Lecture 8:

Functions: Global and Local Variables, Random Variables, and Standard C Functions

Prof. Shervin Shirmohammadi
University of Ottawa

Local Variables: Variables Declared in a C Function

- All variables declared in a function within the opening { and closing } of the function's instruction block are local to the function:
  - Known to the function and not known in other functions in the program.
  - The function parameters are also local variables.
  - They are erased from memory once the function exits.

– E.g.:

```
void main()
{
    int UserInput1, UserInput2, result;
    printf("Please enter an integer, or 0 to exit: ");
    scanf("%d", &UserInput1);
    printf("\nPlease enter another integer: ");
    scanf("%d", &UserInput2);
    printf("\n%d is the bigger one", findTheBiggerOne(UserInput1, UserInput2));
}

int findTheBiggerOne(int a, int b)
{
   int BiggerOne;
   if (a>b)
        BiggerOne = a;
    else
        BiggerOne = b;
   return BiggerOne;
}
```

In this example, `BiggerOne` is **lost** each time the function `findTheBiggerOne` returns.

Global Variables: Variables Declared outside all Functions

- All variables declared outside of function instruction blocks (usually before `main()` and usually after the function prototypes) are known to all of the functions defined in the same C file.
- Global variables should be avoided as much as possible since:
  - A global variable does not appear in the parameter list of the function definition; hence, the function definition is not as clear as it can be; i.e: the I/O's are not properly described.
  - A global variable modified in one function will be modified **everywhere else**, making program debugging more difficult.

```
Int BiggerOne ;
void main()
{
    int UserInput1, UserInput2, result;
    printf("Please enter an integer, or 0 to exit: "); scanf("%d", &UserInput1);
    printf("\nPlease enter another integer: "); scanf("%d", &UserInput2);
    printf("\n%d is the bigger one", findTheBiggerOne(UserInput1, UserInput2));
}

int findTheBiggerOne(int a, int b)
{
   if (a>b)
        BiggerOne = a;
   else
        BiggerOne = b;
   return BiggerOne;
}
```

In this example, `BiggerOne` is **kept** each time the function `findTheBiggerOne` returns.

Local and Global Variable with the Same Name

```
#include <stdio.h>

int x = 10;

void main()
{
   int x = 1;
   printf("x in main is= %d", x);
   fn();
}

void fn()
{
   printf("\nx in fn is= %d", x);
}
```

Global variable

Local variable

Not a good practice though! Makes the code hard to read.

Generation of Random Numbers

- Random numbers are often used to simulate a random process, such as a coin toss or a dice roll.
- The standard functions `rand()` and `srand()` can be used to generate a pseudo-random sequence of numbers. These sequences are called pseudo-random since the numbers eventually start repeating themselves.
- The function `rand()` is of type `int`, thus it returns an integer, and it does not require an argument.
  - The integer returned by `rand()` is between 0 and `RAND_MAX` where `RAND_MAX` is a constant defined by the platform in `stdlib.h` (`#include <stdlib.h>`), and is usually 32767.
- From one execution to another, `rand()` generates the same sequence of pseudo-random numbers. This sequence can be changed via the function `srand()` which changes the root or the **seed** of the random number generator.
  - The function `srand()` does not return a value.
  - The function `srand()` requires one argument which is a positive integer of type `unsigned int` (default value of the seed is 1).

Sequences of Random Integers

- We often wish to generate a sequence of random integers that are within a prescribed range. The modulus operator `%` is quite useful for this task.

E.g.:
```
int integer;
integer = rand() % 8;
integer = 1 + rand() % 6;
integer = rand() % 51 - 25;
```
→ $0 \le integer \le 7$

→ $1 \le integer \le 6$

→ $-25 \le integer \le 25$

- The following general formula can be used to generate random integers within the range $a \le integer \le b$:

```
int a, b, integer;
integer = a + rand() % (b - a + 1);
```

  - The expression `(b - a + 1)` specifies the width of the range and `a` specifies its beginning.

Sequences of Random Real Numbers

- Sometimes we wish to generate a sequence of random real numbers that are within a prescribed range.
  - The following bit of code will produce a random real number within the range: $x \leq real\_no \leq y$.

```
float x, y, real_no;
    ⋮
/* Random real number in the range 0.0 to 1.0. */
real_no = (float) rand() / RAND_MAX;
/* Scale to reduce the range to 0.0 to y - x. */
real_no = real_no * (y-x);
/* Add x to displace the range to x to y. */
real_no = real_no + x;
```

- Note that we may also work with numbers of type `double`.

Example

- Write a program that lets the user “toss a coin” and prints the outcome.

Libraries in C

- There exist in C a set of **library** of functions which are available for general use.
  - Avoids re-inventing the wheel! Familiarize yourselves with the C libraries and use them. The standard functions in the libraeries are defined by ANSI (American National Standards Institute) and are available with any C compiler that adheres to the ANSI C standard.
  - We find in the libraries: mathematical functions, input/output functions, file manipulation functions, and functions to manipulate character strings, and many others.
  - The course textbook (Appendix A) lists and describes some of the C standard functions
    - `printf` and `scanf` are ANSI C functions in the standard input output library (`stdio.h`).

Standard Math Functions

- The course textbook contains a list and description of standard C math functions (see appendix A).
  - The most popular math functions are available.
- The preprocessor directive `#include <math.h>` provides the C compiler with definitions used by the standard math functions in the program. This directive is normally placed after the directive `#include <stdio.h>` before `main()`.
- All standard math functions except three (`frexp`, `ldexp` and `modf`) require arguments of type `double` and all standard math functions return a number of type `double`.
- The value returned by a math function can be assigned to a variable for future use:
  - `x = sin(angle);`

  used in an arithmetic expression, or as an argument sent in to another function:
  - `c = sqrt(PI + pow(z, 3.0));`
- An argument to a math function can be a symbolic constant, a variable or a mathematical expression.

- The rules of promotion are applied to the arguments of math functions if they are not of type `double`. These same rules are applied to the value returned by the function if it is not stored into a variable of type `double`.
- Some of the most commonly encountered math functions:

Math Function	Standard C Math Function	Note on Usage
– $\sqrt{x}$	`sqrt(x)`	`x` must be ≥ 0.0
– x^y	`pow(x,y)`	
– e^x	`exp(x)`	
– $\log_e x$	`log(x)`	
– $\log_{10} x$	`log10(x)`	
– $\|x\|$	`fabs(x)`	
– sin(x)	`sin(x)`	`x` is in radians.
– cos(x)	`cos(x)`	
– tan(x)	`tan(x)`	
arcsin(x)	`asin(x)`	$-1 \le x \le 1$ and $-\pi/2 \le$ return $\le \pi/2$, in radians.
– arccos(x)	`acos(x)`	$-1 \le x \le 1$ and $0 \le$ return $\le \pi$, in radians.
– arctan(x)	`atan(x)`	$-\pi/2 \le$ return $\le \pi/2$, in radians.
– sinh(x)	`sinh(x)`	Hyperbolic functions.
– cosh(x)	`cosh(x)`	
– tanh(x)	`tanh(x)`	

- The function `ceil(x)` rounds `x` to the smallest integer that is greater than or equal to `x`.
  - E.g.: `ceil(9.0)` yields 9.0
    - `ceil(9.2)` yields 10.0
    - `ceil(9.5)` yields 10.0
    - `ceil(9.8)` yields 10.0
    - `ceil(-9.8)` yields -9.0
    - `ceil(-9.5)` yields -9.0
    - `ceil(-9.2)` yields -9.0
    - `ceil(-9.0)` yields -9.0
- The function `floor(x)` rounds `x` to the largest integer that is less than or equal to `x`.
  - E.g.: `floor(9.0)` yields 9.0
    - `floor(9.2)` yields 9.0
    - `floor(9.5)` yields 9.0
    - `floor(9.8)` yields 9.0
    - `floor(-9.8)` yields -10.0
    - `floor(-9.5)` yields -10.0
    - `floor(-9.2)` yields -10.0
    - `floor(-9.0)` yields -9.0

Example

- Write a program to calculate ***x*** to the power of ***y***, both input by the user.

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

void main()
{
    double x, y;
    printf("Please enter X: ");
    scanf("%lf", &x);
    printf("X is: %f", x);
    printf("Please enter Y: ");
    scanf("%lf", &y);
    printf("\nX to the power of Y is: %f", pow(x,y));
}
```

The C Standard Library

<assert.h> : Diagnostics
<ctype.h> : Character Class Tests
<errno.h> : Error Codes Reported by (Some) Library Functions
<float.h> : Implementation-defined Floating-Point Limits
<limits.h> : Implementation-defined Limits
<locale.h> : Locale-specific Information
<math.h> : Mathematical Functions
<setjmp.h> : Non-local Jumps
<signal.h> : Signals
<stdarg.h> : Variable Argument Lists
<stddef.h> : Definitions of General Use
<stdio.h> : Input and Output
<stdlib.h> : Utility functions
<string.h> : String functions
<time.h> : Time and Date functions

C Standard Library