Basics about C program

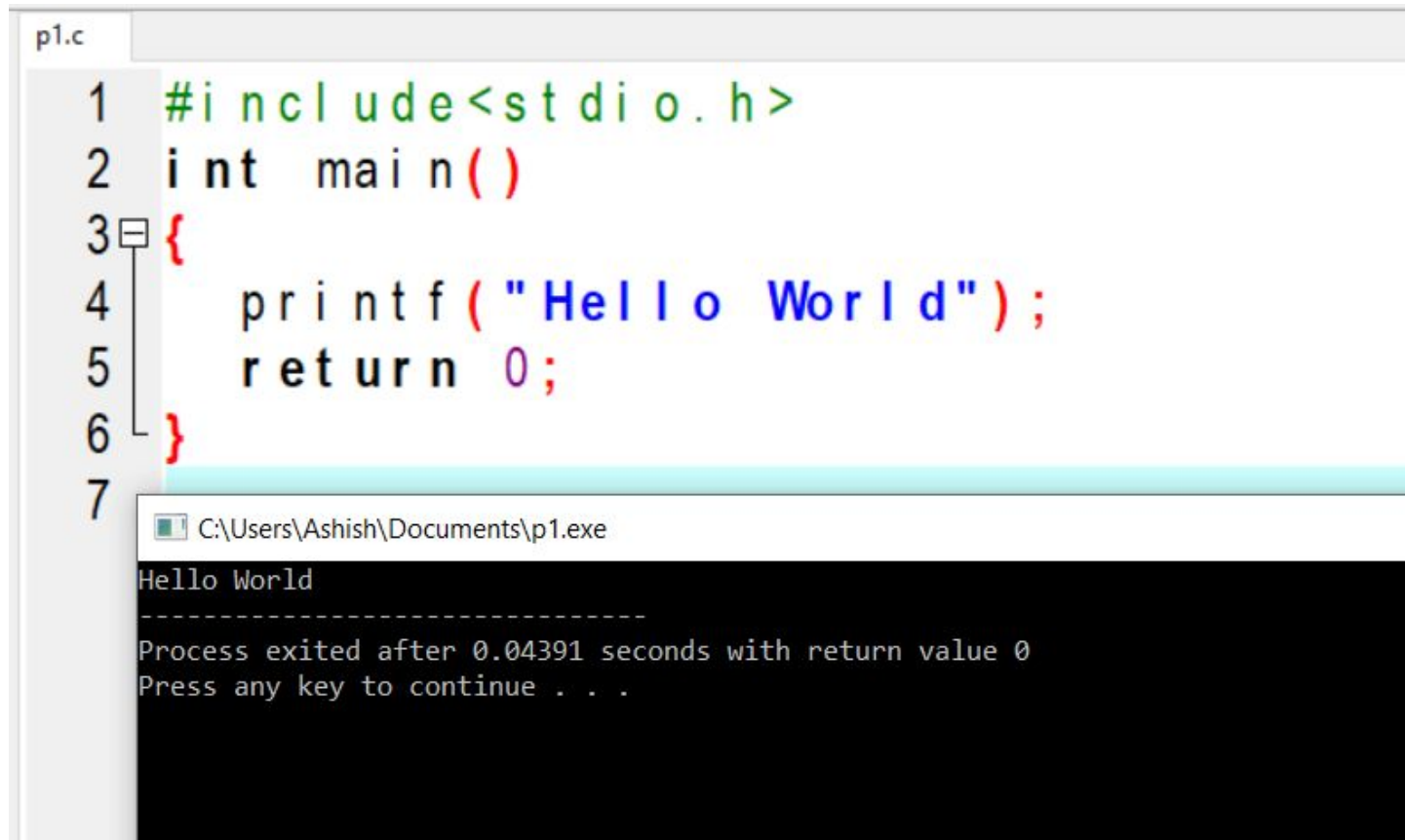**Topics covered**

-Structure of C Program

-Life Cycle of Program from Source code to Executable

-Compiling and Executing C Code

-Keywords

-Identifiers

-Primitive Data types in C

-Variables

-Constants

-Input/Output statements in C

-Operators

-Type conversion and type casting

What is programming?

Computer programming language expresses a set of detailed instructions for a computer.

Editor and Console

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

C:\Users\Ashish\Documents\p1.exe

```
Hello World
---------------------------------
Process exited after 0.04391 seconds with return value 0
Press any key to continue . . .
```

Compiler to run the program

https://www.onlinegdb.com/online_c_compiler

DevC++ (Find some other compiler for MAC)

CppDroid app for **Android phones**

Structure of a C program

A C program is divided into different sections. There are six main sections to a basic c program :

1. Documentation/ Comments Section

2. Preprocessor Directives/ Link Section

3. Definition Section

4. Global Declaration Section

5. Main Function

6. Sub-program Section

BASIC STRUCTURE OF A 'C' PROGRAM:

Example:

Documentation section [Used for Comments]	//Sample Prog Created by:Bsource
Link Section - **Header files, preprocessing definition**	#include<stdio.h> #include<conio.h>
Definition Section	**#define PI 3.14**
Global declaration section [Variable used in more than one function]	int a=10;

main()
{
Declaration part
Executable part
}
	void main()
	{
	printf("a value inside main(): %d",a);
	fun();
	}

Subprogram section
[User-defined Function]
Function1
Function 2
:
Function n
	void fun()
	{
	printf("\na value inside fun(): %d",a);
	}

1. Document/ Comments Section

- The documentation section is the part of the program where the programmer gives the details associated with the program.

- He usually gives- Name of the program,
- The Author And Other Details, which the programmer would like to use later.

- Comment means explanations or annotations that are included in a program for documentation and clarification purpose.

- Comments are completely ignored by the compiler during compilation and have no effect on program execution.

- This section is optional

1. Documentation/ Comments Section (cont..)

- Multiline Comments starts with '/*' and ends with '*/'
- Single line comments starts with '//'

Example:
/* Comments Section Structure of C Program Author: XXXX
Date : 28/07/20
*/
// My first C Program

2. Link Section

- This part of the code is used to declare all the header files that will be used in the program.
- This leads to the compiler being told to link the header files to the system libraries.
- **Example:**
- **#include<stdio.h>**
- These are also called preprocessor directives.
- Preprocessor Directives are always preceded with '#' sign .
- These are the first statement to be checked by the compiler.
- There are many preprocessor directives but most important is #include <stdio.h>

2. Link Section (cont…)

#include<stdio.h>

- Tells the compiler to include the file stdio.h during compilation

- 'stdio.h' stands for standard input output header file and contains function prototype for input output operations e.g. printf() for output and scanf() for input, etc.

2. Link Section (cont...)

Other popular header files:

#include<string. h> (String header)

#include<stdlib. h> (Standard library

header) #include<math. h> (Math header)

3. Definition Section

- In this section, we define different constants. The keyword define is used in this part.

- #define PI 3.14

4. Global Declaration Section

- The global variables that can be used anywhere in the program are declared in global declaration section.

- This section also declares the user defined functions.

- float area(float r);

- int a=7;

5. Main Function

- It is necessary to have one main() function section in every C program.
- This section contains two parts, declaration and executable part.
- The declaration part declares all the variables that are used in executable part.
- These two parts must be written in between the opening and closing braces.
- Each statement in the declaration and executable part must end with a semicolon (;).
- The execution of program starts at opening braces and ends at closing braces.

Example of a C Program

```c
/* My first C program to print Hello, World! */  ◄━━━━Comment

#include <stdio.h>  ◄━━━━Pre-processor directive

int main()  ◄━━━━Main function
{
    printf("Hello, World!");  ◄━━━━Function
    
    return 0;
}
```

Body of main function

6. Sub-program Section

- All the user-defined functions are defined in this section of the program.

- User can define their own functions in this section which performs particular task as per the user requirement. So user creates this according to their needs.
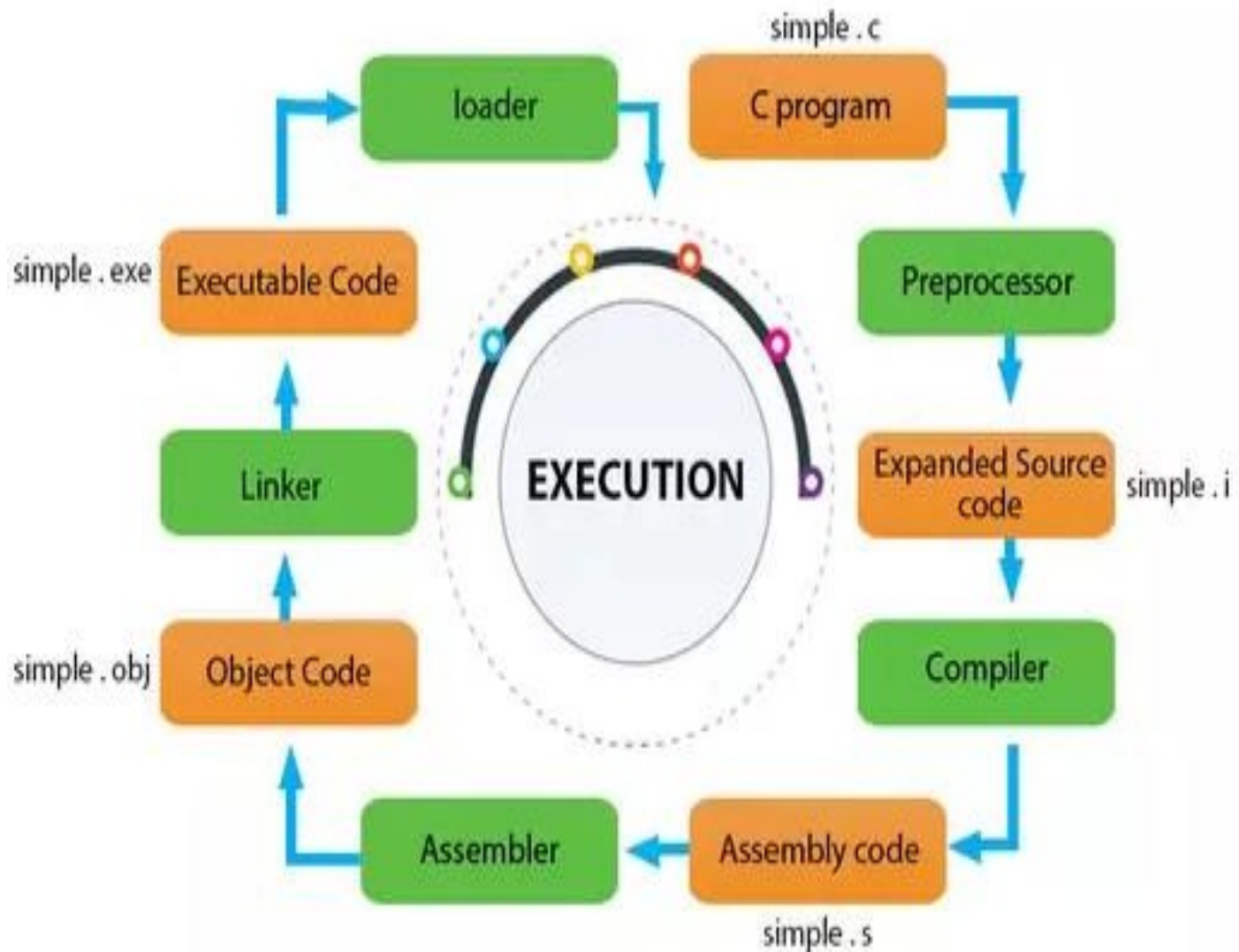
Example:

```
int add(int a, int b)
{
return a+b;
}
```

Life Cycle of a C Program

```
/* My first C program to print Hello, World! */ ◄——————Comment

#include <stdio.h> ◄——————Pre-processor directive

int main() ◄——————Main function
{
        printf("Hello, World!"); ◄——————Function          ⎤
                                                            ⎬——Body of main function
        return 0;                                          ⎦

}
```

Let us suppose, we write program above Notepad and save the file as in simple.c

Life Cycle of a C

Preprocessor

Compiler

Assembler

Linker

loader

```
/* this is demo */
#include<stdio.h>
void main()
{
printf ("hello");
}
```

Preprocessor

Expanded Source code   simple . i

```
code of stdio.h file
void main()
{
printf ("hello");
}
```

Preprocessor remove comments and include header files in source code, replace macro name with code.

Preprocessor

Compiler

Assembler

Linker

loader

Expanded Source code simple . i

code of stdio.h file
void main()
{
printf ("hello");
}

Compiler

Assembly code

simple . s

```
push    ebp
mov     ebp, esp
and     esp, -16
sub     esp, 16
mov     eax, OFFSET
FLAT:.LC0
mov     DWORD PTR
[esp], eax
call    printf
leave
ret
```

Compiler generate assebmly code.

Assembler

Preprocessor

Compiler

Assembler

Linker

loader

Assembly code

simple . s

```
push    ebp
mov     ebp, esp
and     esp, -16
sub     esp, 16
mov     eax, OFFSET
FLAT:.LC0
mov     DWORD PTR
[esp], eax
call    printf
leave
ret
```

Assembler

Object Code

```
00101001100101001
10101111010100101
01010010101011010
01010101010010101
01001100111111100
10001010010101001
01011111011111111
11111101000000011
10010010
```

simple . obj
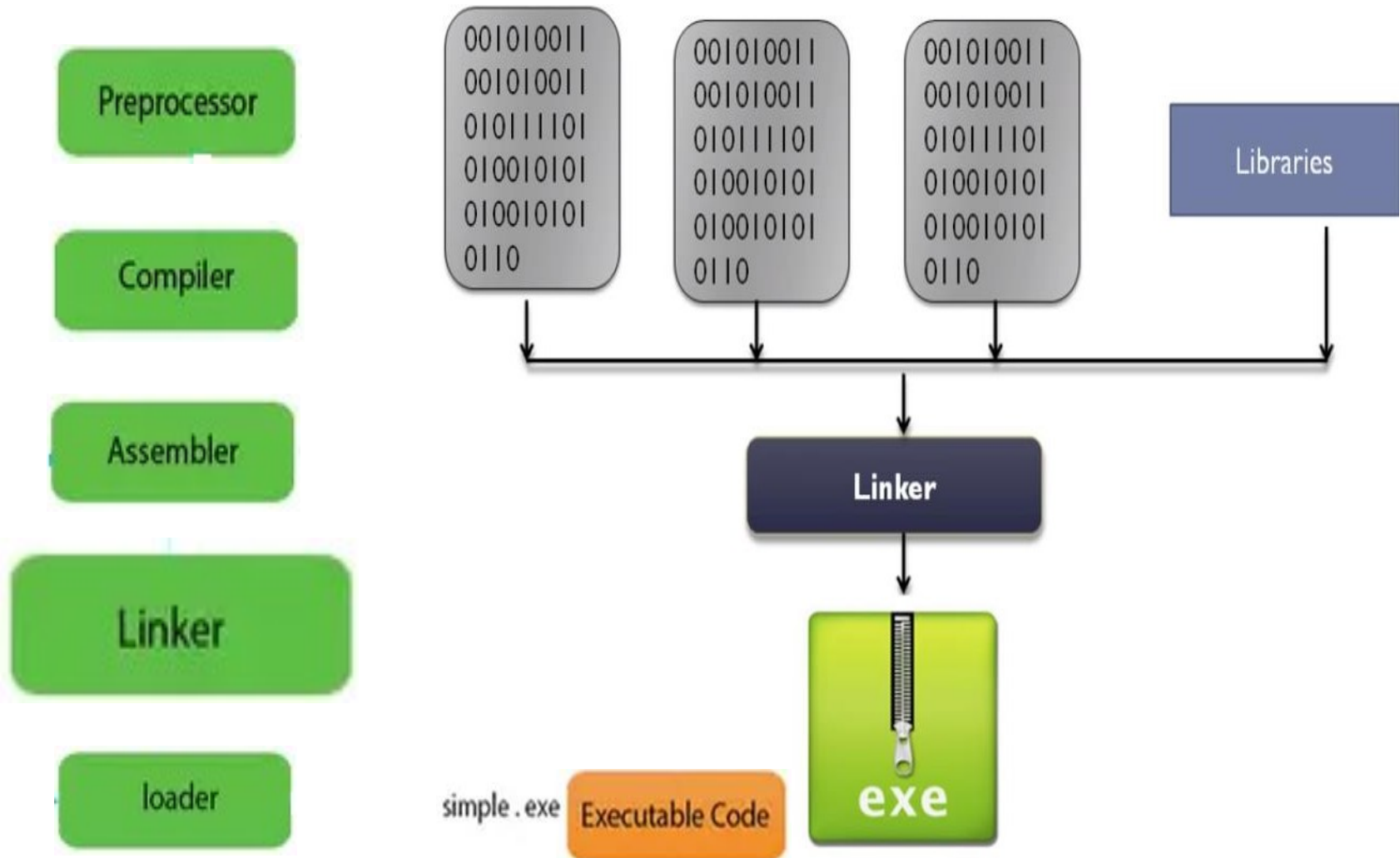
Assebler convert assemble code into object code.
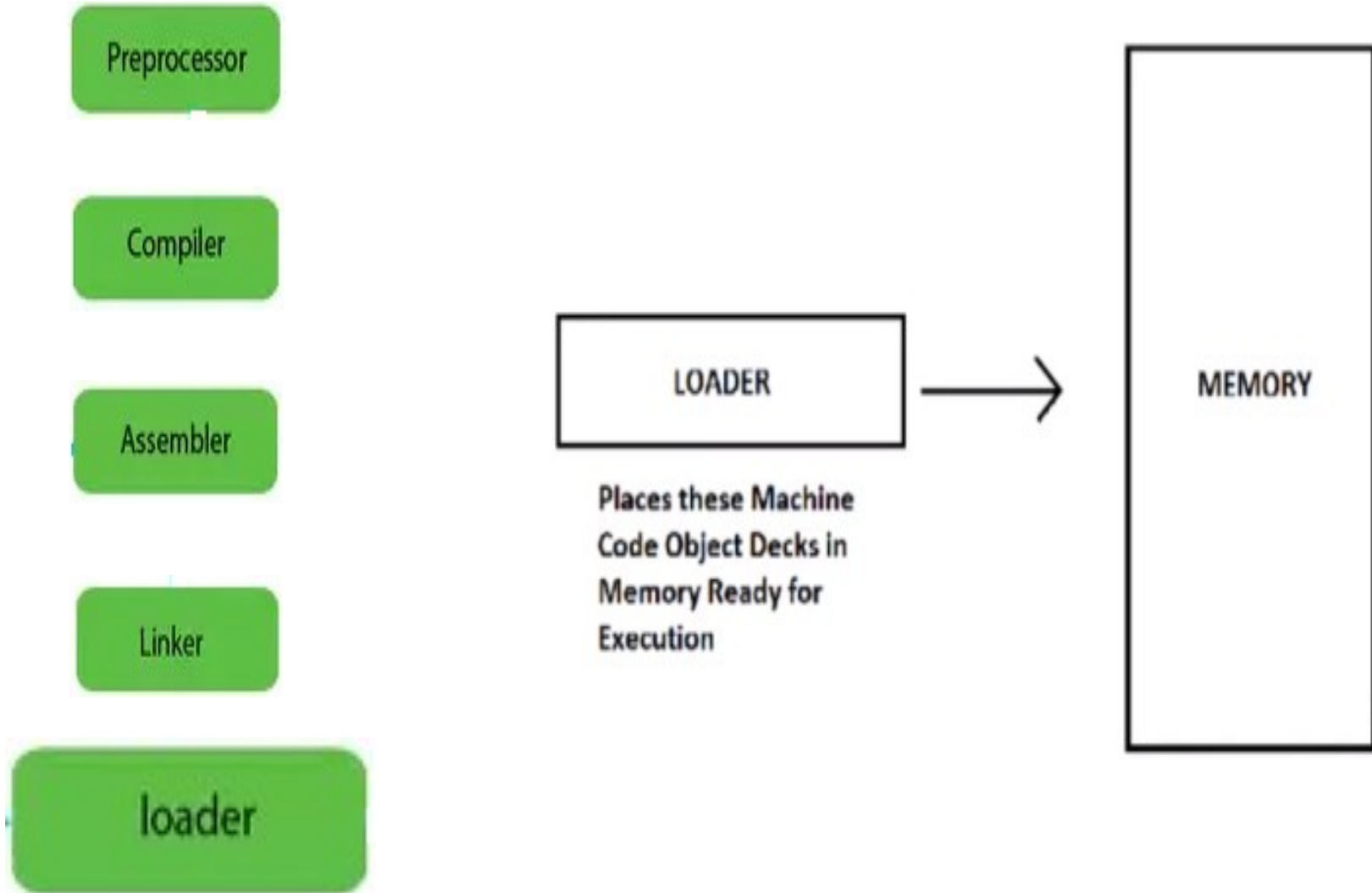
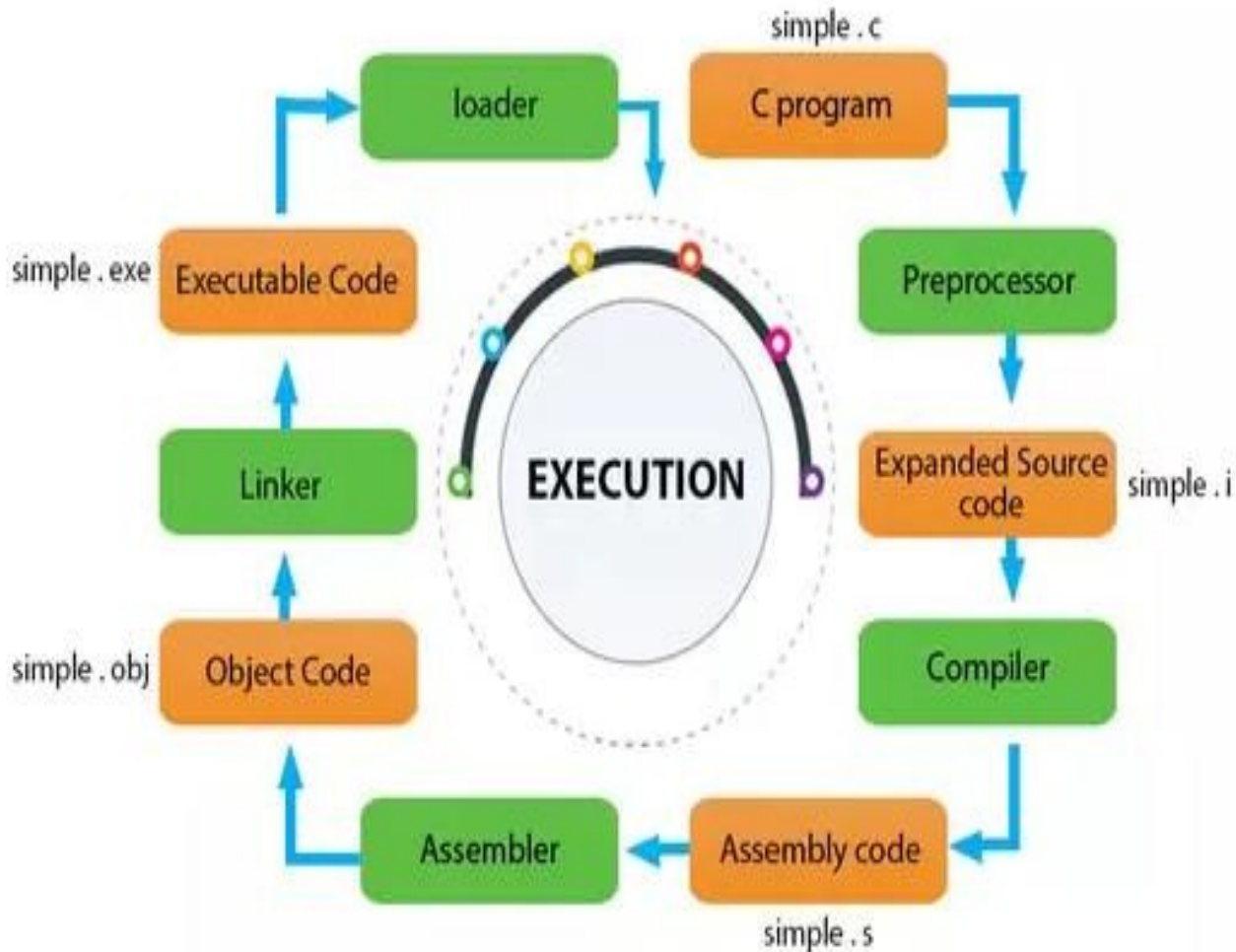# Linke

Preprocessor

Compiler

Assembler

Linker

loader

001010011
001010011
010111101
010010101
010010101
0110

001010011
001010011
010111101
010010101
010010101
0110

001010011
001010011
010111101
010010101
010010101
0110

Libraries

Linker

simple . exe Executable Code

exe

Loader

Preprocessor

Compiler

Assembler

Linker

loader

LOADER → MEMORY

Places these Machine Code Object Decks in Memory Ready for Execution

Life Cycle of a C

C Character Set

Letters	Digits		Escape Sequences
Capital A to Z	All decimal digits 0 to 9		Blank space
Small a to z			Horizontal tab
			Vertical tab
			New line
			Form feed

C Character Set (Special Characters)

,	Comma	&	Ampersand
.	dot	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus
'	Apostrophe	+	Plus
"	Quotation mark	<	Less than
!	Exclamation mark	>	Greater than
|	Vertical bar	()	Parenthesis left/right
/	Slash	[ ]	Bracket left/right
\	Back slash	{}	Braces left/right
~	Tilde	%	Percent
_	Underscore	#	Number sign or Hash
$	Dollar	=	Equal to
?	Question mark	@	At the rate

C Character Set (Special Characters)

- Each character is assigned a unique numeric value.

- These values put together are called character codes.

Delimiters

Delimiters	Use
: Colon	Useful for label
; Semicolon	Terminates the statement
( ) Parenthesis	Used in expression and function
[ ] Square Bracket	Used for array declaration
{ } Curly Brace	Scope of the statement
# hash	Preprocessor directive
, Comma	Variable separator

Keywords or reserved words

The following list shows the reserved words in C. You cannot use them for variable names

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

Identifiers in C

C identifier is a name used to identify a variable, function, or any other user-defined item.

An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).

C is a **case-sensitive** programming language. Thus, *Manpower* and *manpower* are two different identifiers in C.

Rules for forming Identifier Names

- It cannot include any special characters or punctuation marks (like #, $, ^, ?, . , etc.) except underscore ' _ '

- There cannot be two successive underscores

- Keywords cannot be used as identifiers

- Case of alphabetic characters is significant

Rules for forming Identifier Names

- It can be of any reasonable length

- It can be mainly upto 63 characters long

- Underscore can be used to separate parts of the name in an identifier

Data Types

Data types in C refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

C supports a number of qualifiers that can be applied to the basic data types e.g. short, long, unsigned and signed etc.

Data Types
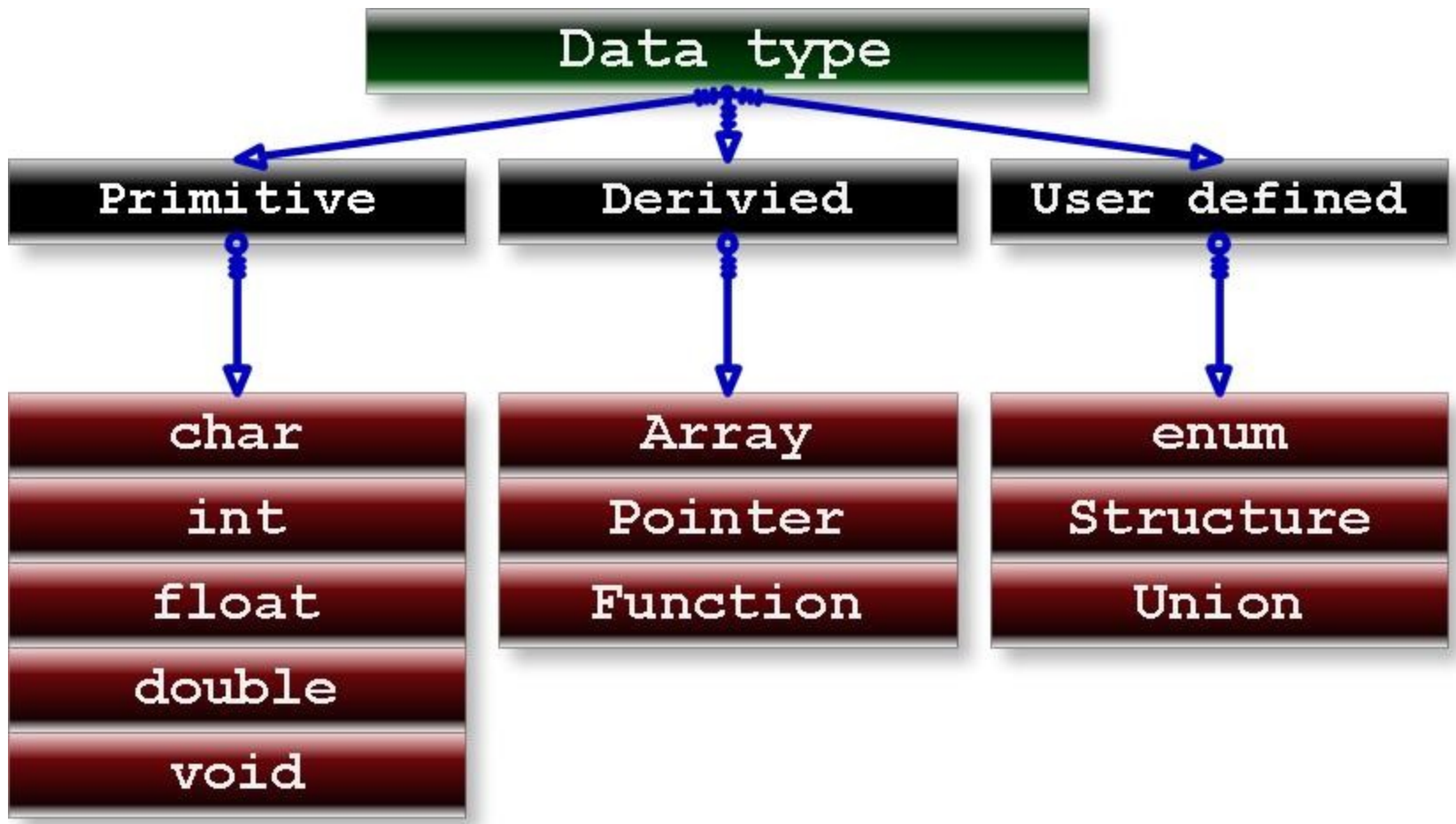
Primitive Data types in C

Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.

Primitive Data Types

Primary Data type

Character
- char
- Signed char
- Unsigned char

Integer
signed
- int
- short int
- long int
unsigned
- int
- short int
- long int

Float
- float
- double
- long double

void
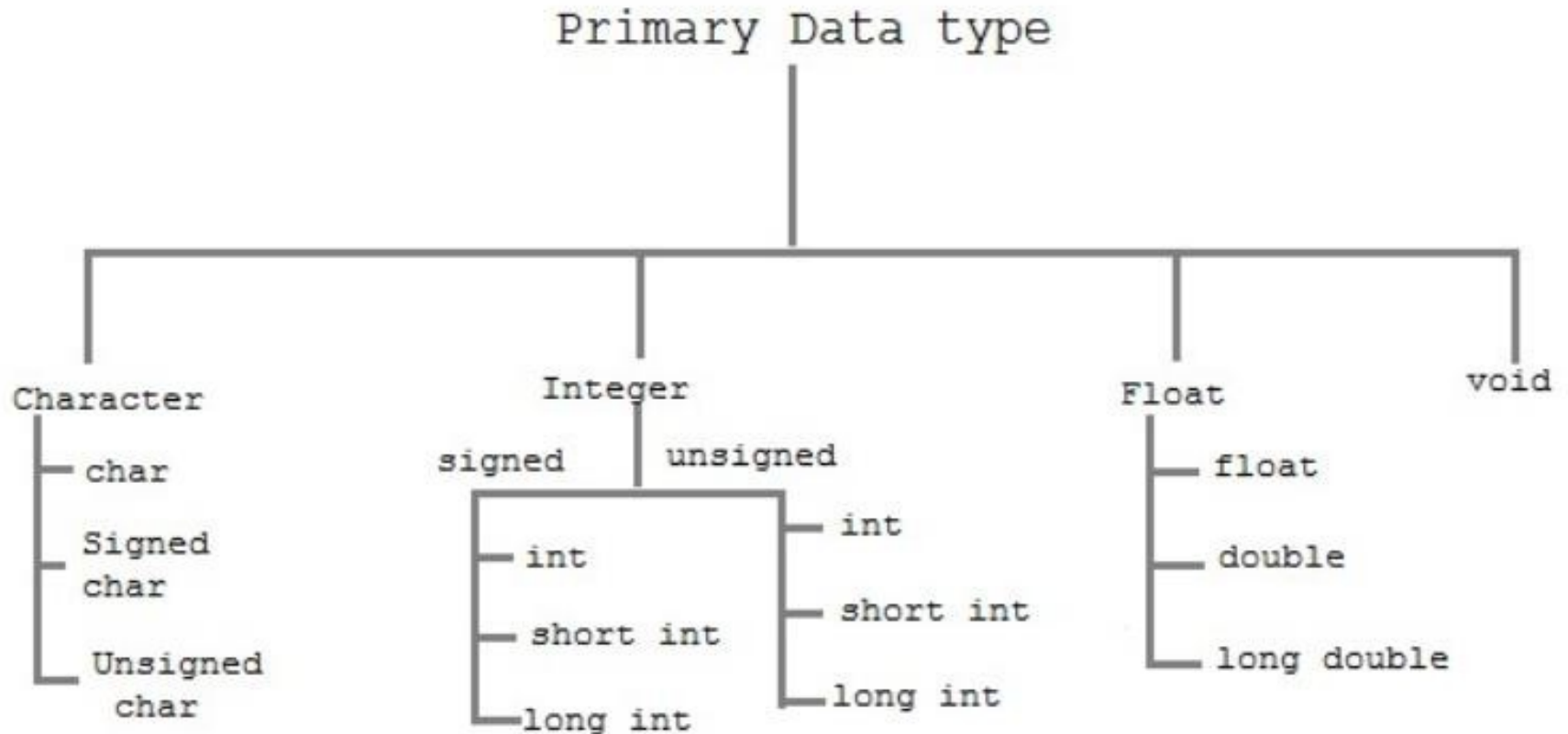
Variables

- Variable is a meaningful name given to the data storage location in computer memory.

- When using variable, we actually refer to address of the memory where the data is stored.

- It can store one value at a time.

- Its value may be changed during the program execution.

Variables

```
#include<stdio.h>
int main()
{   int a =10;
printf("%d",a);
return 0;
}
```

Rules for defining Variables

- It must begin with character or an underscore without spaces.

- Length of variables varies from compiler to compiler. However, ANSII standard recognizes max length of variable up to 31 characters.

- Its name should not be a C keyword

- It can be a combination of upper and lower case.

- It should not start with a digit

- Blanks and commas are not permitted

Examples of Variables

- int emp_num;
- float salary;
- char grade;
- double balance_amount;
- unsigned short int velocity;

Initializing Variables

- int emp_num = 7;
- float salary = 1100.50;
- char grade = 'A';
- double balance_amount = 5000;

Constants

- A constant is an entity whose value does not change during the execution of the program
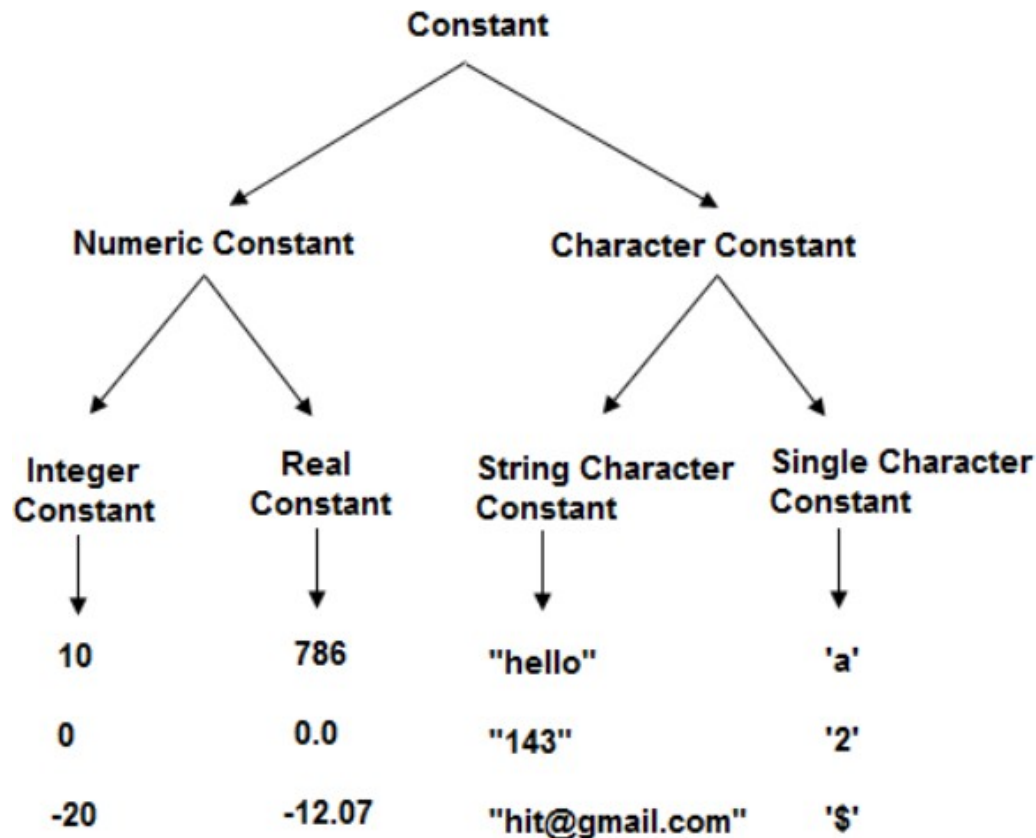
Constants

- Declaration

const int var = 100;

Constants

```c
#include<stdio.h>
int main()
{ const int a=10;
 printf("%d",a);
a=20; // gives error you can't modify const
return 0;
}
```

Variables and constants

Variables   Constants

Input/ Output Statements

- C language supports two input/output functions printf and scanf.

- printf is used to convert data stored in a program into a text stream for output to the monitor.

- scanf is used to convert the text stream coming from the keyboard to data values and stores them in program variables.

keyboard

standard input file

scanf(...)

monitor

standard output file

printf(...)

Memory
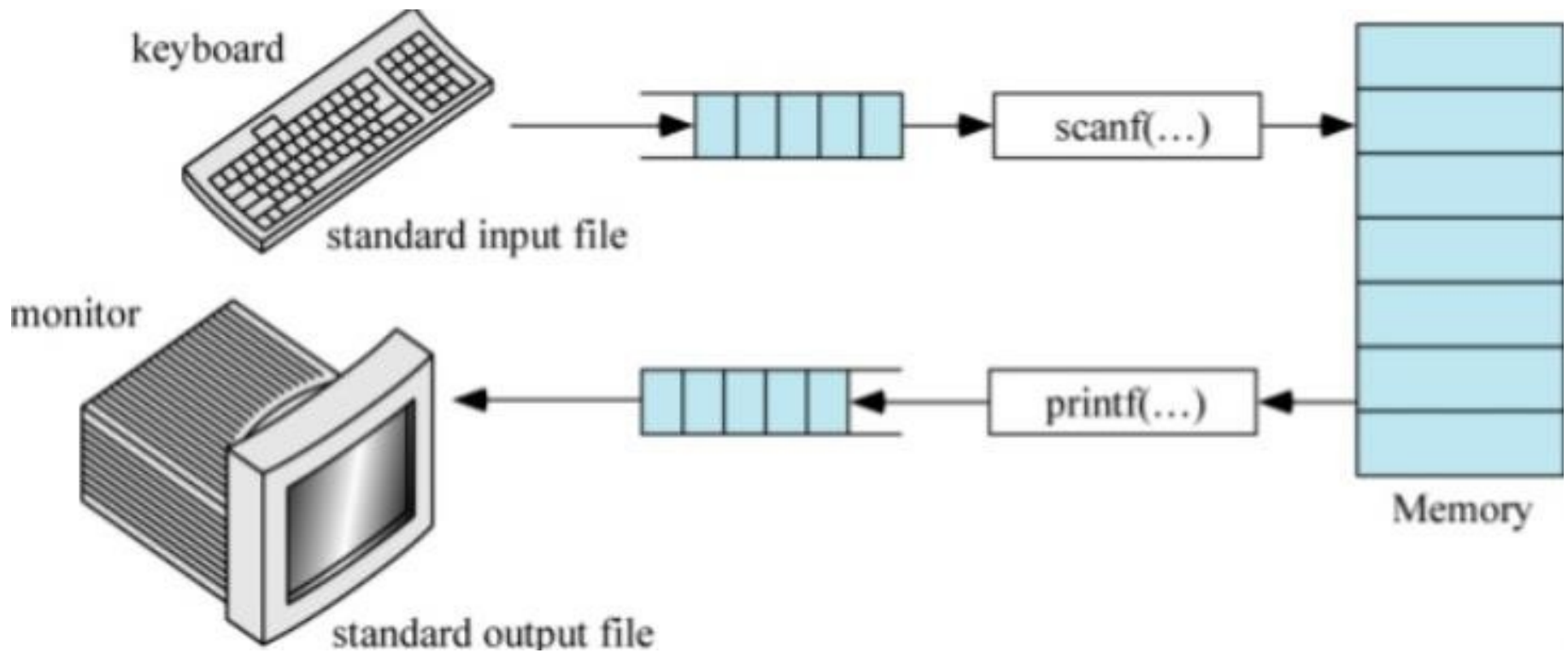
printf()

- Print Formatting.

- Displays information required by the user.

- Prints values of the variables.

  - Takes data values, converts them to text stream using formatting functions specified in a control string and passes resulting text stream to standard output.

printf()

- The control string may contain zero or more conversion specifications, textual data, and control characters to be displayed.

- printf("control string" , arg1, arg2………, argn);

- Control characters can also be included in printf statement.

- \n , \t , \r , \a etc.

printf()

- printf("This is a message \n");

- How do we print the value of a variable?

- int a = 22, b = 14;
- Answer: Use special   format   specifiers depending on the type of the variable
- Format specifiers begin with % sign

"%d\n%d"   a, b

printf ("%d\n%d", a, b);   a $\boxed{22}$ b $\boxed{14}$

... 22¬14 ...

Output stream
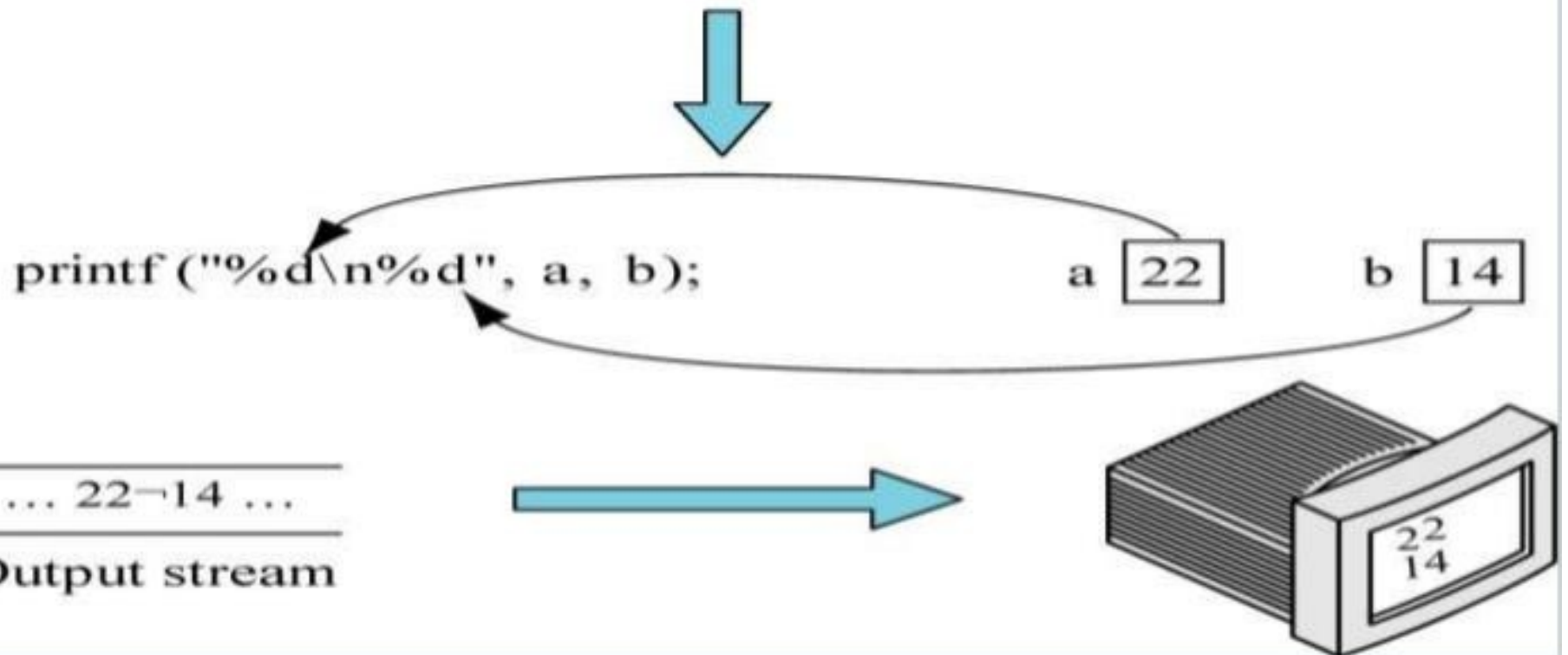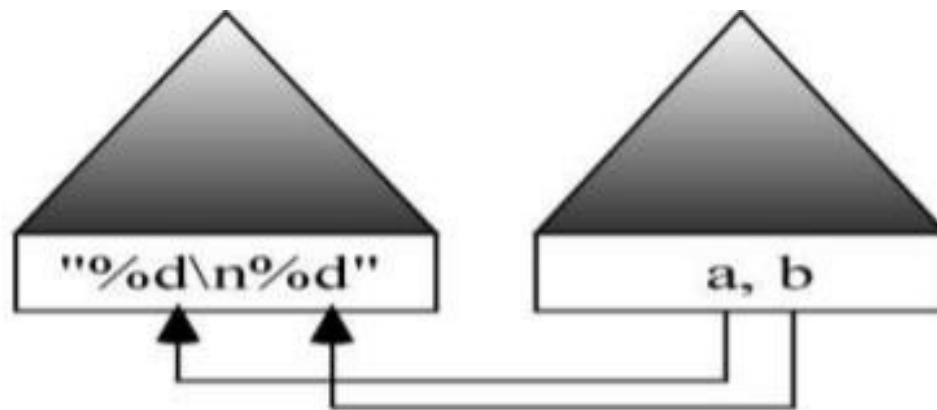
Format Specifier	Description
%d	Integer Format Specifier
%f	Float Format Specifier
%c	Character Format Specifier
%s	String Format Specifier
%u	Unsigned Integer Format Specifier
%ld	Long Int Format Specifier

scanf()

- Scan Formatting.

- Read formatted data from the keyboard.

 - Takes text stream from the keyboard, extracts and formats data from the stream according to a format control string and then stores the data in specified program variables.

scanf()

- scanf("control string" , arg1, arg2........., argn);

- The control specifies type and format of data obtained from the keyboard and stored in memory locations pointed by the arguments.

- The arguments are actually variable addresses where each piece of data are to be stored.
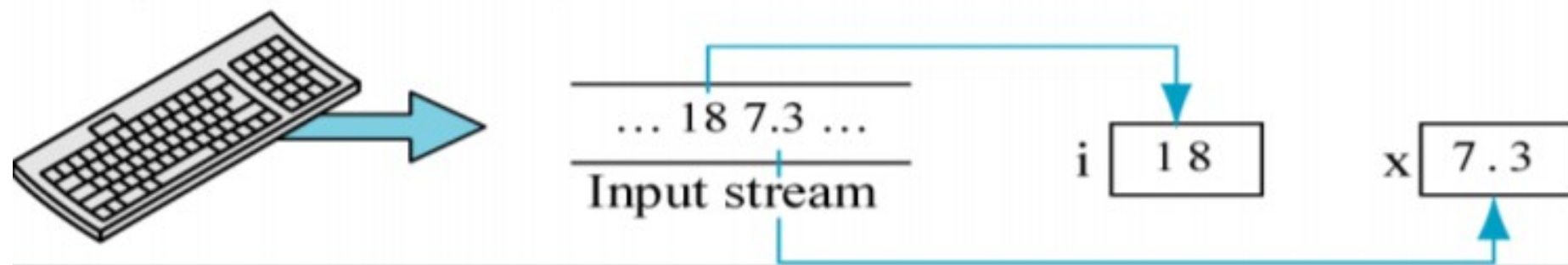
- Ignores blank spaces, tabs, newline

scanf()

```c
#include <stdio.h>
int main()
{
int     i ;
float x;
scanf("%d%f", &i, &x);
  return 0;
}
```

"%d%f" &i, &x

scanf("%d%f", &i, &x);

... 18 7.3 ...
Input stream

i 18

x 7.3
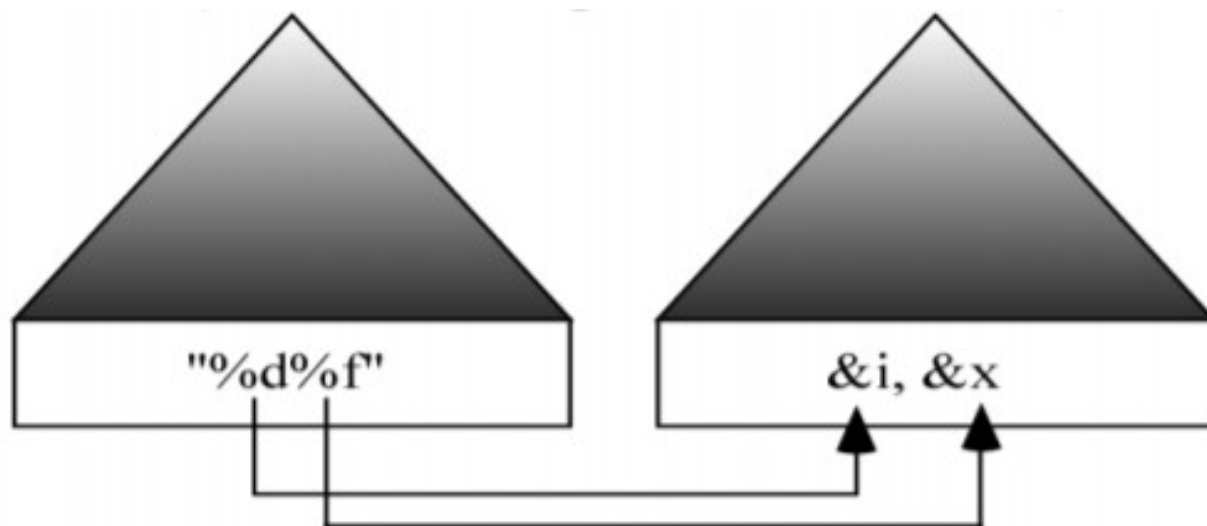
Example

#include <stdio.h> int main()

{

printf(" \n Result: %d %c%f",   12, 'a', 2.3);

  return 0;

}

Result: 12a2.3

C operators

- To build an expression, operators are used with operands like 4 + 5, Here, 4 and 5 are operands and '+' is operator

C contains following group of operators –
1) Arithmetic
2) Assignment
3) Logical/Relational
4) Bitwise
5) Miscellaneous

Arithmetic Operators

Symbol	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo (remainder)
--	Decrement
++	Increment

Pre increment/Decrement (++i or --i):
First increment/decrement current value by 1 and then use   the incremented/decremented value
Post increment/Decrement (i++ or i--):
First use the current value and then increment/decrement current value by 1 for next time use

Arithmetic operators Example

```
#include<stdio.h>  void main()

{
int i=7,j=5;
printf("%d %d %d %d \n",i+j, i-j,  i*j, i/j );  printf("%d
\n",      i%j );

printf("%d %d \n", i++, ++j );
}
```

Assignment Operators

- Each expression contains two parts: lvalue and rvalue,
- Arithmetic operations are performed to calculate rvalue,
- Assignment operators are used to assign the rvalue to lvalue Example:

i=i+1

can also be reduced to i+=1

Symbol	Name
=	Assign
*=	Multiply
/=	Divide
-=	Subtract
+=	Add
%=	Modulus

Assignment Operators Example

```c
#include<stdio.h>
void main()
{
int i=6;
printf("%d \n",i=8);
i+=1;
printf("%d \n",i);
i*=2;
printf("%d \n",i);
i/=3;
printf("%d \n",i);
i%=4;
printf("%d \n",i);

}
```

```
8
9
18
6
2
```

Relational Operators

A **relational operator** is a programming language construct or operator that tests or defines some kind of relation between two entities.

Symbol	Name
==	Equal to
<	Less than
>	Greater Than
<=	Less or equal
>=	Greater or equal
!=	Not equal

•If condition is true it returns 1 else returns 0.

Relational Operators (Cont..)

```
#include<stdio.h>
void main()
{
int i=7,j=1,k=0;
printf("%d \n",i==j);          0
printf("%d \n",i<j);           0
printf("%d \n",i>j);           1
printf("%d \n",i<=j);          0
printf("%d \n",i>=j);          1
printf("%d \n",i!=j);          1
}
```

Relational Operators (Cont..)

```c
/* C program to find largest number using if...else statement
*/ #include <stdio.h>

int main()
{      float a, b, c;
printf("Enter three numbers: ");  scanf("%f %f %f", &a, &b, &c);
if (a>=b) {
if(a>=c)
printf("Largest number = %f",a);
else
printf("Largest number = %f",c);
}
else {
if(b>=c)
printf("Largest number = %f",b);
else
printf("Largest number =
}
return 0;
}
```

```
Enter three numbers: 10
30
%60
; Largest number = 60.000000
```

Logical Operators

- Allow a program to make a decision based on multiple conditions.

- Each operand is considered a condition that can be evaluated to a true or false value.

- **Logical operator returns 0 or 1.**
- The Logical operators are: &&, || and !
- **op1 && op2**-- Performs a logical AND of the two operands.
- **op1 || op2**-- Performs a logical OR of the two operands.
- **!op1**-- Performs a logical NOT of the operand.
- Op1 and op2 may be one condition or single operand
- **C considers non-zero value to true and zero to false.**
- Examples:(a>b)&&(a>c) , a&&b, 5&&8 etc.

Logical Operators

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

OR

A	not A
0	1
1	0

NOT

&&(Logical AND)

- The && operator is used to determine whether both operands or conditions are true
- For example:
- a=50,b=9
- if (a == 10 && b == 9)

printf("Hello!");

If either of the two conditions is false or incorrect, then the printf command is bypassed.

Example of Logical

```c
/* C program to find largest number using if statement only */

#include <stdio.h>
int main(){
        float a, b, c;
        printf("Enter three numbers: ");
        scanf("%f %f %f", &a, &b, &c);
        if(a>=b && a>=c)
            printf("Largest number = %.2f", a);
        if(b>=a && b>=c)
            printf("Largest number = %.2f", b);
        if(c>=a && c>=b)
            printf("Largest number = %.2f", c);
        return 0;
}
```

Output:

```
Enter three numbers: 12.2
13.452
10.193
Largest number = 13.45
```

|| (Logical OR)

- The || operator is used to determine whether either of the condition is true.
- For example:
- a=50,b=9
- if (a== 10 || b == 9) // either of the condition should be true for printing hello! printf("Hello!");

If both of the two conditions are false, then only the printf command is bypassed.

Ex2: if(0||9)

printf("Correct !");

else

printf("Incorrect !");

Caution: If the first operand of the || operator evaluates to true, the second operand will not be evaluated. This could be a source of bugs if you are not careful.

- For instance, in the following code fragment:
- if (9 || X++) {
- print("X=%d\n",X); } variable X will not be incremented because first condition is evaluates to true.

|| (Logical OR)

Caution: If the first operand of the || operator evaluates to true, the second operand will not be evaluated. This could be a source of bugs if you are not careful.

For instance, in the following code fragment:

```
if (9 || X++)
 {
print("X=%d\n",X);
}
```

variable X will not be incremented because first condition is evaluates to true.

! (Logical NOT)

- The ! operator is used to convert true values to false and false values to true. In other words, it inverts a value

- For example:

- a=50,b=9

- if (!(a== 10)) //
printf("Hello!");

Ex2: if(!(0||9))
printf("Correct !");
else
printf("Incorrect !");

Bitwise Operators

•In the C programming language, operations can be performed on a bit level using bitwise operators.

•Following are the bitwise Operators

Symbol	Operator
&	bitwise AND
|	bitwise inclusive OR
^	bitwise exclusive OR
<<	left shift
>>	right shift
~	bitwise NOT (one's complement) (unary)

Bitwise AND

•A **bitwise AND** takes two binary representations of equal length and performs the logical AND operation on each pair of corresponding bits.

The result in each position is 1 if the first bit is 1 *and* the second bit is 1;
otherwise, the result is 0.

Example: a=5, b=3;
printf("%d",a&b); will print 1

bit a	bit b	a & b (a AND b)
0	0	0
0	1	0
1	0	0
1	1	1

    0101--5

  0011--3

    0001--1

Bitwise OR(|)

A **bitwise OR** takes two bit patterns of equal length and performs the logical inclusive OR operation on each pair of corresponding bits.

The result in each position is 1 if the first bit is 1 *or* the second bit is 1 *or* both bits are 1; otherwise, the result is 0.

Example: a=5, b=3;
printf("%d",a|b);
will print 7

  0101--5

  0111--7

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

OR

Bitwise
XOR(^)

A **bitwise XOR(^)** takes two bit patterns of equal length and performs the logical exclusive OR operation on each pair of corresponding bits.

The result in each position is 1 if only the first bit is 1 *or* only the second bit is 1, but will be 0 if both are 0 or both are 1.

printf("%d",a^b); will print 6

```
  0101--5
^ 0011--3
  0110--6
```

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Shift Operators

Left Shift Operator (<<): The left shift operator will shift the bits towards left for the given number of times.

       int a=2<<1;

       printf("%d",a);//_{will print 4}

If you left shift like 2<<2, then it will give the result as 8.

Therefore left shifting n time, is equal to multiplying the value by 2^n.

Shift Operators

Right shift Operator (>>)

The right shift operator will shift the bits towards right for the given number of times

      int b=4>>1

      printf("%d",b);/ _{will print 2}

      /

Right shifting n time, is equivalent to dividing the value by 2^n.

Bitwise Operators Example

```c
#include<stdio.h>
void main()
{
int i=5,j=7,c;
c=i & j;
printf("%d \n",c);
c=i | j;
printf("%d \n",c);
c=i ^ j;
printf("%d \n",c);
i-=1;
printf("%d %d\n",i<<=3,i>>=2);

}
```
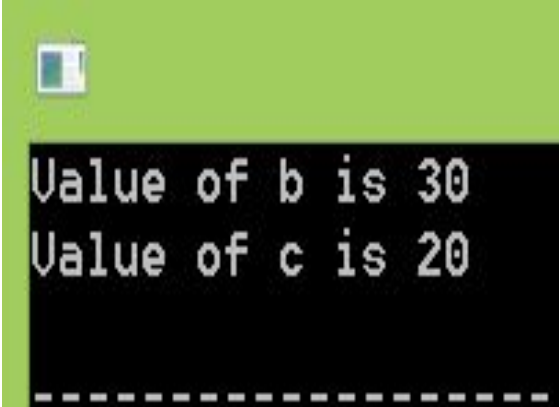
Miscellaneous Operators

- There are few other important operators including **sizeof** and **? :** supported by C Language.

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is interger, will return 4.
&	Returns the address of an variable.	&a; will give actaul address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Examples of **conditional operator(**Ternary Operator)

```
main() {
int a , b,c;
a = 10;
b = (a == 1) ? 20: 30;
printf( "Value of b is %d\n", b );
c = (a == 10) ? 20: 30;
printf( "Value of c is %d\n", c );
}
```

WAP to check even or odd using
conditional operator

```c
#include<stdio.h>  main() {

int n;
printf("Enter number n");

 scanf("%d",&n);

(n%2 == 0) ? printf("%d is even",n): printf("%d is odd",n);
}
```

Operators available in C can be classified in following ways:

1. **unary** – that requires only one
    operand. For example: &a, ++a, *a ,--b
    etc.

2. **binary** - that requires two operands.
    For example: a + b,   a * b, a<<2

3. **ternary** - that requires three operands.
    For example: (a>b) ? a : b [? :]

Precedence and Associativity

- If more than one operators are involved in an expression then, C language has predefined rule of priority of operators. This rule of priority of operators is called operator precedence.

- Associativity indicates in which order two operators of same precedence (priority) executes.

Operators	Associativity
(expression) [index] → .	LR
! ~ ++ -- (type) sizeof Unary * &	RL
* / %	LR
+ -	LR
>> <<	LR
> >= < <=	LR
== !=	LR
Binary &	LR
Binary ^	LR

Operators	Associativity
Binary |	LR
&&	LR
||	LR
+= -= *= /= %= etc. (all assignments)	RL
,	LR

LR : Left to Right   RL: Right to Left

Examples of precedence of operators

```
(a>b+c&&d)
This expression is equivalent to:
((a>(b+c))&&d)
i.e, (b+c) executes first
then, (a>(b+c)) executes
then, (a>(b+c))&&d) executes
```

- precedence of arithmetic operators(*,%,/,+,-) is higher than relational operators(==,!=,>,<,>=,<=) and precedence of relational operator is higher than logical operators(&&, || and !).

Example of Associativity of Operators

Example 1:
a==b!=c

Here, operators == and != have same precedence.

The associativity of both == and != is left to right, i.e, the expression in left is executed first and execution take place towards right.
Thus, a==b!=c equivalent to : (a==b)!=c

Example of Associativity of Operators

Example 2:

Associativity is important, since it changes the meaning of an expression. Consider the division operator with integer arithmetic, which is left associative

4 / 2 / 3 (4 / 2) / 3 2 / 3 = 0

If it were right associative, it would evaluate to an undefined expression, since you would divide by zero

4 / 2 / 3 4 / (2 / 3) 4 / 0 = undefined

What will be the value of i

$i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8$

$i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8$	operation: *
$i = 1 + 4 / 4 + 8 - 2 + 5 / 8$	operation: /
$i = 1 + 1 + 8 - 2 + 5 / 8$	operation: /
$i = 1 + 1 + 8 - 2 + 0$	operation: /
$i = 2 + 8 - 2 + 0$	operation: +
$i = 10 - 2 + 0$	operation: +
$i = 8 + 0$	operation : -
$i = 8$	operation: +

Arithmetic expressions vs C expressions

•Although Arithmetic instructions look simple to use one often commits mistakes in writing them.

•following points should keep in mind while writing C programs:

(a)C allows only one variable on left-hand side of **= That is, z =k * l is legal, whereas k * l = z is illegal.**

(b)Other than division operator(/) C has modulus operator(%). Thus the expression 10 / 2 yields 5, whereas, 10 % 2 yields 0.

•Note that the modulus operator (**%) cannot be applied on a float.**
•Also note that on using % the sign of the remainder is always same as the sign of the numerator. Thus –5 % 2 yields –1, whereas, 5 % -2 yields 1.
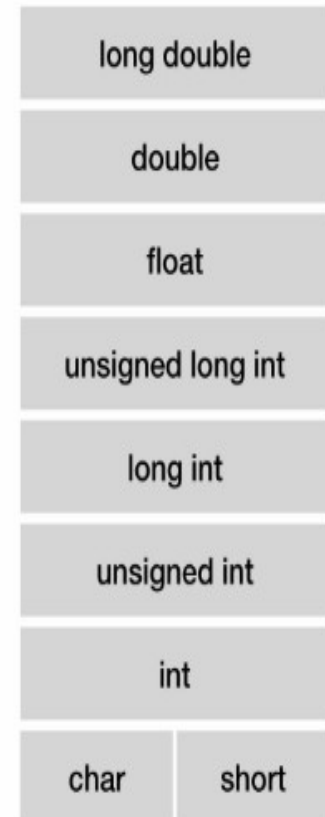
Type Conversion & Type Casting

- Type conversion or type casting of variables refers to changing a variable of one data type into another.

- Type conversion is done implicitly whereas,

- Type casting has to be done explicitly

Type Conversion

It is done when the expression has variables of different data types.

To evaluate the expression, the type is promoted from lower to level where the hierarchy of data is given in the figure

conversion hierarchy

long double
double
float
unsigned long int
long int
unsigned int
int

char	short

Type Conversion

float x;

int y = 3;

x = y;

Output:
x = 3.000000

```c
#include<stdio.h>
 int main(){
float x=81.2;
printf("%f",x);
}
// 81.199997 due to digital
// conversion error
```

Type Conversion Example

```c
#include <stdio.h>  main()
{ int number = 1;
char character = 'k';
 int sum;
sum = number + character;
printf("Value of sum : %d\n", sum );
}
```

OUTPUT
Value of sum :
10 8

Important regarding Type Conversion

- Type conversion is also called as implicit or standard type conversion.

- We do not require any keyword or special statements in implicit type casting.

- Converting from smaller data type into larger data type is also called as **type promotion**.

- The type conversion always happens with the compatible data types.

Important regarding Type Conversion

- Converting   float   to an int will   truncate the fraction part hence losing the meaning of the value.

- Converting double to float will round up the digits.

- Converting   long   int to int will   cause dropping of excess high order bits.

Type Casting

- Type casting is also known as forced conversion or explicit type conversion

- It is done when the value of a higher data type has to be converted into the value of a lower data type.

float salary = 10000.00; int sal;

sal = (int) salary;

Type Casting

res = (int) 9.5;

res = (int) 12.3/ (int) 4.2;

res = (double) total / n;

res = (int) a + b;

res = cos ((double) x);

Thank You