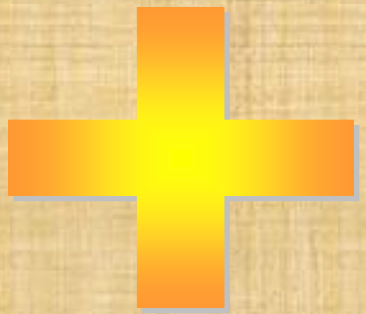
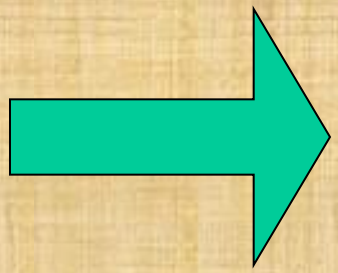
Lecture 1&2
Programming Essentials in C

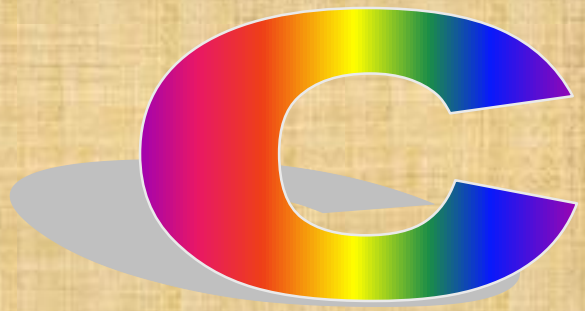
C – A Middle Level Language

In 1972 **Dennis Ritchie** at Bell Labs writes C and in 1978 the publication of **The C Programming Language** by Kernighan & Ritchie caused a revolution in the computing world

High-Level Language







Low Level Language

Why use C?

- Mainly because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:
  - Operating Systems
  - Language Compilers
  - Assemblers
  - Text Editors
  - Data Bases
  - Language Interpreters

Why C Still Useful?

- **C provides:**
  - Efficiency, high performance and high quality
  - flexibility and power
  - many high-level and low-level operations → middle level
  - Stability and small size code
  - Provide functionality through rich set of function libraries
  - Gateway for other professional languages like C → C++ → Java

- **C is used:**
  - System software Compilers, Editors, embedded systems
  - data compression, graphics and computational geometry, utility programs
  - databases, operating systems, device drivers, system level routines
  - there are zillions of lines of C legacy code
  - Also used in application programs

Terminologies of ‘C’

1. Keywords
2. Identifiers
3. Variables
4. Constants
5. Special Symbols
6. Character & String
7. Operators

1. Keywords

- Keywords are the reserved words whose meaning has already been explained to the C compiler.
- C has 32 keywords.
- These keywords combined with a formal syntax form a C programming language.
- Rules to be followed for all programs written in C:
  - All keywords are lower-cased.
  - C is case sensitive, do-while is different from DO WHILE.
  - Keywords cannot be used as a variable or function name.

2. Identifiers

- Identifiers refer to the name of variables, functions and arrays.
- These are user-defined names and consist of sequence of letters and digits, with a letter as a first character.
- Both uppercase and lowercase letters are permitted, although lowercase letters are commonly used.
- The underscore character is also permitted in identifiers. It is usually used as a link between two words in long identifiers.

Identifier Names

- Some correct identifier names are -

arena, s_count
marks40
class_one

- Some erroneous identifier names are -

1stsst
oh!god
start….end

- The number of characters in the variable that are recognized differs from compiler to compiler
- An identifier cannot be the same as a C keyword

3. Variables

- Variables are named locations in memory that are used to hold a value that may be modified by the program.

- Unlike constants that remain unchanged during the execution of a program.

- A variable may take different values at different times during execution.

- The syntax for declaring a variable is –

  DataType IdentifierName ;

- Example- int num;

  long int sum , a;

Type	Description
char	Typically a single octet (one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

4. Constants

- Constants are the fixed values that do not change during the execution of a program.
- C supports several types of constants.
  - Numeric Constants
    - Integer constants
    - Real constants
  - Character Constants
    - Single character constant
    - String Constants

5. Special Symbols

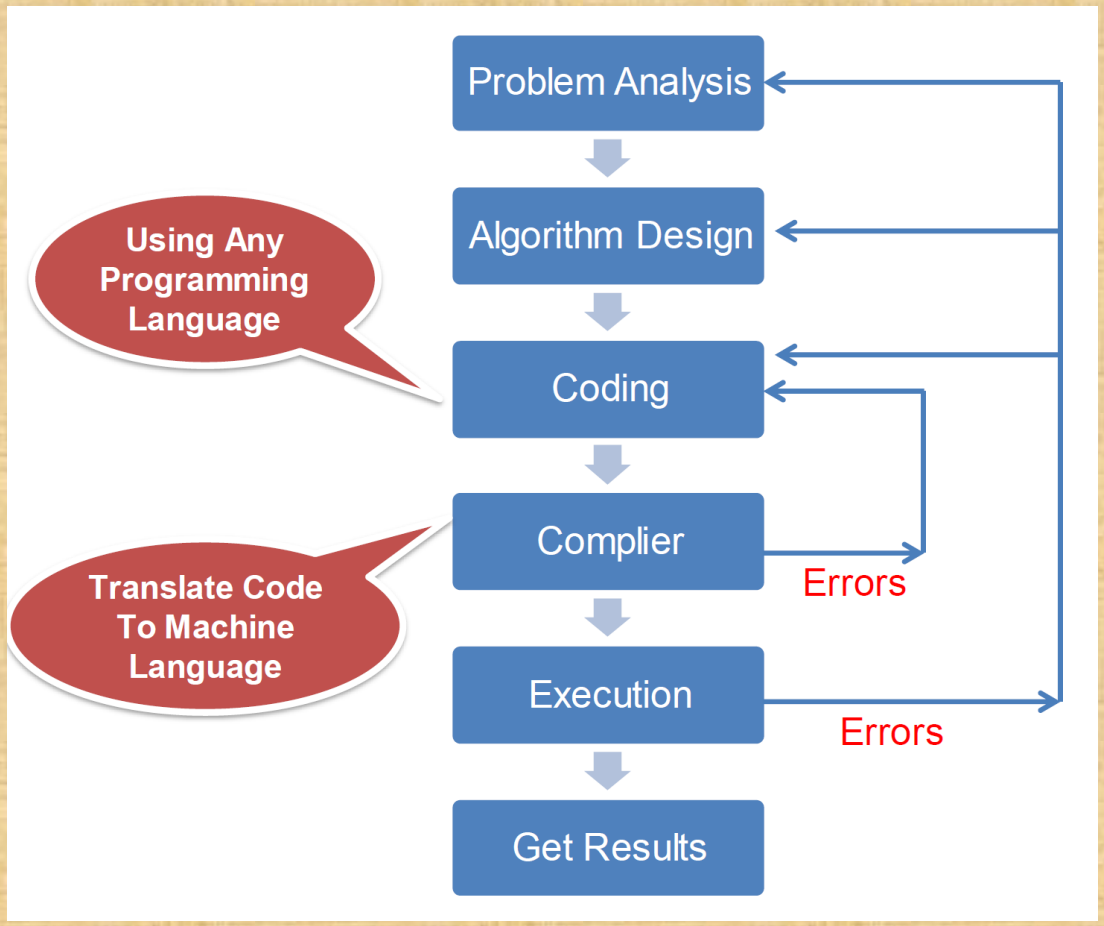
- ! , @, #, $, & , * , ….. These all symbols that can be find on Keyboard, are called Special Symbols.

- Every symbol has its special meaning in different respect at different place that’s why it is called Special Symbols.

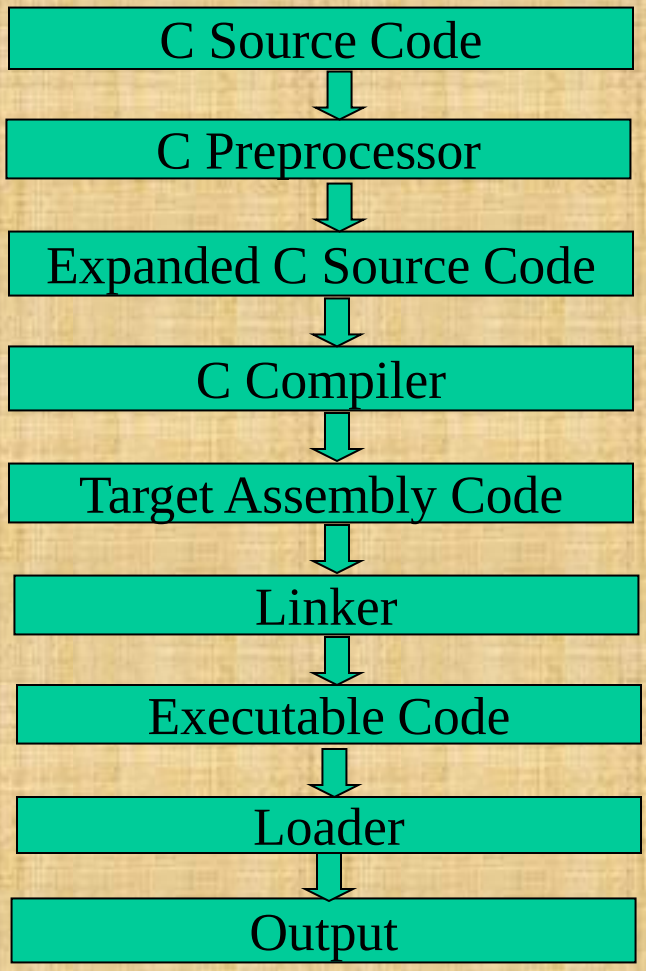
7. Character & String

- The Characters that can be used to form words, numbers and expressions depend upon the computer on which the program is running.
- The characters in C are grouped into the following categories :
  - Letters
  - Digits
  - Special characters
  - White Spaces
- Remember that a character ‘a’ is not equivalent to the string “a”.

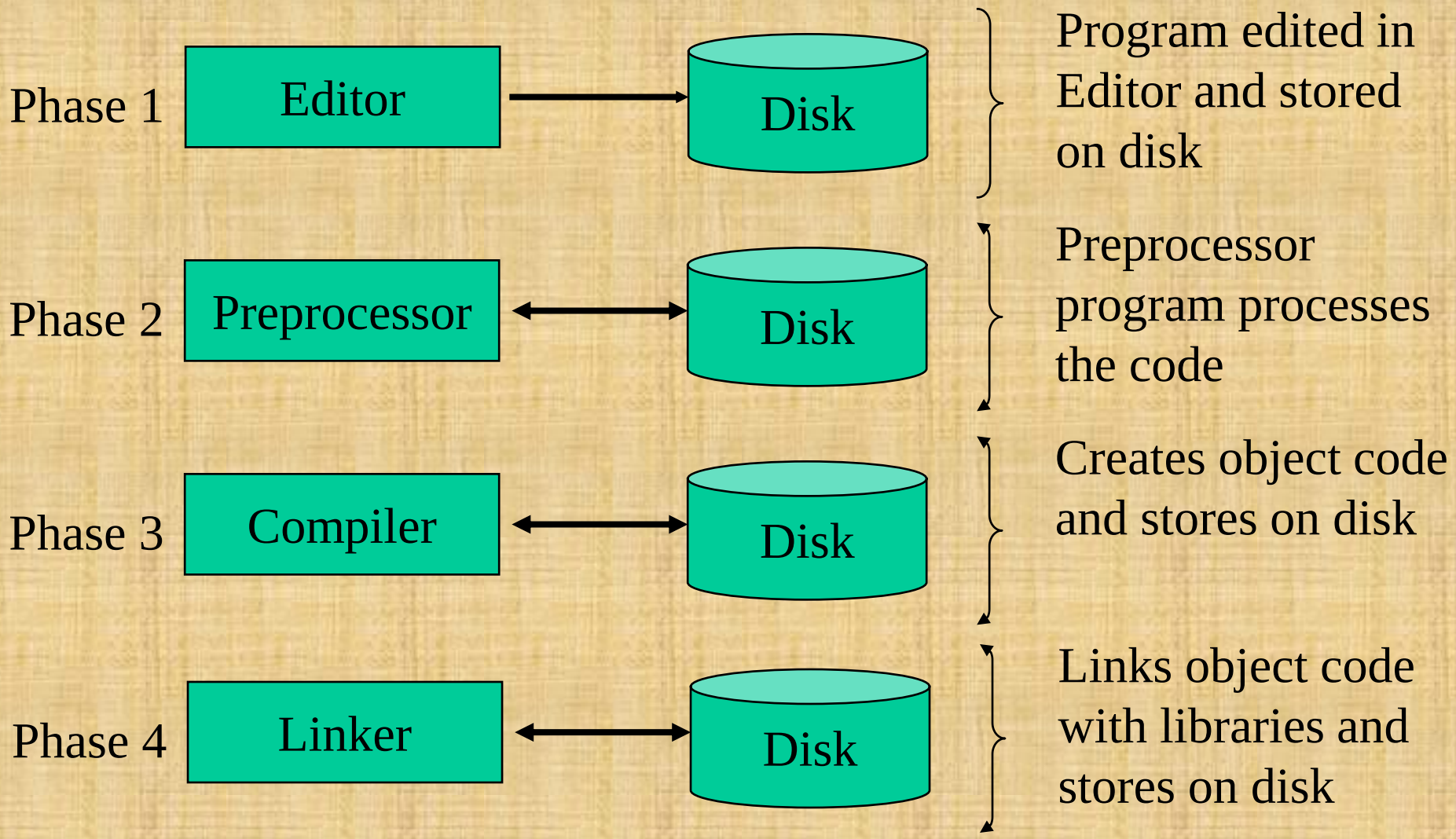
Program Development Life Cycle



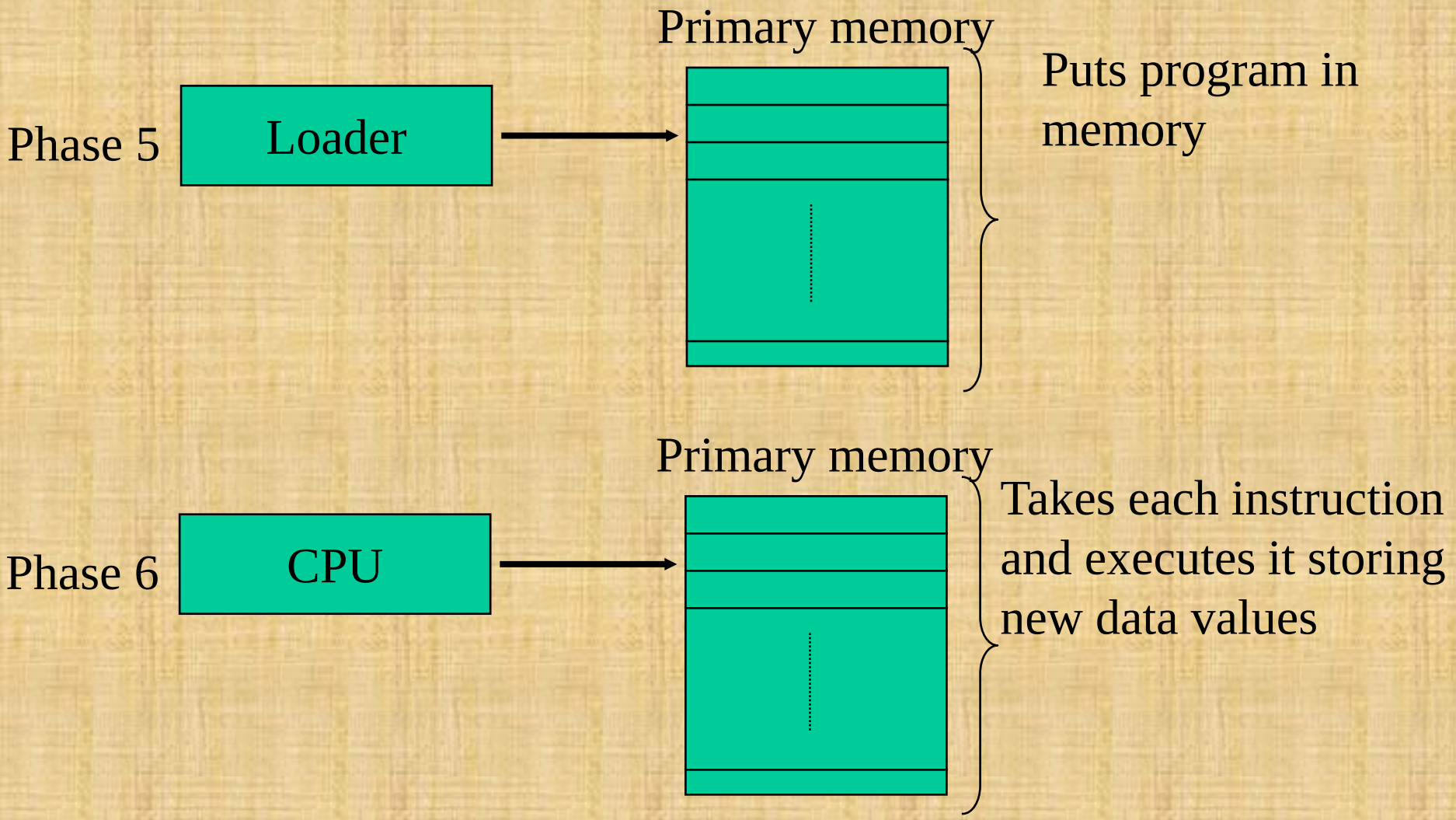
Compilation & Execution of a Program



Basics of C Environment



Basics of C Environment



A simple Program of C

```
/* Write a program to print a message */
#include<stdio.h>
void main()
    {
        printf(“ C Programming”);
     }
```

- Comment about the program should be enclosed within ‘/*’ & ‘*/’.
- printf() is a function which is used to print messages on the screen.
- main() is a function from which execution of the program starts.
- Here stdio.h is a library file which contains standard input/output functions , keywords etc.
- #include<> is used to define the library file that is to be used in the program for compiler information.

Basics of C Environment

- C systems consist of 3 parts
  - Environment
  - Language
  - C Standard Library
- Development environment has 6 phases
  - Edit
  - Pre-processor
  - Compile
  - Link
  - Load
  - Execute

Execution Process of a Program

PROCESS	SHORT CUT	FILE NAME
Save	F2	abc.c
Compile	Alt + F9	abc.obj
Execute	Ctrl + F9	abc.exe
Back up	Again F2	abc.bak

Escape Sequence

- \n new line
- \t tab
- \r carriage return
- \a alert
- \\ backslash
- \” double quote

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");

    return 0;
}
```

Outputs

```
Hello, World!
```

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
    printf("Hello, World!\n");
    return 0;
}
```

Outputs

```
Hello, World!
Hello, World!
```

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!    ");
    printf("Hello, World!\n");
    return 0;
}
```

Outputs

```
Hello, World!    Hello, World!
```

Program to Display "Hello, World!"

```c
#include <stdio.h>
int main() {
   // printf() displays the string inside quotation
   printf("Hello, World!");
   return 0;
}
```

Output

```
Hello, World!
```

6. Operators

- Operator is a symbol that operates on one or more operands and produces output.

Example - z = x + y ;

- In the above Example the symbols + and = are operators that operate on operands x, y , and z .

```c
#include<stdio.h>

int main() {
    int x=10;
    int y=25;
    int z=x+y;
    printf("Sum of x+y = %d", z);
}
```

Outputs

```
Sum of x+y = 35
```

6. Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the

- following types of operators:
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Misc Operators
- We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20, then:

Operator	Description	Example
+	Adds two operands.	A + B = 30
-	Subtracts second operand from the first.	A - B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Example

```c
#include <stdio.h>
main()
{
    int a = 21;
    int b = 10;
    int c ;
    c = a + b;
    printf("Line 1 - Value of c is %d\n", c );
    c = a - b;
    printf("Line 2 - Value of c is %d\n", c );
    c = a * b;
    printf("Line 3 - Value of c is %d\n", c );
    c = a / b;
    printf("Line 4 - Value of c is %d\n", c );
    c = a % b;
    printf("Line 5 - Value of c is %d\n", c );
    c = a++;
    printf("Line 6 - Value of c is %d\n", c );
    c = a--;
    printf("Line 7 - Value of c is %d\n", c );
}
```

outputs

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 21
Line 7 - Value of c is 22
```

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Example

```c
#include <stdio.h>
main()
{
    int a = 21;
    int b = 10;
    int c ;
    if( a == b )
    {
        printf("Line 1 - a is equal to b\n" );
    }
    else
    {
        printf("Line 1 - a is not equal to b\n" );

    }
    if ( a < b )
    {
        printf("Line 2 - a is less than b\n" );
    }
    else
    {
        printf("Line 2 - a is not less than b\n" );
    }
    if ( a > b )
    {
        printf("Line 3 - a is greater than b\n" );
    }
    else
    {
        printf("Line 3 - a is not greater than b\n" );
```

```
31      }
32      /* Lets change value of a and b */
33      a = 5;
34      b = 20;
35      if ( a <= b )
36      {
37          printf("Line 4 - a is either less than or equal to b\n" );
38      }
39      if ( b >= a )
40      {
41          printf("Line 5 - b is either greater than or equal to b\n" );
42      }
43  }
```

When you compile and execute the above program, it produces the following result:

```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to b
Line 5 - b is either greater than or equal to b
```

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Example

```c
#include <stdio.h>
main()
{
    int a = 5;
    int b = 20;
    int c ;
    if ( a && b )
    {
        printf("Line 1 - Condition is true\n" );
    }
    if ( a || b )
    {
        printf("Line 2 - Condition is true\n" );
    }
    /* lets change the value of a and b */
    a = 0;
    b = 10;
    if ( a && b )
    {
        printf("Line 3 - Condition is true\n" );
    }
    else
    {
        printf("Line 3 - Condition is not true\n" );
    }
    if ( !(a && b) )
    {
        printf("Line 4 - Condition is true\n" );
    }
}
```

outputs

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

Bitwise Operators

Bitwise operators work on bits and perform bit-by-bit operation. The truth table for &, |, and ^ is as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

- Assume A = 60 and B = 13; in binary format, they will be as follows:
- A = 0011 1100
- B = 0000 1101
- ----------
- A&B = 0000 1100
- A|B = 0011 1101
- A^B = 0011 0001
- ~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -61, i.e., 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240, i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15, i.e., 0000 1111

```c
#include <stdio.h>
main()
{
    unsigned int a = 60; /* 60 = 0011 1100 */
    unsigned int b = 13; /* 13 = 0000 1101 */
    int c = 0;
    c = a & b; /* 12 = 0000 1100 */
    printf("Line 1 - Value of c is %d\n", c );
    c = a | b; /* 61 = 0011 1101 */
    printf("Line 2 - Value of c is %d\n", c );
    c = a ^ b; /* 49 = 0011 0001 */
    printf("Line 3 - Value of c is %d\n", c );
    c = ~a; /*-61 = 1100 0011 */
    printf("Line 4 - Value of c is %d\n", c );
    c = a << 2; /* 240 = 1111 0000 */
    printf("Line 5 - Value of c is %d\n", c );
    c = a >> 2; /* 15 = 0000 1111 */
    printf("Line 6 - Value of c is %d\n", c );
}
```

When you compile and execute the above program, it produces the following result:

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

Assignment Operators

		= C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C

```c
#include <stdio.h>
main()
{
    int a = 21;
    int c ;
    c = a;
    printf("Line 1 - = Operator Example, Value of c = %d\n", c );
    c += a;
    printf("Line 2 - += Operator Example, Value of c = %d\n", c );
    c -= a;
    printf("Line 3 - -= Operator Example, Value of c = %d\n", c );
    c *= a;
    printf("Line 4 - *= Operator Example, Value of c = %d\n", c );
    c /= a;
    printf("Line 5 - /= Operator Example, Value of c = %d\n", c );
    c = 200;
    c %= a;
    printf("Line 6 - %= Operator Example, Value of c = %d\n", c );
    c <<= 2;
    printf("Line 7 - <<= Operator Example, Value of c = %d\n", c );
    c >>= 2;
    printf("Line 8 - >>= Operator Example, Value of c = %d\n", c );
    c &= 2;
    printf("Line 9 - &= Operator Example, Value of c = %d\n", c );
    c ^= 2;
    printf("Line 10 - ^= Operator Example, Value of c = %d\n", c );
    c |= 2;
    printf("Line 11 - |= Operator Example, Value of c = %d\n", c );
}
```

When you compile and execute the above program, it produces the following result:

```
Line 1 - = Operator Example, Value of c = 21
Line 2 - += Operator Example, Value of c = 42
Line 3 - -= Operator Example, Value of c = 21
Line 4 - *= Operator Example, Value of c = 441
Line 5 - /= Operator Example, Value of c = 21
Line 6 - %= Operator Example, Value of c = 11
Line 7 - <<= Operator Example, Value of c = 44
Line 8 - >>= Operator Example, Value of c = 11
Line 9 - &= Operator Example, Value of c = 2
Line 10 - ^= Operator Example, Value of c = 0
Line 11 - |= Operator Example, Value of c = 2
```

Misc Operators

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Online c Compiler

https://www.onlinegdb.com/online_c_compiler

https://www.tutorialspoint.com/compile_c_online.php

https://onecompiler.com/c

https://www.jdoodle.com/c-online-compiler/