BIL 104E Introduction to Scientific and Engineering Computing

Lecture 2

Constants and Variables, Basic I/O Functions

Constants and Variables

- Constants and variables represent values that we use in our programs.
- **Constants** are specific values such as 2, 3.14, or -5.78.
- **Variables** are memory locations that are assigned a **name** or **identifier**.
- **Identifier** is used to reference the value stored in the memory location.
- The values of variables that were not given initial values are unspecified. These values are called **garbage values** because they are values left in memory from previous program.

Rules for selecting a valid identifier

- Begin with an **alphabetic character** or the underscore character _.
- Alphabetical characters can be lowercase or uppercase. (C is **case sensitive**)
- **Digits** can be used but not as the first character.
- Can be of any length but first **31 characters** should be unique.
- **Keywords** with special meanings to the compiler should not be used.
- The name should reflect the **content** of the variable.

Scientific Notation

- **Floating-point value**: Can represent integer and non-integer values such as 2.5, -0.004, 15.0.

- **Scientific notation**: A floating-point number is expressed as a mantissa times a power of 10, where mantissa has an absolute value greater than or equal to 1.0 and less than 10.0.

  Example: 25.6 = 2.56×10^{1} -0.004 = -4.0×10^{-3} 1.5 = 1.5×10^{0}

- In **exponential notation** letter **e** is used to separate the mantissa from the exponent of the power of ten.

  Example: 25.6 = 2.56e1 -0.004 = -4.0e-3 1.5 = 1.5e0

- **Precision**: The number of digits allowed by the computer for the decimal portion of the mantissa determines the precision or accuracy and the remaining digits are **truncated** or **chopped**.

  Example: 35.004 has 4 digits of precision **(why??)** and if the computer allows three digits of precision the number will be stored as 3.500e1 which will produce inaccurate computation.

- **Range**: The number of digits allowed for the exponent determines the range.

Possible Problems

- Despite all their accuracy computers sometimes produce erroneous results or fail to provide answers.
  - **Overflow:** If an operation produces a number that is **too large** for the computer to store, it will stop and display an error message. Example: 1000^{1000}.
  - **Underflow**: If an operation produces a number whose absolute value is **too small** for the computer to store then there will be an **underflow** problem and the number will be treated as zero. Example: 0.001^{1000}.
  - **Difference Error** is a problem that can occur when subtracting two nearly equal numbers. Example: x=12345.2 and y=12345.1 (both has 5 digits of precision)

    x-y=0.1 (has 1 digit of precision, loss of 4 significant digits)

- We can rearrange the formula we calculate in order to minimize these problems.

$$\frac{1000^{1000}}{999^{1000}} \text{ as } \left(\frac{1000}{999}\right)^{1000}$$

$$\frac{237^{1000}}{237^{998}} \text{ as } 297^{1000-998} = 237^{2}$$

$$\frac{(0.012)^{1000}}{(0.011)^{1000}} \text{ as } \left(\frac{0.012}{0.011}\right)^{1000}$$

$$\frac{1}{\sqrt{25000}-\sqrt{24999}} \text{ as } \sqrt{25000}+\sqrt{24999}$$

Numeric Data Types

- In C, numeric values are either **integers** or **floating-point values**. There are also non-numeric data types (such as **characters**) which will be discussed later.

- **Integers**:

  - Specified by **short, int** and **long** according to the required range. Ranges of values are **system dependent**.

  - C also allows **unsigned** qualifier where unsigned integer represents only positive values. Signed and unsigned integers represent same number of values but the ranges are different.

- **Floating Point Numbers:**

  - Specified by **float** (single-precision), **double** (double-precision), and **long double** (extended precision) according to the required **precision** and **range** which are also system dependent.

For most systems ranges are:

INTEGERS	Min	Max
short	-32768	32767
int	-32768	32767
long	-2147483648	2147483647
unsigned short	0	65535

FLOATING POINT NUMBERS	Precision	Max Exponent	Maximum Value
float	6 digits	38	3.402823e+38
double	15 digits	308	1.797693e+308
long double	19 digits	4932	1.189731e+4932

printf Function

- The preprocessor directive **#include <stdio.h>** gives the compiler the information that it needs to check referenced to the input/output functions in the Standard C library.

- **printf** function allows to print to the screen.

  Example:

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

- The first argument which is enclosed in double quotation marks is the **control string**. The control string can contain text or conversion specifiers or both.
  - The **conversion specifier** (in the example it is **%f**) describes the format to use in printing the value of a variable.
  - The **newline indicator (\n)** causes a skip to a new line on the screen after the information has been printed.

- The second argument is the **variable** which is matched to the conversion specifier in the control string.

Specifiers for Output

	Variable Type	Output Type	Specifier for output
INTEGER VALUES	short, int	int	**%i** (integer) , **%d** (decimal)
	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** (floating-point), **%e** (exponential form), **%E** (exponential form) , **%g** (general), **%G** (general)
	long double	long double	**%Lf**, **%Le**, **%LE**, **%Lg**, **%LG**

minimum field width Specifier

- **minimum field width specifier,** which may be given between the percent sign (%) and the letter in a format specifier, ensures that the output reaches the minimum width.
- For example, %10f ensures that the output is at least 10 character spaces wide.
- If the field width specifies more positions than are needed for the value, the value is printed **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.

minimum field width Specifier

Specifier	**Value Printed** **(**□ represents blank)
%i	-145
%4d	-145
%3i	-145
%6i	□□-145
%06i	-00145
%-6i	-145□□

precision Specifier

Specifier	**Value Printed** **(**□ represents blank)
%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

Escape Character, backslash (\)

Sequence	Character Represented
\b	backspace, moves cursor to the left one character
\f	formfeed, goes to the top of a new page
\n	newline
\r	carriage return, returns to the beginning of the current line
\t	horizontal tab
\v	vertical tab
\	backslash
\"	double quote

scanf Function

- **scanf** function allows to enter values from the keyboard while the program is being executed.
- 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 remaining arguments are the memory locations that correspond to the specifiers in the control string.
- The memory locations are indicated with the **address operator (&)**.
- **Example**:

```
scanf("%i",&year);
printf("Enter the distance (m) and velocity (m/s): \n");
scanf("%lf %lf", &distance, &velocity);
```

Specifiers for Input

Variable Type	Specifier of Input
int	%i , %d
short	%hi, %hd
long int	%li, %ld
unsigned int	%u
unsigned short	%hu
unsigned long	%lu
float	%f, %e, %E, %g, %G
double	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG