FY BBA-CA (Semester I) – C Programming

# C Programming Structures

**Prof. Ashvini swapnil Tanpure**
**Department of BBA(CA)**
**Hutatma Rajguru Mahavidyalay, Rajgurunagar**

# Alphabets in C:

| | |
|---|---|
| Alphabets | A, B, ….., Y, Z<br>a, b, ……, y, z |
| Digits | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| Special Symbols | + - * / = % & #<br>! ? ^ “ „ ~ \ \|<br>< > ( ) [ ] { } |

# C – Character Set:

Character Set:

In C, the characters are grouped in to

Letters, Digits, Special Characters, and While Spaces

Tokens:

The smallest individual units are known as tokens.

C program is written using these tokens.

Keywords:

The words that are having special predefined meaning to the compiler, that can not be changed.

Keywords are basic building blocks of C program.

It must be written in small case letters only.

32 keywords are available in C.

## C – Character Set:

Identifiers:

It is the name given to variables, functions, and arrays.

Both uppercase and lower case letter are permitted, But lowercase letters are used commonly.

No other special character is allowed in Identifier, except underscore.

Underscore used to link two words in long identifiers.

Constants: In C, constants refer to fix value, that can not be change during the execution of the program.

## Constants:

Integer: Sequence of digits.

Types: Decimal (0-9) => 124, 5264, -5478

Octal (0-7), => 037, 0551, 0456

Hexadecimal (0-10 & A-F). => 0x2, 0x9F, 0x2D

Real: The sequence of digits with decimal points.

The numbers can be used to represent distance, height, prices.

0.0083

-0.75

425.36

7500000000 => 7.5E9

-0.000000368 => -3.68E-7

## Character:

Single Character Constant: It contains single character enclosed in pair of single quote.

e.g.

„S“, „p“, „6“

String Constant: Sequence of characters enclosed in pair of double quote.

e.g.

“Hello”, “Computer”, “1972”, “X”

## Back slash characters / Escape Sequences :

These characters are used in output function to perform special function.

| Characters | Meaning |
|---|---|
| „\a“ | Audible alert (bell) |
| „\b“ | Back space |
| „\n“ | new line feed |
| „\r“ | Carriage return |
| „\t“ | Horizontal tab |
| „\v“ | Vertical tab |
| „\‘“ | Single Quote |
| „\”“ | Double quote |
| „\\“ | Back slash |
| „\0“ | null character |

## Variables in C:

Variable is a name given to a memory location where we can store different values of same type during the program execution.

Every variable must be declared in the declaration section before it is used.

Every variable must have a type that determines the range and type of values to be stored and size of the memory to be allocated.

Variable name may contains letters, digits, and underscore (_).

SRules for naming Variable:

1) Must start with letter and not with digit.
2) Should not contain any keywords defined in C.
3) Should not contain any special character defined in C except underscore (_).
4) White Space is not allowed in variable name.
5) Recognize length is 31 characters, but first 8 characters are used by many compilers.

## Variables in C:

Declaration:

It tells to the compiler to allocate required amount of memory with specified variable name and allows only specified type values into that memory location.

Declaration can be performed either before the function as global variables or inside any block or function.

But it must be at the beginning of block or function.

Syntax:

data_type variable_name;

e.g.

int number;

## Variables in C:

Definition:

It tells the compiler the initialization of variable declared earlier.

We can assign any initial value to the variable declared earlier as per the type of variable.

This initial value can be changed during execution of the program.

Example:

```
int number;
number=10;
```

We can do declaration and definition/initialization in one step as follows.

```
int number = 10;
```

## Data Types:

C language has rich set of data types. It is used to represent storage type of data. The variety of data types available, can be used as per the need of an application / program.

Data types are used to specify what kind of value can be stored in a variable.

The memory size and type of value of a variable are determined by variable data type.

1) Primitive Data Types.
2) Derived Data Types.
3) User Defined Data Types.

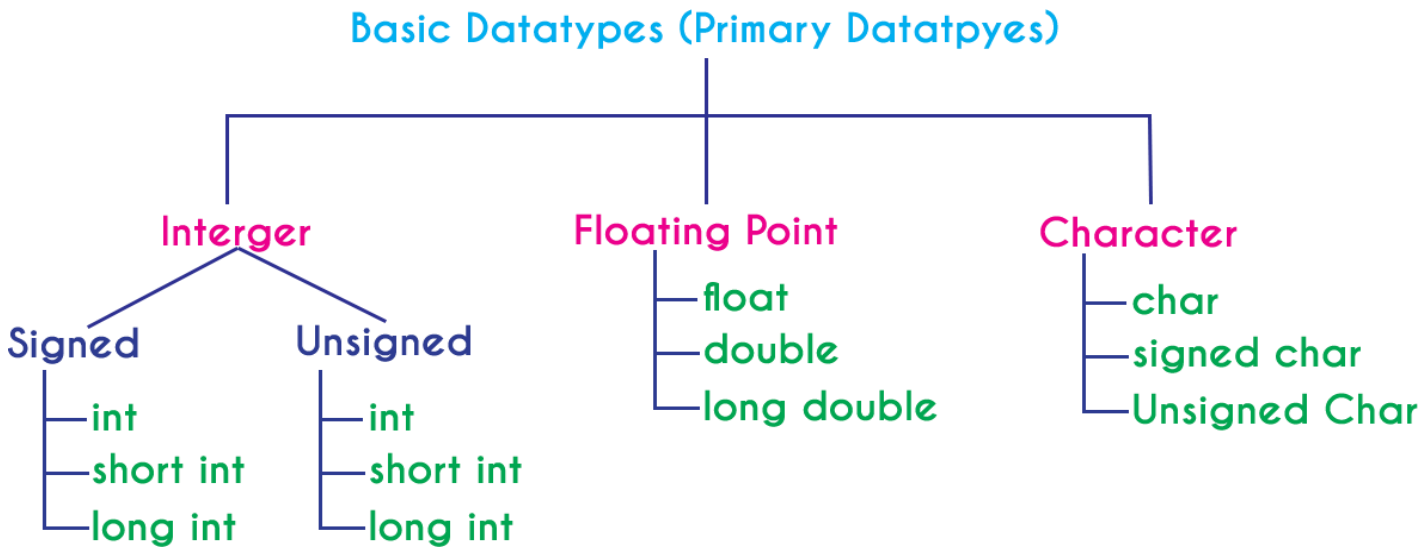
## Data Types:

1) Primitive Data Types:

These data types are already defined in the system. All the compilers support these primitive data types. It is also called as built in data type.

There are 4 types of primitive data types:

1. Integer (int)
2. Floating Point (float)
3. Double precision floating point (double)
4. Character (char)



## Data Types:

1. Integer (int):

Integers are whole number with range of values specified by a compilers.

We used keyword int to specify the integer data type.

The integer data type is used with different type modifiers like short, long, signed and unsigned.

The following table provides complete details about integer data type:

| Type | Size | Range | format Specifier |
|---|---|---|---|
| int | 2 Byte | -32768 to 32767 | %d |
| short int | 2 Byte | -32768 to 32767 | %d |
| long int | 4 Byte | -2,147,483,648 to 2,147,483,647 | %d |
| unsigned int | 2 Byte | 0 to 65325 | %u |
| unsigned long int | 4 Byte | 0 to 4,294,967,295 | %u |

## Data Types:

2. Floating Point (float):

It is set of numbers with decimal point value. It has two variants.

Float, and Double.

Float: It uses 32 bits (4Bytes) with 6 digits of precision to store floating point value.

Keyword float is used to defined float type of variable.

Double: It uses 64 bits (8Bytes) with 12 digits of precision to store floating point value.

Keyword double is used to defined double type of variable.

| Type | Size | Range | format Specifier |
|---|---|---|---|
| float | 4 Byte | 1.2E-38 to 3.4 E+38 | %f |
| double | 8 Byte | 2.3E-308 to 1.7E+308 | %ld |
| long double | 10 Byte | 3.4 E-4932 to 1.1E+4932 | %ld |

## Data Types:

4. Character (char):

A single character can be defined as character type data.

A character can be stored in single quote.

Keyword char is used to defined character type variable.

Character type can be either signed or unsigned.

| Type | Size | Range | Format Specifier |
|---|---|---|---|
| char | 1 Byte | -128 to 127 | %c |
| unsigned char | 1 Byte | 0 to 255 | %c |

## Derived Data Types:

It is user defined data types or secondary data types.

The derived data types are created using following concepts.

1. Array
2. Structure
3. Union
4. Enumeration

## User Defined Data Types:

Typedef:

C language supports feature type definition, which allows us to define an identifier that can be used as existing data types.

Syntax:

typedef type identifier;

Where,

typedef is keyword,

type is an existing data type,

identifier is an new name given to existing data type.

e.g.

typedef int marks;

Now marks can be used to define new identifier name to int.

marks sub1, sub2, sub3;

## User Defined Data Types:

Enumeration:

Enumeration (or enum) is a user defined data type in C.

It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

The keyword „enum“ is used to declare new enumeration types in C.

e.g.

```
enum State {Working = 1, Failed = 0};
```

Variables of type enum can also be defined. They can be defined in two ways:

```
enum week{Mon, Tue, Wed}; enum day;
```

Or

```
enum week{Mon, Tue, Wed}day
```

## User Defined Data Types:

Enumeration:

```
#include<stdio.h>
enum year{Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

int main()
{
   int i;
   for (i=Jan; i<=Dec; i++)
      printf("%d ", i);

   return 0;
}
```

O/P:

```
    0 1 2 3 4 5 6 7 8 9 10 11
```

## Operators & Expressions:

An operator is a symbol used to perform mathematical and logical operations in a program.

An operator is a special symbol that tells to the compiler to perform mathematical or logical operations.

Arithmetic:

The arithmetic operators are the symbols that are used to perform basic mathematical operations like addition, subtraction, multiplication, division and percentage modulo.

| | |
|---|---|
| Addition (+) : | 10 + 5 = 15 |
| Subtraction (-): | 10 – 5 = 5 |
| Multiplication(*): | 10 * 5 = 50 |
| Division (/): | 10 / 5 = 2 |
| Remainder(%): | 10 % 5 = 0 |

## Operators & Expressions:

Relational:

The relational operators are the symbols that are used to compare two values.

Every relational operator has two results TRUE or FALSE.

| | |
|---|---|
| Less than (<): | 10 < 5 is false |
| Greater than (>): | 10 > 5 is true |
| Less than or equal to (<=): | 10 <= 5 is false |
| Greater than or equal to(>=): | 10 >= 5 is true |
| Equal to (= =): | 10 = = 5 is false |
| Not equal to (!=): | 10 != 5 is true |

## Operators & Expressions:

Logical:

The logical operators are the symbols that are used to combine multiple conditions into one condition.

| | | |
|---|---|---|
| Logical And (&&): | 10 < 5 && 15 > 10 | is false |
| Logical OR (\| \|): | 10 < 5 \|\| 15 > 10 | is true |
| Logical Not(~): | !(10 < 5 && 15 > 10) | is true |

## Operators & Expressions:

Conditional:

The conditional operator is also called as ternary operator because it requires three operands.

This operator is used for decision making. In this operator, first we verify a condition, then we perform one operation out of the two operations based on the condition result.

If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed.

Syntax:

Condition ? TRUE Part : FALSE Part ;

e.g.

A = (10<15) ? 100 : 200 ;

A has value 100

## Operators & Expressions:

Assignment:

The assignment operators are used to assign right hand side value (Rvalue) to the left hand side variable (Lvalue).

The assignment operator is used in different variants along with arithmetic operators.

| Operators | Example | Meaning |
|---|---|---|
| = | X=10 | |
| += | X+=10 | X=X+10 |
| -= | X-+10 | X=X-10 |
| *= | X*=10 | X=X*10 |
| /= | X/=10 | X=X/10 |
| %= | X%=10 | X=X%10 |

## Operators & Expressions:

Bitwise:

The bitwise operators are used to perform bit level operations in C programming language.

When we use the bitwise operators, the operations are performed based on the binary values.

Let us consider two variable A=25 (11001) and B=20 (10100)

| Operator | Example | Meaning |
|---|---|---|
| & (Bitwise AND) | A & B | 10000 (16) |
| \| (Bitwise OR) | A \| B | 11101 (29) |
| ^ (Bitwise XOR) | A ^ B | 01101(13) |
| ~ (Bitwise NOT) | ~A | 00110 (6) |
| << (Left Shift) | A << 2 | 1100100 (100) |
| >> (Right Shift) | A >> 2 | 00110 (2) |

## Operators & Expressions:

Increment & Decrement Operator:

The increment operators adds one to the existing value of the operand and the decrement operator subtracts one from the existing value of the operand.

The increment and decrement operators are called as unary operators because, both needs only one operand.

Assume X=10;

| **Operator** | **Example** | **Meaning** |
|---|---|---|
| Increment (++) | Y=X++ | X=11 & Y=10 |
| | Y=++X | X=11 & Y=11 |
| Decrement (--) | Y=X-- | X=9 & Y=10 |
| | Y=--X | X=9 & Y=9 |

## Operator Precedance & Associativity:

Precedence:

Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence.

Associativity :

Operator associativity is used to determine the order of operators with equal precedence evaluated in an expression.

Associativity can be either Left to Right or Right to Left.

e.g.

10 + 20 * 30 is calculated as 10 + (20 * 30) and not as (10 + 20) * 30.

## Operator Precedance & Associativity:

Associativity is only used when there are two or more operators of same precedence.

e.g.

The expression “100 / 10 * 10” is treated as “(100 / 10) * 10”.

All operators with same precedence have same associativity.

e.g.

+ and – have same associativity.

Precedence and associativity of postfix ++ and prefix ++ are different.

e.g.

Precedence of postfix ++ is more than prefix ++

Associativity of postfix ++ is left to right and associativity of prefix ++

## Operator Precedance & Associativity:

Comma has the least precedence among all operators and should be used carefully.

e.g.

```
int a;
a = 1, 2, 3; // Evaluated as (a = 1), 2, 3
```

There is no chaining of comparison operators in C.

e.g.

```
int a = 10, b = 20, c = 30;
(c > b > a)          is  not treated as 30 > 20 > 10
```

(c > b > a) is treated as ((c > b) > a),
associativity of '>' is left to right.

Therefore the value becomes ((30 > 20) > 10)
which becomes (1 > 20)

## Operator Precedance & Associativity:

| Category | Operator | Associativity |
|---|---|---|
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)* & sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |

## Expression:

Arithmetic Expression: It is combination of variables, constants, operators arranged as per the syntax of the language.

Syntax:

variable = expression;

Example:

| | | |
|---|---|---|
| a * b – c | => | a X b - c |
| (m+n) * (x+y) | => | (m+n)(x+y) |
| a * b / c | => | ab/c |
| 3*x*x + 2*x +1 | => | $3x^2+2x+1$ |
| x/y + c | => | x/y+c |

## Expression:

Example:

x = a-b/3+c*2-1;

Where, a=9, b=12, c=3, find x.

x = 9 – 12 / 3 + 3 * 2 – 1

x = 9 – 4 + 6 – 1

x = 5 + 6 – 1

x = 11 – 1

x = 10

## Type Conversion:

When constants and variables of different types are mixed in an expression, they are all converted to same type. This procedure is called as Type Conversion.

Compiler converts all operands to the type of largest operand.

This is also called as type promotion.

There are two ways of Types conversion.

1. Implicit: Compiler converts all lower type operand to the higher type before the operation.

2. Explicit: Forcefully converting / casting the type of one operand to other operand to simplify the operation and to get correct result.

## Tye Conversion:

Sequence of rules applied for evaluating an expression:

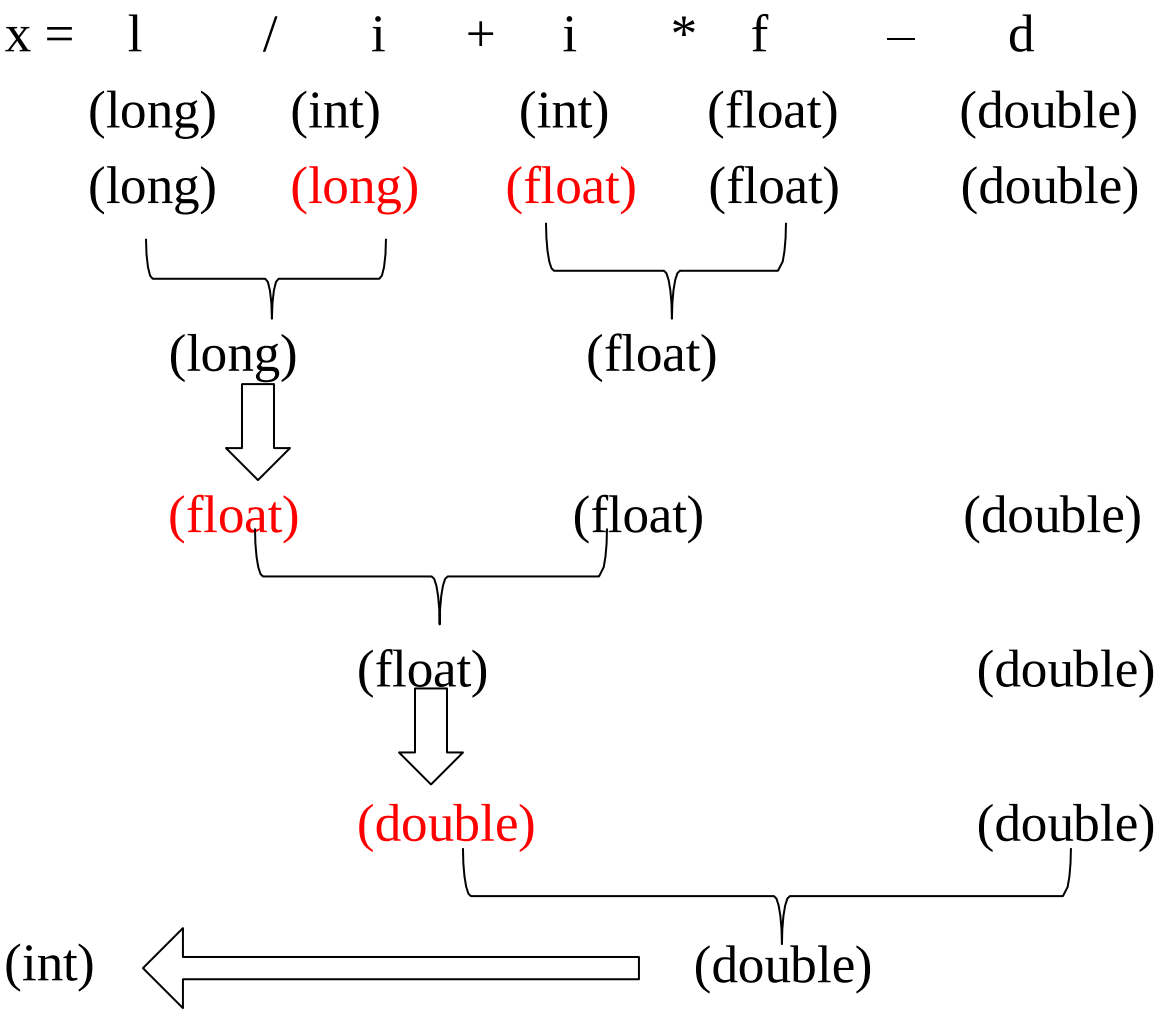
1. All short and char are automatically converted to int.
2. Always Lower type get converted to Higher Type

i.e. Unsigned Integer => Long Integer => Unsigned Long Integer => Float => Double => Long Double

3. If one operand is Unsigned Integer and other is Long Integer then Unsigned Integer will be converted to Long Integer & result in Long Int
4. If float is converted to Integer, It causes truncation of fraction part.
5. If double converted to float, it causes rounding of digits.
6. If long int converted to int. it causes dropping of the excess higher order bits.

## Implicit Tye Conversion:

int i, x; float f, double d; long int l;



## Explicit Tye Conversion:

There are certain instances where, we need to do type conversion forcefully.

i.e. Explicit Type Conversion.

e.g.

```
total = subjet1 + subject2 + subject3 + subject4;
average = total / 3;
```

We can do explicit type casting as follows…

```
average  = (float) total / 3;
```

Syntax:

```
var = (type) expression;
```

## Explicit Tye Conversion:

Example:

| | | |
|---|---|---|
| x = (int)7.6; | => | 7.6 is converted to integer. |
| a=(int)21.3 / (int)4.5; | => | Evaluate 21/4, and result is 5. |
| y = (int) (a+b); | => | Result of a+b is converted to integer. |
| z = (int) a+b; | => | a is converted to integer and added to b. |
| n = cos((double)x); | => | x is converted to double then used. |

## Built in I/O Functions:

For any program 3 operations are important.

1. Input,
2. Process, and
3. Output.

Any programs needs some Input from user, perform some Operation (Process) on it, and produces the Output.

Input: Can be accepted by user from console (Command Line).
from disk file (Reading File).

Process: Performs some operations on accepted Input.

Output: Can be produced to user to console (Command Line).
to disk file (Writing File).

## Built in I/O Functions:

C language provides us two type of I/O operations.

1. Unformatted I/O Operation
2. Formatted I/O Operation

1. Unformatted I/O Operation:

It is simplest form of input / output operation.

We can perform Input and Output operation in the forms of characters & strings.

We have to use several I/O functions, which are defined in stdio.h header file.

We need to include stdio.h header file every time we used these functions.

## Reading a Character:

getchar(): This function is used to read a single character from the console.

Syntax:

var_name = getchar();

where,

var_name is character type variable.

It reads one single character from the console, typed by user and store that character in variable “var_name”.

e.g.

```
char ch;
ch = getchar();
```

## Reading a Character:

getc(): This function is used to read a single character from the console.

Syntax:

var_name = getc(stdin);

where,

var_name is character type variable.

It reads one single character from the console, typed by user and store that character in variable “var_name”.

e.g.

```
char ch;
ch = getc();
```

## Reading a Character:

getch(): It reads single character from keyboard. But it does not use any buffer, so the entered character is immediately returned without waiting for the enter key.

It is non standard function, written in conio.h header file

Syntax:

```
var_name = getch();
```

where,

var_name is character type variable.

It reads one single character from the console, typed by user and store that character in variable “var_name”.

e.g.

```
char ch;
ch = getch();
```

## Difference between getchar(), getch(), and getc():

getc() can read from any input stream (standard / file).

getchar() can reads from standard input.

Similar to getch() one more function is available i.e.

getche(): It also a non-standard function present in conio.h.

It reads a single character from the keyboard and displays immediately on output screen without waiting for enter key.

## Writing a Character:

putchar(): It writes one character at time to the console.

Syntax:

putchar(ch);

Where,

ch is any character type variable.

putc(): It also writes one character at time to the console / standard output device.

Syntax:

putc(ch,stdout);

where,

ch is any character type variable,

stdout is standard output stream.

## Reading a String:

gets(): This function reads string (line of characters) from user including spaces and tabs.

It is defined in stdio.h header file.

The newline character is read and converted to a null character (\O) before it is stored in s.

Syntax:

```
gets(var_name);
```

E.g.

```
char name[10];

gets(name);
```

## Reading a String:

puts(): This function is used to print a string back to console.
It is also defined in stdio.h header file.

Syntax:

```
puts(var_name);
```

E.g.

```
puts(name);
```

## Formatted I/O operations:

Formatted I/O means to perform input and output operations in some specific format.

e.g.

| | |
|---|---|
| 1234 | Integer Format. |
| 45.75 | Floating Format. |
| “Amit” | String Format. |

C provides us two special functions to perform formatted input and output.

1. printf(): It is used for formatted output.

2. scanf(): It is used for formatted input.

## Formatted Input:

scanf() function is used for formatted input operations.

Syntax:

scanf("control string", arg1,arg2, arg3);

Where,

control string specifies the field format in which data is to be entered and stoared.

arg1, arg2, arg3 are the variable where the actual values are stored.

Field format: It consist of

1. conversion character (format specifier) and optional number that specifies field width.
2. blanks, new line character, tabs.

## Formatted Input:

e.g.

1. scanf(%d %d", &num1,&num2);

Where,

%d is format specifier used for Integer type data.

&num1 specifies the address space of the variable, where it stores the value.

2. scanf("%5d %4d", &num1, &num2);

Where,

%5d is format specifier used for Integer type and 5 specifies the filed width used to store the accepted value in the variable.

## Format Specifiers:

| Character | Meaning |
|---|---|
| %d | Reads single decimal value |
| %c | Reads single character value |
| %f | Reads single floating point value |
| %lf | Reads double type of value |
| %l | Reads long value |
| %o | Reads octal representation of integer. |
| %s | Reads a string without bank, new line, tab |
| %u | Reads unsigned integer |
| %i | Reads signed integer |
| %x | Reads hexadecimal representation of integer |

## Formatted Input:

e.g.

```
scanf("%f %f %f", &x, &y, &z);

scanf("%s", &name);

scanf("%d %c %f %s", &rno, &grade, &per, &name);
```

## Formatted Output:

printf(): This function is used to do formatted output operation.

We can use this function to print captions, text message, numerical values as result back to user.

Syntax:

printf("text message");

Where,

text message is any text message that we want to display back to user.

Syntax:

printf("control string",arg1, arg2,arg3);

Where,

control string contains:

characters message,

format specification string,

Escape Seequences characters such as \n, \t, \r etc.

## Formatted Output:

e.g.

```
printf("C – Programming Language");

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

printf("Welcome to \n C programming");

printf("a = %f \n b=%f ", a, b);

printf("\n");
```

## Formatted Output:

e.g.

printf("%5d %7.4f", a, b); //a=9876 & b=98.7654

o/p=> _9876 98.7654

printf("%-7.2f" , a); //a=98.7654 - is used for left Alignment

o/p=> 98.77_ _

printf("%s", city); // city="New Delhi"

o/p=> New Delhi

printf("%20s",city); // by default right Alignment

o/p=> _ _ _ _ _ _ _ _ _ _ _ _ New Delhi

printf("%-20s",city); // - specifies left Alignment

o/p=> New Delhi _ _ _ _ _ _ _ _ _ _ _ _

References:

https://www.javatpoint.com/c-programming-language-tutorial

www.google.com

http://www.btechsmartclass.com/c_programming/C-Datatypes.html

# THANK YOU…

Prof.Ashvini S.Tanpure

HRM College,Rajgrunagar