Lecture 3.1

Topics

1. Data Output with `printf()`
2. Examples

---

1. Data Output with `printf()`

Compile and run the following program.

Exercise 1

```
/**
 *Program Name:  cis26Lec0311.c
 *Discussion:    Breaking line at wrong place
 */
#include <stdio.h>

int main( void ) {
  printf( "Size of char : %d\n", sizeof( char ) );
  printf( "Size of short : %d\n", sizeof( short ) );
  printf( "Size of int : %d\n", sizeof( int ) );
  printf( "Size of long %d\n", sizeof( long ) );
  printf( "Size of unsigned short : %d\n",
           sizeof( unsigned short ) );
  printf( "Size of unsigned int :
          %d\n", sizeof( unsigned int ) );
  printf( "Size of unsigned long : %d\n",
          sizeof( unsigned long ) );

  printf( "\nSize of float : %d\n", sizeof( float ) );
  printf( "Size of double : %d\n", sizeof( double ) );
  printf( "Size of long double : %d\n",
          sizeof( long double ) );

  return 0;
}
```

What do you get? Copy and paste what being displayed.

OUTPUT

The following exercises will involve screen output and `printf()`.

In C, data may be entered to a program from keyboard, file, etc. and the results may be then sent to memory, screen, file, etc.

Generally, when a program is running, there are three files opened for usage: **standard input, standard output, and standard error**.

In general,

- **Standard Output** : Monitor/Display Screen
- **Standard Input** : Keyboard
- **Standard Error** : Monitor/Display Screen

Let's look at standard output and `printf()` function next.

1.1 Formatted Output – `printf()`

The standard formatted output function in C is referred to as `printf()` (print formatted). This `printf()` function requires two things:

(1) The format instruction, and

(2) The data to be printed.

Putting things together, the syntax of `printf()` is as follows,

```
printf( "format string/instruction", data1, data2, … );
```

The first argument "**`format string`**" contains the constant text string to be sent to screen and zero or more **field specifications**.

1.2 Field Specification

- The **field specification** describes how the data should be printed.
- Each **field specification** begins with a percent sign (%) for each data point; such as `data1, data2`, etc.

For `printf()`, the syntax of a **field specification** is given as follows,

```
%<flag><minimum_width><precision><size>conversion_code
```

Conversion Code

There are several **conversion codes** that one can use. We will be working with four of them: character (**`c`**), integer (**`d`**), floating-point (**`f`**), and address (**`p`**). The fields that are enclosed by the angle brackets are optional and may not be needed.

Size

The **size** is used to modify the type that was specified by the conversion code. There are three sizes: **`h`**, **`l`**, and **`L`**.

- The **`h`** is used with integer codes to indicate `short int`.
- The **`l`** or **`L`** is used with integer codes to indicate `long int`.
- The **`L`** is used with floating-point numbers to indicate `long double`.
- There is no **`size`** for `char, int, float`, and `double`.

Size	Code	Type	Example
none	c	char	%c
h	d	short int	%hd
none	d	int	%d
l or L	d	long int	%Ld
none	f	float	%f
none	f	double	%f
L	f	long double	%Lf

Minimum Width

The **`minimum_width`** specifier is used to specify the minimum number of position in the output. The function `printf()` will override the width if it needs more space.

Precision

If a floating-point number is to be printed, the **`precision`** specifier indicates the number of decimal places with the format of **`.m`** where **`m`** is the number of decimal digits.

The default is six (6) decimal digits.

For examples,

Value	%d	%4d
12	12	12
123	123	123
1234	1234	1234
12345	12345	12345

When both the minimum width (or width) and the precision are used, the width must be large enough to contain the integral value of the number, the decimal point and the number of digits in the decimal position.

For examples,

```
%2hd            /*short integer; 2 print positions*/
%4d             /*int; 4 print positions*/
%8Ld            /*long int; 8 print positions*/
%7.2f           /*float; 7 print positions: xxxx.dd*/
%10.3Lf         /*long double; 10 print positions: xxxxxx.ddd*/
```

Flag

The **`flag`** modifier is used to specify two print modifications.

- If the flag is **`0`** and there is a width specification then leading zeros may be printed along with the number.
- If the flag is a minus sign (**`-`**) then the data are left justified and filled with spaces in the right positions.

For examples,

```
%-8d      /*int; 8 print positions; left justified*/
%08d      /*int; 8 print positions; leading zeros*/
```

2. Examples

Exercise 2

What would be the output of the following program?

```
/**
 *Program Name:  cis26Lec0312.c
 *Discussion:    Formatted Outputs
 */
#include <stdio.h>

int main() {
```

```
  char cVar;
  int iVar;
  double dVar;

  printf( "12345678901234567890\n" );

  printf( "%d%c%f\n", 23, 'z', 4.1 );
  printf( "%c%d%f\n", 23, 'z', 4.1 );
  printf( "%c%d%f\n", 'z', 23, 4.1 );
  printf( "%c%f%d\n", 'z', 4.1, 23 );

  printf( “\n” );
  cVar = 'z';
  iVar = 23;
  dVar = 4.1;

  printf( "%d%c%f\n", iVar, cVar, dVar );
  printf( "%c%d%f\n", iVar, cVar, dVar );
  printf( "%c%d%f\n", cVar, iVar, dVar );
  printf( "%c%f%d\n", cVar, dVar, iVar );

  return 0;
}
```

OUTPUT

Exercise 3

What is the output of the program below?

```
/**
 *Program Name:  cis26Lec0313.c
 *Discussion:    Formatted Outputs with WIDTH Option
 */
#include <stdio.h>

int main() {
  char cVar;
  int iVar;
  double dVar;

  printf( "12345678901234567890123456789012345678901234567890\n" );

  printf( "%8d%8c%10f\n", 23, 'z', 4.1 );

  printf( "\n" );

  cVar = 'z';
  iVar = 23;
  dVar = 4.1;

  printf( "%8c%08d%10.4f\n", cVar, iVar, dVar );

  printf( "%-8d%8c\t%10f\n", iVar, cVar, dVar );

  return 0;
}
```

OUTPUT