CSE101-Lec#8 and 9

- Formatted and Unformatted Input/Output functions
- Type conversion
- Type modifiers

Outline

- Formatted Input/Output functions
  - printf() function
  - scanf() function
- Conversion Specifiers

Introduction

- Presentation of output is very important.
- Formatted functions scanf and printf :
  - these functions input data from standard input stream and
  - output data to standard output stream.
- Include the header `#include<stdio.h>`

Standard I/O Functions

- There are many library functions available for standard I/O.
- These functions are divided into two categories:
  - **Unformatted functions**
  - **Formatted functions**

Formatted Functions

- With Formatted functions, input and output is formatted as per our requirement
  - For example, if *different values are to be displayed*, how much field width i.e., how many columns on screen, is to be used, and how much space between two values is to be given. If a value to be displayed is of real type, then how many decimal places to output
- Formatted functions are:
  - printf()
  - scanf()

Unformatted functions

- The unformatted functions work only with character data type.
- They do not require format conversion symbol for formatting of data types because they work only with character data type
- Unformatted functions are:
  - getchar() and putchar()
  - getch() and putch()
  - gets() and puts()

Formatted output with printf function

- **The printf() function: (Formatted output)**

  printf() is an output function that takes text and values from within the program and sends it out onto the screen.

  In general terms, the printf function is written as:

Syntax

```
printf("format-control-string", arg1,arg2,……..,argN);
```

- The format-control-string can contain:
  - Characters that are simply printed as they are
  - Conversion specifications that begin with a % sign
  - Escape sequences that begin with a \ sign

✔ The arguments can be written as constants, single variable or array names, or more complex expressions.

Example

```
printf("Area of circle is %f units
  \n", area);
```

In this :-

`"Area of circle is %f units \n"-` is a control string.

`area -` is a variable whose value will be printed.

`%f-` is the conversion specifier indicating the type of corresponding value to be printed.

Printing Integers

- Integer values can be 0, 890, -328.

Conversion Specifier	Description	Example
d	Display as a signed decimal integer.	printf("%d", -890);
i	Display as a signed decimal integer.	printf("%i", -890);
u	Display as an unsigned decimal integer.	printf("%u", 890);
h or l	Used before any integer conversion specifier to indicate that a short or long integer is displayed, respectively	printf("%hd", 890); printf("%ld", 800000000L)

```
#include <stdio.h>
int main( void )
{
   printf( "%d\n", 890);
   printf( "%i\n", 890); // i same as d in printf
   printf( "%d\n", +890 ); // plus sign does not //print
   printf( "%d\n", -890 ); // minus sign prints
   printf( "%hd\n", 32000 );
   printf( "%ld\n", 2000000000L ); // L suffix makes
   //literal a long
   printf( "%u\n", 890 );
   printf( "%u\n", -890 );  // Not allowed
}
```

```
890
890
890
-890
32000
200000000
890
3246435674
```

Printing Floating-Point number

- Decimal point numbers like 0.01, 98.07 or -23.78

Conversion Specifier	Description	Example
e or E	Display a floating-point value in exponential notation.	printf(“%e”,-1234567.89);
f or F	Display floating-point values in fixed-point notation	printf(“%f”,1234567.89);
g or G	Display a floating-point value in either the floating-point from f or the exponential form e based on the magnitude of the value	printf(“%g”, 1234567.89);
L	Used before any floating-point conversion specifier to indicate that a long double is displayed.	printf(“%lf”,1234567.89L);

```
#include <stdio.h>
int main( void )
{
   printf( "%e\n", 1234567.89 );
   printf( "%e\n", -1234567.89 );//minus
  prints
   printf( "%E\n", 1234567.89 );
   printf( "%f\n", 1234567.89 );
   printf( "%g\n", 1234567.89 );
   printf( "%G\n", 1234567.89 );
}
```

```
1.234568e+006
-1.234568e+006
1.234568E+006
1234567.890000
1.234568e+006
1.234568E+006
```

Printing Strings and Characters

Character = ‘A’ and String= “This is string”

Conversion Specifier	Description	Example
c	Display a single character argument.	printf(“%c”, ‘A’);
s	Displays a string and requires a pointer to a character argument.	printf(“%s”, “This is String”);

Conversion Specifier s causes characters to be printed until a terminating null(‘\0’) character is encountered.

```c
#include <stdio.h>
int main( void )
{
   char character = 'A'; // initialize char
   char string[] = "This is a string"; // initialize char
   array

   printf( "%c\n", character );
   printf( "%s\n", "This is a string" );
   printf( "%s\n", string );

}
```

```
A
This is string
This is string
```

Other Conversion Specifier

Pointer holds the address of another variable.

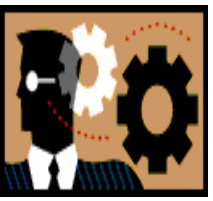
Conversion Specifier	Description	Example

p	Display a pointer value	Int *ptr=&score;
		printf("%p", ptr);
		printf("%p", &score);

%	Displays the percent character.	printf("a%%");

```c
#include <stdio.h>
int main( void )
{
   int *ptr; // define pointer to int
   int x = 12345; // initialize int x

   ptr = &x; // assign address of x to ptr
   printf( "The value of ptr is %p\n", ptr );
   printf( "The address of x is %p\n\n", &x );

   printf( "Printing a %% in a format control string\n" );
}
```

```
The value of ptr is 002ER443
The address of x is 002ER443

Printing a % in a format control string
```



How?

- Till now we have displayed numbers in left justified manner
- Consider the program that displays

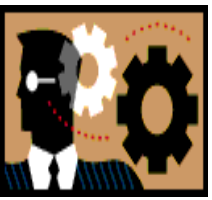
```
    1
   12
  123
 1234
12345
```

Printing with Field widths

- Field width: the exact size of field in which data is printed is specified by field width.
- The data is printed in the specified field and **right justified**.
- The integer representing the width size is inserted between percent sign(%) and the conversion specifier(e.g. %8d).

```
#include <stdio.h>

int main()
{
  printf( "%4d\n", 123 );
   printf( "%4d\n", 1234 );
   printf( "%4d\n\n", 12345 );

  }
```

```
 123
1234
12345

-
```



How?

- Dividing 7 by 3

Answer :

2.33333333333……

But the required output is

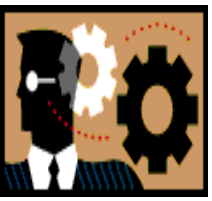
2.3333

Printing with Precision

- Specifies precision with which data is printed.
- Precision with **integer** conversion specifier indicates **the minimum number of digits to be printed.**
- Precision with **floating-point** conversion specifier indicates **the number of digits to appear after the decimal point.**
- Precision with **string** specifier indicates **the maximum number of characters to be written from the string.**

```c
#include <stdio.h>
int main( void )
{
   int i = 873; // initialize int i
   double f = 123.94536; // initialize double f
   char s[] = "Happy Birthday"; // initialize char array s

   printf( "Using precision for integers" );
   printf( "%.4d \n %.9d \n\n", i, i );

   printf( "Using precision for floating-point numbers" );
   printf( "%.3f \n %.3e\n %.3g \n\n", f, f, f );

   printf( "Using precision for strings" );
   printf( "%.11s \n", s );
}
```

```
Using precision for integers
0873
000000873

Using precision for floating-point numbers
123.945
1.239e+002
124

Using precision for strings
Happy Birth
```



How?

Suppose the output required is

```
First\one
there "
There's
```

Printing literals and escape sequences

Escape sequence	Description
\’	Output the single quote(‘) character
\”	Output the double quote(“) character
\\	Output the backslash (\) character
\a	Cause an audible(bell)
\b	Move the cursor back one position on the current line
\n	Move the cursor to the beginning of the next line
\t	Move the cursor to the next horizontal tab position

Formatted Functions

The scanf() function: (Formatted input)

scanf() is a function that reads data from the keyboard. It interprets character input to the computer and stores the interpretation in specified variable(s).

In general terms, the scanf function is written as:

Syntax

```
scanf (format-control-string, arg1, arg2,……., argN);
```

- The format-control-string can contain:
  - Describes the format of the input.
  - Conversion specifications that begin with a % sign.
- The arguments are the pointers to variables in which the input will be stored.

Example:

```
scanf("%s %d %f", name, &age,
  &salary);
```

In this :-

"%s %d %f"- is a control string.

name – is a string argument and it's a array name and implicit memory address reference.

age – is a decimal integer variable preceded by &.

salary – is floating-point value preceded by &.

Reading data

Conversion Specifier	Description
d	Read signed decimal integer
i	Read a signed decimal integer
u	Read an unsigned decimal integer
h or l	Used before any integer conversion specifier to indicate that a short or long integer is to be input, respectively
e, E, f, g, G	Read a floating-point value
c	Read a character
s	Read a string
p	Read an address
%	Skip the percent sign(%) in the input

```c
#include <stdio.h>
int main( void )
{  int a, c;
   float f;
   char day[10];
   printf( "Enter integers: " );
   scanf( "%d %u", &a, &c);

   printf( "Enter floating-point numbers:" );
   scanf( "%f", &f);

   printf( "%s", "Enter a string: " );
   scanf( "%8s", day );
}
```

```
Enter integers: -89 23
Enter floating-point numbers:
1.34256
Enter a string:
monday
```

Outline

- Unformatted Input/Output functions
  - getchar()
  - putchar()
  - getch()
  - putch()
  - gets()
  - puts()

Unformatted Functions

- C has three types of I/O functions:
  i. Character I/O
  ii. String I/O
  iii. File I/O

getchar()

- This function reads a character-type data from standard input.
- It reads one character at a time till the user presses the enter key.

Syntax

```
Variable-name = getchar();
```

Example:

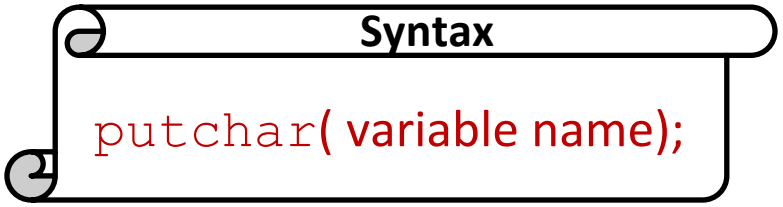
```
char c;
c = getchar();
```

```
#include<stdio.h>
 int main()
{
 char c;
 printf("enter a character");
 c=getchar();
 printf("c = %c ",c);
 }
```

```
Enter a character   k
c = k
```

putchar()

- This function prints one character on the screen at a time which is read by standard input.



Example: char c= ‘c’;

putchar (c);

```
#include<stdio.h>
 int main()
{
char ch;
printf("enter a character: ");
scanf("%c", &ch);
putchar(ch);
}
```

```
enter a character: r
r
```

getch() & getche()

- These functions read any alphanumeric character from the standard input device
- The character entered is not displayed by the getch() function until enter is pressed
- The **getche()** accepts and displays the character.
- The **getch()** accepts but does not display the character.

Syntax

```
getche();
```

```
#include<stdio.h>
 int main()
{
    printf("Enter two alphabets:");
    getche();
    getch();
}
```

```
Enter two alphabets a
```

putch()

This function prints any alphanumeric character taken by the standard input device

```
#include<stdio.h>
 int main()
{
      char ch;
      printf("Press any key to continue");
      ch = getch();
      printf(" you pressed:");
      putch(ch);
}
```

```
Press any key to continue
You pressed : e
```

gets()

String I/O

- This function is used for accepting any string until enter key is pressed (string will be covered later)

Syntax

```
char str[length of string in number];
gets(str);
```

```
#include<stdio.h>
 int main()
     {
     char ch[30];
     printf("Enter the string:");
     gets(ch);
     printf("Entered string: %s", ch);
     }
```

```
Enter the string: Use of data!
Entered string: Use of data!
```

puts()

- This function prints the string or character array. It is opposite to gets()

Syntax

```
char str[length of string in number];
gets(str);
puts(str);
```

```
#include<stdio.h>
 int main()
       {
       char ch[30];
       printf("Enter the string:");
       gets(ch);
       puts("Entered string:");
       puts(ch);
       }
```

```
Enter the string: puts is in use
Entered string: puts is in use
```

Q1

```
#include<stdio.h>
int main()
{
float x=12.6784;
printf("%.3f",x);
return 0;
}
```

A. 12.678
B. 12.6
C. 12.679
D. 12.0

Q2

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=3.456;
printf("%lf",floor(x));
return 0;
}
```

A. 3.460000
B. 3.000000
C. 4.000000
D. 3.500000

Q3

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=3.001;
printf("%lf",ceil(x));
return 0;
}
```

A. 3.010000
B. 3.000000
C. 4.000000
D. 3.500000

Q4

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=10.0,y=7.0;
printf("%lf",fmod(x,y));
return 0;
}
```

A. 1.000000
B. 3.000000
C. 1.428571
D. Error

Q5

```
#include<stdio.h>
#include<math.h>
int main()
{
int x;
x=printf("ABC");
printf(" %d",x);
return 0;
}
```

A. ABC
B. ABC 1
C. 3 ABC
D. ABC 3

Q6

Which of the following is a non-standard input unformatted function in C?

A. printf()
B. getch()
C. getchar()
D. scanf()

Q7

Which of the following unformatted function waits for the user to press the enter key, after the character input is provided?

A. putch()

B. getch()

C. getchar()

D. getche()

Type conversion

- Conversion from one type to another
- It can happen in two ways:

1) Implicit type conversion
2) Explicit type conversion

Implicit type conversion(or automatic type conversion)

- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.
- char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double
- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Example of implicit conversion

```
// An example of implicit conversion
#include<stdio.h>
int main()
{
        int x = 10; // integer x
        char y = 'a'; // character c

        // y implicitly converted to int. ASCII
        // value of 'a' is 97
        x = x + y;

        // x is implicitly converted to float
        float z = x + 1.0;

        printf("x = %d, z = %f", x, z);
        return 0;
}
```

Output:

x = 107, z = 108.000000

Explicit type conversion

- This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.
- The syntax in C:

(type) expression

- Type indicated the data type to which the final result is converted.

```
// C program to demonstrate explicit type casting
#include<stdio.h>
int main()
{
        double x = 1.2;

        // Explicit conversion from double to int
        int sum = (int)x + 1;

        printf("sum = %d", sum);

        return 0;
}
```

Output:

sum=2

Type modifiers

- Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
- short, long, signed and unsigned are the type modifiers in C

Type modifiers