IE1005 From Computational Thinking to Programming

Lesson 5

Standard Library Functions

(Reference: Harry H. Cheng, Chapter 6, 9, 12)

Dr. Tan Chee Wah, Wesley (School of EEE)

Email: wesleytan@ntu.edu.sg
Office: S1-B1b-54
Phone:6790 6009

"Be Prepared, Give Feedback"

The C Standard Library

Functions are the ***building blocks*** of C programs.

e.g. `printf()`, `scanf()`, `sqrt()`

CodeBlocks (other C systems as well) provides a rich collection of powerful functions known as the C Standard Library for our use.

Header Files

The library has a set of associated header files such as **`stdio.h`**. These contain

- ***prototypes*** of functions in the library (Prototypes will be covered in a later lesson). .
- ***macro*** definitions
- other programming elements.

To use a function (e.g. **printf**( )), we need to include the corresponding header file (**`stdio.h`** for **printf**( )).

Table 5.1 Selected Header Files

Header File	*Description*
`ctype.h`	Character handling
`math.h`	Math functions
`stdio.h`	Input/Output
`stdlib.h`	Misc functions*
`string.h`	String functions
`time.h`	Time functions

* C Standard General Utilities Library

cmath (math.h)

cos	Compute cosine (function)
sin	Compute sine (function)
tan	Compute tangent (function)
acos	Compute arc cosine (function)
asin	Compute arc sine (function)
atan	Compute arc tangent (function)
atan2	Compute arc tangent with two parameters (function)
cosh	Compute hyperbolic cosine (function)
sinh	Compute hyperbolic sine (function)
tanh	Compute hyperbolic tangent (function)
exp	Compute exponential function (function)
frexp	Get significand and exponent (function)
ldexp	Generate number from significand and exponent (function)
log	Compute natural logarithm (function)
log10	Compute common logarithm (function)
modf	Break into fractional and integral parts (function)
pow	Raise to power (function)
sqrt	Compute square root (function)
ceil	Round up value (function)
fabs	Compute absolute value (function)
floor	Round down value (function)
fmod	Compute remainder of division (function)

How to use standard functions

We need to know:

- The name of the function
- What the function does
- Input Arguments (if any) of the function
- Data type of Output Result returned (if any)
- The associated header file name

stdio.h

Some I/O functions/macros:

- **printf()**
- **putchar()** Displays one char on screen

  Example: **putchar('A');**
- **scanf()**
- **getchar()** Gets ***one*** char from input stream
- **fflush()** Flushes I/O buffers

Input Stream & Input Buffer

The characters you type at the keyboard form an **input stream**.

Usually this stream of characters first goes into an area of memory known as the **input buffer** before being retrieved (e.g. by **`scanf()`** or **`getchar()`**) in a running program.

Why Use fflush(stdin)?

This is used to flush away any unwanted characters in the input buffer.

Consider

```
scanf("%d", &num);
scanf("%c", &ch);
```

to read an integer and a character from the keyboard.

The second statement will not execute as expected (i.e. character not captured)! Why?

`Stdin:` standard input stream

scanf()

Some points to note about **scanf()**:

(a) When using **%c** to read in characters, remember that ***whitespaces*** (spaces, tabs, newlines) are characters.

```
scanf("%c%c", &ch1, &ch2);
```

Be careful about how you enter these two characters. DO NOT separate them using a whitespace. IF you enter "a b" using keyboard, 'a' will be stored in ch1 and space character (NOT 'b') in ch2!

scanf()

(b) When reading in numbers using **%d, %f** or **%lf** , **scanf()** skips leading whitespaces.

In inputting two numbers using

```
scanf("%d%d", &num1, &num2);
```

we can separate them using one or more whitespaces which will be discarded by the computer.

scanf()

(c) When reading numeric data, it will continue reading until it meets a trailing whitespace.

```
scanf("%d", &a);
```

Suppose you type 123 456. This input is considered as two numbers. **scanf()** will only read 123, leaving the rest behind in the input buffer.

scanf()

(d) When reading numeric data, it will stop reading any subsequent characters in the input stream if it meets an invalid character.

```
scanf("%d", &a);
```

Suppose you type **123xyz**. **scanf()** will read the **123** part but leave **xyz** behind in the input buffer.

IF you type '**123**' and press ENTER key, **123** will be stored in **a**, leaving input (whitespace) by ENTER key to be retrieved by next **scanf()**.

Use of fflush(stdin)

Thus, should write

```
scanf("%d", &num); //get integer
fflush(stdin); //remove whitespace
                input when ENTER key is
                pressed
scanf("%c", &ch); //get actual char
```

stdin refers to the input stream created by input from the keyboard.

Getting two integers as below does not need `fflush(stdin)` as whitespaces are discarded:

```
scanf("%d", &num1);
scanf("%d", &num2);
```

getchar()

getchar() reads the next available character from the **input buffer** and returns its value as an integer.

We can assign the character read to a variable as in the following:

```
ch = getchar();
```

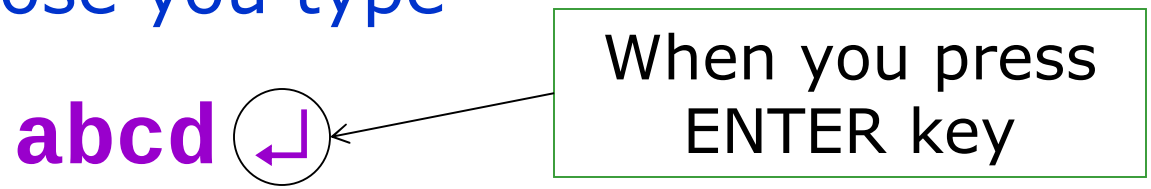
or simply call the function without doing any assignment:

```
getchar();
```

getchar()

It will only read a character from the input buffer after the user presses the ENTER key.

Suppose you type



when **`getchar()`** is expecting input, **`getchar()`** will only read the first character **`'a'`**, leaving behind the others.

Example: getchar()

getchar() can be used to stop program execution temporarily until the user presses the ENTER key.

```
{    . . .
     fflush(stdin);    /* Need to flush if
     there is an input statement before this */
      getchar(); /* Program pauses when
     nothing is entered. When ENTER is pressed,
     getchar() will read this whitespace */

      //Proceed with subsequent statements
}
```

math.h

Some mathematical functions:

fabs(x) Absolute (or positive) value of a number, e.g. fabs(-5.78) returns 5.78. Return data type (which is 5.78 here) is double/float.

Trigonometric functions:

sin(x), cos(x), tan(x) **x must be in radians**

asin(x), acos(x), atan(x) Inverse functions

Exponential and log functions

exp(x) e^x

log(x) $\log_e x$

log10(x) $\log_{10} x$

math.h

Hyperbolic Functions:

sinh(x), cosh(x), tanh(x)

Other mathematical functions:

pow(x,y)	x raised to power y
sqrt(x)	Square root of x
ceil(x)	Ceiling function
floor(x)	Floor function

Example: sqrt(), pow()

```
double sqrt(double x);
```

It normally takes an argument of type ***double*** but ***float x*** (or even ***int***) is also acceptable.

```
double pow(double x, double y);

   double x = 2.718;

   y = pow(x,6.0);      → y = 2.718^6.0

   z = pow(log(x),2.5); → z = (log_e(2.718))^2.5
```

Return data type of sqrt and pow is double.

Example: ceil() & floor()

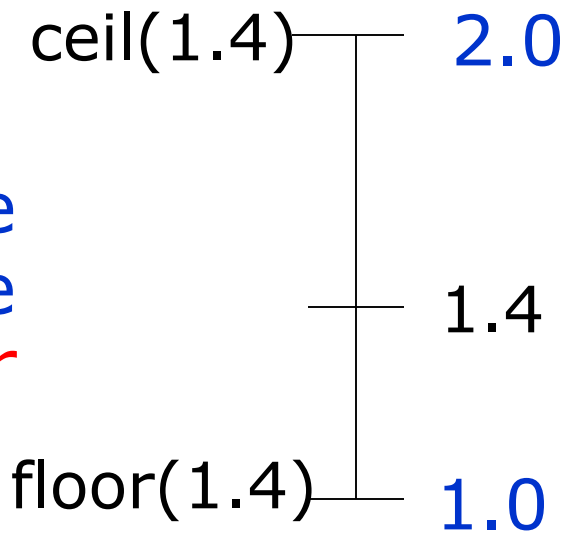
double x;

ceil(x) returns the integer (as a value of type double!) that is just larger than or equal to x.

ceil(1.4) returns 2.0

floor(x) returns the integer (as a value of type double) that is just smaller than or equal to x.

floor(1.4) returns 1.0

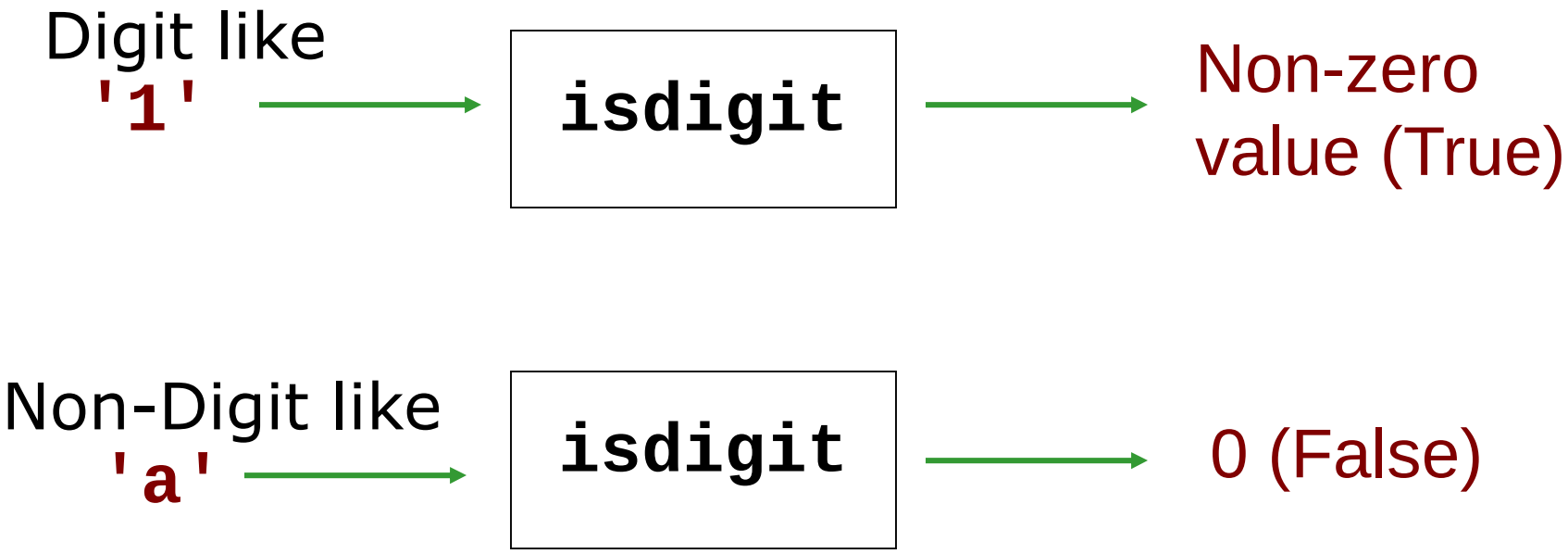


ctype.h

Some character functions:

- **`isalpha`** Alphabetic character like ‘a’?
- **`isdigit`** Numerical Digit character like ‘1’?
- **`isalnum`** Alphabetic or Numerical Digit character?
- **`islower`** Lower case character?
- **`isupper`** Upper case character?
- **`tolower`** Upper case --> lower case
- **`toupper`** Lower case --> upper case

Example (`isdigit`)



Example (isdigit)

```
#include <ctype.h>
char ch='1';
if (isdigit(ch) != 0) //isdigit() is TRUE
   printf("%c is a digit.", ch);
else
   printf("%c is not a digit.", ch);
```

Note:

- **isdigit** returns TRUE (not 0) or FALSE (0).
- != means *not equal*
- We'll see later that the **if** statement can be simply written as

```
if (isdigit(ch))
```

stdlib.h

We shall only consider 4 functions:

abs(), system(), rand() and **srand()**

abs(x) returns the absolute (or positive **integer**) value of a number **x**, e.g. abs(-5.78) returns 5. Return data type (which is 5 here) is integer.

system() enables us to execute operating system commands.

Example:

```
system("Pause");
```

causes program execution to pause until we press any key.

stdlib.h: rand()

rand() and **srand()** are for generating random integers.

The function **rand()** generates a random integer within the range: 0 to 32767 (This maximum value is defined as a macro **RAND_MAX** in **stdlib.h**).

Program 5.3 (rand())

```
/* Example on the use of rand() */

#include <stdio.h>
#include <stdlib.h> //Library for rand()

int main(void)
{
  printf("Three random integers are:\n %d
  %d %d\n", rand(), rand(), rand());

  return 0;
}
```

Program 5.3 (rand())

Output:

```
Three random integers are:
 6334 18467 41
```

The program always produces the ***same*** output when the program is run repeatedly. This is normally not satisfactory. To make the result unpredictable, we need to provide a "seed" to the random number generator in **`rand()`**.

Programmers frequently use the time, in seconds, provided by the computer clock as this seed number.

*Seed: A integer used to set the starting point for generating a series of random numbers.

Program 5.4 (srand())

```
/* Example on the use of srand() */
#include <stdio.h>
#include <stdlib.h>
#include <time.h> //Library for time()

int main(void)
{
   srand((unsigned) time(NULL));
   printf("Three random integers are:\n
   %d %d %d\n", rand(),rand(),rand());

   return 0;
}
```

Program 5.4 (srand())

Output (different each time the program is run):

```
Three random integers are:
 12066 20374 22199
```

time(NULL) returns the current calendar time. Value is in terms of number of seconds elapsed, *usually* since midnight January 1, 1970 GMT).

srand() takes an unsigned integer argument as seed, **time(NULL)**in this example, and **rand()** will generate random numbers based on seed

Summary

1. **stdio.h**
   - See Slide 5-7 for list of IO functions
2. **math.h**
   - See Slide 5-18 and 19 for list of mathematical functions
3. **ctype.h**
   - See Slide 5-22 for list of character handling functions
4. **stdlib.h**
   - See Slide 5-25 for list of miscellaneous functions
5. **time.h** – time (NULL) function