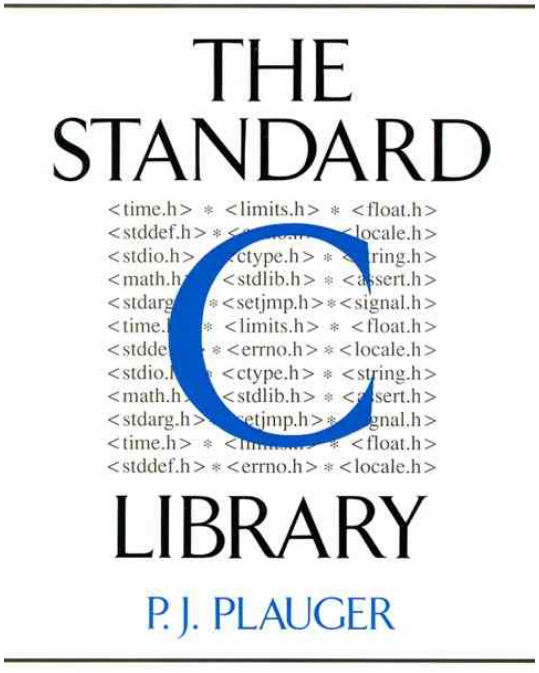
CS 241
Data Organization
Standard Libraries

Brooke Chenoweth

University of New Mexico

Fall 2014

The Standard C Library by Plauger



- Comprehensive treatment of ANSI and ISO standards for the C Library.
- Contains the complete code of the Standard C Library and includes practical advice on using all 15 headers.
- Focus on the concepts, design issues, and trade-offs associated with library building.
- Using this book, programmers will make the best use of the C Library and will learn to build programs with maximum portability and reusability.

Alternatively, Use The Internet

- `http://en.wikipedia.org/wiki/C_standard_library`
- `http://www.acm.uiuc.edu/webmonkeys/book/c_guide/`
- `http://c-faq.com/`
- Be careful! Not all information is made equal.
  - Which library? (ANSI C or something else?)
  - Which language? (Are you actually looking at C++?)
  - Does source actually make sense?

Standard Library: `stdio.h`

`stdio.h`: "**St**andard **I**nput/**O**utput **h**eader"

```
#include <stdio.h>
```

Functions defined in `stdio.h` include:

- `printf` – Formatted output to standard out stream
- `fprintf` – Formatted output to file
- `scanf` – Formatted input from standard in stream
- `getchar` – Read character from standard in stream
- `fopen` – File open
- `fclose` – File close
- `rewind` – Return to the beginning of a file

Constants defined `stdio.h` include:

- `EOF`
- `NULL`

Example: cat

```c
#include <stdio.h>

void filecopy(FILE* in, FILE* out)
{ int c;
  while ((c = getc(in)) != EOF)
  { putc(c, out);
  }
}
int main(int argc, char** argv)
{ FILE* fp;
  int i;
  if(argc == 1) filecopy(stdin, stdout);
  else
  { for(i = 1; i < argc; ++i)
    { fp = fopen(argv[i], "r");
      if(fp == NULL) return 1;
      else
      { filecopy(fp, stdout);
        fclose(fp);
      }
    }
  }
  return 0;
}
```

Using Standard C Library Functions

Include the library's `.h` file in your source code.

- The `.h` file defines as `extern` the name, return type and argument list of each "*public*" function in the library.
- A `.h` file can also define `extern` variables and constants.
- The `.h` file is used at compile time.

Compile/Link: `gcc -l`*library* `options`

Your source code will compile with references to functions and variables declared `extern`.

After your source code compiles, the linker needs to attach to the executable code for each library function you referenced.

`gcc -l`*library*

- On Unix-like systems, the rule for naming libraries is `lib`*x*`.a`, where *x* is some string.
- Link library `lib`*x*`.a` with the `gcc` option: `-l`*x*. Example:
  - Date and time functions are defined in `time.h`. Thus, to use a time function: `#include <time.h>`
  - In C, most library files end with `.a`. The library containing executable code for functions in `time.h` is `libtime.a`
  - The `gcc` option for compiling with this library is: `gcc -ltime`
- Not all libraries require this.

Standard Library: time.h

```c
#include <stdio.h>
#include <time.h>
void main(void)
{
  time_t clock = time(NULL);
  long sec = (long)clock;
  printf("Seconds since Unix Epoch: %ld\n",sec);
  printf("Current time: %s\n", ctime(&clock));
}
```

```
Seconds since Unix Epoch: 1332286966
Current time: Tue Mar 20 17:42:46 2012
```

On moons.cs.unm.edu, time_t is a long.

On moons, gcc -ltime not needed.

Standard Library: `limits.h`

The example below shows just a few of the constants defined in `limits.h`.

```
#include <stdio.h>
#include <limits.h> /* no linker lib option needed. */
void main(void)
{
   printf("%d\n", INT_MIN); /* -2147483648 */
   printf("%d\n", INT_MAX); /*  2147483647 */
   printf("%d\n", CHAR_MIN);  /*       -128 */
   printf("%d\n", CHAR_MAX);  /*        127 */
   printf("%d\n", UCHAR_MAX); /*        255 */
}
```

Standard Library: `stdlib.h`

`#include <stdlib.h>` (`gcc -l`*library* NOT needed.)

Predefined types include:

- `size_t` – size of memory blocks (`unsigned int`).

Functions include:

- `int atoi(const char *str)` – ASCII string to integer.
- `int atof(const char *str)` – ASCII string to float.
- `void *malloc(size_t size)` – Allocate memory from the heap.
- `void free(void *pointer)` – Free allocated memory to the heap.
- `void exit (short code)` – Closes files and other cleanup, then terminates program.

stdlib.h: The rand Function

```
#include <stdlib.h>

int rand(void)
```

- Generate a uniformly distributed pseudo-random value between 0 and RAND_MAX.
  - On moons.cs.unm.edu: RAND_MAX = 2,147,483,647
  - On many older machines: RAND_MAX = 32,767

```
void srand (unsigned long seed)
```

- Initializes pseudo-random number generator.
- If no seed value provided, the `rand()` function is automatically seeded with a value of 1.
- Usually, called *once and only once* in a program.

Making rand() More Useful

Generally, it is not very useful to get a pseudo-random number between 0 and RAND_MAX.
This utility function returns a uniformly distributed pseudorandom number between 0 and n-1.

```
int randomInt(int n)
{
  int r = rand(); /* r = [0,RAND_MAX] */

  /* x = [0, 1) */
  double x = (double)r / ((double)RAND_MAX + 1.0 );
  /* Without +1.0, there is a 1 in RAND_MAX */
  /* chance of returning n. */

  /* return: [0, n-1] */
  return (int)(x*n);
}
```

Using randomInt(int n)

```
void main(void)
{ int i; int bins[7];
  long seed = (long)time(NULL);
  printf("seed = %ld\n", seed);
  srand(seed);

  for (i=0; i<7; i++)
  { bins[i] = 0;
  }
  for (i=0; i<10000; i++)
  { int r = randomInt(6);
    bins[r]++;
  }

  for (i=0; i<7; i++)
  { printf("bins[%d] = %d\n", i, bins[i]);
  }
}
```

Use of this seed on moons will exactly reproduce these results.

```
seed = 1332289063
bins[0] = 1638
bins[1] = 1669
bins[2] = 1604
bins[3] = 1645
bins[4] = 1690
bins[5] = 1754
bins[6] = 0
```

Explain This Output

```
void main(void)
{ int i;
  int bins[12];
  srand((long)time(NULL));

  for (i=0; i<7; i++)
  { bins[i] = 0;
  }

  for (i=0; i<70000; i++)
  {
    int r = randomInt(6) + randomInt(6);
    bins[r]++;
  }

  for (i=0; i<12; i++)
  { printf("bins[%d] = %d\n", i, bins[i]);
  }
```

```
bins[0] = 2016
bins[1] = 3826
bins[2] = 5879
bins[3] = 7904
bins[4] = 9558
bins[5] = 11733
bins[6] = 9684
bins[7] = 7749
bins[8] = 5779
bins[9] = 3951
bins[10] = 1921
bins[11] = 0
```

string.h: strcpy & strncpy

```
char *strcpy(char *dest, const char *src)
```

- Copies characters from location src until the terminating '\0' character is copied.

```
char *strncpy(char *dest, const char *src, size_t n)
```

- The `strncpy()` function copies no more than `n` bytes of `src`. Thus, if there is no null byte among the first n bytes of `src`, the resulting `dest` will not be null-terminated.
- In the case where the length of `src` is less than `n`, the remainder of `dest` is padded with '\0'.
- Returns pointer to `dest`.

string.h: strlen

```
int strlen(const char* str)
```

Returns the number of bytes in the string to which `str` points, not including the terminating NULL byte.

```
#include <stdio.h>
int strlen(const char str[])
{ int i=0;
  while (str[i]) i++;
  return i;
}

void main(void)
{ char word[] = "Hello";
  printf("%d\n", strlen(word));
}
```

Standard Library: `math.h`

- `double pow(double x, double y)` – x raised to power y.
- `double log(double x)` – Natural logarithm of x.
- `double log10(double x)` – Base 10 logarithm of x.
- `double sqrt(double x)` – Square root of x.
- `double ceil(double x)` – Smallest integer not $< x$.
- `double floor(double x)` – Largest integer not $> x$.
- `double sin(double x)` – sin of x in radians.
- `int abs(int n)` – absolute value of n.
- `long labs(long n)` – absolute value of n.
- `double fabs(double x)` – absolute value of x.

Be careful not to use `abs` when you want `fabs`

Using C's Math Library

```
#include <math.h>
```

Including `math.h` will tell the compiler that the math functions like `sqrt(x)` exist.
The math library file name is: `libm.a`

```
gcc foo.c -lm
```

- `-lm` tells the linker to link with the math library.
- Many installations of `gcc` require using the `-lm` option in order to link with the math library.

pow(x,y): x Raised to Power y: x^y

```c
#include <stdio.h>
#include <math.h>
void main(void)
{
   double x1 = 3.1;
   double x2 = 3.6;
   printf("%f\n", pow(x2-x1, 2.0)); /* 0.250000 */
   printf("%f\n", pow(x1-x2, 2.0)); /* 0.250000 */
   printf("%f\n", pow(x2-x1, 2.5)); /* 0.176777 */
   printf("%f\n", pow(x1-x2, 2.5)); /* -nan ???? */
   printf("%f\n", pow(x2-x1, 2.0) * sqrt(x2-x1));
   /* 0.176777 */
}
```

$$x^{2.5} = x^2 x^{0.5} = x^2\sqrt{x}$$

Distance in 2D: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Could use pow, but both slower AND less accurate.

```
#include <math.h>
double dist(double x1, double y1,
            double x2, double y2)
{
  return sqrt(pow(x1-x2, 2.0) + pow(y1-y2, 2.0));
}
```

Better approach:

```
  double dx = x1 - x2;
  double dy = y1 - y2;
  return sqrt(dx*dx + dy*dy);
```

Often only need *relative* distance.

```
  double dx = x1 - x2;
  double dy = y1 - y2;
  return dx*dx + dy*dy;
```

Generalized Concept of Distance

Distance is a very important concept in computer science:

- Calculating *spatial distance* between two locations.
- Minimizing "distance" in *hue*, *saturation*, and *brightness*.
- Minimizing a weighted, "total distance" to some *high dimensional* set of objectives.
- Almost every *simulation program*, from physics to biology to finances to political interactions to games, uses some abstracted concept of distance.