

# Kick start of C Programs

## STRUCTURE OF THE PROGRAM

```
#include<stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

Header File

Main  
function

Output to be  
displayed

Braces

Return type

# Layout of a C program

Documentation section

---

Link section

---

Definition section

---

Global declaration section

---

main () Function section

{

Declaration part
Executable part

}

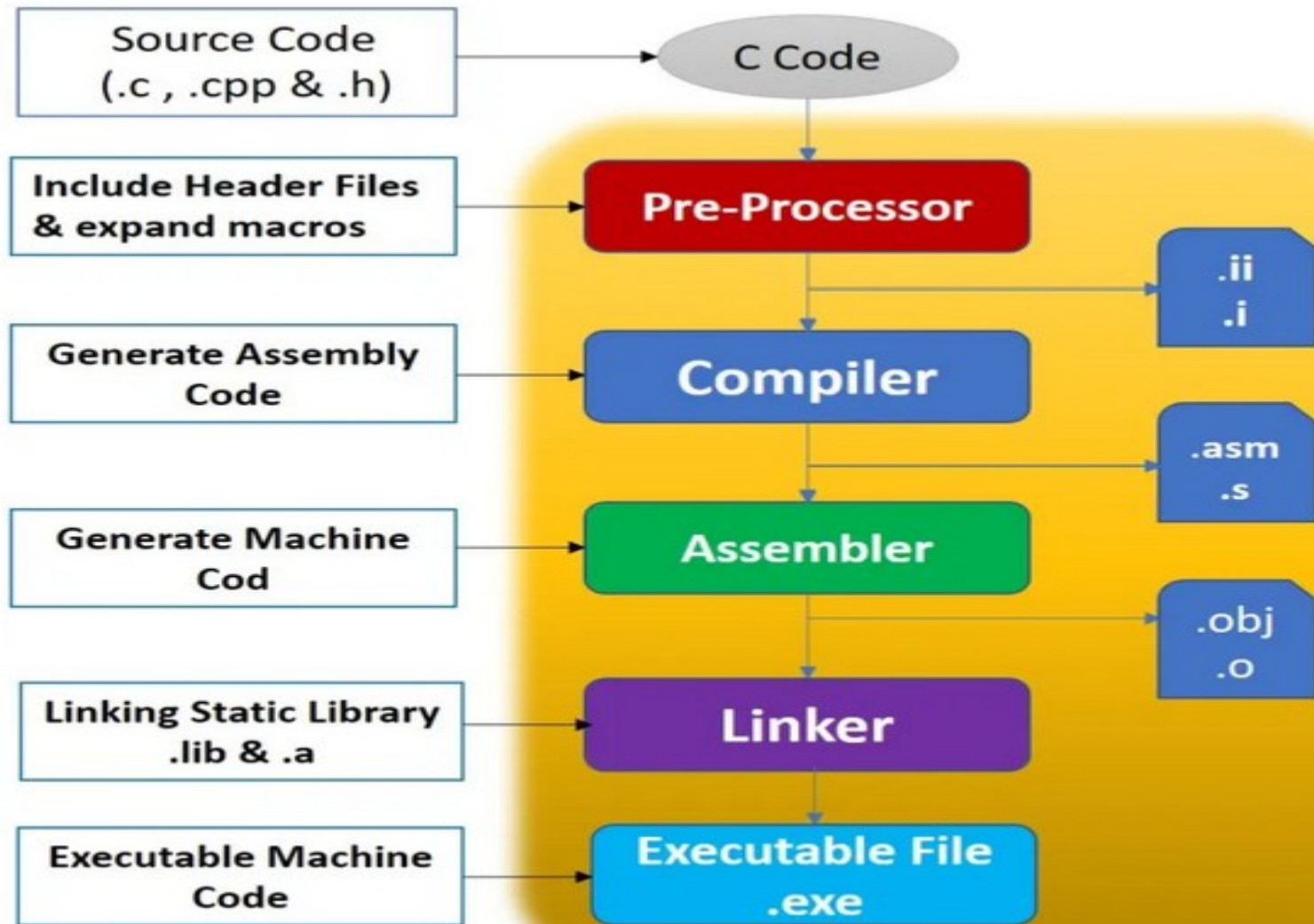
---

Subprogram section

Function 1
Function 2
.....
.....
Function n

(User defined functions)

# C Program Compilation Steps



# Components of a C program

A C program consists of the following parts:

- Comments
- Variables
- Preprocessor Commands
- Functions
- Statements & Expressions

# Comments in C

Two types of comments

Single line and multi line comment

Single Line Comment is double forward slash `'//'` and can be **Placed Anywhere**

# Multiline Comments in C

- Multi line comment can be placed anywhere
- Multi line comment starts with `/*`
- Multi line comment ends with `*/`
- Any symbols written between `/*` and `*/` are ignored by Compiler

# Character set of C

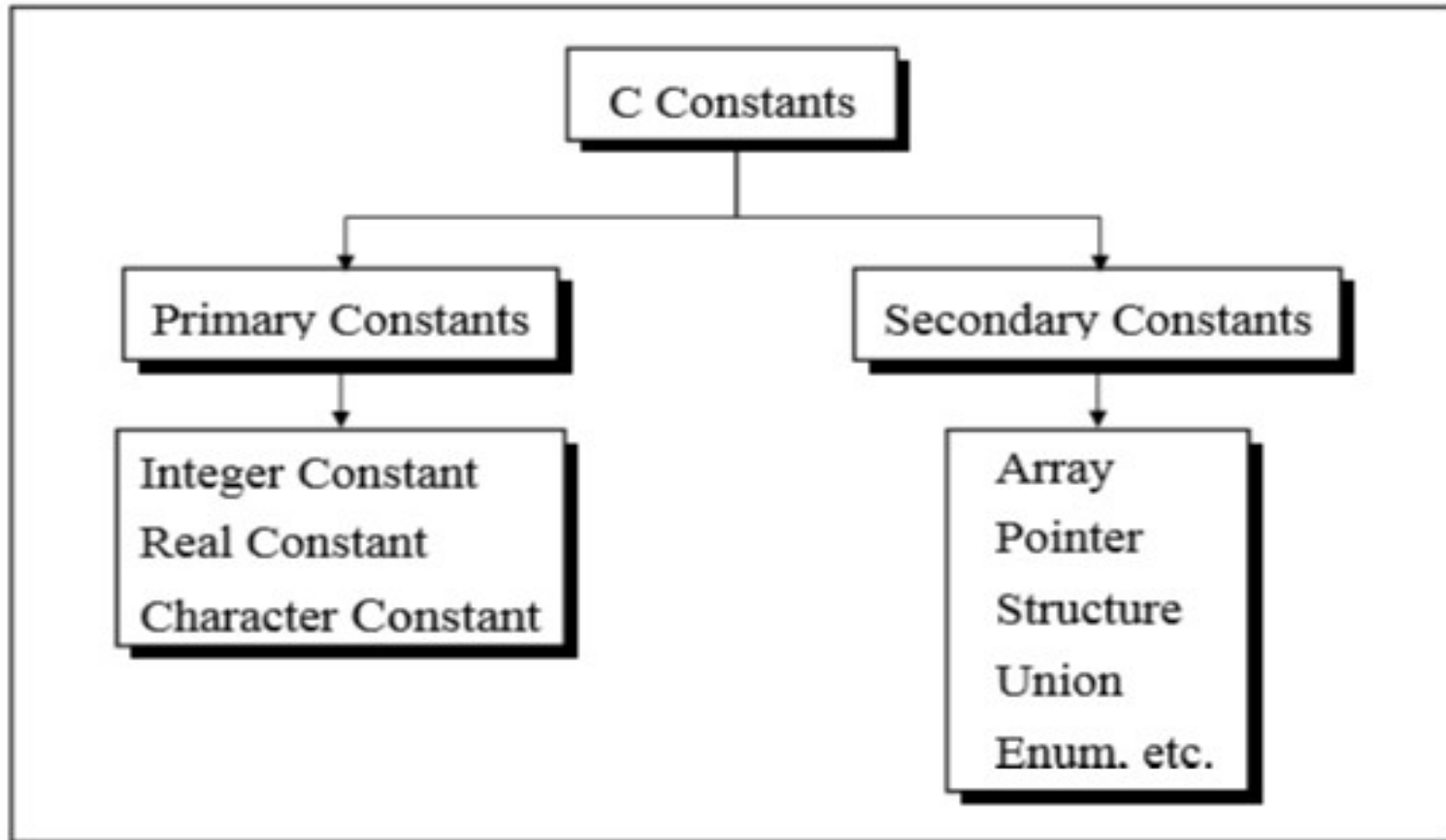
Alphabets	A, B, ....., Y, Z a, b, ....., y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ' ! @ # % ^ & * ( ) _ - + =   \ { } [ ] : ; " ' < > , . ? /



# **Constants, Variables and Keywords**

- alphabets, numbers and special symbols when properly combined form constants, variables and keywords
- Constant - entity that doesn't change
- Variable - entity that may change
- Keyword – reserved in programming language (Equivalent to words in natural language with predefined

# Types of C Constants



**Now Restrict Discussion to Primary Constants**

# Types of C Variables

- Variable names - Names given to locations in memory
- Locations can contain integer, real or character constants
- In any language, types of variables supported depend on the types of constants that it can handle
- Because a particular type of variable can hold only same type of constant

## **Rules for Constructing Variable Names**

- Rules for constructing variable names of all types are same
- Combination of 1 to 31 alphabets, digits or underscores
- Some compilers allow up to 247 characters but safer to 31 characters
- First character must be an alphabet or underscore
- No commas or blanks are allowed

# **Discuss Valid and Invalid variable names**

- **BASICSALARY**
- **\_basic**
- **basic-hra**
- **#MEAN**
- **group.**
- **422**
- **population in 2006**
- **FLOAT**
- **hELLO**

# Data types in C

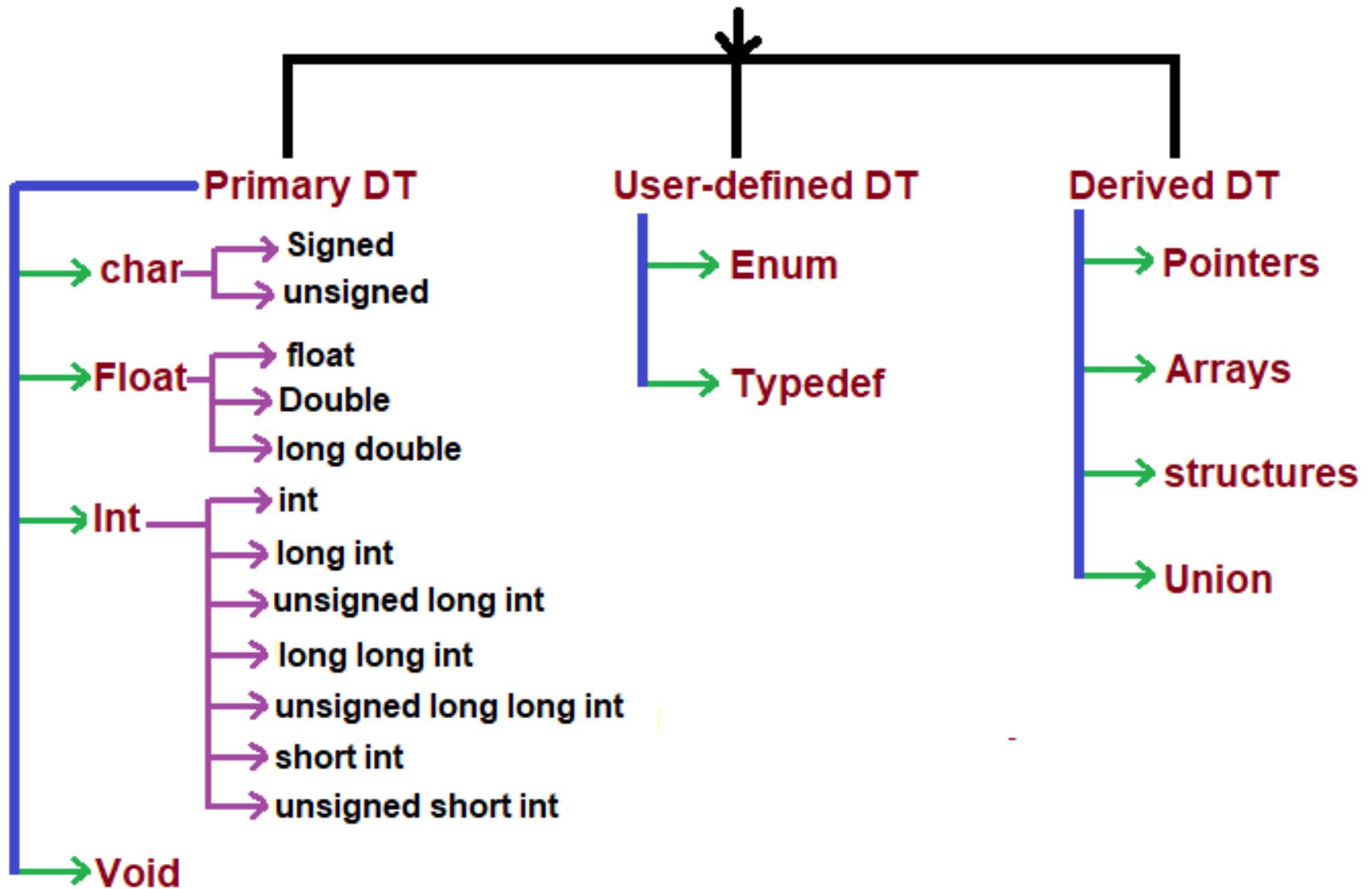
**Basic Arithmetic types** - further classified into: (a) integer types and (b) floating-point types

**Enumerated types** - arithmetic types that are used to define variables that can be assigned only certain discrete integer values throughout the program

**Type void** - indicates that no value is available

**Derived types** - The derived types are (a) Pointers

# Data Types in C



## Character

- ⇒ char
- ⇒ signed char
- ⇒ unsigned char

## Integer

- ⇒ unsigned int
- ⇒ signed char
- ⇒ unsigned char
- ⇒ longint
- ⇒ Signed longint

## Floating Point

- ⇒ Float
- ⇒ Double
- ⇒ Long Double

Data Types	Modifiers	Memory Requirement	
		Bits	Byte
Character	i) Char	8	1
	ii) signed char	8	1
	iii) unsigned char	8	1
Integer	i) int	16	2
	ii) signed int	16	2
	iii) unsigned int	16	2
	iv) short int	16	2
	v) long int	32	4
	vi) signed long int	32	4
	vii) unsigned long int	32	4
Floating point	i) float	32	4(6 digits of precision)
	ii) double	64	8(10 digits of precision)
	iii) long double	128	16(10 digits of precision)



Data type	Size(bytes)	Range	Format string
Char	1	128 to 127	%c
Unsigned char	1	0 to 255	%c
Short or int	2	-32,768 to 32,767	%i or %d
Unsigned int	2	0 to 65535	%u
Long	4	-2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%lu
Float	4	3.4 e-38 to 3.4 e+38	%f or %g
Double	8	1.7 e-308 to 1.7 e+308	%lf
Long Double	10	3.4 e-4932 to 1.1 e+4932	%lf

# Keywords

- 32 keywords available in C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Compiler vendors (like Microsoft, Borland, etc.) provide their own keywords

# I/O in C

- Basic operation in any language
- Input is got through a function scanf which is equivalent to input or raw\_input in Python
- Syntax of scanf
- **int scanf(const char \*format, ...)**
- Basically two or more arguments
- First format string, followed by address of variables that are going to hold values entered by user

# I/O in C

- `int printf(const char *format, ...)`
- contains one or more arguments
- first argument is the format string

# printf and scanf format codes

code	type	format
d	int	decimal (base ten) number
o	int	octal number (no leading '0' supplied in printf)
x or X	int	hexadecimal number (no leading '0x' supplied in printf; accepted if present in scanf) (for printf, 'X' makes it use upper case for the digits ABCDEF)
ld	long	decimal number ('l' can also be applied to any of the above to change the type from 'int' to 'long')

# printf and scanf format codes

code	type	format
u	Unsigned int	decimal number
lu	unsigned long	decimal number
c	char [footnote]	single character
s	char pointer	string
f	float [footnote]	number with six digits of precision
lf	double [footnote]	number with six digits of precision

# **Address of a variable**

- Address of a variable can be obtained by putting a '&' before the variable name

# Arithmetic instructions in C

- To perform arithmetic operations between constants and variables
- Operands shall be either constant or variables
- Variables can be of any type of integer or floating point value or character except for modulus operator
- Modulus operator cannot be applied for floating point values but can be applied for



# Example 1

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
int a = 27;
```

```
int b = 25;
```

```
int c = a - b;
```

```
printf("%d",c);
```

```
}
```

# Output 1

2

# Example 2

```
#include<stdio.h>
void main()
{
char a = 273;
char b = 25;
int c = a%b;
printf("%d",c);

}
```

# Output 2

warning: overflow in implicit constant conversion [-Woverflow]

17

Character – range is 0 to 255

256 is 0

257 is 1 and so on

# Example 3

```
#include<stdio.h>
void main()
{
char a = 27;
char b = 25;
char c = a -b;
printf("%c",c);

}
```

# Output 3

A special character

# Arithmetic Operators in C

Operator	Description
+	Adds two operands
−	Subtracts second operand from the first.
*	Multiplies both operands
/	Divides numerator by denominator
%	Modulus Operator and remainder of after an integer division.
++	Increment operator increases the integer value by one
--	Decrement operator decreases the integer value by one

# Assignment Operators in C

OPERATOR	DESCRIPTION	SYNTAX
=	Assign value of right side of expression to left side operand	$x = y + z$
+=	Add AND: Add right side operand with left side operand and then assign to left operand	$a += b$ $a = a + b$
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	$a -= b$ $a = a - b$
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	$a *= b$ $a = a * b$
/=	Divide AND: Divide left operand with right operand and then assign to left operand	$a /= b$ $a = a / b$
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	$a \% = b$



# Relational Operators in C

OPERATOR	DESCRIPTION	SYNTAX
>	Greater than: True if left operand is greater than the right	<code>x &gt; y</code>
<	Less than: True if left operand is less than the right	<code>x &lt; y</code>
==	Equal to: True if both operands are equal	<code>x == y</code>
!=	Not equal to - True if operands are not equal	<code>x != y</code>
>=	Greater than or equal to: True if left operand is greater than or equal to the right	<code>x &gt;= y</code>
<=	Less than or equal to: True if left operand is less than or equal to the right	<code>x &lt;= y</code>

# Logical Operators in C

Logical Operators		
Operator	Description	Example
<b>&amp;&amp;</b>	<b>AND</b>	<b>x=6 y=3 x&lt;10 &amp;&amp; y&gt;1 Return True</b>
<b>  </b>	<b>OR</b>	<b>x=6 y=3 x==5    y==5 Return False</b>
<b>!</b>	<b>NOT</b>	<b>x=6 y=3 !(x==y) Return True</b>

# Bitwise Operators in C

OPERATOR	DESCRIPTION	SYNTAX
&	Bitwise AND	<code>x &amp; y</code>
	Bitwise OR	<code>x   y</code>
~	Bitwise NOT	<code>~x</code>
^	Bitwise XOR	<code>x ^ y</code>
>>	Bitwise right shift	<code>x &gt;&gt;</code>
<<	Bitwise left shift	<code>x &lt;&lt;</code>

# Bitwise Operators in C

A = 10 => 1010 (Binary)  
B = 7 => 111 (Binary)

A & B = 1010  
    &  
    0111  
= 0010  
= 2 (Decimal)

**Bitwise AND Operator**

A = 10 => 1010 (Binary)  
B = 7 => 111 (Binary)

A | B = 1010  
    |  
    0111  
= 1111  
= 15 (Decimal)

**Bitwise OR Operator**

A = 10 => 1010 (Binary)  
B = 7 => 111 (Binary)

A ^ B = 1010  
    ^  
    0111  
= 1101  
= 13 (Decimal)

**Bitwise XOR Operator**

A = 10 => 1010 (Binary)

~A = ~1010  
    = -(1010+1)  
    = -(1011)  
    = -11 (Decimal)

**Bitwise Ones' Complement Operator**

A = 10 => 1010 (Binary)

A << 2 = 1010 << 2  
        = 101000  
        = 40 (Decimal)

**Bitwise Left Shift Operator**

A = 10 => 1010 (Binary)

A >> 2 = 1010 >> 2  
        = 10  
        = 2 (Decimal)

**Bitwise Right Shift Operator**

# Precedence of Operators in C

++, -- Post increment Operators

++, -- Pre increment Operators

Operators	Description
* / %	multiplication, division, modular division
+ -	addition, subtraction
=	assignment

Parenthesis can be used to override default precedence

# Example 4

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;
b = 2;

c = -a+--b;

printf ( "c = %d", c) ;
}
```

# Output 4

$$c = -3$$

# Example 5

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;
b = 2;

c = -a+ b--;

printf ( "c = %d", c) ;
printf ( "b = %d", b) ;
}
```



# Output 5

c = -2

b = 1

# Example 6

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;

c = ++a + a++;

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

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

}
```

# Output 6

c = 10 a = 6

# Example 7

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;

c = ++a + ++a;

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

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

}
```

# Output 7

c = 12 a = 6

# Example 8

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;

c = a++ + ++a;

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

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

}
```

# Output 8

c = 10 a = 6

# **Associativity of Operators in C**

When precedence of two operators are same then associativity of operator is considered for evaluation



OPERATOR	DESCRIPTION	ASSOCIATIVITY
( )	Parentheses (function call) (see Note 1)	left-to-right
[ ]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ —	Postfix increment/decrement (see Note 2)	
++ —	Prefix increment/decrement	right-to-left
+ —	Unary plus/minus	
! ~	Logical negation/bitwise complement	
( <i>type</i> )	Cast (convert value to temporary value of <i>type</i> )	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ —	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right

	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^=  =	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

# Automatic Type Conversion in C

Convert a variable from one data type to another data type

When the type conversion is performed automatically by the compiler without programmers intervention, such type of conversion is known as **implicit type conversion** or **type promotion**.

The compiler converts all operands into the data type of the largest operand.

# Rules for Implicit Type Conversion in C

- Sequence of rules that are applied while evaluating expressions are given below:
- All short and char are automatically converted to int, then,
- If either of the operand is of type long double, then others will be converted to long double and result will be long double.
- Else, if either of the operand is double, then others are converted to double.
- Else, if either of the operand is float,

# Rules for Type Conversion in C

- Else, if either of the operand is unsigned long int, then others will be converted to unsigned long int.
- Else, if one of the operand is long int, and the other is unsigned int, then
  - if a long int can represent all values of an unsigned int, the unsigned int is converted to long int.
  - otherwise, both operands are converted to unsigned long int.
- Else, if either operand is long int then other will be converted to long int.
- Else, if either operand is unsigned int then others will be converted to unsigned int.

# Example 9

```
#include<stdio.h>
void main()
{
char a = 65;
char b = 100;
int c = a%b;
printf("%c",c);

}
```

# Output 9

A

# Example 10

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
int c = a/b;
printf("%d",c);

}
```



# Output 10

1

# Example 11

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
float c = a/b;
printf("%f",c);

}
```

# Output 11

1.000000

# Example 12

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
float c = a/b;
printf("%f",c);

}
```

# Output 12

1.000000

# Example 13

```
#include<stdio.h>
void main()
{
int a = 165;
float b = 100;
float c = a/b;
printf("%f",c);

}
```

# Output 13

1.650000

# Explicit Type Conversion

Type conversion performed by the programmer is known as explicit type conversion

Explicit type conversion is also known as **type casting**.

Type casting in c is done in the following form:  
**(data\_type)expression;**

where, *data\_type* is any valid c data type,  
and *expression* may be constant, variable or



# Explicit Type Conversion

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

- All integer types to be converted to float.
- All float types to be converted to double.
- All character types to be converted to integer.

# Example 14

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
float c = (float)(a/b);
printf("%f",c);

}
```

# Output 14

1.000000

# Example 15

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
float c = (float) a/b;
printf("%f",c);

}
```

# Output 15

1.650000

# Scope and Lifetime of a Variable in C Language

**Life Time** of a variable - Time for which the particular variable outlives in memory during execution of program

**Scope of a variable** - Area of our program where we can actually access the variable

# Global vs Local Scope

## **Global scope :**

When variable is defined outside all functions. It is then available to all the functions of the program and all the blocks program contains.

## **Local scope :**

When variable is defined inside a function or a block, then it is locally accessible within the block and hence it is a local variable.

# Global vs Local Scope

```
int global = 100;           // global variable declared

void main()
{
    int local = 10;         // local variable declared
    printf("Global variable is %d",global);
    printf("Local variable is %d",local);
    func1();
}

void func1()
{
    printf("Global inside func1 is %d",global);
}
```



# Storage Classes in C

Four Storage Classes, classified based on life and scope of access:

Storage Class	Life	Scope
Auto	Created when a function is called and destroyed automatically when function exits Default – <code>int n;</code>	Can be accessed only within the function where it is declared
Register	Similar to Auto but stored in register, a faster memory. Number of register is limited. Therefore automatically converted to auto if	Similar to Auto

# Storage Classes in C

Storage Class	Life	Scope
Static	<ul style="list-style-type: none"><li>• Created when a function is called</li><li>• Mandatory for programmer to initialize</li><li>• Lives till the program executes</li></ul> <code>static int n =10;</code>	Accessible only inside the function where it is defined
Extern	Created when the program execution starts  <code>extern int n;</code>	Global variables that can be shared across files

# Static in C

```
void test();    //Function declaration (discussed in next topic)
```

```
main()
```

```
{
```

```
    test();
```

```
    test();
```

```
    test();
```

```
}
```

```
void test()
```

```
{
```

```
    static int a = 0;    //Static variable
```

```
    a = a+1;
```

```
    printf("%d\t",a);
```

```
}
```

output :

1

2

3


# Extern in C

file1.c

```
#include<stdio.h>
int a = 7 ; // global variable
void fun()
{
    a++ ;
    printf("%d", a) ;
    .....
    .....
}
```

file2.c

```
#include "file1.c" ;
main()
{
    extern int a ;
    fun() ;
}
```



global variable from one file can be used in other using **extern** keyword.

# Extern in C

## Example Using extern in same file

```
main()
{
    extern int x; //Tells compiler that it is defined somewhere else
    x = 10;
    printf("%d",x);
}

int x; //Global variable x
```

# **Declaration and Definition of Variables in C**

- Declaration – Information to compiler about the variable
- Definition – Memory is allocated for the variable according to type
- For all storage classes declaration and definition are same except for extern
- When a keyword extern precedes, its only a declaration and the compiler waits for definition without raising error

# Activity-1

- Little Bob loves chocolate, and he goes to a store with Rs.  $N$  in his pocket. The price of each chocolate is Rs.  $C$ . The store offers a discount: for every  $M$  wrappers he gives to the store, he gets one chocolate for free. This offer is available only once. How many chocolates does Bob get to eat?

```
#include <stdio.h>
void main()
{
    float n,c;
    int p,m,f,tot;
    printf("Enter amount in hand, price of chocolate and number of free wrappers");
    scanf("%f%f%d",&n,&c,&m);
    //compute number of chocolates bought
    p = (int)(n/c);
    //free chocolates
    f = (int)(p/m);
    tot = p+f;
    printf("Number of chocolates bought %d\n",tot);
}
```



# Activity-2

ABC company Ltd. is interested to computerize the pay calculation of their employee in the form of Basic Pay, Dearness Allowance (DA) and House Rent Allowance (HRA). DA and HRA are calculated as certain % of Basic pay(For example, DA is 80% of Basic Pay, and HRA is 30% of Basic pay). They have the deduction in the salary as PF which is 12% of Basic pay. Propose a computerized solution for the above said problem.

Input : Basic Pay

Process : Calculate Salary

( Basic Pay + ( Basic Pay \* 0.8) + ( Basic Pay \* 0.3 - ( Basic Pay \* 0.12))  
-----allowances ----- --- deductions----

Output : Salary

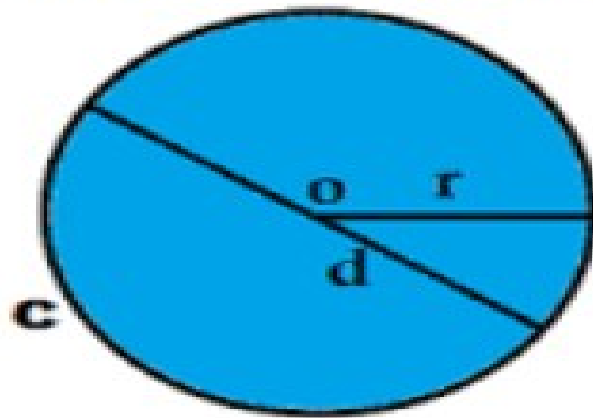
# Activity 3

- Find the average runs scored by a batsman in 4 matches (restricted to 2 decimal places)

# Activity 4

- Write a C program to find the Area, Diameter & Circumference of a circle

*Circle & Formulas*



$$\text{Area} = \pi r^2$$

$$\text{Diameter} = 2r$$

$$\text{Circumference} = 2\pi r$$

# Activity 5

An university is setting up a new lab at their premises. write a C code to determine the approximate cost to be spent for setting up the lab. Cost for setting the lab is sum of cost of computers, cost of furniture's and labour cost.

Use the following formulae for solving the problem:

Cost of computer = cost of one computer \* number of computers

Cost of furniture = Number of tables \* cost of one table + number of chairs \* cost of one chair

Labour cost = number of hours worked \* wages per hour