Unit 3 : Input/output Management

Er. Nipun Thapa

3.1. Input/output Management

- **Input** means to provide the program with some data to be used in the program and **Output** means to display data on screen or write the data to a printer or a file.
- C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result.
- All these built-in functions are present in C header files.
- C language has standard libraries that allow input and output in a program.
- The **stdio.h** or **standard input output library** in C that has methods for input and output.
- The input/output functions are classified into two types:
  1. Formatted functions
  2. Unformatted functions

3.2. Formatted I/O

- Formatted console input/output functions are used to take one or more inputs from the user at console and it also allows us to display one or multiple values in the output to the user at the console.
- These functions allow to supply input or display output in user desired format.
- **printf()** and **scanf()** are examples for formatted input and output functions.
- Formatted input and output functions contain format specifier in their syntax.
- They are used for storing data more user friendly.
- They are used with all data types.

3.2. Formatted I/O

Formatted Input

- The built-in function **scanf()** can be used to input data into the computer from a standard input device.
- The function can be used to input an numerical value, single character or string.
- The general form of **scanf()** is:
  - **scanf("Control string", arg1, arg2,……., argN);**
- The control string refers to the field format in which the data is to be entered.
- The arguments **arg1,arg2,….,argN** specify the address of locations where the data is stored.
- Generally, arguments are precede by ampersand(&) to denote memory address.

3.2. Formatted I/O

Formatted Input

- **scanf()** function reads all types of data value from input devices (or) from a file.
- The address operator **'&'** is to indicate the memory location of the variable.
- This memory location is used to store the data which is read through keyboard.
- The **scanf()** method, in C, reads the value from the console as per the type specified.

3.2. Formatted I/O

Formatted Input

Syntax:

scanf("%X", &variableOfXType);

where **%X** is the format specifier in C.

It is a way to tell the compiler what type of data is in a variable *and* ***&*** *is the address operator in C, which tells the compiler to change the real value of this variable, stored at this address in the memory.*

3.2. Formatted I/O

Format String	Meaning
%d	Scan or print an integer as signed decimal number
%f	Scan or print a floating point number
%c	To scan or print a character
%s	To scan or print a character string. The scanning ends at whitespace.

3.2. Formatted I/O

Formatted Output

- The function printf() is used for formatted output to standard output based on a format specification. The format specification string, along with the data to be output, are the parameters to the printf() function.

Syntax:

printf (format, data1, data2,……..);

- In this syntax format is the format specification string. This string contains, for each variable to be output, a specification beginning with the symbol % followed by a character called the conversion character.

Example:

printf (“%c”, data1);

- The character specified after % is called a conversion character because it allows one data type to be converted to another type and printed.

3.2. Formatted I/O (Example)

```
#include<stdio.h>
int main()
{
   int a;
   printf("Enter value of a:");
   scanf("%d", &a);
   printf(" a = %d", a);
   return 0;
}
```

3.2. Formatted I/O (Example)

- Write a program to show the usage of whitespace character in scanf() function.

```
#include<stdio.h>
int main()
{
        int n1;
        char ch;
        printf("enter a number:");
        scanf("%d",&n1);
        printf("enter a character:");
        scanf("%c",&ch);
        printf("\nnumber: %d \n character : %c",n1,ch);
        return 0;
}
```

3.2. Formatted I/O (Example)

- Write a program to show the use of ordinary character in control string of scanf() function.

```
#include<stdio.h>
Int main()
{
        int day, month, year;
        printf(“enter your date of birth in sequence”);
        printf(“day, month and then year seperated by / character:”);
        scanf(“%d/%d/%d”,&day,&month,&year);
        printf(“your date of birth is : %d day %d month %d year”,day, month, year);
        return 0;
}
```

3.2. Formatted I/O (Example)

- Write a program to read marks of a student in a subject. The input marks should be less than 100 (ie only two digits).

```
#include<stdio.h>
Int main()
{
    int mark;
    printf("enter your mark (<100):");
    scanf("%2d",&mark);
    printf("your entered marks: %d", mark);
    return 0;
}
```

3.2. Formatted I/O (Example)

- Program to show full name

```
#include<stdio.h>
int main()
{
        char name[100];
        printf("Enter your full name:");
        scanf("%[^\n]",&name);
        printf("Your full name is %s",name);
        return 0;
}
```

3.2. Unformatted I/O

- These functions are the most basic form of input and output and they do not allow to supply input or display output in user desired format.
- getch(), getche(), getchar(), gets(), puts(), putchar() etc. are examples of unformatted input output functions.
- Unformatted input and output functions do not contain format specifier in their syntax.
- They are used for storing data more compactly.
- They are used mainly for character and string data types.

3.2. Unformatted I/O

1. getch(), getche() and putch()

The function **getch**() and **getche**() reads a single character in the instant it is typed without waiting for the enter key to be hit.

The difference between them is that **getch**() reads the character typed without echoing it on the screen, while **getche**() reads the character and echoes (displays) it onto the screen.

Syntax:

```
character_variable = getch();
character_variable = getvhe();
```

In both functions, the character typed is assigned to the char type variable character_variable.

The function putch() prints a character onto the screen.

Syntax: putch(character_variable)

3.2. Unformatted I/O

- Program to read two character from keyboard and display the character.

```
#include<stdio.h>
int main()
{
    char ch1,ch2;
    printf("Enter 1st character");
    ch1=getch();
    printf("enter 2nd character");
    ch2=getch();
     printf("\n1st character:");
     putch(ch1);
     printf("\n2nd character:");
     putch(ch2);
     return 0;
}
```

3.2. Unformatted I/O

2. getchar() & putchar() functions

- The **getchar**() function reads a character from the terminal and returns it as an integer. This function reads only single character at a time.
- The **putchar**() function displays the character passed to it on the screen and returns the same character. This function too displays only a single character at a time.
- In case you want to display more than one characters, use **putchar**() method in a loop.

3.2. Unformatted I/O

getchar() & putchar() functions example

```
#include <stdio.h>
int main( )
{
        int c;
        printf("Enter a character");
        c = getchar();
        putchar(c);
        return 0;
}
```

3.2. Unformatted I/O

3. gets() & puts() functions

- The gets() function reads a line from **stdin**(standard input) into the buffer pointed to by str pointer, until either a terminating newline or EOF (end of file) occurs.
- The puts() function writes the string str and a trailing newline to **stdout**.
- str → This is the pointer to an array of chars where the C string is stored. (Ignore if you are not able to understand this now.)

3.2. Unformatted I/O

gets() & puts() functions example

```
#include<stdio.h>
int main()
{
        char str[100];
        printf("Enter a string");
        gets( str );
        puts( str );
        return 0;
}
```

Difference between scanf() and gets()

- The main difference between these two functions is that **scanf**() stops reading characters when it encounters a space, but gets() reads space as character too.
- If you enter name as **Study Tonight** using **scanf**() it will only read and store **Study** and will leave the part after space. But **gets**() function will read it completely.

3.3. Conversion character

- The **Conversion character** for printf() is similar to that of scanf().
- The **Conversion character** depend upon the type of the variable or constant to be displayed.

3.3. Conversion character

Conversion Character	Displays Argument (Variable's Contents) As
%c	Single character
%d	Signed decimal integer (int)
%e	Signed floating-point value in E notation
%f	Signed floating-point value (float)
%g	Signed value in %e or %f format, whichever is shorter
%i	Signed decimal integer (int)
%o	Unsigned octal (base 8) integer (int)
%s	String of text
%u	Unsigned decimal integer (int)
%x	Unsigned hexadecimal (base 16) integer (int)
%%	(percent character)

3.4. Escape Sequences

Escape Sequence	Meaning
\n	New Line
\t	Horizontal Tab
\b	BackSpace
\r	Carriage Return
\a	Audible bell
\'	Printing single quotation
\"	printing double quotation
\?	Question Mark Sequence
\\	Back Slash
\f	Form Feed
\v	Vertical Tab
\0	Null Value
\nnn	Print octal value
\xhh	Print Hexadecimal value

Finished

Unit 3