris, AIX, *BSD, Mac OS X, Linux)
• POSIX: A standard where a bunch of UNIXy types got together and standardized their APIs
• Even Windows supports the core of POSIX
• POSIX is not core C99, but it’s pretty much as well supported.
UNIX through time
Useful POSIX calls

#include <unistd.h>

int chdir(const char *); // Change directory

char *getlogin(void); // Find current username

char *getcwd(char *, size_t); // Find current directory

unsigned int sleep(unsigned int); // Wait time
Exercise 21b

Investigate

• Find out what the logged in user and current directory are when you run something through ideone
Any Questions?