Part D - Modularity

Library Functions

Implement algorithms using standard library procedures to incorporate existing technology

"Programs that confine their system interactions to facilities provided by the standard library can be moved from one system to another without change"
(Kernighan and Ritchie, 1988)

Mathematical | Time | Character | Exercises

The standard libraries that support programming languages perform many common tasks. C compilers ship with the libraries that include functions for mathematical calculations, generation of random events and manipulation and analysis of character data. To access any function within any library we simply include the appropriate header file for that libray and call the function in our source code.

This chapter introduces some of the more common functions in the libraries that ship with C compilers. The GNU Documentation includes a comprehensive description of each function in each library.

MATHEMATICAL FUNCTIONS

The mathematics related libraries that contain the more common mathematical functions are:

- **stdlib** - the standard library
- **math** - the math library

Standard Library

The header file **<stdlib.h>** contains prototypes for the functions that perform the more general mathematical calculations. These calculations include absolute values of integers and random number generation.

Integer Absolute Value

abs(), **labs()**, **llabs()** return the absolute value of the argument. Their prototypes are

```
int abs(int);
long labs(long);
long long llabs(long long);
```

The following program produces the output on the right

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

int main(void)
{
        int x = -12;
        long y = -24L;

        printf("|%d| is %d\n", x, abs(x));
        printf("|%ld| is %ld\n", y, labs(y));
        return 0;
}
```

```
|-12| is 12
|-24| is 24
```

Random Numbers

`rand()` returns a pseudo-random integer in the range `0` to `RAND_MAX`. `RAND_MAX` is implementation dependent but no less than `32767`. The prototype for `rand()` is

```
int rand(void);
```

The following program outputs the same set of 10 pseudo-random integers for each successive run:

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

int main(void)
{
        int i;

        for (i = 0; i < 10 ; i++)
                printf("Random number %d is %d\n", i+1, rand());
        return 0;
}
```

The following program outputs the same set of 10 pseudo-random integers between `6` and `100` inclusive:

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

int main(void)
{
        int i, n, a = 6, b = 100;

        for (i = 0; i < 10 ; i++) {
                n = a + rand() % (b + 1 - a);
                printf("Random number %d is %d\n", i+1, n);
        }
        return 0;
}
```

The following program outputs the same set of 10 pseudo-random floating-point numbers between `3.0` and `100.0` inclusive:

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

int main(void)
{
        int i;
        double x, a = 3.0, b = 100.0;

        for (i = 0; i < 10 ; i++) {
                x = a + ((double) rand() / RAND_MAX * (b - a));
                printf("Random number %d is %.2lf\n", i+1, x);
        }
        return 0;
}
```

`rand()` generates the same set of random numbers for every run of its host application. This is very useful during the debugging stage. To generate a different set of random numbers for every run we add a call to the function `srand()`. `srand()` sets the seed for the random number generator. The prototype for `srand()` is

```
int srand(unsigned seed);
```

unsigned is a type that only holds non-negative integer values. We call **srand()** with **time(NULL)** (see below) as the argument once before the first call to **rand()**, typically at the start of our program. This provides a unique seed for each run and hence different pseudo-random numbers.

The following program outputs a different set of 10 pseudo-random numbers with every run:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h> // prototype for time(NULL)

int main(void)
{
        int i;

        srand(time(NULL));
        for (i = 0; i < 10 ; i++)
                printf("Random number %d is %d\n", i+1, rand());
        return 0;
}
```

math (Optional)

The **math** library contains many functions that perform mathematical calculations. Their prototypes are listed in **<math.h>**. To compile a program that uses one of these function with the **gcc** compiler, we add the **-lm** option to the command line

```
gcc myProgram.c -lm
```

Floating-Point Absolute Value

fabs(), **fabsf()**, **fabsl()** return the absolute value of the argument. Their prototypes are

```
double fabs(double);
float fabsf(float);
long double fabsl(long double);
```

The following program produces the output on the right

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

int main(void)
{
        float w = -12.5;
        double x = -12.5;

        printf("|%f| is %f\n", w, fabsf(w));
        printf("|%lf| is %lf\n", x, fabs(x));
        return 0;
}
```

```
|-12.500000| is 12.500000
|-12.500000| is 12.500000
```

Floor

floor(), **floorf()**, **floorl()** return the largest integer value not greater than the argument. Their prototypes are

```
double floor(double);
float floorf(float);
long double floorl(long double);
```

For example, **floor(16.3)** returns a value of **16.0**.

Ceiling

`ceil()`, `ceilf()`, `ceill()` return the smallest integer value not less than the argument. Their prototypes are

```
double ceil(double);
float ceilf(float);
long double ceill(long double);
```

For example, `ceil(16.3)` has a value of `17.0`.

Rounding

`round()`, `roundf()`, `roundl()` return the integer value closest to the argument. Their prototypes are

```
double round(double);
float roundf(float);
long double roundl(long double);
```

For example, `round(16.3)` has a value of `16.0`, while `round(-16.3)` returns a value of `-16.0`.

Truncating

`trunc()`, `truncf()`, `truncl()` return the integer part of the argument. Their prototypes are

```
double trunc(double);
float truncf(float);
long double truncl(long double);
```

For example, `trunc(16.7)` has a value of `16.0`, while `trunc(-16.7)` returns a value of `-16.0`.

Square Root

`sqrt()`, `sqrtf()`, `sqrtl()` return the square root of the argument. Their prototypes are

```
double sqrt(double);
float sqrtf(float);
long double sqrtl(long double);
```

For example, `sqrt(16.0)` returns a value of `4.0`.

Powers

`pow()`, `powf()`, `powl()` return the result of the first argument raised to the power of the second argument. Their prototypes are

```
double pow(double base, double exponent);
float powf(float base, float exponent);
long double powl(long double base, long double exponent);
```

The following program produces the output on the right

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

int main(void)
{
        double base = 12.5;

        printf("%lf^3 is %lf\n", base,
         pow(base,3));
        return 0;
}
```

```
12.500000^3 is 1953.125000
```

This set of functions was designed for floating-point arguments and each one is computationally intensive. Avoid using them for simpler integer arguments.

Logarithms

`log()`, `logf()`, `logl()` return the natural logarithm of the argument. Their prototypes are

```
double log(double);
float logf(float);
long double logl(long double);
```

For example, `log(2.718281828459045)` returns a value of `1.0`.

Powers of e

`exp(), expf(), expl()` return the natural anti-logarithm of the argument. Their prototypes are

```
double exp(double);
float expf(float);
long double expl(long double);
```

For example, `exp(1.0)` returns a value of `2.718281828459045`.

TIME FUNCTIONS (OPTIONAL)

The `time` library contains functions that return timing data. Their prototypes are listed in `<time.h>`.

Calendar Time

time

`time(NULL)` returns the current calendar time. The prototype is

```
time_t time(time_t *);
```

`time_t` is a type that it is sufficiently large to hold time values - for example, `unsigned long`.

difftime

`difftime()` returns the difference in seconds between two calendar time arguments. The prototype is

```
double difftime(time_t, time_t);
```

`time_t` is a type that it is sufficiently large to hold time values - for example, `unsigned long`.

The following program returns the time in seconds taken to execute the central iteration

```
#include <time.h>
#include <stdio.h>
#define NITER 1000000000

int main(void)
{
        double x;
        int i, j, k;
        time_t t0, t1;

        x = 1;
        t0 = time(NULL);
        for (i = 0; i < NITER; i++)
                x = x * 1.0000000001;
        t1 = time(NULL);
        printf("Elapsed time is %.1lf secs\n",
         difftime(t1, t0));
        printf("Value of x is %.10lf\n", x);

        return 0;
}
```

```
Elapsed time is 40.0 secs
Value of x is 1.1051709272
```

Process Time

clock

clock() returns the approximate process time. The prototype is

```
clock_t clock(void);
```

clock_t is a type that holds time values - for example, **unsigned long**. The value is in units of **CLOCKS_PER_SEC**. We divide by these units to obtain the process time in seconds.

The following program returns the process time in seconds taken to execute the central iteration

```
#include <time.h>
#include <stdio.h>
#define NITER 400

int main(void)
{
        double x;
        int i, j, k;
        time_t t0, t1;
        clock_t c0, c1;

        x = 1;
        t0 = time(NULL);
        c0 = clock();
        for (i = 0; i < NITER; i++)
            for (j = 0; j < NITER; j++)
                for (k = 0; k < NITER; k++)
                    x = x * 1.0000000001;
        t1 = time(NULL);
        c1 = clock();
        printf("Elapsed time is %.1lf secs\n",
               difftime(t1, t0));
        printf("Process time is %.3lf secs\n",
               (double)(c1-c0)/CLOCKS_PER_SEC);
        printf("Value of x is %.10lf\n", x);

        return 0;
}
```

```
Elapsed time is 1.0 secs
Process time is 0.040 secs
Value of x is 1.0001000050
```

The cast to a **double** forces floating-point division.

CHARACTER

The **ctype** library contains functions for character manipulation and analysis. Their prototypes are listed in **<ctype.h>**.

Manipulation

tolower

`tolower()` returns the lower case of the character received, if possible; otherwise the character received unchanged. The prototype is

```
int tolower(int);
```

For example, `tolower('D')` returns `'d'`, while `tolower(';')` returns `';'`.

toupper

`toupper()` returns the upper case of the character received, if possible; otherwise the character received unchanged. The prototype is

```
int toupper(int);
```

For example, `toupper('d')` returns `'D'`, while `toupper(';')` returns `';'`.

Analysis

The character analysis functions return 'true' or 'false'. C represents 'false' by the value 0 and 'true' by any other value.

islower

`islower()` returns a true value if the character received is lower case, a false value otherwise. The prototype is

```
int islower(int);
```

For example, `islower('d')` returns a true value, while `islower('E')` returns a false value.

isupper

`isupper()` returns a true value if the character received is upper case, a false value otherwise. The prototype is

```
int isupper(int);
```

For example, `isupper('d')` returns a false value, while `isupper('E')` returns a true value.

isalpha

`isalpha()` returns a true value if the character received is alphabetic, a false value otherwise. The prototype is

```
int isalpha(int);
```

For example, `isalpha('d')` returns true, while `isalpha('6')` returns false.

isdigit

`isdigit()` returns a true value if the character received is a digit, a false value otherwise. The prototype is

```
int isdigit(int);
```

For example, `isdigit('d')` returns false, while `isdigit('6')` returns true.

isspace

`isspace()` returns a true value if the character received is a whitespace character, a false value otherwise. The prototype is

```
int isspace(int);
```

For example, `isspace('d')` returns false, while `isspace(' ')` returns true. Whitespace characters are `' ', '\t', '\n', '\v',` and `'\f'`.

isblank

`isblank()` returns a true value if the character received is a space or tab, a false value otherwise. The prototype is

```
int isblank(int);
```

For example, **`isblank('\t')`** returns true, while **`isblank('\n')`** returns false.

EXERCISES

- Complete the Workshop on Library Functions