Introduction to functions in C

Lecture Topics

- Introduction to functions in C
- Overview of C standard library

Lecture materials

- Textbook §14.1, 14.2, 14.4

Introduction to functions in C

- A function in C is roughly equivalent to a subroutine in LC-3 assembly language
- It is a segment of code that implements some well-defined function in the program
- Example from some hypothetical game board program
    - Clear board
    - Setup board
    - Display board
        - All these are functions that do some well-defined work.
        - The main program then just calls them when needed
- Using functions enables
    - Hiding low-level details
    - Giving high-level structure to the program
    - Efficiently reusing code

Syntax

- Using functions in C requires:
- 1. A function **prototype**, or function's **declaration**
    - Note we did not need one in LC-3 assembly language
    - Example: **int** Factorial(**int** n);
    - Function prototype specifies three things:
        - Name of the function, e.g., sin, cons, printf, Factorial, etc.
        - Types of all arguments that are passed to the function, e.g., int for n in the example above
        - Type of return value
    - Examples:
        - double cos(double x); <- cos function requires one argument of type double and returns a value of type double
        - int getchar(void); <- reads a character from keyboard and returns its ascii value; does not require any arguments = no input.
        - void clearscreen(void); <- clears screen, takes no arguments, returns nothing.
    - A function may return no value; in this case its return type is "void"
    - A function may not require any arguments; in this case its arguments list is declared as "void"
    - Function prototype must be provided before the function is called in the program
- 2. Function **definition** or **implementation**
    - Note we needed this in LC-3 assembly as well
    - This is the actual source code of the function
    - It includes a formal list of arguments – a list of variables declared and the order in which they are exposed to the user
    - Example

```
int Factorial(int n)
{
    int i, result=1;

    for (i = 0; i <= n; i++)
        result = result * i;

    return result;
}
```

- In this example, we have an implementation of the function to compute factorial of number n.
  - Value of n is passed to the function as an argument.
  - Result is returned back to the calling program using **return** keyword.
- To use this function, we need to just call it from our main function:

```
#include <stdio.h>

/* our Factorial function prototype goes here */
int Factorial(int n);

/* main function */
int main()
{
    int number;
    int answer;

    printf("Enter a number: ");
    scanf("%d", &number);

    answer = Factorial(number); /* number is the argument
                                   that is transmitted from
                                   the calling function (main)
                                   to the called function */

    printf("factorial of %d is %d\n", number, answer);

    return 0;
}

/* implementation of Factorial function goes here */
```

Example

- **Problem statement:** write a program to compute integral of a function f(x) on an interval [a,b]; use functions for f(x) and integral calculation.
- Solution:

```c
#include <stdio.h>

float f(float x);
float Reimann(int n, float a, float b);

int main()
{
    printf("%f\n", Reimann(100, -1.0f, 1.0f));

    return 0;
}

/* f(x) = x*x+2x+3 */
float f(float x)
{
    return (x * x + 2 * x + 3);
}

/* compute integral of f(x) = x*x+2x+3 on [a,b] */
float Reimann(int n, float a, float b)
{
    float s = 0.0f;             /* computed integral value */
    int i;                      /* loop counter */
    float x, y;                 /* x and y=f(x) */
    float dx = (b - a) / n;     /* width of rectangles */

    for (i = 0; i < n; i++)
    {
        x = a + dx * i;
        y = f(x);
        s += y * dx;
    }

    return s;
}
```

Standard C library functions

- The C standard library is the standard library for the C programming language, as specified in the ANSI C standard. The C standard library is also called the ISO C library.

API (header files)

- The application programming interface (API) of the C standard library is declared in a number of header files. Each header file contains one or more function declarations, data type definitions, and macros. There are now 29 header files.
- Original header files are
    - <assert.h> Contains the assert macro, used to assist with detecting logical errors and other types of bug in debugging versions of a program.
    - <ctype.h> Defines set of functions used to classify characters by their types or to convert between upper and lower case in a way that is independent of the used character set (typically ASCII or one of its extensions).

  - <errno.h> For testing error codes reported by library functions.
  - <float.h> Defines macro constants specifying the implementation-specific properties of the floating-point library.
  - <limits.h> Defines macro constants specifying the implementation-specific properties of the integer types.
  - <locale.h> Defines localization functions.
  - <math.h> Defines common mathematical functions.
  - <setjmp.h> Declares the macros setjmp and longjmp, which are used for non-local exits.
  - <signal.h> Defines signal handling functions.
  - <stdarg.h> For accessing a varying number of arguments passed to functions.
  - <stddef.h> Defines several useful types and macros.
  - <stdio.h> Defines core input and output functions
  - <stdlib.h> Defines numeric conversion functions, pseudo-random numbers generation functions, memory allocation, process control functions
  - <string.h> Defines string handling functions.
  - <time.h> Defines date and time handling functions
- Normative Addendum 1 (NA1), an addition to the C Standard from 1995, included three more header files:
  - <iso646.h> Defines several macros that implement alternative ways to express several standard tokens. For programming in ISO 646 variant character sets.
  - <wchar.h> Defines wide string handling functions.
  - <wctype.h> Defines set of functions used to classify wide characters by their types or to convert between upper and lower case
- Six more header files were added in C99 standard:
  - <complex.h> A set of functions for manipulating complex numbers.
  - <fenv.h> Defines a set of functions for controlling floating-point environment.
  - <inttypes.h> Defines exact width integer types.
  - <stdbool.h> Defines a boolean data type.
  - <stdint.h> Defines exact width integer types.
  - <tgmath.h> Defines type-generic mathematical functions.
- Five more header files were added in C11 standard:
  - <stdalign.h> C11 For querying and specifying the alignment of objects.
  - <stdatomic.h> C11 For atomic operations on data shared between threads.
  - <stdnoreturn.h> C11 For specifying non-returning functions.
  - <threads.h> C11 Defines functions for managing multiple Threads as well as mutexes and condition variables.
  - <uchar.h> C11 Types and functions for manipulating Unicode characters.

Implementation

- Unix-like systems typically have a C library in shared library form (.so), or in static library form (.a), or both.

- The C library is considered part of the operating system on Unix-like systems. The C functions, including the ISO C standard ones, are widely used by programs, and are regarded as if they were not only an implementation of something in the C language, but also de facto part of the operating system interface. Unix-like operating systems generally cannot function if the C library is erased.
- On Microsoft Windows, the core system dynamic libraries (DLLs) provide an implementation of the C standard library for the Microsoft Visual C++ compiler
    - The C standard library for newer versions of the Microsoft Visual C++ compiler is provided by each compiler individually, as well as redistributable packages.
    - Compiled applications written in C are either statically linked with a C library, or linked to a dynamic version of the library that is shipped with these applications, rather than relied upon to be present on the targeted systems. Functions in a compiler's C library are not regarded as interfaces to Microsoft Windows.

Documentation

- On Unix-like systems, the authoritative documentation of the actually implemented API is provided in the form of man pages. On most systems, man pages on standard library functions are in section 3; section 7 may contain some more generic pages on underlying concepts (e.g. man 7 math_error in Linux).

I/O (from stdio.h)

- We are already familiar with scanf() and printf():
    - int scanf(const char *format, ...) function reads input from the standard input stream stdin and scans that input according to format provided.
    - int printf(const char *format, ...) function writes output to the standard output stream stdout and produces output according to a format provided.
- There are other I/O functions, such as getchar() & putchar(), gets() & puts():
    - **int getchar(void)** reads the next available character from the screen and returns it as an integer. This function reads only single character at a time.
    - **int putchar(int c)** puts the passed character on the screen and returns the same character. This function puts only single character at a time.
    - Example:

```
#include <stdio.h>

int main( )
{
   int c;

   printf( "Enter a value :");
   c = getchar( );

   printf( "\nYou entered: ");
   putchar( c );
```

```
    return 0;
}
```

Math functions (from math.h)

- The math.h header defines various mathematical functions. All the functions available in this library take *double* as an argument and return *double* as the result:
  - double acos(double x) returns the arc cosine of x in radians.
  - double asin(double x) returns the arc sine of x in radians.
  - double atan(double x) returns the arc tangent of x in radians.
  - double atan2(doubly y, double x) returns the arc tangent in radians of y/x based on the signs of both values to determine the correct quadrant.
  - double cos(double x) returns the cosine of a radian angle x.
  - double cosh(double x) returns the hyperbolic cosine of x.
  - double sin(double x) returns the sine of a radian angle x.
  - double sinh(double x) returns the hyperbolic sine of x.
  - double tanh(double x) returns the hyperbolic tangent of x.
  - double exp(double x) returns the value of e raised to the xth power.
  - double frexp(double x, int *exponent) returned value is the mantissa and the integer pointed to by exponent is the exponent. The resultant value is x = mantissa * 2 ^ exponent.
  - double ldexp(double x, int exponent) returns x multiplied by 2 raised to the power of exponent.
  - double log(double x) returns the natural logarithm (base-e logarithm) of x.
  - double log10(double x) returns the common logarithm (base-10 logarithm) of x.
  - double modf(double x, double *integer) returned value is the fraction component (part after the decimal), and sets integer to the integer component.
  - double pow(double x, double y) returns x raised to the power of y.
  - double sqrt(double x) returns the square root of x.
  - double ceil(double x) returns the smallest integer value greater than or equal to x.
  - double fabs(double x) returns the absolute value of x.
  - double floor(double x) returns the largest integer value less than or equal to x.
  - double fmod(double x, double y) returns the remainder of x divided by y.
- Many math library implementations also provide two additional forms of these functions. They are typically available starting with C99 standard:
  - float NAMEf(float x) – 32-bit implementation, e.g.,
    - float sinf(float x);
  - long double NAMEl(long double x); – 128-bit implementation, e.g.,
    - long double sinl(long double x);
- When using functions declared in math.h, linking with libm.a is also required:
  - gcc -sdt=c99 myprg.c -o myprg **-lm**
- Example using a math function:

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

int main( )
{
   float x, y;

   printf( "Enter an angle value in radians: ");
   scanf("%f", &x);

   y = sinf(x);

   printf( "sin(%f)=%f", x, y);
   putchar( c );

   return 0;
}
```

Testing and mapping characters (ctype.h)

- All the functions declared in ctype.h accept int as a parameter, whose value must be EOF or representable as an unsigned char. All the functions return non-zero (true) if the argument c satisfies the condition described, and zero(false) if not.
    - int isalnum(int c) checks whether the passed character is alphanumeric.
    - int isalpha(int c) checks whether the passed character is alphabetic.
    - int iscntrl(int c) checks whether the passed character is control character.
    - int isdigit(int c) checks whether the passed character is decimal digit.
    - int isgraph(int c) checks whether the passed character has graphical representation using locale.
    - int islower(int c) checks whether the passed character is lowercase letter.
    - int isprint(int c) checks whether the passed character is printable.
    - int ispunct(int c) checks whether the passed character is a punctuation character.
    - int isspace(int c) checks whether the passed character is white-space.
    - int isupper(int c) checks whether the passed character is an uppercase letter.
    - int isxdigit(int c) checks whether the passed character is a hexadecimal digit.
    - int tolower(int c) converts uppercase letters to lowercase.
    - int toupper(int c) converts lowercase letters to uppercase.
- Example (convert to lower case):

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

int main()
{
   char c;

   c = getchar();
```

```
    while( isalpha(c) )
    {
       putchar(tolower(c));
       c = getchar();
    }

    return(0);
}
```

Various general functions and macros (stdlib.h)

- Some examples of macros:
  - NULL - this macro is the value of a null pointer constant.
  - EXIT_FAILURE - this is the value for the exit function to return in case of failure.
  - EXIT_SUCCESS - this is the value for the exit function to return in case of success.
  - RAND_MAX - this macro is the maximum value returned by the rand function.
- Just a few function examples (we will study more in later lectures):
  - void abort(void) causes an abnormal program termination.
  - void exit(int status) causes the program to terminate normally.
  - int abs(int x) returns the absolute value of x.
  - int rand(void) returns a pseudo-random number in the range of 0 to RAND_MAX.
  - void srand(unsigned int seed) seeds the random number generator used by the function rand().
- Programming example: compute random value on an interval [a,b]:

```
#include <stdio.h>   /* needed for printf */
#include <stdlib.h>  /* needed for rand(), EXIT_SUCCESS, RAND_MAX */

double get_randon_sampe(double lo, double hi);

int main()
{
   double a=0.0, b=3.1415; /* region of interest */
   double r;

   r = get_random_sample(a, b);  /* compute integral */

   printf("Random value = %f\n", I);

   return EXIT_SUCCESS;
}

/* this function returns a random value on [lo,hi] */
double get_random_sample(double lo, double hi)
{
    /* rand() returns a random integer value on [0, RAND_MAX]*/
    return (lo + rand() * (hi - lo) / RAND_MAX);
}
```