Introduction to Computing and Programming

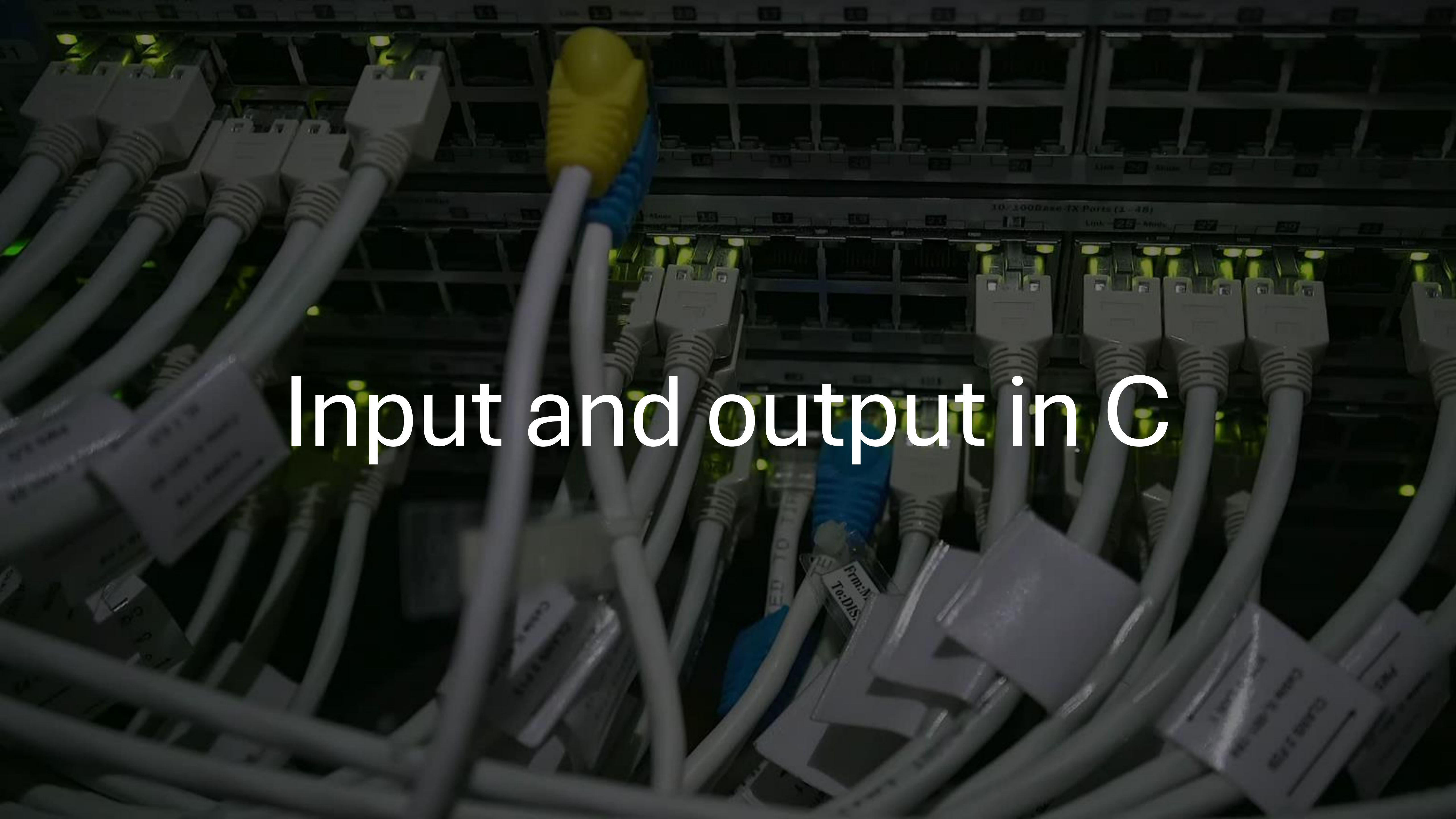
Operators, Logical Expressions and Conditional Statements, Number System

Content

- Quick Recap
- Input and Output in C
- Comments
- Operators
- Logical Expression
- Conditional Statements
- Number System

Recap

- Expressions
- Variables
- Operators: **Arithmetic & Logical operator**
- Conditional Statements: **Ternary Operator**



Input and output in C

Basic Methods : scanf()

C performs input and output operations using **stdio.h** or **standard input output library**

scanf()

- The scanf() reads the value from the console as per the type specified and store it in the given address.

  scanf("%X", &variableOfXType);

  where **%X** is the format specifier and **&** is the address operator which tells the compiler to change the real value of **variableOfXType**, stored at this address in the memory.

Basic Methods : printf()

- The printf() method prints the value passed as the parameter to it, on the console screen.
- Syntax:

```
printf("%X", variableOfXType);
```

where %X is the format specifier which tells the compiler what type of data is in a variable and variableOfXType is the variable to be printed.

How to take input and output of basic types in C?

- Integer:

  Input: scanf("%d", &intVariable);

  Output: printf("%d", intVariable);

- Float:

  Input: scanf("%f", &floatVariable);

  Output: printf("%f", floatVariable);

- Character:

  Input: scanf(“ %c", &charVariable);

  Output: printf("%c", charVariable);

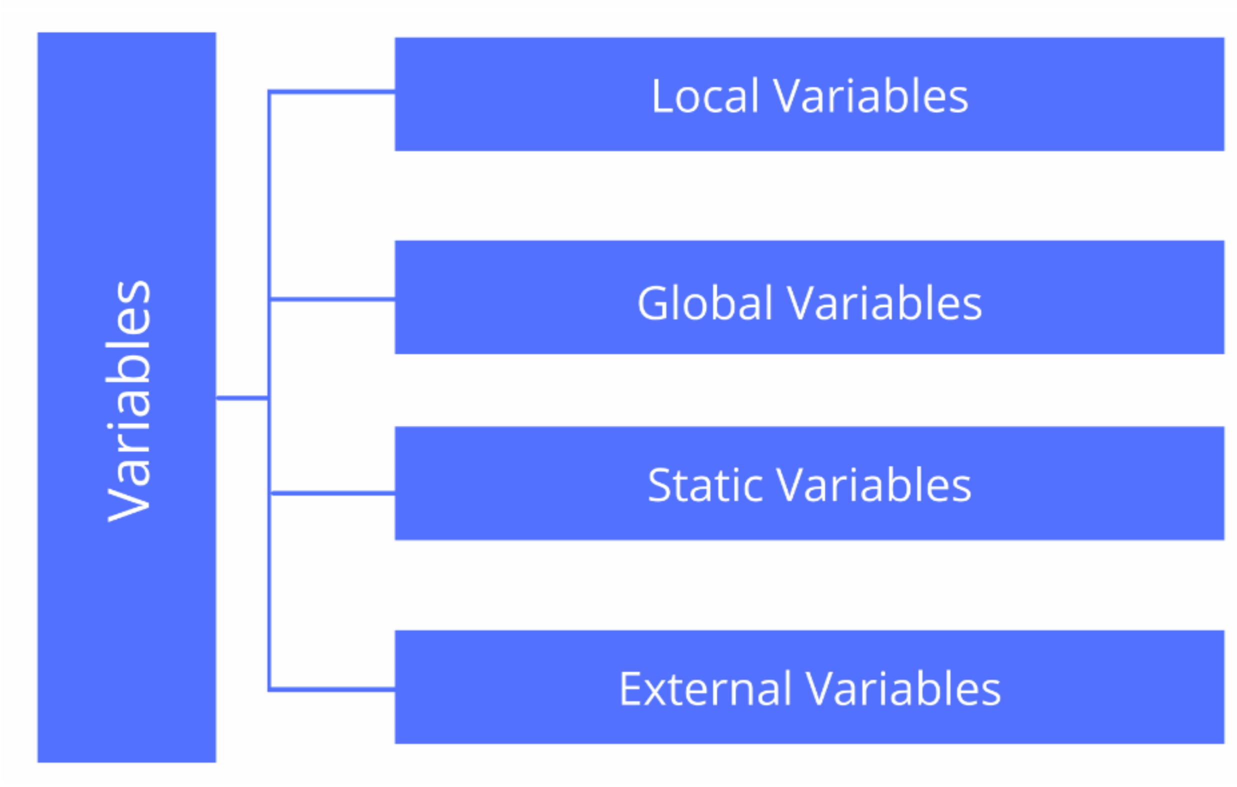
C program to show input and output operation

```
#include <stdio.h>
int main()
{
   int num;
   char ch;
   float f;
    printf("Enter the integer: ");
    scanf("%d", &num);                          // Input integer value
    printf("\nEntered integer is: %d", num);    // Output integer value
    printf("\n\nEnter the float: ");
    scanf("%f", &f);                            // Input float value
    printf("\nEntered float is: %f", f);        // Output float value
    printf("\n\nEnter the Character: ");
    scanf(“ %c", &ch);                          // Input character value
    printf("\nEntered character is: %c", ch);   // Output character value
    return 0;
}
```

User input and output for string

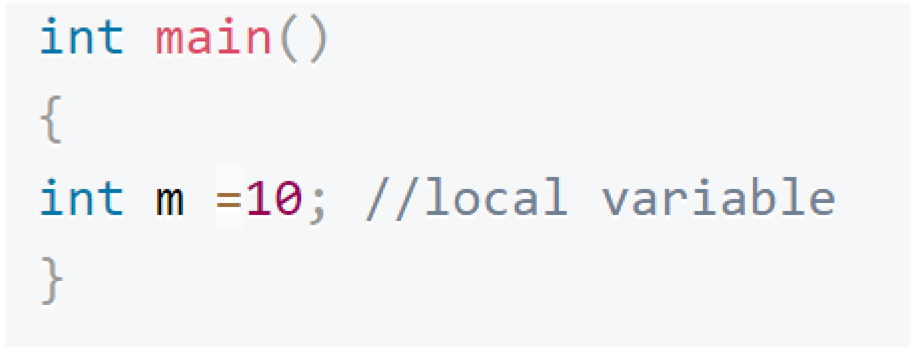
```
#include <stdio.h>
int main()
{
char str[50];
printf("Enter a single Word: ");
scanf("%s\n", str);
printf("Entered Word is: %s", str);
printf("Enter the complete sentence: ");
scanf("%[^\n]s", str);
printf("\nEntered Sentence is: %s", str);
return 0;
}
```

Types of variables



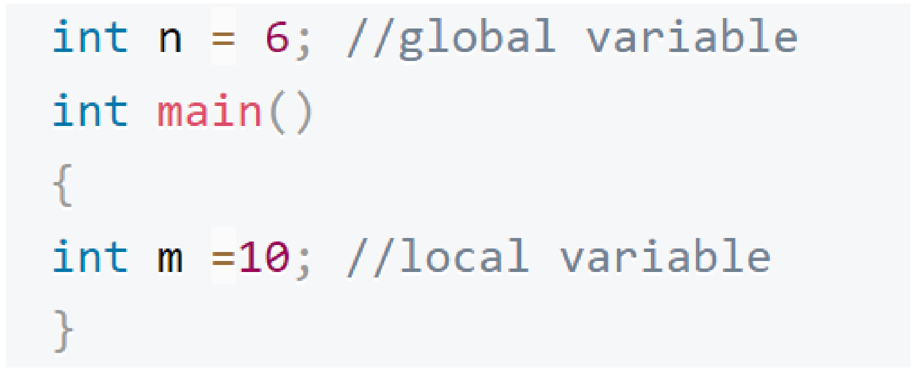
Types of Variable

- **Local variable:** Variables declared inside the functions and only local functions can change the value of variables.



```
int main()
{
int m =10; //local variable
}
```

- **Global variable:** Variables are declared outside the functions and any functions can change the value of variables.



```
int n = 6; //global variable
int main()
{
int m =10; //local variable
}
```

Program on local variable

```
#include <stdio.h>
void subfunc()
{
 int a;
 a = 10;
 printf("Sub function: Variable a = %d\n", a);
}
int main(void)
{
 subfunc();
 printf("Main: Variable a = %d\n", a);
 return (0);
}
```

```
#include <stdio.h>

void foo(int a)
{
 // Changing the value of 'a' in foo but not in main
 // since both variables are distinct
 a = 145;
 printf("Foo: Variable a = %d\n", a); // a == 145
}

int main(void)
{
 int a;

 a = 10;
 printf("Main: Variable a = %d\n", a); // a == 10
 foo(a);
 printf("Main: Variable a = %d\n", a); // a == 10
 return (0);
}
```

https://www.codequoi.com/en/local-global-static-variables-in-c/

Program on global variable

```
#include <stdio.h>
int a;
void foo(void)
{
 a = 42;
printf("Foo: a = %d\n", a); // a == 42
}
int main(void)
{
 printf("Main: a = %d\n", a); // a == 0
 foo();
 printf("Main: a = %d\n", a); // a == 42
 a = 200;
 printf("Main: a = %d\n", a); // a == 200
 return (0);
}
```

Program on static variable

```
#include <stdio.h>
void foo(void)
{
 int  a = 100;
 static int b = 100;
 printf("a = %d, b = %d\n", a, b);
 a = a+1;
 b = b+1;
}
int main(void)
{
 foo();
 foo();
 foo();
 return (0);
}
```

Program on extern variable

File1.c

```
#include <stdio.h>

extern int a;

void foo(void); // Foo prototype, defined elsewhere

int main(void)

{

 printf("Main: a = %d\n", a);

 foo();

 printf("Main: a = %d\n", a);

 a = 200;

 printf("Main: a = %d\n", a);

 return (0);

}
```

File2.c

```
#include <stdio.h>


int a = 100;


void foo(void)

{

 a = 42;

 printf("Foo: a = %d\n", a);

}
```

Comments in C

Comments can be used to explain code, and to make it more readable.

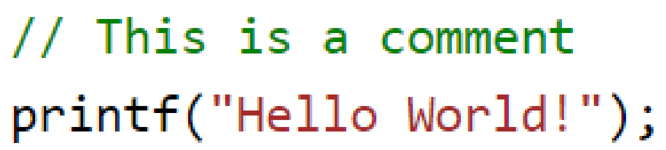
It can also be used to prevent execution when testing alternative code.

Comments can be **singled-lined** or **multi-lined**.

Comments in C

Single-line comments:

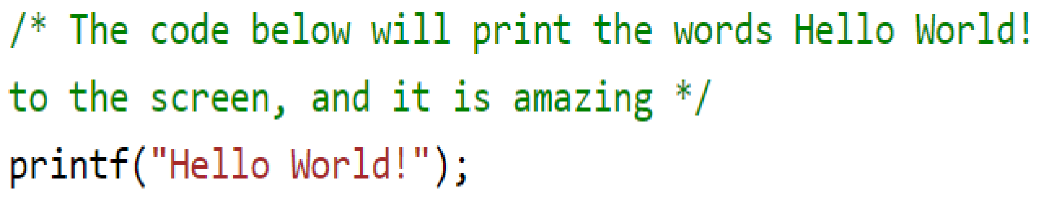
- Start with two forward slashes (//).
- Any text between // and the end of the line is ignored by the compiler (will not be executed).



```c
// This is a comment
printf("Hello World!");
```

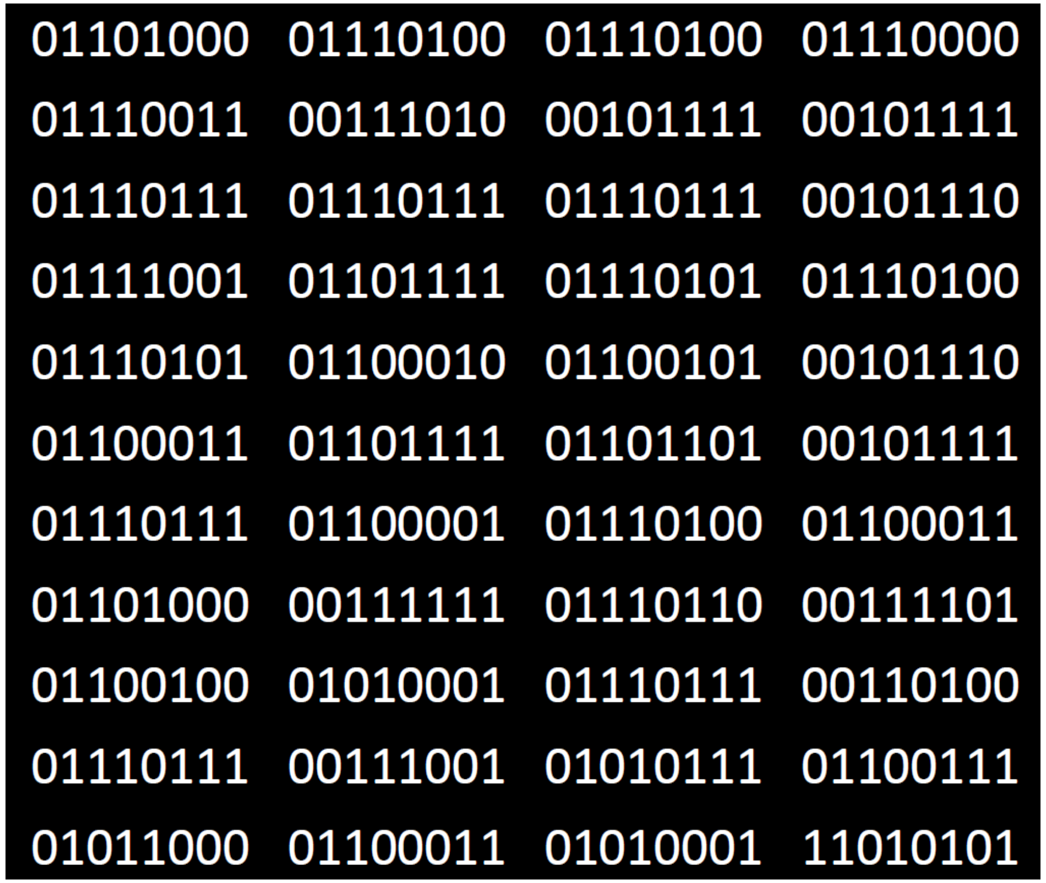
Multi line Comments:

- Multi-line comments start with /* and ends with */.
- Any text between /* and */ will be ignored by the compiler



```c
/* The code below will print the words Hello World!
to the screen, and it is amazing */
printf("Hello World!");
```

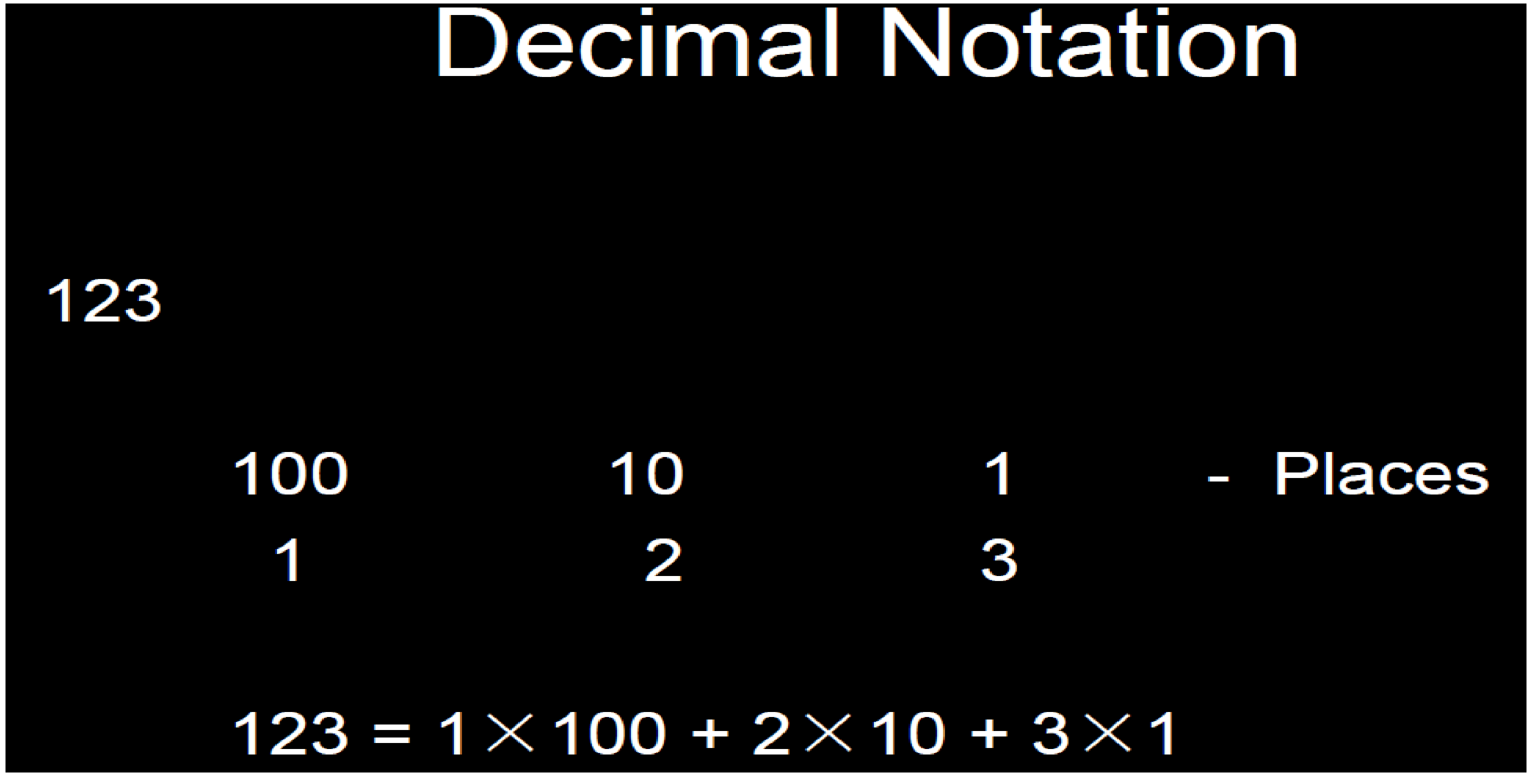
What do computers understand?



01101000	01110100	01110100	01110000
01110011	00111010	00101111	00101111
01110111	01110111	01110111	00101110
01111001	01101111	01110101	01110100
01110101	01100010	01100101	00101110
01100011	01101111	01101101	00101111
01110111	01100001	01110100	01100011
01101000	00111111	01110110	00111101
01100100	01010001	01110111	00110100
01110111	00111001	01010111	01100111
01011000	01100011	01010001	11010101

- Binary
  - 0/1
  - True/False
  - On/Off

Number System: Represents a quantity through a set of Numbers



Decimal Notation

123

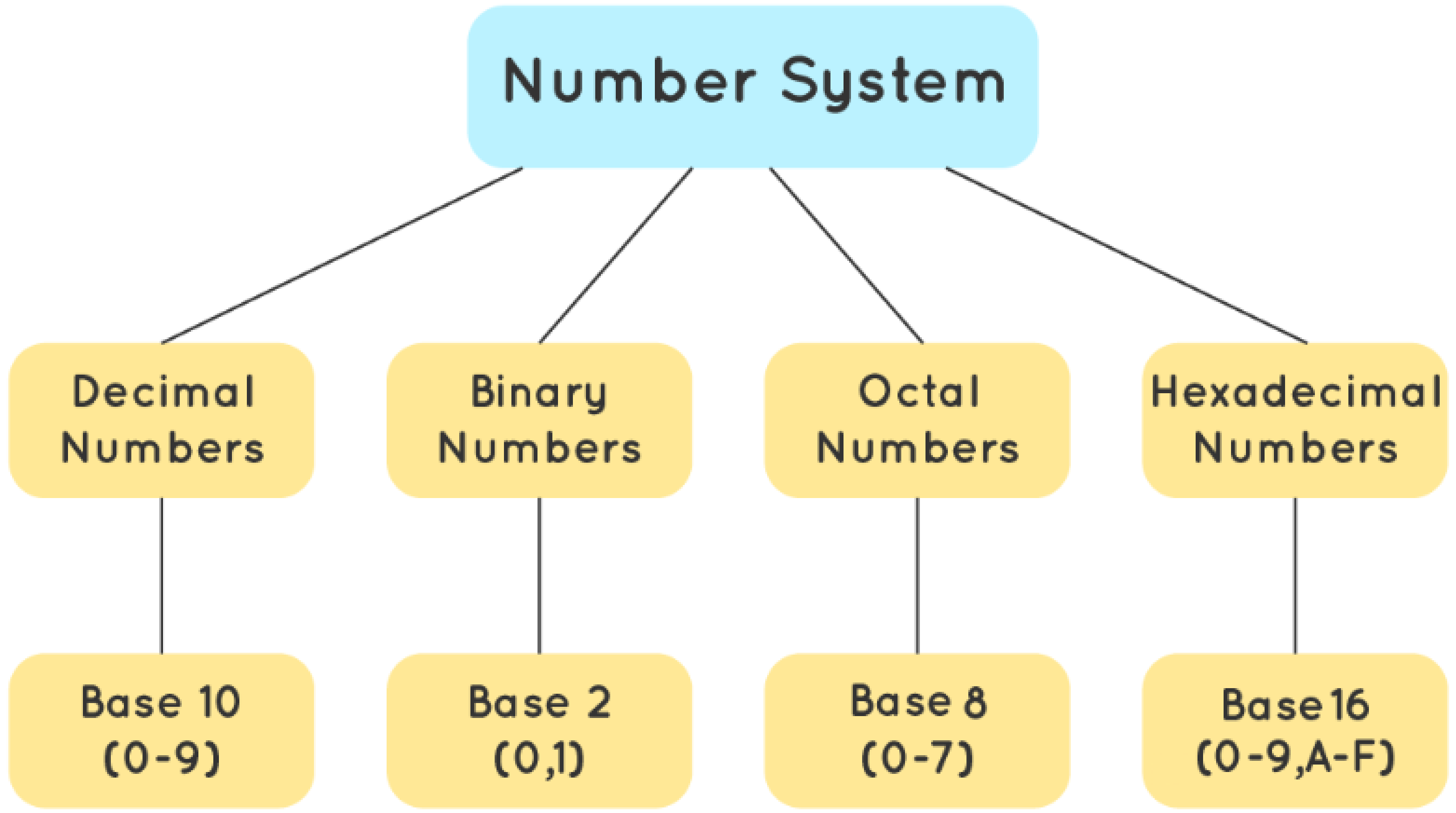
100	10	1	- Places
1	2	3	

$$123 = 1 \times 100 + 2 \times 10 + 3 \times 1$$

Decimal Number representation

- Examples are:
- 762

Types of Number System



Number System Cont..

From Decimal to Binary

- Examples are:
- $(15)_{10}$ to $(\ )_2$
- $(17.35)_{10}$ to $(\ )_2$

From Binary to Decimal

- Examples are:
- $(1111)_2$ to $()_{10}$
- $(10001.1011)_2$ to $(\ )_{10}$

Number System Cont..: Try your own

From Decimal to Binary

- (256)10 to ( )2
- (198.29)10 to ( )2

From Binary to Decimal

- (100000000)2 to ()10
- (1011.011)2 to ( )10

Number System Cont..

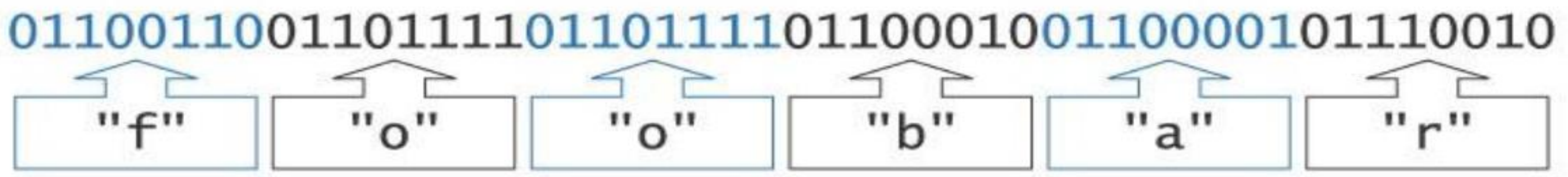
01001000 01001001

72 73

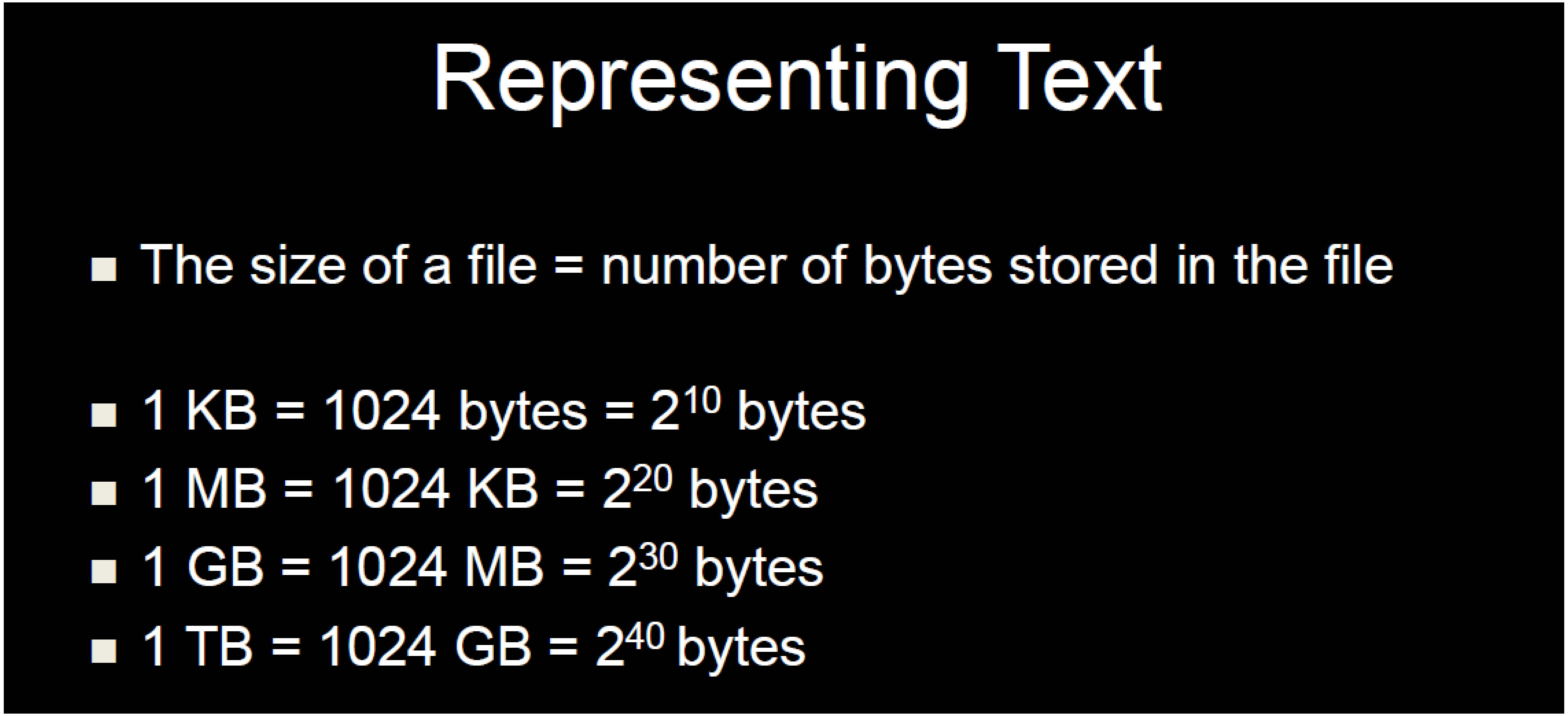
H I

Number System Cont..

Representing Text



Number System Cont..



Representing Text

- The size of a file = number of bytes stored in the file
- 1 KB = 1024 bytes = 2^{10} bytes
- 1 MB = 1024 KB = 2^{20} bytes
- 1 GB = 1024 MB = 2^{30} bytes
- 1 TB = 1024 GB = 2^{40} bytes

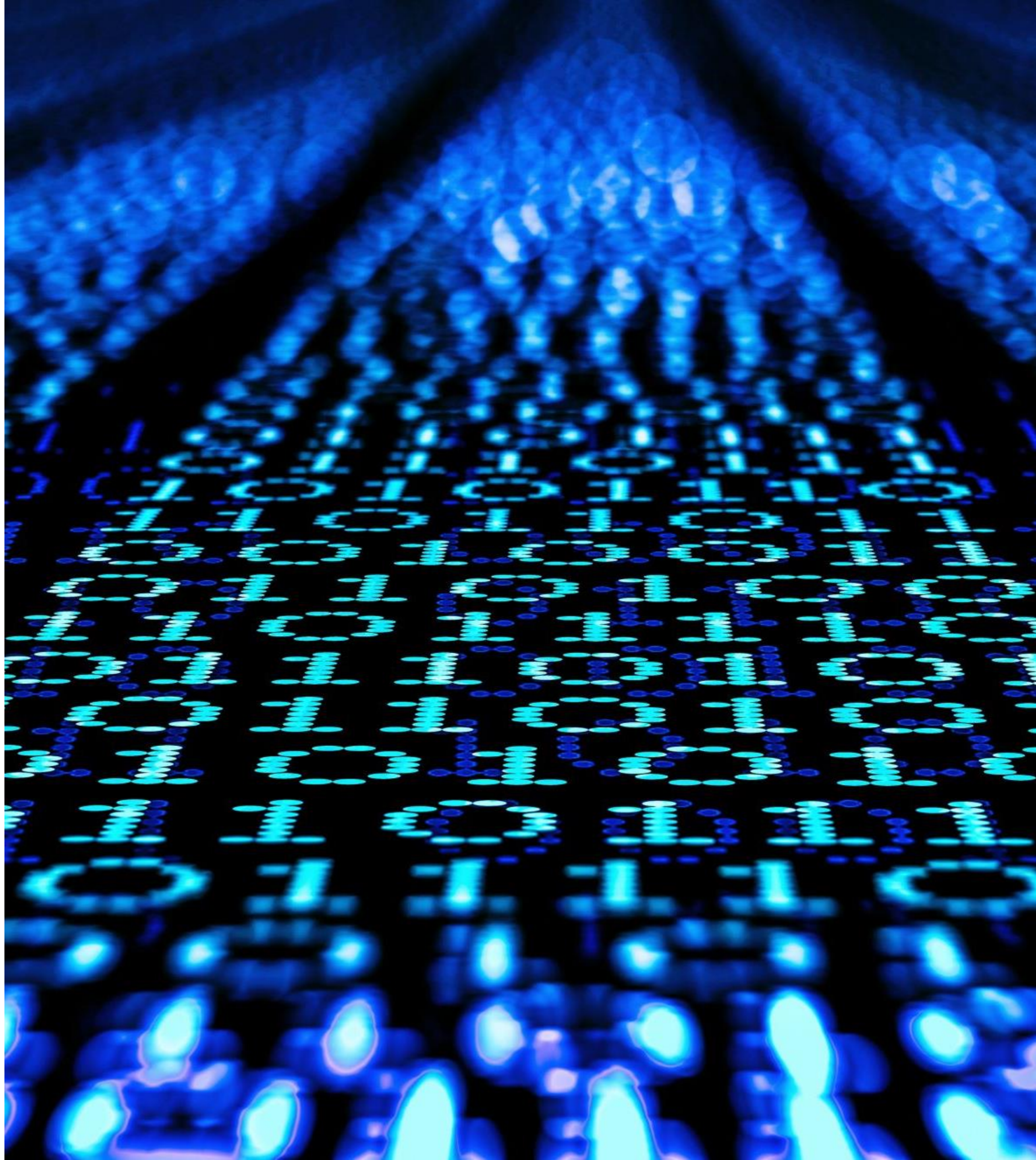
Number system conversion code

- printf("Enter value of a in decimal format:");
- scanf("%d", &a);

- printf("Enter value of b in octal format: ");
- scanf("%i", &b);

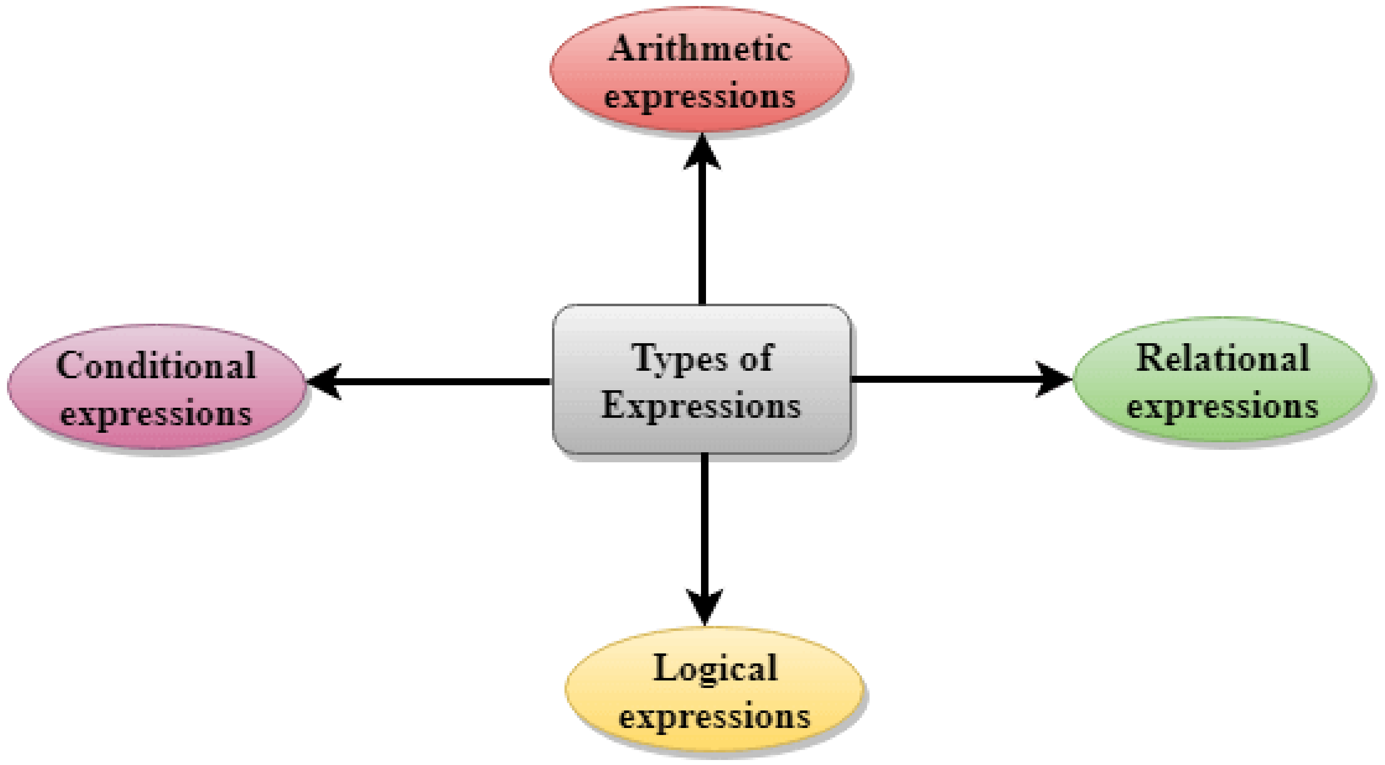
- printf("Enter value of c in hexadecimal format: ");
- scanf("%i", &c);

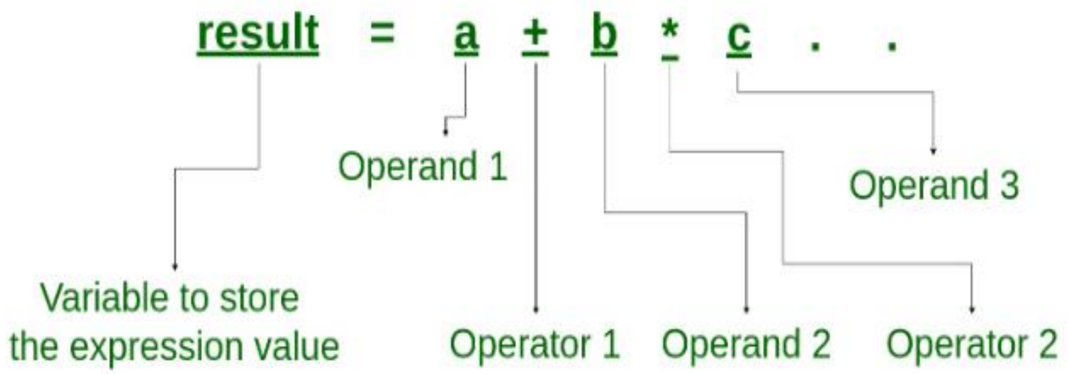
- printf("a = %i, b = %i, c = %i", a, b, c);

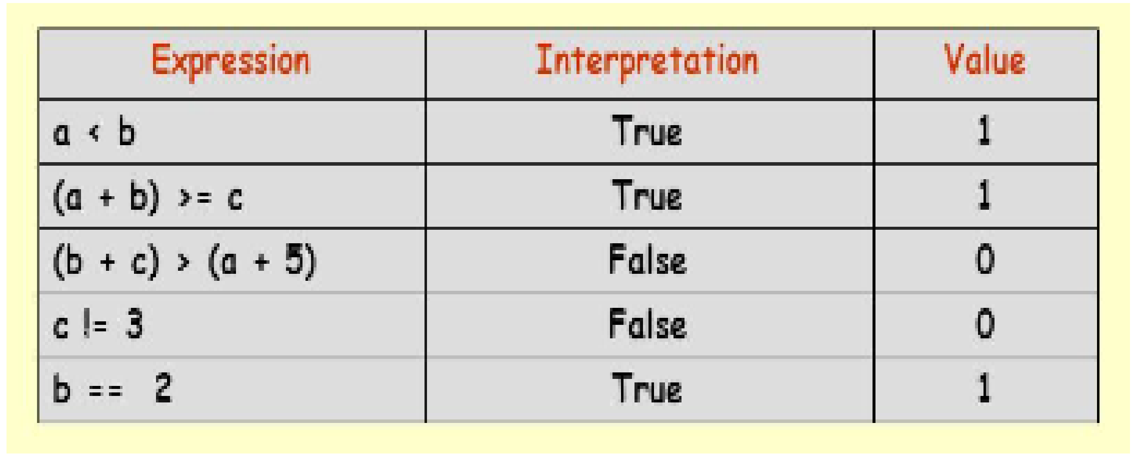


Expressions

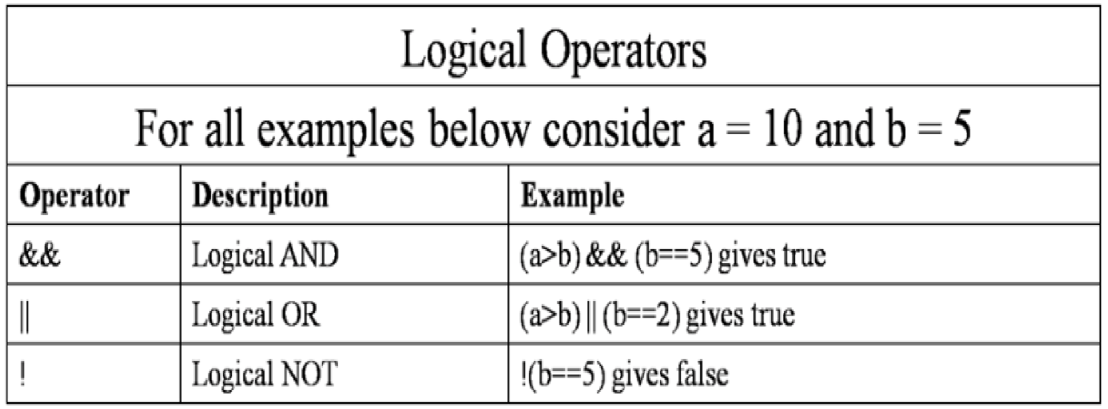
An expression is a combination of operators, constants, variables and functions







Expression	Interpretation	Value
a < b	True	1
(a + b) >= c	True	1
(b + c) > (a + 5)	False	0
c != 3	False	0
b == 2	True	1



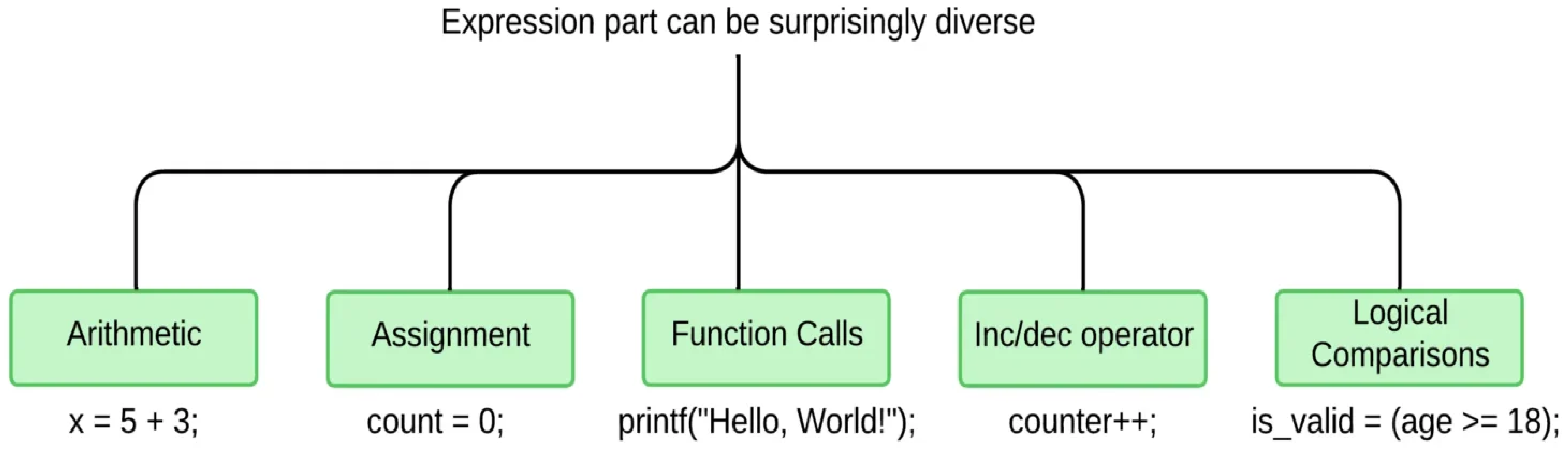
Logical Operators

For all examples below consider a = 10 and b = 5

Operator	Description	Example
&&	Logical AND	(a>b) && (b==5) gives true
||	Logical OR	(a>b) || (b==2) gives true
!	Logical NOT	!(b==5) gives false

```
int a = 10, b = 5, c;

c = (a>b) ? 20 : 12;
```



```
x = 5 + 3;
count = 0;
printf("Hello, World!");
counter++;
is_valid = (age >= 18);
```

Statements

- Instructions are written in the form of statements.
- An executable part of program to carry out some actions

What are Operators?

Operators are symbols that tell the compiler to perform specific mathematical, logical, or relational operations.

They act on variables and values (operands).

Types of Operator

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Unary Operators
- Ternary (Conditional) Operator
- Miscellaneous Operators

Arithmetic Operators

- +: Addition
- -: Subtraction
- *: Multiplication
- /: Division
- %: Modulus

```
int main()
{
    int a = 10, b = 4, res;
    printf("a is %d and b is %d\n", a, b);
    res = a + b; // addition
    printf("a + b is %d\n", res);
    res = a - b; // subtraction
    printf("a - b is %d\n", res);
    res = a * b; // multiplication
    printf("a * b is %d\n", res);
    res = a / b; // division
    printf("a / b is %d\n", res);
    res = a % b; // modulus
    printf("a %% b is %d\n", res);
    return 0;
}
```

Relational Operators

==: Equal to

!=: Not equal to

>: Greater than

<: Less than

>=: Greater than or equal to

<=: Less than or equal to

```
int main()
{
    int a, b;
    printf("Enter the value of a:");
    scanf("%d",&a);
    printf("Enter the value of b:");
    scanf("%d",&b);
    if (a > b)
        printf("a is greater than b\n");
    if (a >= b)
        printf("a is greater than or equal to b\n");
    if (a < b)
        printf("a is lesser than b\n");
    if (a <= b)
        printf("a is lesser than or equal to b\n");
    if (a == b)
        printf("a is equal to b\n");
    if (a != b)
        printf("a is not equal to b\n");
    return 0;
}
```

Logical Operators

- &&: Logical AND
- ||: Logical OR
- !: Logical NOT

```
#include <stdio.h>
int main()
{
    int a = 10, b = 20;
    if (a > 0 && b > 0 || a != b) {
        printf("Both values are greater than 0 or they are not equal \n");
    }
    else {
        printf("Both values are less than 0 or equal \n");
    }
    return 0;
}
```

Bitwise Operators

- &: Bitwise AND
- |: Bitwise OR
- ^: Bitwise XOR
- ~: Bitwise NOT
- <<: Left shift
- >>: Right shift

```
#include <stdio.h>
int main()
{
    short int a = 5, b = 9;
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a & b);
    printf("a|b = %d\n", a | b);
    printf("a^b = %d\n", a ^ b);
    printf("~a = %d\n", a = ~a);
    printf("b<<1 = %d\n", b << 1);
    printf("b>>1 = %d\n", b >> 1);
    return 0;
}
```

Assignment Operators

- =: Assign
- +=: Add and assign
- -=: Subtract and assign
- *=: Multiply and assign
- /=: Divide and assign
- %=: Modulus and assign

```
int main()
{  int a;
   printf("Enter the value of a");
   scanf("%d", &a);
   a += 10;
   printf("Value of a is %d\n", a);
   a -= 10;
   printf("Value of a is %d\n", a);
   a *= 10;
   printf("Value of a is %d\n", a);
   a /= 10;
   printf("Value of a is %d\n", a);
   a %= 10;
   printf("Value of a is %d\n", a);
   return 0;
}
```

Unary Operators

- ++: Increment
- --: Decrement
- +: Unary plus
- -: Unary minus
- &: address of a variable
- sizeof: size of its operand in byte

```
int main()
{
    int a = 5;
    int b = 5;
    printf("Positive Integer = %d\n", a);
    printf("Negative Integer = %d\n", -a);
    printf("Pre-Incrementing a = %d\n", ++a);
    printf("Post-Incrementing b = %d\n", b++);
    printf("Pre-Decrementing a = %d\n", --a);
    printf("Post-Decrementing b = %d\n", b--);
    if (!(a > b))
        printf("b is greater than a\n");
    else
        printf("a is greater than b\n");
    printf("Address of a = %p\n", &a);
    printf("Size of int: %d Byte\n", sizeof(short int));
    printf("Size of int: %d Byte \n", sizeof(int));
    printf("Size of int: %d Byte \n", sizeof(float));
    printf("Size of int: %d Byte \n", sizeof(char));
    printf("Size of int: %d Byte \n", sizeof(double));
    return 0; }
```

Ternary (Conditional) Operator

- **Syntax**: condition ? expression_if_true : expression_if_false
- Acts as a shorthand for **if-else**

Example:

```
#include <stdio.h>
int main()
{
    int m = 5, n = 4;
    (m > n) ? printf("m is greater than n that is %d > %d", m, n) : printf("n is greater than m that is %d > %d", n, m);
    return 0;
}
```

Miscellaneous Operators

- *: Pointer dereference
- ,: Comma operator
- ->: Structure pointer

Operands vs. Operators

X+y: X and y are operand; + is operator

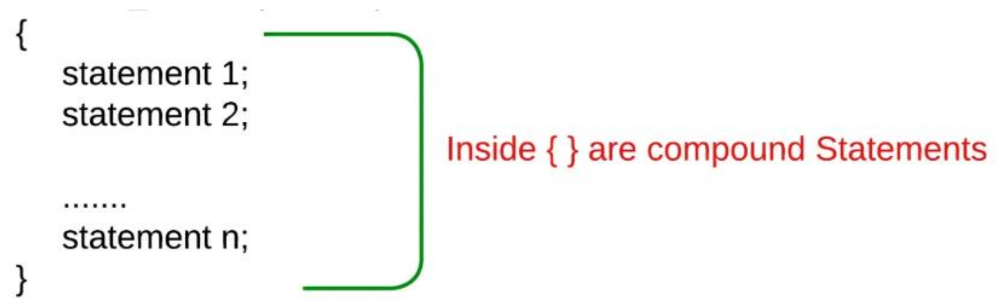
Unery operator: 1 operand	++
Binary operator: 2 operands	+
Ternary operator: 3 operands	?:

Compound Statement

Compound statement in C is also called a block.

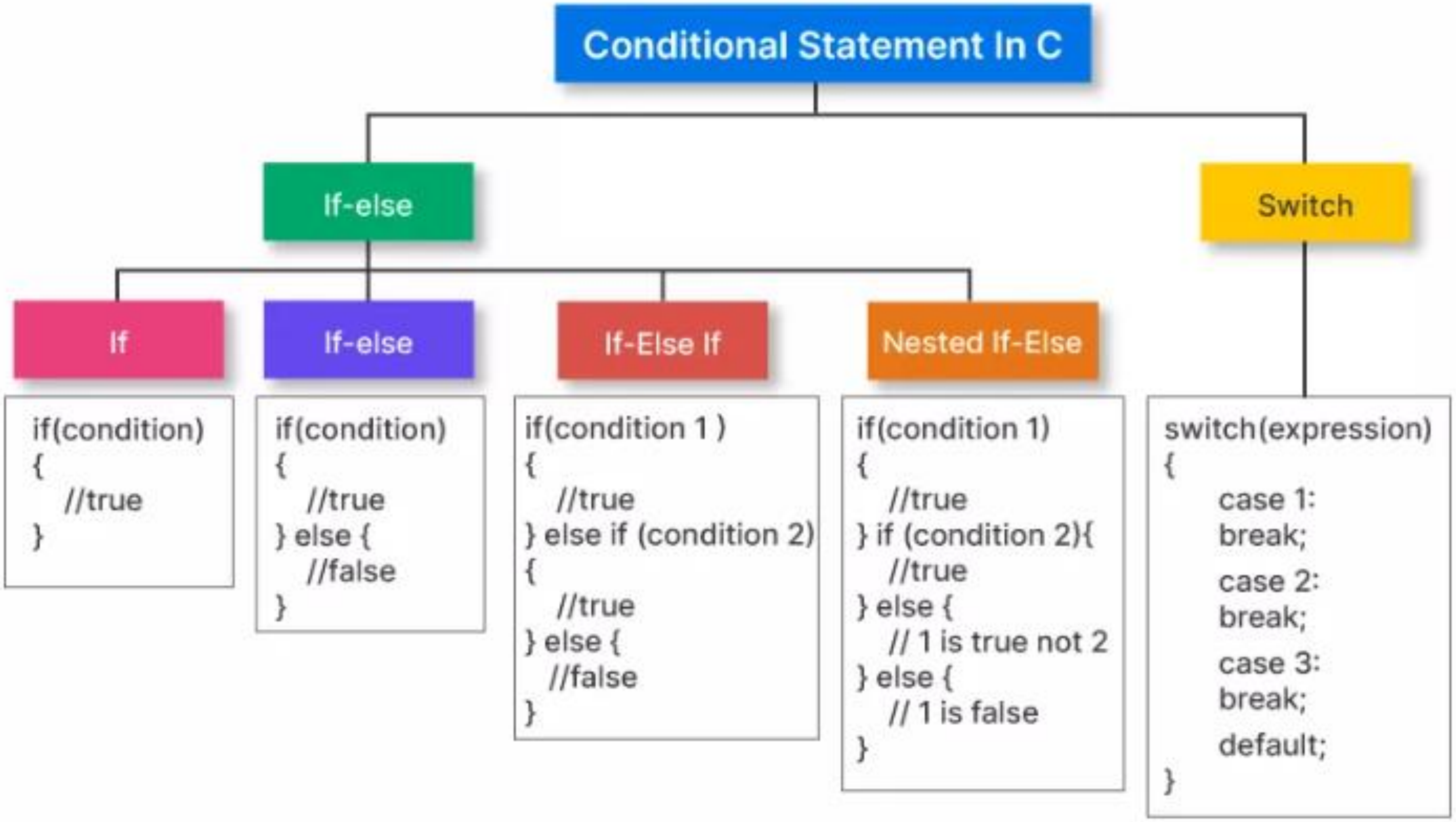
A block is a set of statements that are enclosed within the { }.

This grouping allows you to treat multiple statements as a single unit.



```
{
    statement 1;
    statement 2;

    .......
    statement n;
}
```



```
if(condition)
{
   //true
}
```

```
if(condition)
{
   //true
} else {
   //false
}
```

```
if(condition 1 )
{
   //true
} else if (condition 2)
{
   //true
} else {
  //false
}
```

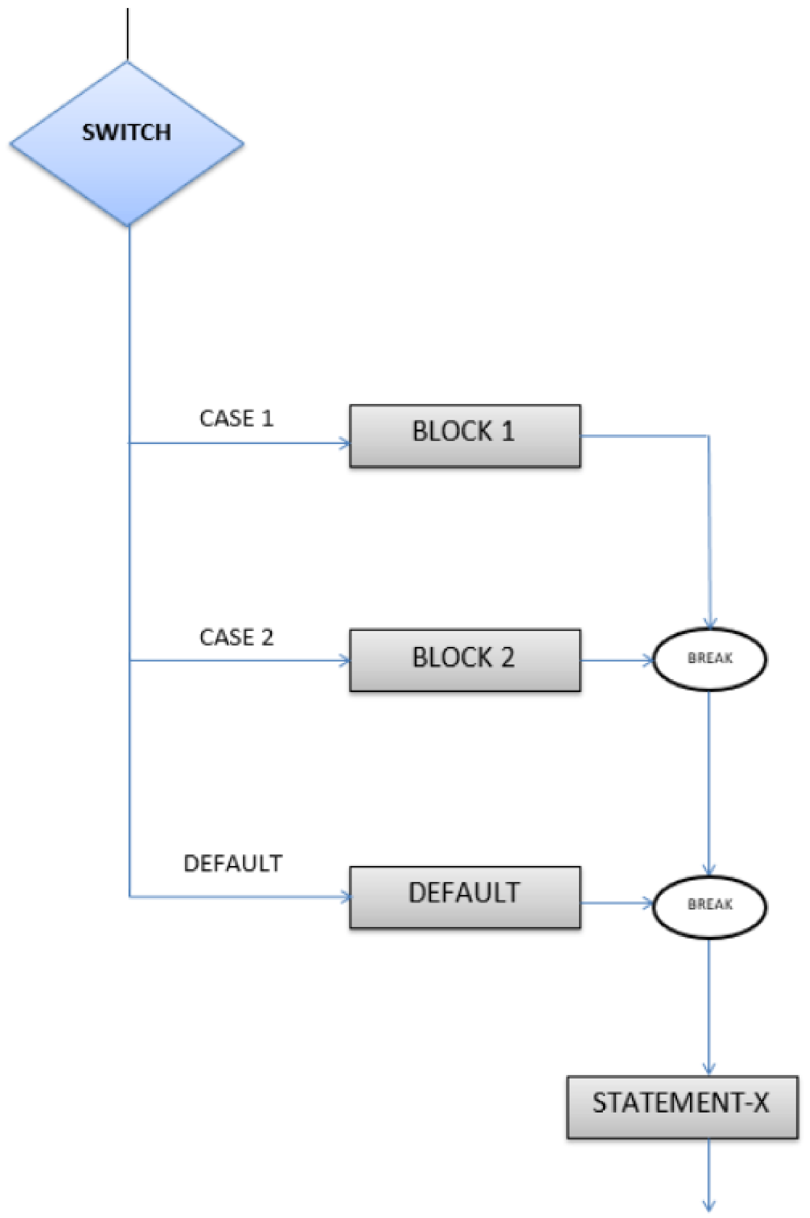
```
if(condition 1)
{
   //true
} if (condition 2){
   //true
} else {
   // 1 is true not 2
} else {
   // 1 is false
}
```

```
switch(expression)
{
      case 1:
      break;
      case 2:
      break;
      case 3:
      break;
      default;
}
```

Conditional Statement

```
#include<stdio.h>
int main ()
{
 int num1, num2;
 num1=12;
 num2=13;
 if (num1 == num2){
   printf("both are equal");}
 else if (num1 > num2){
   printf("%d is greater", num1);}
 else{
   printf("%d is greater", num2);}
 return 0;
}
```

Switch case


SWITCH
CASE 1
BLOCK 1
CASE 2
BLOCK 2
BREAK
DEFAULT
DEFAULT
BREAK
STATEMENT-X

Switch case

```
switch( expression )
{
        case value-1:
                        Block-1;
                        Break;
        case value-2:
                        Block-2;
                        Break;
        case value-n:
                        Block-n;
                        Break;
        default:
                        Block-1;
                        Break;
}
Statement-x;
```

Switch case Example

```
#include <stdio.h>
int main() {
   int num1, num2, result;
   char operator;

   // Asking user for input
   printf("Enter first number: ");
   scanf("%d", &num1);
   printf("Enter an operator (+, -, *, /): ");
   scanf(" %c", &operator);  // Notice the space before
%c to consume any leftover newline character
   printf("Enter second number: ");
   scanf("%d", &num2);

   // Switch case to perform the chosen operation
   switch (operator) {
      case '+':
         result = num1 + num2;
         printf("Result: %d + %d = %d\n", num1, num2,
result);
         break;
case '-':
         result = num1 - num2;
         printf("Result: %d - %d = %d\n", num1, num2, result);
         break;
      case '*':
         result = num1 * num2;
         printf("Result: %d * %d = %d\n", num1, num2, result);
         break;
      case '/':
         if (num2 != 0) {
            result = num1 / num2;
            printf("Result: %d / %d = %d\n", num1, num2, result);
         } else {
            printf("Error: Division by zero is not allowed.\n");
         }
         break;
      default:
         printf("Error: Invalid operator.\n");
         break;
   }
return 0; }
```

Upcoming Slides

- Precedence of operator
- Conditional statement and Loop