CS101: Problem Solving through C Programming

Sachchida Nand Chaurasia

Assistant Professor

Department of Computer Science

Banaras Hindu University

Varanasi

Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

January 29, 2021

Use of & in scanf() but not in printf() I

Why there is need of using & in case if scanf function while not in case of printf function.

Examples:

```c
scanf("%d %d", &a, &b);
printf("%d %d", a, b);
```

✓ As a and b above are two variable and each has their own address assigned but instead of a and b, we send the address of a and b respectively.

✓ The reason is, scanf() needs to modify values of a and b and but they are local to scanf().

Use of & in scanf() but not in printf() II

✓ So in order to reflect changes in the variable a and b of the main function, we need to pass addresses of them. We cannot simply pass them by value.

✓ But in case of printf function as we are only going to print the values of the variables in output console, there are no changes going to be made in variable a and b's values.

✓ So it is not required to send their addresses.

Another answer:

✓ scanf() stands for scan formatted string.

✓ Now while scanning input from standard input stream, scanf() needs to put that input data into somewhere.

Use of & in scanf() but not in printf() III

✓ To store the formatted input data, scanf() needs to know the memory location of a variable of same data type.

✓ That is why scanf() needs a pointer (a pointer in C stores memory location of a variable or an expression) to store input.

✓ The address-of operator () preceeding a variable i.e. var Indicates the memory location of variable 'var'.

```c
int var;
scanf("%d",&var);

char str[20];
scanf("%s",str);
```

Use of & in scanf() but not in printf() IV

✓ For the second example we do not need the address-of operator because C treats array name variable as a pointer.

✓ printf() is the reverse function of scanf().

✓ It prints the formatted string to the standard output.

✓ Printf doesn’t need any memory location to print the output, it only needs the variable to get the data and formats it according to the format specifier.

```
printf("%c in ASCII is %d",65,65);
The output will be: A in ASCII is 65
```

Format specifiers in C I

✓ The format specifier is used during input and output.

✓ It is a way to tell the compiler what type of data is in a variable during taking input using scanf() or printing using printf().

✓ Some examples are %c, %d, %f, etc.

✓ This function prints the character on standard output and returns the number of character printed the format is a string starting with % and ends with conversion character (like c, i, f, d, etc.).

✓ Between both, there can be elements governing the printing format. Below is its description

1. A minus(-) sign tells left alignment.

Format specifiers in C II

2. A number after % specifies the minimum field width to be printed if the characters are less than the size of width the remaining space is filled with space and if it is greater than it printed as it is without truncation.
3. A period(.) symbol separate field width with the precision.

Precision tells the maximum number of digits in integer, characters in string and number of digits after decimal part in floating value.

Format specifiers in C III

Format Specifier	Type
%c	Character
%d	Signed integer
%e or %E	Scientific notation of floats
%f	Float values
%g or %G	Similar as %e or %E
%hi	Signed integer (short)
%hu	Unsigned Integer (short)
%i	Unsigned integer
%l or %ld or %li	Long
%lf	Double

Format Specifier	Type
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character

Format specifiers in C IV

```c
#include <stdio.h>
main()
{
        char ch = 'B';
        printf("%c\n", ch); //printing character data
        //print decimal or integer data with d and i
        int x = 45, y = 90;
        printf("%d\n", x);
        printf("%i\n", y);
        float f = 12.67;
        printf("%f\n", f); //print float value
        printf("%e\n", f); //print in scientific notation
        int a = 67;
```

Format specifiers in C V

```
    printf("%o\n", a); //print in octal format
    printf("%x\n", a); //print in hex format
    char str[] = "Hello World";
    printf("%s\n", str);
    printf("%20s\n", str); //shift to the right 20 characters including the string
    printf("%-20s\n", str); //left align
    printf("%20.5s\n", str); //shift to the right 20 characters including the string, and print string up to 5 character
    printf("%-20.5s\n", str); //left align and print string up to 5 character
}
```

Format specifiers in C VI

Character format specifier : %c:

```
#include <stdio.h>
int main()
{
    char ch = 'A';
    printf("%c\n", ch);
    return 0;
}

Output: ???
```

Format specifiers in C VII

Integer format specifier : %d, %i:

```
#include <stdio.h>
int main()
{
    int x = 45, y = 90;
    printf("%d\n", x);
    printf("%i\n", x);
    return 0;
}

Output: ???
```

Format specifiers in C VIII

Floating-point format specifier : %f, %e or %E:

```
#include <stdio.h>
int main()
{
    float a = 12.67;
    printf("%f\n", a);
    printf("%e\n", a);
    return 0;
}

Output: ???
```

Format specifiers in C IX

Unsigned Octal number for integer : %o:

```
#include <stdio.h>
int main()
{
    int a = 67;
    printf("%o\n", a);
    return 0;
}

Output: ???
```

Format specifiers in C X

Unsigned Hexadecimal for integer : %x, %X:

```
#include <stdio.h>
int main()
{
    int a = 15;
    printf("%x\n", a);
    return 0;
}

Output: ???
```

Format specifiers in C XI

String printing : %s:

```
#include <stdio.h>
int main()
{
    char a[] = "BSCPMKSMK";
    printf("%s\n", a);
    return 0;
}

Output: ???
```

Format specifiers in C XII

More formatting:

```
int main()
{
    char str[] = "BSCPMKSMK";
    printf("%20s\n", str);
    printf("%-20s\n", str);
    printf("%20.5s\n", str);
    printf("%-20.5s\n", str);
    return 0;
}

Output: ???
```

printf(), sprintf() and fprintf() in C I

printf() : It returns total number of Characters Printed, Or negative value if an output error or an encoding error.

✓ The format specifier in printf():

```
int printf(char *format, arg1, arg2,...);
```

```
#include <stdio.h>
int main()
{
    char st[] = "CODING";
    printf("%s", st);
    printf("While printing ");
    printf(", the value returned by printf() is : %d",printf("%s"
    , st));
```

printf(), sprintf() and fprintf() in C II

```
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    long int n = 123456789;
    printf("While printing ");
    printf(", the value returned by printf() is : %d",printf("%ld"
    , n));
    return 0;
}
```

printf(), sprintf() and fprintf() in C III

sprintf(): String print function instead of printing on console store it on char buffer which are specified in sprintf.

```c
int sprintf(char *str, const char *string,...);
```

```c
// Example program to demonstrate sprintf()
#include<stdio.h>
int main()
{
    char buffer[50];
    int a = 10, b = 20, c;
    c = a + b;
    sprintf(buffer, "Sum of %d and %d is %d", a, b, c);

    // The string "sum of 10 and 20 is 30" is stored
```

printf(), sprintf() and fprintf() in C IV

```
    // into buffer instead of printing on stdout
    printf("%s", buffer);

    return 0;
}
Output :

Sum of 10 and 20 is 30
```

fprintf(): fprintf is used to print the string content in file but not on stdout console.

```
int fprintf(FILE *fptr, const char *str, ...);
```

printf(), sprintf() and fprintf() in C V

```
#include<stdio.h>
int main()
{
    int i, n=2;
    char str[50];

    //open file sample.txt in write mode
    FILE *fptr = fopen("sample.txt", "w");
    if (fptr == NULL)
    {
        printf("Could not open file");
        return 0;
    }
```

printf(), sprintf() and fprintf() in C VI

```
    puts("Enter a name");
    gets(str);
    fprintf(fptr,"%s\n",str);
    fclose(fptr);
    return 0;
}
Input: GeeksforGeeks
Output :  GeeksforGeeks
```

scanf() and fscanf() in C I

scanf() : scanf reads formatted input from stdin.

Following is the declaration for scanf() function.

```c
int scanf(const char *format, ...)
```

✓ It returns total number of Inputs Scanned successfully, or EOF if input failure occurs before the first receiving argument was assigned.

```c
#include <stdio.h>
int main ()
{
        char str1[20], str2[30];
        printf("Enter name: ");
        scanf("%s", str1);
        printf("Enter your website name: ");
```

scanf() and fscanf() in C II

```c
        scanf("%s", str2);
        printf("Entered Name: %s\n", str1);
        printf("Entered Website:%s", str2);
        return(0);
}
```

```c
#include <stdio.h>
int main()
{
    char a[100], b[100], c[100];
    // scanf() with one input
    printf("\n First scanf() returns : %d",scanf("%s", a));
    // scanf() with two inputs
    printf("\n Second scanf() returns : %d",scanf("%s%s", a, b));
```

scanf() and fscanf() in C III

```
    // scanf() with three inputs
    printf("\n Third scanf() returns : %d",scanf("%s%s%s", a, b, c
    ));
    return 0;
}
-----------
Input:
Hey!
welcome to
BSC PMKSMK Class
--------------------
Output: ????
```

scanf() and fscanf() in C IV

fscanf() : Tired of all the clumpsy syntax to read from files? well, fscanf comes to the rescue.

```
int fscanf(FILE *ptr, const char *format, ...)
```

✓ fscanf reads from a file pointed by the FILE pointer (ptr), instead of reading from the input stream.

Consider the following text file abc.txt:

NAME AGE CITY abc 12 hyderbad bef 25 delhi cce 65 bangalore

Now, we want to read only the city field of the above text file, ignoring all the other fields. A combination of fscanf and the trick mentioned above does this with ease

scanf() and fscanf() in C V

```c
/*c program demonstrating fscanf and its usage*/
#include<stdio.h>
int main()
{
        FILE* ptr = fopen("abc.txt","r");
        if (ptr==NULL)

    {
                printf("no such file.");
                return 0;

    }

```

scanf() and fscanf() in C VI

```
        /* Assuming that abc.txt has content in below
           format
           NAME    AGE   CITY
           abc     12    hyderbad
           bef     25    delhi
           cce     65    bangalore */
        char buf[100];
        while (fscanf(ptr,"%*s %*s %s ",buf)==1)
            printf("%s\n", buf);

        return 0;
}
Output:
```

scanf() and fscanf() in C VII

```
CITY
hyderbad
delhi
bangalore
```

Scansets in C

✓ scanf family functions support scanset specifiers which are represented by %[].

✓ Inside scanset, we can specify single character or range of characters.

✓ While processing scanset, scanf will process only those characters which are part of scanset.

✓ We can define scanset by putting characters inside squre brackets.

✓ Please note that the scansets are case-sensitive.

scanf() and fscanf() in C VIII

We can also use scanset by providing comma in between the character you want to add.

example: scanf(%s[A-Z,_,a,b,c]s,str);

```c
/* A simple scanset example */
#include <stdio.h>

int main(void)
{
    char str[128];
    printf("Enter a string: ");
    scanf("%[A-Z]s", str);
    printf("You entered: %s\n", str);
    return 0;
```

scanf() and fscanf() in C IX

```
}
  Enter a string: GEEKs_for_geeks
  You entered: GEEK
```

```c
/* Another scanset example with ^ */
#include <stdio.h>
int main(void)
{
    char str[128];
    printf("Enter a string: ");
    scanf("%[^o]s", str);
    printf("You entered: %s\n", str);
    return 0;
}
```

scanf() and fscanf() in C X

```
Enter a string: http://geeks for geeks
You entered: http://geeks f
```

getchar(), getc() and gets() I

gets():

Reads characters from the standard input (stdin) and stores them as a C string into str until a newline character or the end-of-file is reached.

Syntax:

```
char * gets ( char * str );
str :Pointer to a block of memory (array of char)
where the string read is copied as a C string.
returns : the function returns str
```

getchar(), getc() and gets() II

✓ It is used to read string from user until newline character not encountered.

Example : Suppose we have a character array of 15 characters and input is greater than 15 characters, gets() will read all these characters and store them into variable.Since, gets() do not check the maximum limit of input characters, so at any time compiler may return buffer overflow error.

```c
// C program to illustrate
// gets()
#include <stdio.h>
#define MAX 15
int main()
{
```

getchar(), getc() and gets() III

```c
    char buf[MAX];
    printf("Enter a string: ");
    gets(buf);
    printf("string is: %s\n", buf);
    return 0;
}
```

getchar(), getc() and gets() IV

gets() is risky to use!

✓ It is not safe to use because it does not check the array bound.

```c
void read()
{
        char str[20];
        gets(str);
        printf("%s", str);
        return;
}
```

getchar(), getc() and gets() V

✓ The code looks simple, it reads string from standard input and prints the entered string, but it suffers from Buffer Overflow as gets() doesn't do any array bound testing. gets() keeps on reading until it sees a newline character.

✓ To avoid Buffer Overflow, fgets() should be used instead of gets() as fgets() makes sure that not more than MAX_LIMIT characters are read.

```
#define MAX_LIMIT 20
void read()
{
        char str[MAX_LIMIT];
        fgets(str, MAX_LIMIT, stdin);
        printf("%s", str);
```

getchar(), getc() and gets() VI

```
}
```

Problem with scanf() when there is fgets()/gets()/scanf() after it I

```
// C program to demonstrate the problem when
// fgets()/gets() is used after scanf()
#include<stdio.h>
int main()
{
        int x;
        char str[100];
        scanf("%d", &x);
        fgets(str, 100, stdin);
        printf("x = %d, str = %s", x, str);
        return 0;
}
Input:
10
```

Problem with scanf() when there is fgets()/gets()/scanf() after it II

```
test

Output:
x = 10, str =
```

✓ The problem with above code is scanf() reads an integer and leaves a newline character in buffer. So fgets() only reads newline and the string “test” is ignored by the program.

The similar problem occurs when scanf() is used in a loop.

Problem with scanf() when there is fgets()/gets()/scanf() after it III

```c
// C program to demonstrate the problem when
// scanf() is used in a loop
#include<stdio.h>

int main()
{
        char c;
        printf("......Enter q to quit......\n");
        do

    {
                printf("Enter a character\n");
                scanf("%c", &c);
```

Problem with scanf() when there is fgets()/gets()/scanf() after it IV

```
                printf("%c\n", c);

    }
        while (c != 'q');
        return 0;
}
Input

a
b
q
Output:

......Enter q to quit......
```

Problem with scanf() when there is fgets()/gets()/scanf() after it V

```
Enter a character
a
Enter a character


Enter a character
b
Enter a character


Enter a character
q
```

Problem with scanf() when there is fgets()/gets()/scanf() after it VI

✓ We can notice that above program prints an extra "Enter a character" followed by an extra new line. This happens because every scanf() leaves a newline character in buffer that is read by next scanf.

How to solve above problem?

✓ We can make scanf() to read a new line by using an extra "
n";; i.e., scanf("%d", &x) . In fact scanf("%d ", &x) also works (Note extra space).

✓ We can add a getchar() after scanf() to read an extra newline.

putc(), puts(), fputc(), fputs() I

putc():

putc(int char, FILE *stream) writes a character (an unsigned char) specified by the argument char to the specified stream and advances the position indicator for the stream.

```
int putc(int char, FILE *stream)
```

```
#include <stdio.h>

int main ()
{
        FILE *fp;
        int ch;

```

putc(), puts(), fputc(), fputs() II

```
    fp = fopen("file.txt", "w");
    for( ch = 33 ; ch <= 100; ch++ )
  {
      putc(ch, fp);

  }
    fclose(fp);

    return(0);
}
```

putc(), puts(), fputc(), fputs() III

puts() : The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console.

```c
// C program to show the use of puts
#include <stdio.h>
int main()
{
    puts("Geeksfor");
    puts("Geeks");

    getchar();
    return 0;
```

putc(), puts(), fputc(), fputs() IV

```
}
```

fputc() : fputc writes a character (an unsigned char) specified by the argument char to the specified stream and advances the position indicator for the stream.

```
f
```

putc(int char, FILE *stream)

putc(), puts(), fputc(), fputs() V

```c
#include <stdio.h>

int main ()
{
        FILE *fp;
        int ch;

        fp = fopen("file.txt", "w+");
        for( ch = 33 ; ch <= 100; ch++ )
    {
          fputc(ch, fp);

    }
```

putc(), puts(), fputc(), fputs() VI

```c
        fclose(fp);

        return(0);
}
```

fputs() : fputs() is a function in C programming language that writes an array of characters to a given file stream.

```c
int fputs(const char *str, FILE *stream)
```

putc(), puts(), fputc(), fputs() VII

```c
// C program to show the use of fputs and getchar
#include <stdio.h>
int main()
{
    fputs("Geeksfor", stdout);
    fputs("Geeks", stdout);

    getchar();
    return 0;
}
```