CS1010E Lecture 3
Simple C Programs – Part 2

Joxan Jaffar

Block COM1, Room 3-11, +65 6516 7346
www.comp.nus.edu.sg/~joxan
cs1010e@comp.nus.edu.sg

Semester II, 2016/2017

Lecture Outline

- Standard Input and Output.
- Numerical Technique: Linear Interpolation.
- Problem Solving Applied: Freezing Temperature of Seawater.
- Mathematical Functions.
- Character Functions.
- System Limitations.

Lecture Objectives

- Develop problem solutions containing:
  - User-supplied information from the keyboard.
  - Information printed on the screen.
  - Linear interpolation techniques.

Standard Input and Output

- Need a statement to print values of variables.
- Need a statement that allows us to enter values from the keyboard when the program is executed.
- Before using these statements, we must include the following preprocessor directive:
  - `#include <stdio.h>`
- This directive gives the compiler the information it needs to check references to the input / output functions in the Standard C library.

The `printf` Function

- The `printf` function allows us to print values and explanatory text to the screen.

- The following statement prints the value of a `double` variable named `angle` along with the corresponding units:

  - `printf("Angle = %f radians \n",angle);`

- The `printf` statement contains two argument: a control string and an identifier to specify the value to be printed.

The `printf` Function

- A **control string** is enclosed in double quotation marks, and can contain text, conversion specifiers, or both.
- A **conversion specifier** describes the format to use in printing the value of a variable.
- The control string specifies that the characters `Angle =` are to be printed.
- The next group of characters (`%f`) represents a conversion specifier that indicates that a value is to be printed next.

The `printf` Function

- This is followed by the control string `radians`.
- The next combination of characters (`\n`) represents a new line indicator; it causes a skip to a new line on the screen after the information has been printed.
- The second argument in the `printf` statement is a variable `angle`; it is matched to the conversion specifier in the control string.
- The value in `angle` is printed according to the specification `%f`, explained later.

The `printf` Function

- If the value of `angle` is 2.84, then the output generated by the previous statement is:
  - `Angle = 2.840000 radians`

- Now we can take a closer look at the conversion specifiers.

Conversion Specifiers (Output)

Variable Type	Output Type	Specifier
	Integer Values	
short, int	int	%i, %d
int	short	%hi, %hd
long	long	%li, %ld
int	unsigned int	%u
int	unsigned short	%hu
long	unsigned long	%lu
	Floating-Point Values	
float, double	double	%f, %e, %E, %g, %G
long double	long double	%LF, %Le, %LE, %Lg, %LG
	Character Values	
char	char	%c

Conversion Specifiers

- To select a conversion specifier for a value to be printed, first select the correct type of specifier as indicated in the table on the previous slide.

- To print a `short` or an `int`, use an `%i` (integer) or `%d` (decimal) specifier (either specifier gives the same results).

- To print a `long`, use an `%li` or `%ld` specifier.

Conversion Specifiers

- To print a `float` or a `double`, use an `%f` (floating-point form), `%e` (exponential form, as in `2.3e+02`), or `%E` (exponential form, as in `2.3E+02`).

- The `%g` (general) specifier prints the value using an `%f` or `%e` specifier, depending on the size of the value.

- The `%G` specifier is the same as the `%g` specifier, except that it prints the value using an `%f` or `%E` specifier.

Field Width and Precision

- After selecting the correct specifier, additional information can be added.
- A minimum **field width** can be specified, along with an optional **precision** that controls the number of characters printed.
- The field width and the precision can be used together or separately.
- If the precision is omitted, a default of 6 is used for the `%f` specifier.

Field Width and Precision

- The decimal portion of a value is rounded to the specified precision; thus, the value 14.51678 will be printed as `14.52` if a `%.2f` specification is used.

- The specification `%5i` indicates that a `short` or `int` is to be printed with a minimum field width of 5.

- The field width will be increased if necessary to print the corresponding value.

Field Width and Precision

- If the field width specifies more positions than are needed for the value, the value is **right justified**, which means that the extra positions are filled with blanks on the left of the value.

- To **left justify** a value, a minus sign is inserted before the field width, as in `%-8i`.

- If a plus sign is inserted before the field width, as in `%+6f`, a sign will always be printed with the value.

Examples

- The list on the next slide shows several conversion specifiers and the resulting output fields for a given value.

- The characters $\texttt{b}$ or $\texttt{␣}$ is used to indicate the location of blanks within the field.

- In these examples, assume that the corresponding integer value is -145.

Examples

Specifier	Value Printed
`%i`	`-145`
`%4d`	`-145`
`%3i`	`-145`
`%6i`	`bb-145`
`%-6i`	`-145bb`

Examples

The next list shows several conversion specifiers and the resulting output fields for the `double` value 157.8926:

Specifier	Value Printed
`%f`	`157.892600`
`%6.2f`	`157.89`
`%+8.2f`	`␣+157.89`
`%7.5f`	`157.89260`
`%e`	`1.578926e+02`
`%.3E`	`1.579E+02`
`%g`	`157.893`

Multiple Specifiers

- If a control argument contains three conversion specifiers, then three corresponding identifiers or expressions would need to follow the control string, as in this statement:

```
printf("Results: x = %5.2f, y = %5.2f, z = %5.2f \n",
       x,y,z+3);
```

- An example output from this statement is:

```
Results: x =  4.52, y =  0.15, z = -1.34
```

- Note that the last conversion specifier matches to an arithmetic expression instead of a simple variable.

Escape Character

- The backslash (\) is called an **escape character** when it is used in a control string.
- The compiler combines it with the character that follows it and then attaches a special meaning to the combination of characters.
- For example, we have already seen that `\n` represents a skip to a new line.
- In addition, the sequence `\\` is used to insert a single backslash in a control string.
- The sequence `\"` will insert a double quote in a control string.

Escape Character

- Thus, the output of the following statement

```
printf("\"The End.\"\n");
```

- is a line containing

```
"The End."
```

- The other escape sequences recognized by C are shown in the next two slides.

Escape Sequences

Specifier	Value Printed
`\a`	alert (bell) character
`\b`	backspace
`\f`	formfeed
`\n`	*newline
`\r`	carriage return
`\t`	*horizontal tab
`\v`	vertical tab
`\\`	*backslash
`\'`	*single quote
`\"`	*double quote

Splitting Lines

- If a `printf` statement is too long, you should split it over two lines.
- In general, long lines should be split at a point that preserves readability.
- To split text that is contained in quotation marks, split the text into two separate pieces of text, each in its own set of quotation marks.

Splitting Lines

- The following statements show several different ways to correctly separate a statement:
  - `printf("The distance between the points is %5.2f \n", distance);`
  - `printf("The distance between the points is" " %5.2f \n", distance);`
  - `printf("The distance between the " "points is %5.2f \n", distance);`

Style

- Conversion specifiers can be used to make the output of your program readable and usable.

- For engineering values, it is also very important to include the corresponding units in the output along with the numerical values.

- Although the purpose of the `printf` function is to print information, it also returns a value that represents the number of characters printed.

The `scanf` Function

- The `scanf` function allows you to enter values from the keyboard when the program is executed.

- For example, suppose that a program computes the number of acres of new forest growth after a specified period elapses.

- If the time elapsed is a constant in the program, we would have to change the value of the constant, and then recompile and reexecute the program to obtain the output for a different period.

The `scanf` Function

- Alternatively, if we use the `scanf` function to read the time period, we do not need to recompile the program.
- We only need to reexecute it and enter the desired period from the keyboard.
- The first argument of the `scanf` function is a control string that specifies the types of the variables whose values are to be entered from the keyboard.
- The type specifiers are shown in the table on the next slide.

Conversion Specifiers (Input)

Variable Type	Specifier
Integer Values	
int	%i, %d
short	%hi, %hd
long int	%li, %ld
unsigned int	%u
unsigned short	%hu
unsigned long	%lu
Floating-Point Values	
float	%f, %e, %E, %g, %G
double	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG
Character Values	
char	%c

Conversion Specifiers

- The specifiers for an integer variable are `%i` or `%d`.
- The specifiers for a `float` variable are `%f`, `%e`, and `%g`.
- The specifiers for a `double` variable are `%lf`, `%le`, and `%lg`.
- The correct specifier must be used; for example, errors will occur if you use an `%f` specifier to read the value for a `double` variable.

Arguments to `scanf`

- The remaining arguments in the `scanf` function are memory locations that correspond to the specifiers in the control string.

- These memory locations are indicated with the **address operator** `&`.

- This operator is a unary operator that determines the memory address of the identifier with which it is associated.

Arguments to `scanf`

- Thus, if the value to be entered through the keyboard is an integer that is to be stored in the variable `year`, we could use this statement to read the value:

```
scanf("%i",&year);
```

- The precedence level of the address operator is the same as the other unary operators; if there are several unary operators in the same statement, they are associated from right to left.

- A common error in `scanf` statements is to omit the address operator for the identifiers.

Multiple Input

- If we wish to read more than one value from the keyboard, we can use statements such as the following:

```
scanf("%i %lf", &age, &weight_kg);
```

- When this statement is executed, the program will read two values from the keyboard and convert them into one `int` value and one `double` value.

- The values must be separated by at least one blank; they can be on the same line or on different lines.

Multiple Input

In order to **prompt** the program user to enter the values, a `scanf` statement is usually preceded by a `printf` statement that describes the information that the user should enter from the keyboard:

```
printf("Enter age in years and weight in kg: ");
scanf("%i %lf", &age, &weight_kg);
```

Multiple Input

Thus, after the previous statements are executed and the user has responded to the prompt, an example of the information on the screen is

```
Enter age in years and weight in kg: 23 63.4
```

Erroneous Input

- If the characters entered by the user cannot be successfully converted to the types of values indicated by conversion specifiers in the `scanf` statement, the result is system dependent.

- These conversion errors include entering values such as 14.2 for integer values, including commas in large values, and forgetting to separate values with blanks.

Return Value

- Although the main purpose of the `scanf` statement is to read input from the keyboard, it also returns a value that is equal to the number of successful conversions.

- This value is used in programs in later chapters.

Linear Interpolation

- The collection of data from an experiment or from observing a physical phenomenon is an important step in developing a problem solution.

- These data points can generally be considered to be coordinates of points of a function $f(x)$.

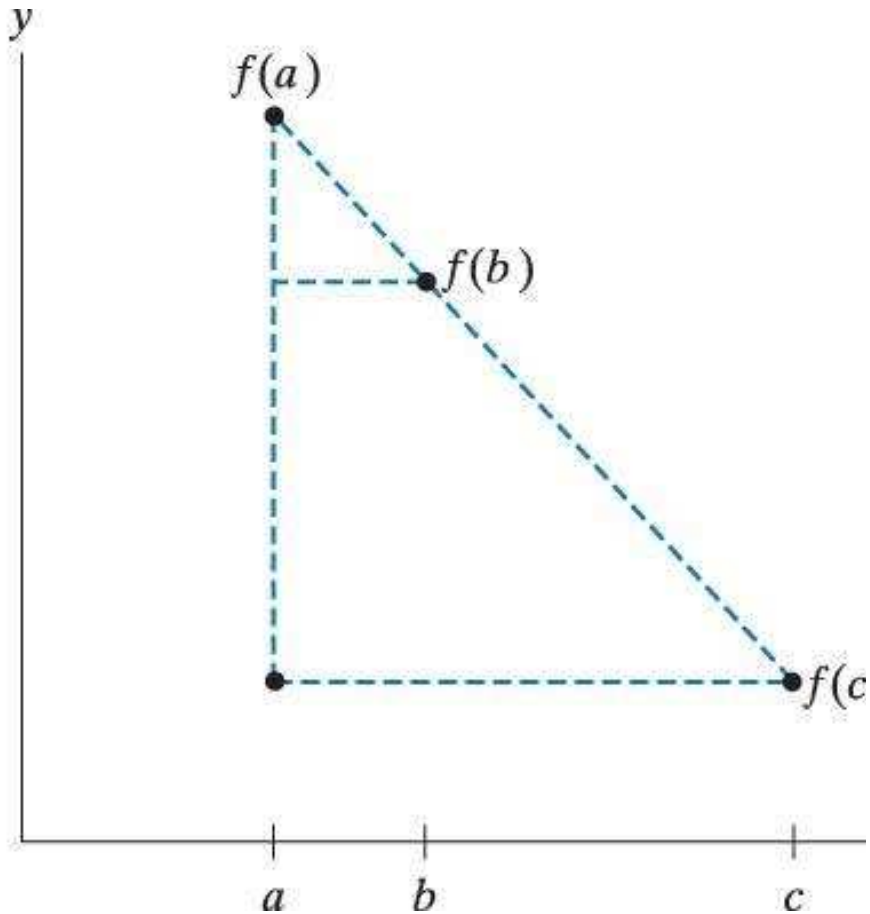
- We would often like to use these data points to determine estimates of the function $f(x)$ for values of x that were not part of the original set of data.

Linear Interpolation

- For example, suppose that we have data points $(a, f(a))$ and $(c, f(c))$.
- If we want to estimate the value of $f(b)$, where $a < b < c$, we could assume that a straight line joined $f(a)$ and $f(c)$, and then use **linear interpolation** to obtain the value of $f(b)$.
- If we assume that the points $f(a)$ and $f(c)$ are joined by a cubic (third-degree) polynomial, we could use a cubic spline interpolation method to obtain the value of $f(b)$.
- Most interpolation problems can be solved using one of these two methods.

Similar Triangles

$$\frac{f(a) - f(b)}{b - a} = \frac{f(a) - f(c)}{c - a}$$



Similar Triangles

- A graph with two arbitrary data points $f(a)$ and $f(c)$ is shown in the previous slide.

- If we assume that the function between the two points can be estimated by a straight line, we can then compute the function value at any point $f(b)$ using an equation derived from similar triangles:

$$f(b) = f(a) + \frac{b-a}{c-a}[f(c) - f(a)].$$

- Recall that we are also assuming that $a < b < c$.

EXAMPLE: Freezing Temperature

- We will now use the new statements we have learned along with linear interpolation to solve a problem related to the **salinity** of water.

- Salinity is often measured using an instrument that measures the electrical conductivity of the water.

- The more dissolved materials in the water, the better it conducts electricity.

- Measurements of salinity are especially important in the colder regions because the temperature at which seawater freezes is dependent upon its salinity.

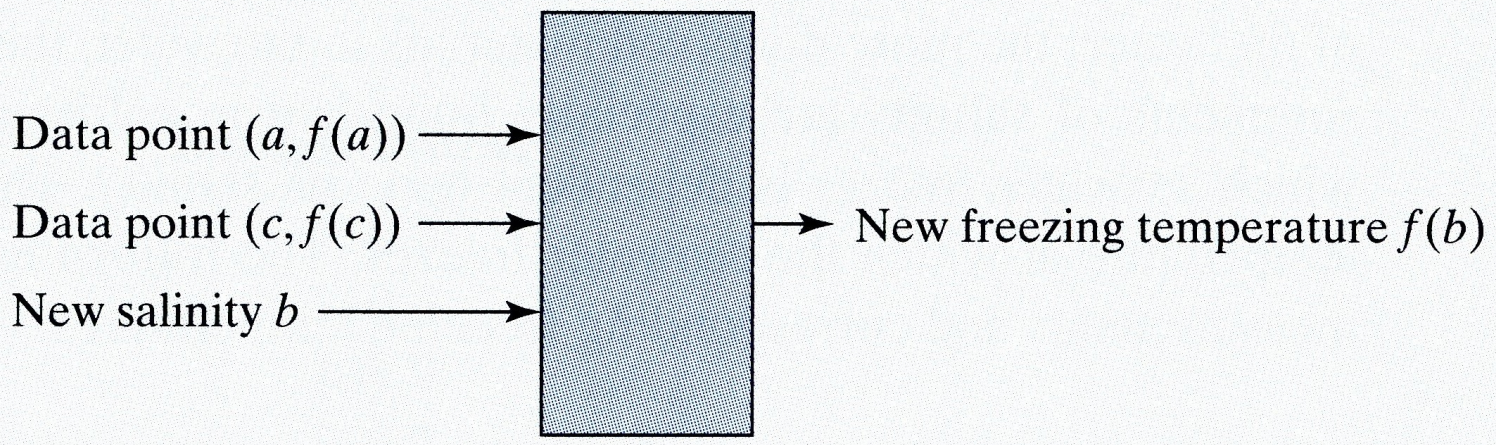
Freezing Temperature (Seawater)

- The higher the salinity, the lower the temperature at which water freezes.

- The following table contains a set of salinity measurements and freezing temperatures:

Salinity (ppt)	**Freezing Temperature, $^\circ$F**
0 (fresh water)	32
10	31.1
20	30.1
24.7	29.6
30	29.1
35	28.6

Freezing Temperature (Seawater)



Hand Example

- Using the linear equation formula, we can compute $f(b)$:

$$\begin{aligned} f(b) &= f(a) + \frac{b-a}{c-a}[f(c) - f(a)] \\ &= 29.1 + \frac{3}{5}(28.6 - 29.1) \\ &= 28.8. \end{aligned}$$

Algorithm Development

- The first step in the development of an algorithm is the decomposition of the problem solution into a set of sequentially executed steps:

- The decomposition outline is:
  - 1. Read the coordinates of the adjacent points and the new salinity value.
  - 2. Compute the new freezing temperature.
  - 3. Print the new freezing temperature.

- This program has a simple structure, so we can convert the decomposition directly into C.

Algorithm Development

```
/*-------------------------------------------------------*/
/*  Program chapter2_2                                   */
/*                                                       */
/*  This program uses linear interpolation to            */
/*  compute the freezing temperature of seawater.        */

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

Algorithm Development

```
int main(void)
{
   /*  Declare variables.  */
   double a, f_a, b, f_b, c, f_c;
   /*  Get user input from the keyboard.  */
   printf("Use ppt for salinity values. \n");
   printf("Use degrees F for temperatures. \n");
   printf("Enter first salinity and freezing temperature: \n");
   scanf("%lf %lf", &a, &f_a);
   printf("Enter second salinity and freezing temperature: \n");
   scanf("%lf %lf", &c, &f_c);
   printf("Enter new salinity: \n");
   scanf("%lf", &b);

   /*  Use linear interpolation to compute  */
   /*  new freezing temperature.            */
   f_b = f_a + (b-a)/(c-a)*(f_c - f_a);

   /*  Print new freezing temperature.  */
   printf("New freezing temperature in degrees F: %4.1f \n",f_b);
   /*  Exit program.  */
   return 0;
}
```

Testing

- We first test the program using the data from the hand example. This generates the following interaction:

```
Use ppt for salinity values.
Use degrees F for temperatures.
Enter first salinity and freezing temperature:
30 29.1
Enter second salinity and freezing temperature:
35 28.6
Enter new salinity:
33
New freezing temperature in degrees F: 28.8
```

- The value computed matches the hand example, so we can then test the program with other time values.

Mathematical Functions

- Arithmetic expressions that solve engineering problems often require computations other than addition, subtraction, multiplication, and division.

- For example, many expressions require the use of exponentiation, logarithms, exponentials, and trigonometric functions.

Mathematical Functions

- The following preprocessor directive should be used in programs referencing the mathematical functions:

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

- This directive specifies that information be added to the program to aid the compiler when it converts references to the mathematical functions in the Standard C library.

An Example

- The following statement computes the sine of an angle `theta` and stores the result in the variable `b`:

```
b = sin(theta);
```

- The `sin` function assumes that the argument is in radians.

- If the variable `theta` contains a value in degrees, we can convert the degrees to radians with a separate statement.

An Example

- Recall that $180^\circ = \pi$ radians:

```
#define PI 3.141593
...
theta_rad = theta*PI/180;
b = sin(theta_rad);
```

- The conversion can also be specified within the function reference: `b = sin(theta*PI/180);`

- Performing the conversion with a separate statement is usually preferable, because it is easier to understand.

Arguments

- A function reference, such as `sin(theta)`, represents a single value.
- The parentheses following the function name contain the inputs to the function, which are called **parameters** or **arguments**.
- A function may contain no arguments, one argument, or many arguments, depending on its definition.
- If a function contains more than one argument, it is very important to list the arguments in the correct order.

Arguments

- Some functions also require that the arguments be in specific units.

  The trigonometric functions assume that arguments are in radians

- Most of the mathematical functions assume that the arguments are `double` values.

- If a different type argument is used, it is converted to a `double` before the function is executed.

Composition

- A function reference can also be part of the argument of another function reference.

- The following statement computes the logarithm of the absolute value of `x`:

```
b = log(fabs(x));
```

- When one function is used to compute the argument of another function, be sure to enclose the argument of each function in its own set of parentheses.

- This nesting of functions is also called **composition** of functions.

Elementary Math Functions

`fabs(x)`	Computes the absolute value of x.
`sqrt(x)`	Computes the square root of x, where $x \geq 0$.
`pow(x,y)`	Used for exponentiation. Computes the value of x^y. Errors occur if $x = 0$ and $y \leq 0$, or if $x < 0$ and y is not an integer.
`ceil(x)`	Rounds x to the nearest integer toward ∞ (infinity). For example, `ceil(2.01)` is equal to 3.

Elementary Math Functions

`floor(x)` Rounds x to the nearest integer toward $-\infty$ (negative infinity). For example, `floor(2.01)` is equal to 2.

`exp(x)` Computes the value of e^x, where e is the base for natural logarithms, or approximately 2.718282.

`log(x)` Returns $\ln x$, the natural logarithm of x to the base e. Errors occur if $x \leq 0$.

`log10(x)` Returns $\log_{10} x$, the common logarithm of x to the base 10. An error occurs if $x \leq 0$.

Elementary Math Functions

- Remember that the logarithm of a negative value or zero does not exist, and thus an execution error occurs if you use a logarithm function with a negative value for its argument.

- An additional mathematical function that you may find useful is the `abs` function.

- This function computes the absolute value of an integer, and returns an integer value.

- The header file containing information relative to this function is `stdlib.h`, and it should be included in programs using this function.

Trigonometric Functions

- The **trigonometric functions** assume that all arguments are of type `double` and that they return values of type `double`.

- The trigonometric functions also assume that angles are represented in radians.

- To convert radians to degrees, or degrees to radians, use the following statements:

```
#define PI 3.141593
...
angle_deg = angle_rad*(180/PI);
angle_rad = angle_deg*(PI/180);
```

Trigonometric Functions

`sin(x)`	Computes the sine of x, where x is in radians.
`cos(x)`	Computes the cosine of x, where x is in radians.
`tan(x)`	Computes the tangent of x, where x is in radians.
`asin(x)`	Computes the arcsine or inverse sine of x, where x must be in the range $[-1, 1]$. The function returns an angle in radians in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

Trigonometric Functions

`acos(x)`	Computes the arccosine or inverse cosine of x, where x must be in the range $[-1, 1]$. The function returns an angle in radians in the range $[0, \pi]$.
`atan(x)`	Computes the arctangent or inverse tangent of x. The function returns an angle in radians in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$.
`atan2(y,x)`	Computes the arctangent or inverse tangent of the value $\frac{y}{x}$. The function returns an angle in radians in the range $[-\pi, \pi]$.

Trigonometric Functions

- Note that the `atan` function always returns an angle in Quadrant I or IV, whereas the `atan2` function returns an angle that can be in any quadrant, depending on the signs of x and y.

- Thus, in many applications, the `atan2` function is preferred over the `atan` function.

- Using degrees instead of radians is a common error in programs with trigonometric functions.

Trigonometric Functions

The other trigonometric and inverse trigonometric functions can be computed with the use of the following equations:

$$\sec x = \frac{1}{\cos x} \qquad \operatorname{asec} x = \operatorname{acos}\left(\frac{1}{x}\right)$$

$$\csc x = \frac{1}{\sin x} \qquad \operatorname{acsc} x = \operatorname{asin}\left(\frac{1}{x}\right)$$

$$\cot x = \frac{1}{\tan x} \qquad \operatorname{acot} x = \operatorname{acos}\left(\frac{x}{\sqrt{1+x^2}}\right)$$

Hyperbolic Functions

`sinh(x)` Computes the hyperbolic sine of x, which is equal to $\frac{e^x - e^{-x}}{2}$.

`cosh(x)` Computes the hyperbolic cosine of x, which is equal to $\frac{e^x + e^{-x}}{2}$.

`tanh(x)` Computes the hyperbolic tangent of x, which is equal to $\frac{\sinh x}{\cosh x}$.

Hyperbolic Functions

Additional hyperbolic functions and the inverse hyperbolic functions can be computed with these relationships:

$$\coth x = \frac{\cosh x}{\sinh x} \quad (\text{for } x \neq 0)$$

$$\operatorname{sech} x = \frac{1}{\cosh x} \qquad \operatorname{csch} x = \frac{1}{\sinh x}$$

$$\operatorname{asinh} x = \ln(x + \sqrt{x^2 + 1})$$

$$\operatorname{acosh} x = \ln(x + \sqrt{x^2 - 1}) \quad (\text{for } x \geq 1)$$

Hyperbolic Functions

$$\operatorname{atanh} x = \frac{1}{2} \ln\left(\frac{1+x}{1-x}\right) \quad (\text{for } |x| < 1)$$

$$\operatorname{acoth} x = \frac{1}{2} \ln\left(\frac{x+1}{x-1}\right) \quad (\text{for } |x| > 1)$$

$$\operatorname{asech} x = \ln\left(\frac{1+\sqrt{1-x^2}}{x}\right) \quad (\text{for } 0 < x \leq 1)$$

$$\operatorname{acsch} x = \ln\left(\frac{1}{x} + \frac{\sqrt{1+x^2}}{|x|}\right) \quad (\text{for } x \neq 0)$$

Character Functions

- Numerous functions are available to use with character data.

- These are functions designed for character input and output, functions to convert characters between uppercase and lowercase, and functions to perform character comparisons.

Character I/O

- Although `printf` and `scanf` functions can be used to print and read characters using the `%c` specifier, C also contains special functions for reading and printing characters.

- The `getchar` function reads the next character from the keyboard and returns the integer value of the character.

- The `putchar` function prints a character to the computer screen.

The `putchar` Function

- The `putchar` takes one integer argument and returns an integer value.
- The execution of the `putchar` function causes the character that corresponds to the integer argument to be written to the computer screen.
- If several `putchar` function references are made in a row, the characters are printed one after another on the same line, until a newline character is printed.

The `putchar` Function

- The following statements cause the characters `ab` to be printed on one line, followed by the character `c` on the next line:

```
putchar('a');
putchar('b');
putchar('\n');
putchar('c');
```

The `putchar` Function

- The same information could be printed using the integer values that correspond to the characters (see ASCII table from previous lecture).

```
putchar(97);
putchar(98);
putchar(10);
putchar(99);
```

The `getchar` Function

- The `getchar` function reads the next character from the keyboard and returns the integer value of the character.

- The following statements cause one character to be read from the keyboard and printed on a new line:

```
int c;
...
putchar('\n');
c = getchar();
putchar(c);
putchar('\n');
```

Character Functions

- **Character functions** in the Standard C library fall into two categories; one set of functions is used to convert characters between uppercase and lowercase; the other set is used to perform character comparisons.

- The following preprocessor directive should be used in programs referencing these character functions:

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

Character Functions

- The following statement converts the lowercase letter stored in the variable `ch` to an uppercase character and stores the result in the character variable `upper`:

```
upper = toupper(ch);
```

- If `ch` is a lowercase letter, the function `toupper` returns the corresponding uppercase letter; otherwise, the function returns `ch`.

- Note that no change is made to the variable `ch`.

Character Functions

- Each character function requires an integer argument, and each function returns an integer value.
- The character comparison functions return a nonzero value if the comparison is true; otherwise they return zero.
- The functions are listed with a brief explanation.

Character Functions

`tolower(ch)`	*If `ch` is an uppercase letter, returns the corresponding lowercase letter; otherwise, it returns `ch`.
`toupper(ch)`	*If `ch` is a lowercase letter, returns the corresponding uppercase letter; otherwise, it returns `ch`.

Character Functions

`isdigit(ch)`	*Returns a nonzero value if `ch` is a decimal digit; otherwise, it returns a zero.
`islower(ch)`	*Returns a nonzero value if `ch` is a lowercase letter; otherwise, it returns a zero.
`isupper(ch)`	*Returns a nonzero value if `ch` is an uppercase letter; otherwise, it returns a zero.

Character Functions

`isalpha(ch)`	*Returns a nonzero value if `ch` is an uppercase letter or a lowercase letter; otherwise, it returns a zero.
`isalnum(ch)`	*Returns a nonzero value if `ch` is an alphabetic character or a numeric digit; otherwise, it returns a zero.

Character Functions

`iscntrl(ch)`	Returns a nonzero value if `ch` is a control character; otherwise, it returns a zero. (The **control characters** have integer codes of 0 through 21, and 127.)
`isgraph(ch)`	Returns a nonzero value if `ch` is a character that can be printed (excludes spaces); otherwise, it returns a zero. (The printing characters have integer codes from 32 through 126.)

Character Functions

`isprint(ch)`	Returns a nonzero value if `ch` is a printing character (does include a space); otherwise, it returns a zero.
`ispunct(ch)`	*Returns a nonzero value if `ch` is a printing character (with the exception of a space or a letter or a digit); otherwise, it returns a zero.

Character Functions

`isspace(ch)`	*Returns a nonzero value if `ch` is a space, formfeed, newline, carriage return, horizontal tab, or vertical tab (these characters are also referred to as white space); otherwise, it returns a zero.
`isxdigit(ch)`	Returns a nonzero value if `ch` is a hexadecimal digit, which is a decimal digit or an alphabetic character A through F (or *a* through *f*); otherwise, it returns a zero.

System Limitations

```
/*-------------------------------------------------------*/
/*  Program chapter2_4                                   */
/*                                                       */
/*  This program prints the system limitations.          */

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

System Limitations

```
int main(void)
{
   /*  Print integer type maximums.  */
   printf("short maximum: %i \n", SHRT_MAX);
   printf("int maximum: %i \n", INT_MAX);
   printf("long maximum: %li \n\n", LONG_MAX);
   /* Print float precision, range, maximum */
   printf("float precision digits: %i \n", FLT_DIG);
   printf("float maximum exponent: %i \n", FLT_MAX_10_EXP);
   printf("float maximum: %e \n\n", FLT_MAX);

   /*  Print double precision, range, maximum.  */
   printf("double precision digits: %i \n",DBL_DIG);
   printf("double maximum exponent: %i \n",
          DBL_MAX_10_EXP);
   printf("double maximum: %e \n\n", DBL_MAX);

   /*  Print long precision, range, maximum.  */
   printf("long double precision digits: %i \n", LDBL_DIG);
   printf("long double maximum exponent: %i \n", LDBL_MAX_10_EXP);
   printf("long double maximum: %Le \n", LDBL_MAX);
   /*  Exit program.  */
   return 0;
}
```

References

Etter Sections 2.4 to 2.10

Next Lecture

Control Structures and Data Files

(Part 1)