Fundamentals of Programming
CCS1063/CSE1062
Lecture 4

Professor Noel Fernando

A simple program in C

- Consider the following simple C program

- ```
  #include <stdio.h>

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

- h" permits the programmer to carry out input/output operations .
- Standard Input Output Header, or "stdio.h," contains features for handling file and console I/O, including the "printf()" function for formatted output
- and the "scanf()" function for formatted input.
  ```

The enhancement of the simple program

- #include <stdio.h>

```
int main() {
  printf(“Welcome to JSP!\n");
  printf("I am studying  C.");
  return 0;
}
```

What is \n exactly?

The newline character(\n)

```
Welcome to JSP!
I am studying  C
```

Note : return 0: A return **0** means that the program will execute successfully and did what it was intended to do.

Few valid escape sequences

Escape Sequence	Description	Example	Output

\t	Creates a horizontal tab	#include <stdio.h>
		
		int main() {
		printf("Welome to SJP University!\t");
		printf("I am Studying C.");
		return 0;
		}
			Welcome to SJP University I am studying C

\\	Inserts a backslash character (\)	#include <stdio.h>
		
		int main() {
		printf("Welcome to SJP University!\\");
		printf("I am studying C.");
		return 0;
		}
			Welcome to SJP University!\I am studying C.

\"	Inserts a double quote character	#include <stdio.h>
		
		int main() {
		printf("They call as \"Sri Jayewardenepura University\".");
		return 0;
		}
			They call as "Sri Jayewardenepura University".

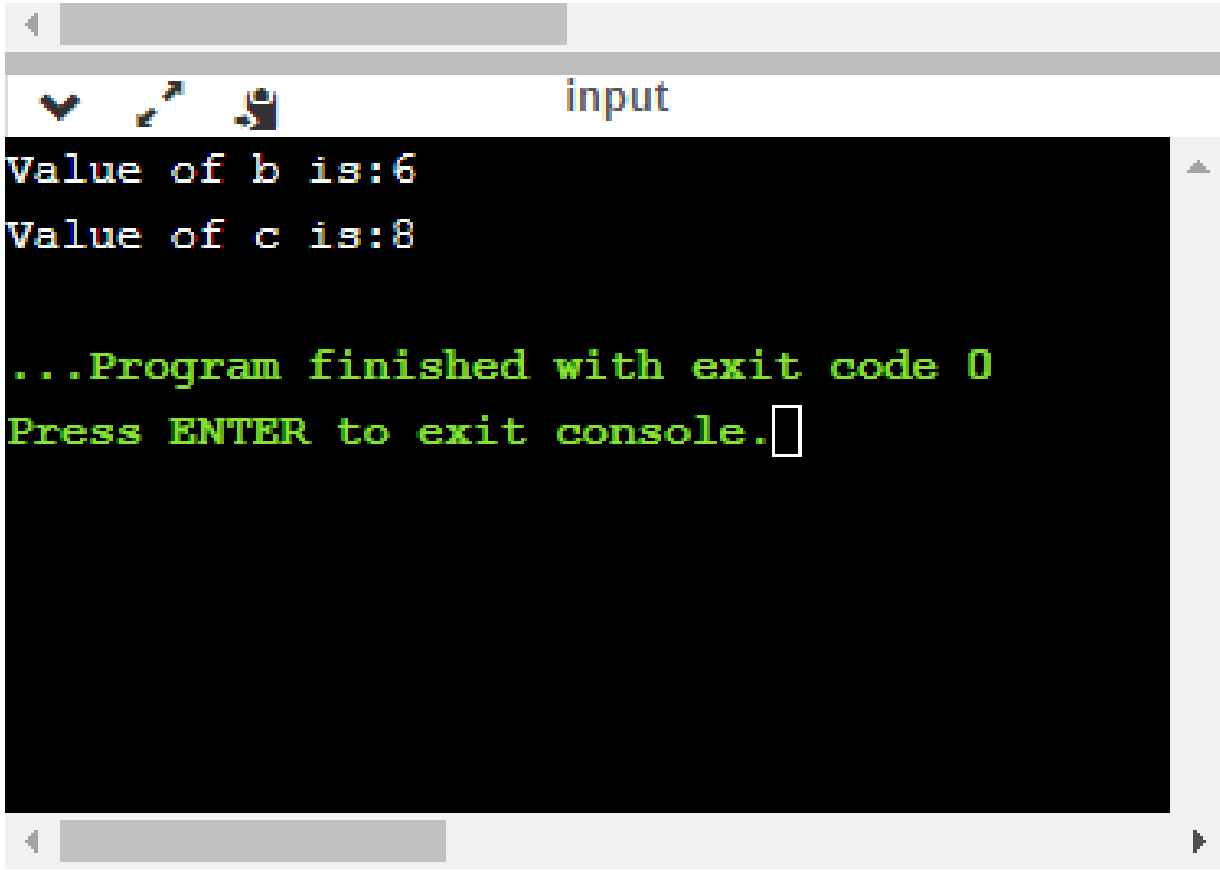
Format Specifiers in C

Format specifier	Description
%d or %i	It is used to print the signed integer value
%u	It is used to print the unsigned integer value
%o	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value
%x	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value
%e/%E	It is used for scientific notation. It is also known as Mantissa or Exponent. (e.g. i.e. $0.12*10^2$ Here the 0.12 is the mantissa and the 10^2 is the exponent.)
%g	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.
%p	It is used to print the address in a hexadecimal form.
%c	It is used to print the unsigned character.
%s	It is used to print the strings.
%f	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'

Let's understand the format specifiers in detail through an example. (Example 1)

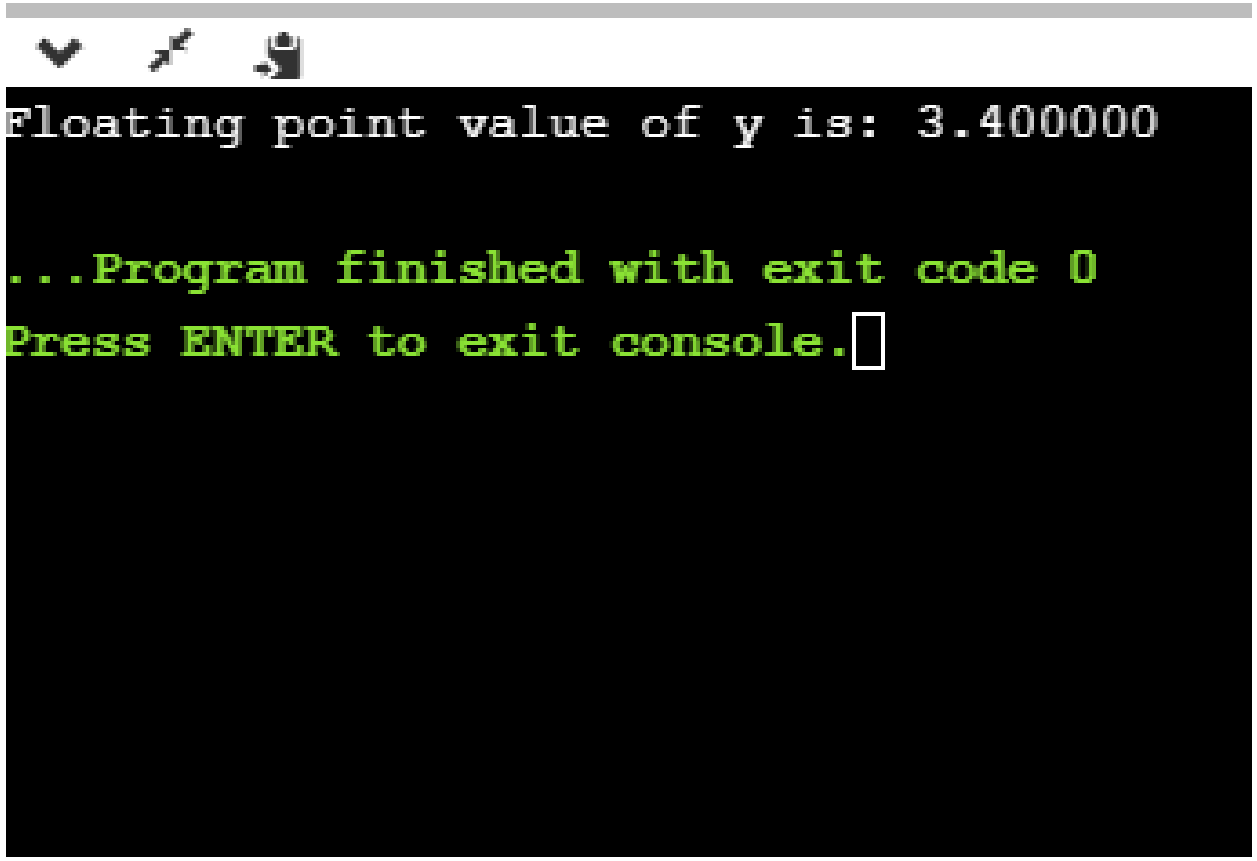
```
int main()
{
  int b=6;
  int c=8;
  printf("Value of b is:%d", b);
  printf("\nValue of c is:%d",c);

    return 0;
}
```



Let's understand the format specifiers in detail through an example. (example 2)

```
int main()
{
  float y=3.4;
  printf("Floating point value of y is
: %f", y);
  return 0;
}
```

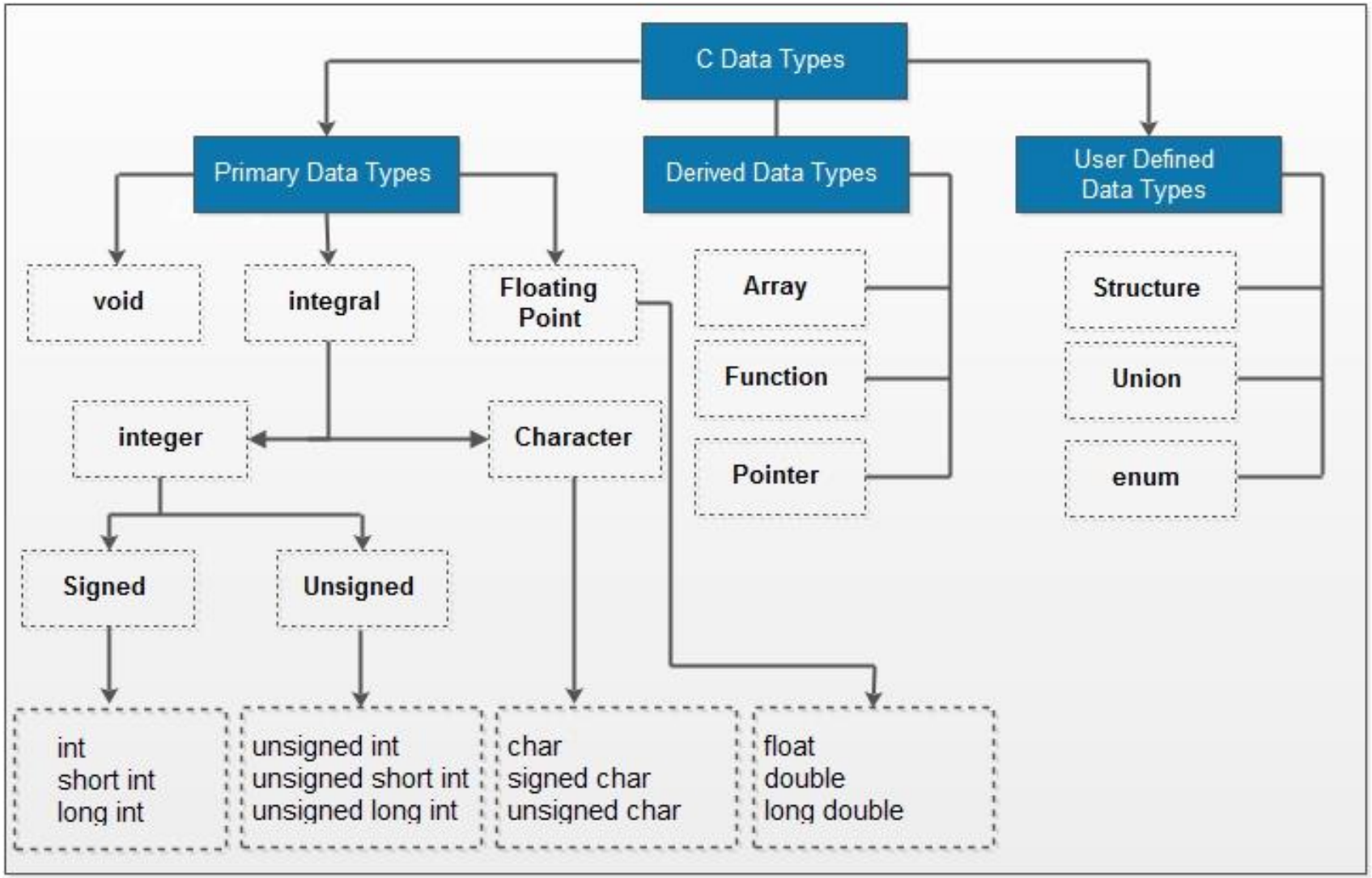


Comments in C language

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

- Example 1
- // This is a comment
  printf("Hello World!");
- Example 2
- printf("Hello World!"); // This is a comment
- Example 3
- /* The code below will print the words Hello World!
  to the screen, and it is amazing */
  printf("Hello World!");

C language data types



Basic data types

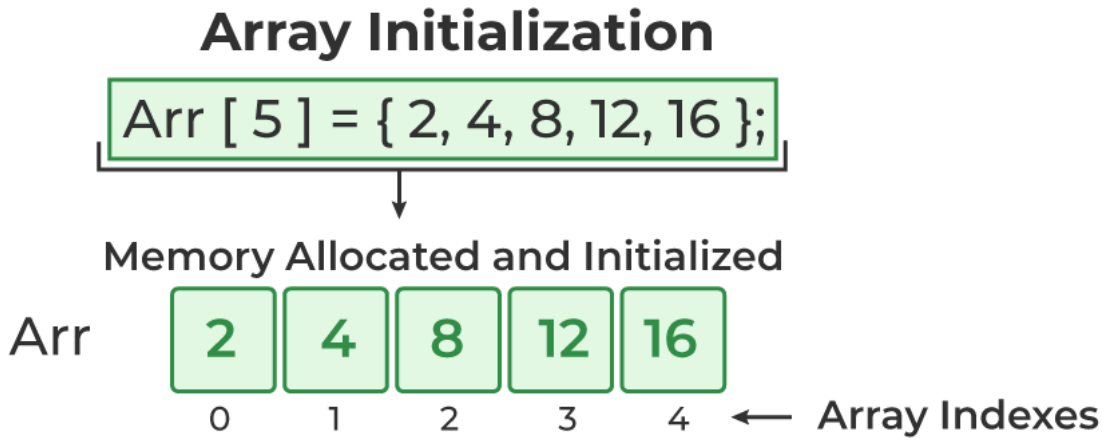
Data Types	Description
int	The most natural size of integer for the machine. An is a whole that can be positive, negative, or zero. Examples of integers are: -5, 1, 5, 8, 97, and 3,043
float	.A single-precision floating point value. A floating point number, is a positive or negative whole number with a decimal point. For example, 5.5, 0.25, and -103.342 are all floating point numbers, while 91, and 0 are not.
double	A double-precision floating point value. The **double in C** is a data type that is used to store **high-precision floating-point** data or numbers (up to 15 to 17 digits). It is used to store large values of decimal numbers
void	Represents the absence of type
char	Typically a single octet(one byte ***char*** is the data type that represents a single character. The char values are encoded with one byte code, by the ASCII table

Basic data Types

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

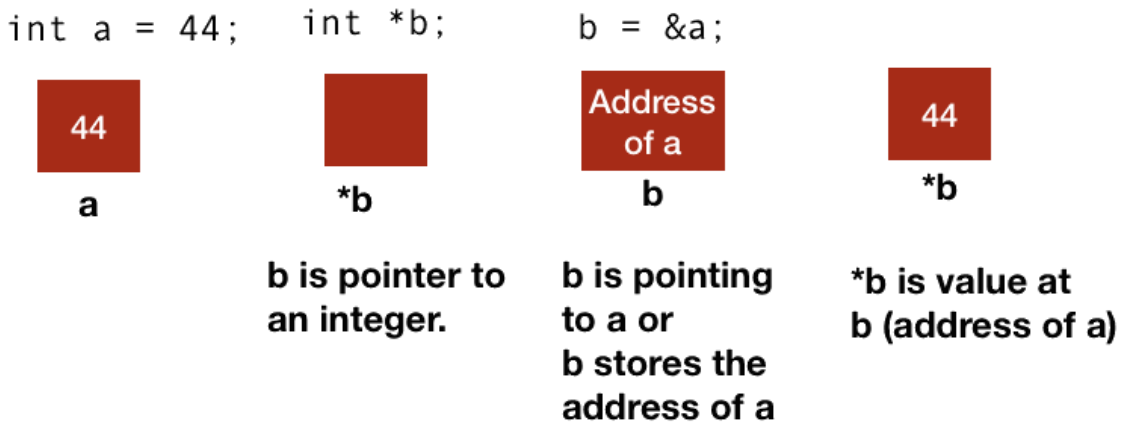
Derived data Types

- The C language supports a few derived data types:
- **Arrays –** The *array* basically refers to a sequence (ordered sequence) of a finite number of data items from the same data type sharing one common name.
- We can form arrays by collecting various primitive data types such as int, float, char, etc.
- We create a collection of these data types and store them in the contiguous memory location.



Derived data Types

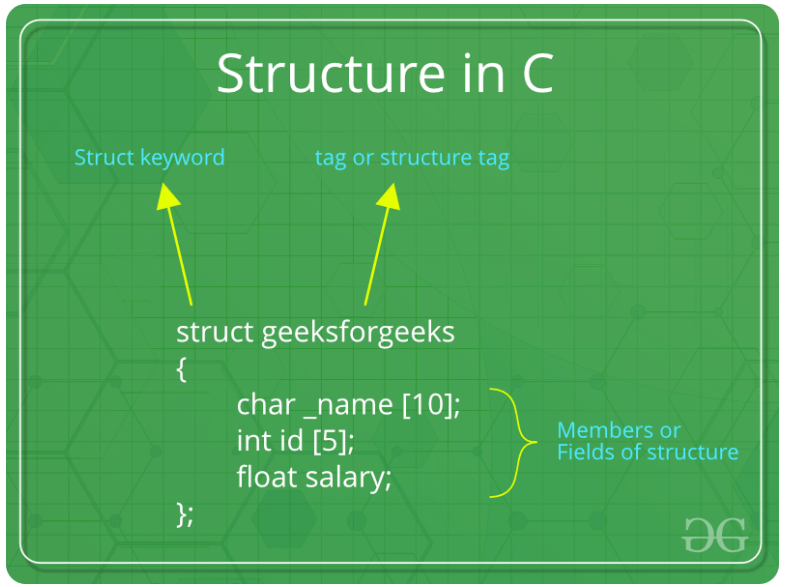
- Pointers: The *Pointers in C* language refer to some special form of variables that one can use for holding other variables' addresses

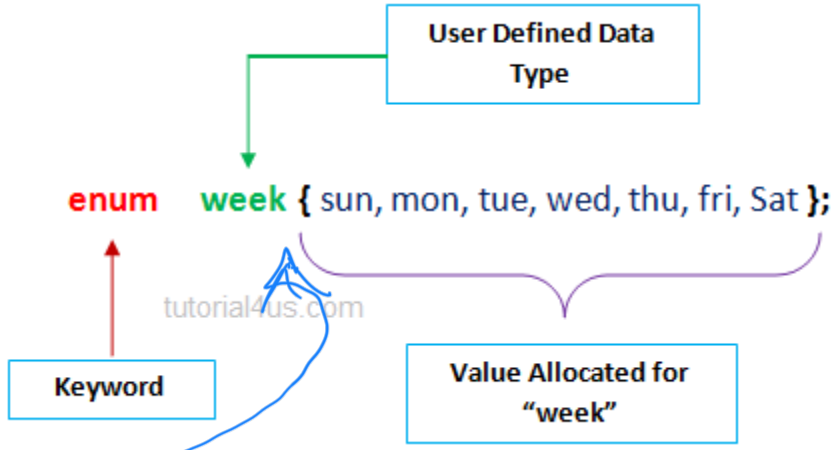


Derived data Types

create an object (oop)

- **Structures** – A collection of various different types of data type items that get stored in a contagious type of memory allocation is known as ***structure*** *in C*.
- Enumeration or Enum in C is a special kind of data type defined by the user. It consists of constant integrals or integers that are given names by a user





something like an Object, create an own variable. When we made this variable, then in the main method,

enum week x = sun;
printf("%d", x); → 0 (default)
sun = 1
mon = 2 ....

enum week {
sun = 10
= = =
}

main () {
enum week x = sun;
printf("%d", x) → 10; ← sun = 10 ← we assigned.
mon = 11 (default)

1st rule of Programming:

If it works.... don't touch it!..

Keywords

- Keywords are predefined, reserved words used in programming that have special meanings to the compiler.
- Keywords are part of the syntax and they cannot be used as an identifier.
- example:

```
int money;
```

Here, int is a keyword that indicates 'money' is a variable of type integer.

Keywords

- As C is a case sensitive language, all its 32 keywords must be written in lowercase.

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

Identifiers

- Identifier refers to name given to entities such as variables, functions, structures etc.
- Identifier must be unique. They are created to give unique name to a entity to identify it during the execution of the program.
- For example:

```
int money;
double accountBalance;
```

  - Here, money and accountBalance are identifiers.

- Identifier names must be different from keywords. You cannot use int as an identifier because int is a keyword.

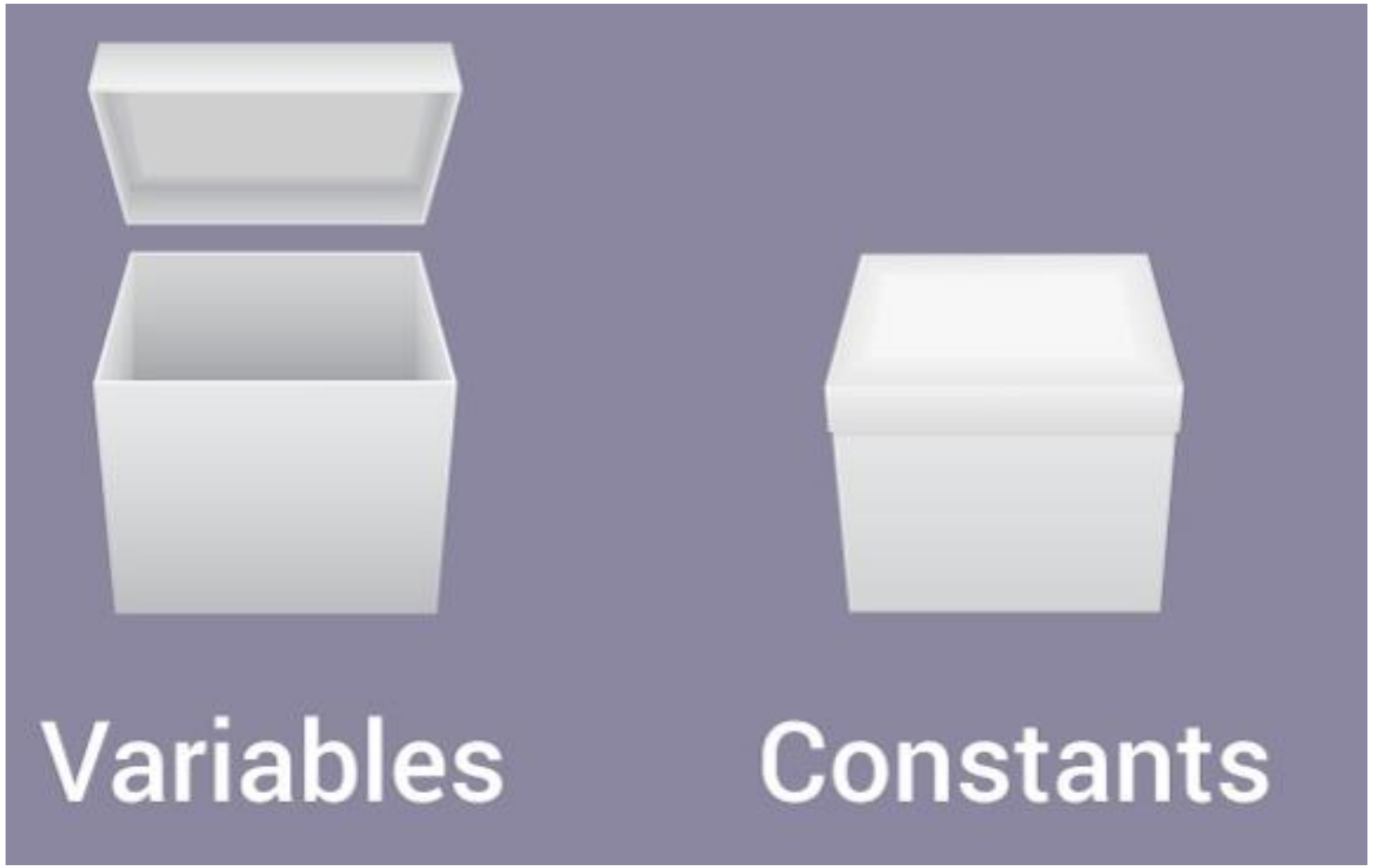
Rules for define an Identifier

1. **A valid identifier can have letters (both uppercase and lowercase letters), digits and underscore only.**

2. **The first letter of an identifier should be either a letter or an underscore.**
   - However, it is discouraged to start an identifier name with an underscore.

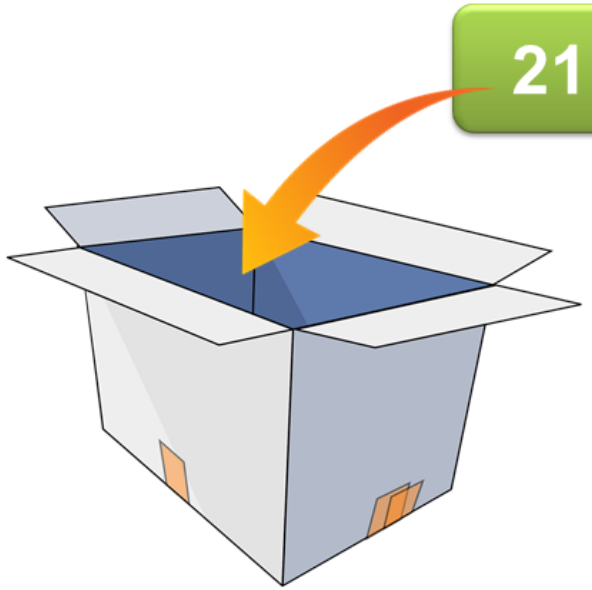
3. **There is no rule on length of an identifier.**
   - However, only the first 31 characters of a identifier are checked by the compiler.

Variables & Constants



Variables

- In programming, a variable is a container (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier).
- A variable is something that may change in value.
- Variable names are just the symbolic representation of a memory location.
- Example : page number, the air temperature each day, or the exam marks given to a class of school children.



We can think that variable is one type of Container where we can store some element

```
int age = 21 ;
```

```
int = which type of element we
                can store
age = name of the container box

21 = type of element
```

Rules for naming a variable in C

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore.
   - However, it is discouraged to start an identifier name with an underscore.
3. There is no rule on length of an identifier.
   - However, only the first 31 characters of a identifier are checked by the compiler.
4. C is a strongly typed language. What this means it that, **the type of a variable cannot be changed.**

Sample programming C -Variables and Math

```
/*
  Variables and Calculations.c

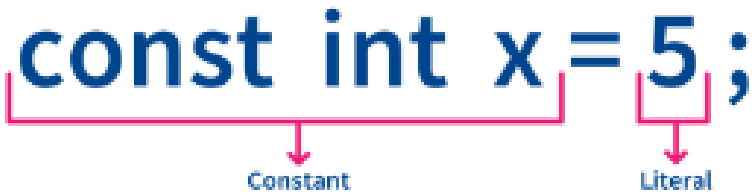
  Add two integer values together and display the result.
*/

#include "simpletools.h"                      // Include simpletools

int main()                                    // main function
{
  int a = 25;                                 // Initialize a variable to 25
  int b = 17;                                 // Initialize b variable to 17
  int c = a + b;                              // Initialize c variable to a + b
  print("c = %d ", c);                        // Display decimal value of c
}
```

Constants/Literals

- **A constant is a value or an identifier whose value cannot be altered in a program**.
- For example; `const double PI = 3.14;`
  - Here, PI is a constant.
  - Basically what it means is that, PI and 3.14 is same for this program.



Sample program with use of constants

main.c

```c
#include <stdio.h>

int main(){

    const int RADIUS = 7;
    const double PI = 3.14;
    float area;
    area = PI  * RADIUS * RADIUS;
    printf("Area : %5.2f",area);
    return 0;
}
```

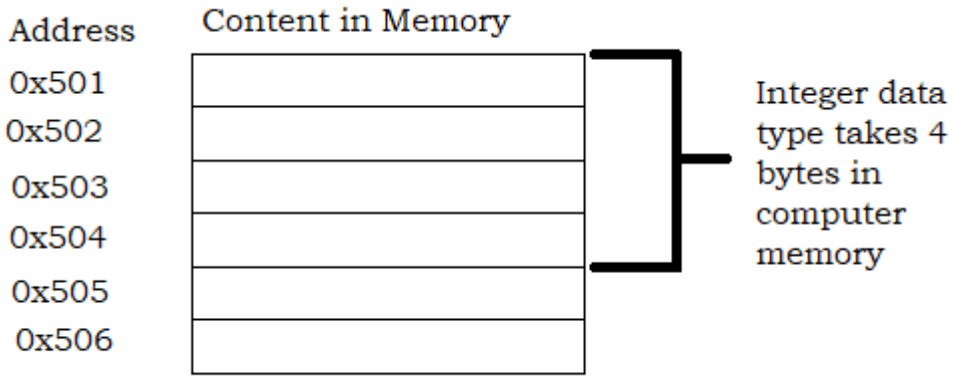
Answer :

Area: 153.86

if your number is 3.14159 and you're using %5.2f, it'll reserve a total of 5 spaces in which 2 are for precision digits but since there's just one digit before the decimal part so it'll print like, 03.14(5 places with 2 precision places).

sizeof()

- A computer's memory is a collection of byte-addressable chunks.



- sizeof() is a built-in function that is used to calculate the size (in bytes)that a data type occupies in the computer's memory.
  - Suppose that a variable x is of type integer and takes four bytes of the computer's memory, then sizeof(x) would return four.

sizeof()

- This function is a unary operator (i.e., it takes in one argument).
- This argument can be a;
  - Data type: The data type can be primitive (e.g., int, char, float) or user-defined (e.g., struct).
  - Expression

- IMORTANT NOTICE:
  - The result of the sizeof() function is machine-dependent since the sizes of data types in C varies from system to system.
  - a unary operator is a single operator that operates on a single operand to produce a new value.
  - Unary operators can perform operations such as negation, increment, decrement, and others.

sizeof operator in C

- **Usage of sizeof() operator**
- *sizeof()* operator is used in different ways according to the operand type.
- **When the operand is a Data Type:** When *sizeof()* is used with the data types such as int, float, char... etc
- it simply returns the amount of memory allocated to that data types

- Sample program – size()

```c
// C Program To demonstrate
// sizeof operator
#include <stdio.h>
int main()
{
    printf("%lu\n", sizeof(char));
    printf("%lu\n", sizeof(int));
    printf("%lu\n", sizeof(float));
    printf("%lu", sizeof(double));
    return 0;
}
```

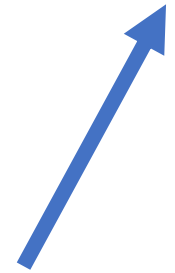
Output

```
1
4
4
8
```

sizeof()

```c
#include<stdio.h>

int main() {
    int x = 20;
    char y = 'a';
    //Using variable names as input
    printf("The size of int is: %d\n", sizeof(x));
    printf("The size of char is %d\n", sizeof(y));
    printf("The size of x + y is: %d\n",sizeof(x+y));
    //Using datatype as input
    printf("The size of float is: %d\n", sizeof(float));
    printf("The size of double is: %d\n", sizeof(double));
    return 0;
}
```

Note: The first case size of int is 4 and char is 1 respectively. the output of our program is 4 bytes



```
The size of int is: 4
The size of char is 1
The size of x + y is: 4
The size of float is: 4
The size of double is: 8
```

Size qualifiers

- Size qualifiers alters the size of a basic type.
- There are two size qualifiers, long and short.
- For example:

```
long double x;
```

```
double x;
long double lx;

printf("size of x: = %d\n", sizeof(x));
printf("size of lx: = %d", sizeof(lx));
```

- The size of double is 8 bytes.
- However, when long keyword is used, that variable becomes 16 bytes.

Size qualifiers

- Size qualifiers alters the size of a basic type.
- There are two size qualifiers, long and short.

- The size of int is 4 bytes.
- However, when short keyword is used, that variable becomes 2 bytes.

```
int x;
short sx;

printf("size of x: = %d\n", sizeof(x));
printf("size of sx: = %d", sizeof(sx));
```

Sign qualifiers

- Integers and floating point variables can hold both negative and positive values.
- However, if a variable needs to hold positive value only, unsigned data types are used.
- For example:

```
unsigned int x = 3;
```

- An int is signed by default, meaning it can represent both positive and negative values.
- An unsigned is an integer that can never be negative.

Operators

- C programming has various types of operators to perform tasks including arithmetic, conditional and bitwise operations.
- Operators in C programming are;
  - Arithmetic Operators
  - Increment & Decrement Operators
  - Assignment Operators
  - Relational Operators
  - Logical Operators
  - Conditional Operators
  - Bitwise Operators
  - Special Operators

Arithmetic Operators

- An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multifaction
/	Division
%	Remainder after division (modulo division)

Arithmetic Operators

```c
int main() {
    int a = 9, b = 4, c;
    c = a+b;
    printf("a+b = %d\n", c);
    c = a-b;
    printf("a-b = %d\n", c);
    c = a*b;
    printf("a*b = %d\n", c);
    c = a/b;
    printf("a/b = %d\n", c);
    c = a%b;
    printf("Remainder = %d\n", c);
}
```

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder = 1
```

Increment & Decrement Operators

- C programming has two increment and decrement operators to change the value of an operand (constant or variable) by 1.
  - Increment ++ increases the value by 1
  - Decrement -- decreases the value by 1
- These two operators are unary operators, meaning they only operate on a single operand.
- The operators ++ and -- can be used as prefix or postfix.

Increment & Decrement Operators

- **++ and -- operator as prefix and postfix**
- If you use the ++ operator as a prefix like:++var the value of var is incremented by 1; then it returns the value.
- If you use the ++ operator as a postfix like : var++, the original value of var is returned first; then var is incremented by 1.
- The -- operator works in a similar way to the ++ operator except -- decreases the value by 1.

Increment & Decrement Operators

```c
int main() {
    int a = 7;
    float b = 5.5;
    printf("++a = %d\n", ++a);
    printf("--b = %.2f\n", --b);
    printf("a++ = %d\n", a++);
    printf("b-- = %.2f\n", b--);
    printf("Final Values: a = %d, b = %.2f\n", a, b);
}
```

Increment & Decrement Operators

```c
int main() {
    int a = 7;
    float b = 5.5;
    printf("++a = %d\n", ++a);
    printf("--b = %.2f\n", --b);
    printf("a++ = %d\n", a++);
    printf("b-- = %.2f\n", b--);
    printf("Final Values: a = %d, b = %.2f\n", a, b);
}
```

```
++a = 8
--b = 4.50
a++ = 8
b-- = 4.50
Final Values: a = 9, b = 3.50
```

Assignment Operators

- An assignment operator is used for assigning a value to a variable.
  - The most common assignment operator is =

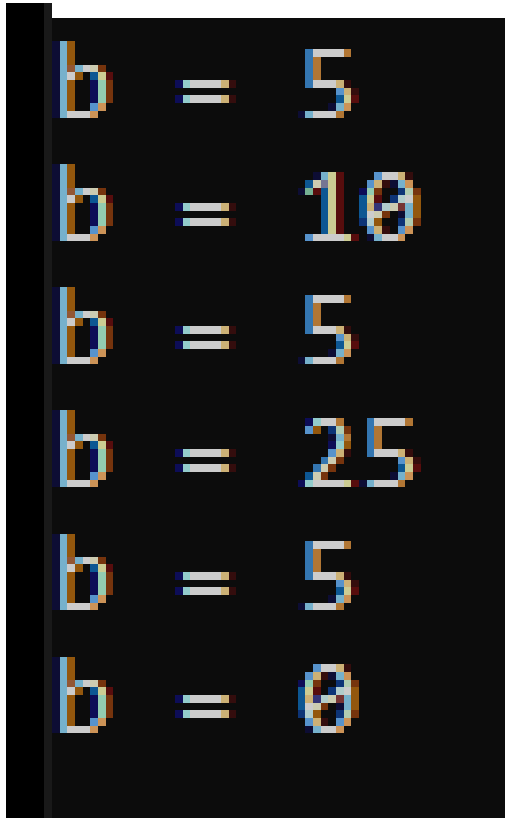
Operator	Example	Same as…
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Assignment Operators

```c
int main() {
    int a = 5, b;
    b = a;
    printf("b = %d\n", b);
    b += a;
    printf("b = %d\n", b);
    b -= a;
    printf("b = %d\n", b);
    b *= a;
    printf("b = %d\n", b);
    b /= a;
    printf("b = %d\n", b);
    b %= a;
    printf("b = %d\n", b);
}
```

Assignment Operators

```c
int main() {
    int a = 5, b;
    b = a;
    printf("b = %d\n", b);
    b += a;
    printf("b = %d\n", b);
    b -= a;
    printf("b = %d\n", b);
    b *= a;
    printf("b = %d\n", b);
    b /= a;
    printf("b = %d\n", b);
    b %= a;
    printf("b = %d\n", b);
}
```



Relational Operators

- A relational operator checks the relationship between two operands
- If the relation is true, it returns 1; if the relation is false, it returns value 0
- Relational operators are used in decision making and loops.

Operator	Meaning	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 returns 0

Relational Operators

```c
int main() {

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

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

    printf("%d > %d = %d\n", a, b, a > b);
    printf("%d > %d = %d\n", a, c, a > c);

    printf("%d < %d = %d\n", a, b, a < b);
    printf("%d < %d = %d\n", a, c, a < c);

    printf("%d != %d = %d\n", a, b, a != b);
    printf("%d != %d = %d\n", a, c, a != c);

    printf("%d >= %d = %d\n", a, b, a >= b);
    printf("%d >= %d = %d\n", a, c, a >= c);

    printf("%d <= %d = %d\n", a, b, a <= b);
    printf("%d <= %d = %d\n", a, c, a <= c);

}
```

Relational Operators

```c
int main() {

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

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

    printf("%d > %d = %d\n", a, b, a > b);
    printf("%d > %d = %d\n", a, c, a > c);

    printf("%d < %d = %d\n", a, b, a < b);
    printf("%d < %d = %d\n", a, c, a < c);

    printf("%d != %d = %d\n", a, b, a != b);
    printf("%d != %d = %d\n", a, c, a != c);

    printf("%d >= %d = %d\n", a, b, a >= b);
    printf("%d >= %d = %d\n", a, c, a >= c);

    printf("%d <= %d = %d\n", a, b, a <= b);
    printf("%d <= %d = %d\n", a, c, a <= c);

}
```

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

Logical Operators

expression

- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND, true only if all operands are true	If c = 5 & d = 2, then, expression ((c==5)&&(d==5)) equals to 0
||	Logical OR, true only if either operand is true	If c = 5 & d = 2, then, expression ((c==5)&&(d==5)) equals to 1
!	Logical NOT, true only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0

Logical Operators

```c
int main() {

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

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d\n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d\n", result);

    result = (a == b) || (c > b);
    printf("(a == b) || (c > b) equals to %d\n", result);

    result = (a != b) || (c > b);
    printf("(a != b) || (c > b) equals to %d\n", result);

    result = !(a != b);
    printf("!(a != b) equals to %d\n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d\n", result);
}
```

Logical Operators

```c
int main() {

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

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d\n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d\n", result);

    result = (a == b) || (c > b);
    printf("(a == b) || (c > b) equals to %d\n", result);

    result = (a != b) || (c > b);
    printf("(a != b) || (c > b) equals to %d\n", result);

    result = !(a != b);
    printf("!(a != b) equals to %d\n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d\n", result);
}
```

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c > b) equals to 1
(a != b) || (c > b) equals to 1
!(a != b) equals to 1
!(a == b) equals to 0
```

Bitwise Operators

- During computation, mathematical operations like: addition, subtraction, Multiplication and division are converted to bit-level which makes processing faster and saves power.
- Bitwise operators are still used for those working on embedded devices that have memory limitations.
- Bitwise operators are used in C programming to perform bit-level operations.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Basics of Bitwise Operations

Name	Symbol	Usage	What it does
Bitwise And	&	a &b	Returns 1 only if both the bits are 1
Bitwise Or	|	a | b	Returns 1 if one of the bits is 1
Bitwise Not	~	~a	Returns the complement of a bit
Bitwise Xor	^	a^b	Returns 0 if both the bits are same else 1
Bitwise Left shift	<<	a<<n	Shifts a towards left by n digits
Bitwise Right shift	>>	a>>n	Shifts a towards right by n digits

Bitwise AND (&)

- The output of bitwise AND is 1 if the corresponding bits of two operands are 1.
  - If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

```
12  =  00001100 (in Binary)
25  =  00011001 (in Binary)

Bit operation of 12 and 25

     00001100
&    00011001
     --------
     00001000  =  8 (in Decimal)
```

C code for Bitwise AND

```
#include <stdio.h>

int main() {

int a = 12, b = 25;

printf("Output = %d", a & b);
return 0;}
```

- Output is :
- Output = 8

Don’t get confused with Binary Addition

- **Examples of Binary Addition**
- **Example 1:** 10001 + 11101
- **Solution:**

```
      1
   1 0 0 0 1
(+) 1 1 1 0 1
-------------
  1 0 1 1 1 0
```

- **Rules of Binary Addition**
- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 =10

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	0 (where 1 s carried over)

Bitwise OR (|)

- The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1

```
12  =  00001100 (in Binary)
25  =  00011001 (in Binary)

Bit operation of 12 and 25

    00001100
  & 00011001
    ________

    00011101  =  29 (in Decimal)
```

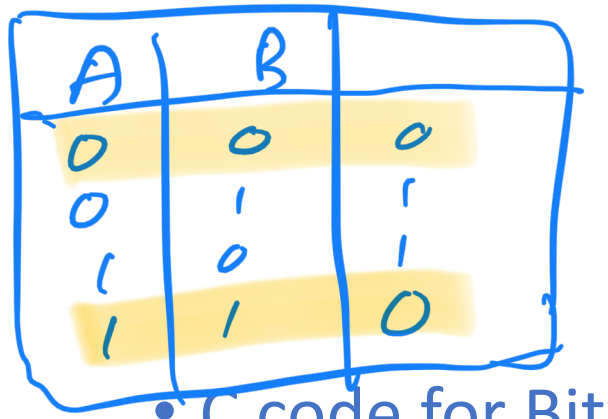
C code for Bitwise OR

```
#include <stdio.h>
int main() {
int a = 12, b = 25;
printf("Output = %d", a | b);
return 0;}
```

Output is :

Output = 29

Bitwise XOR (^)



- The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite.

```
12  =  00001100 (in Binary)
25  =  00011001 (in Binary)

Bit operation of 12 and 25

    00001100
^   00011001
    --------
    00010101  =  21 (in Decimal)
```

- C code for Bitwise XOR

```
#include <stdio.h>
int main() {
 int a = 12, b = 25;
printf("Output = %d", a ^ b);
return 0;}
```

Output :

Output = 21

Bitwise complement (~)

- Bitwise compliment operator is an unary operator (works on only one operand) which changes 1 to 0 and 0 to 1.

```
12  =  00001100 (in Binary)

Bitwise complement of 12

   ~ 00001100
     ----------
     11110011  =  243 (in Decimal)
```

- Example :C code for Bitwise complement

```
#include <stdio.h>

int main() {

printf("Output = %d\n", ~35);
printf("Output = %d\n", ~-12);
return 0;}
```

Outputs:

Output = -36

Output = 11

Bitwise Operators

```c
int main() {

    int a = 12, b = 25;

    printf("Bitwise AND = %d\n", a&b);
    printf("Bitwise OR = %d\n", a|b);
    printf("Bitwise XOR = %d\n", a^b);
    printf("Complement of a = %d\n", ~a);
    printf("Complement of b = %d\n", ~b);

}
```

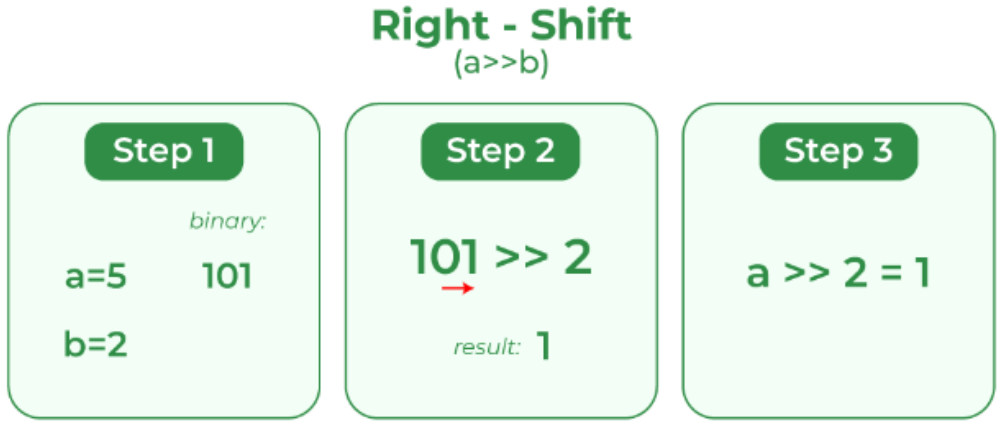
Bitwise Operators

```c
int main() {

    int a = 12, b = 25;

    printf("Bitwise AND = %d\n", a&b);
    printf("Bitwise OR = %d\n", a|b);
    printf("Bitwise XOR = %d\n", a^b);
    printf("Complement of a = %d\n", ~a);
    printf("Complement of b = %d\n", ~b);

}
```

```
Bitwise AND = 8
Bitwise OR = 29
Bitwise XOR = 21
Complement of a = -13
Complement of b = -26
```

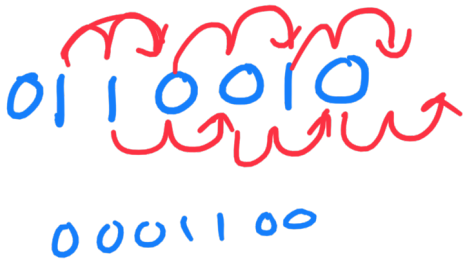
Shift Operators

- Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.



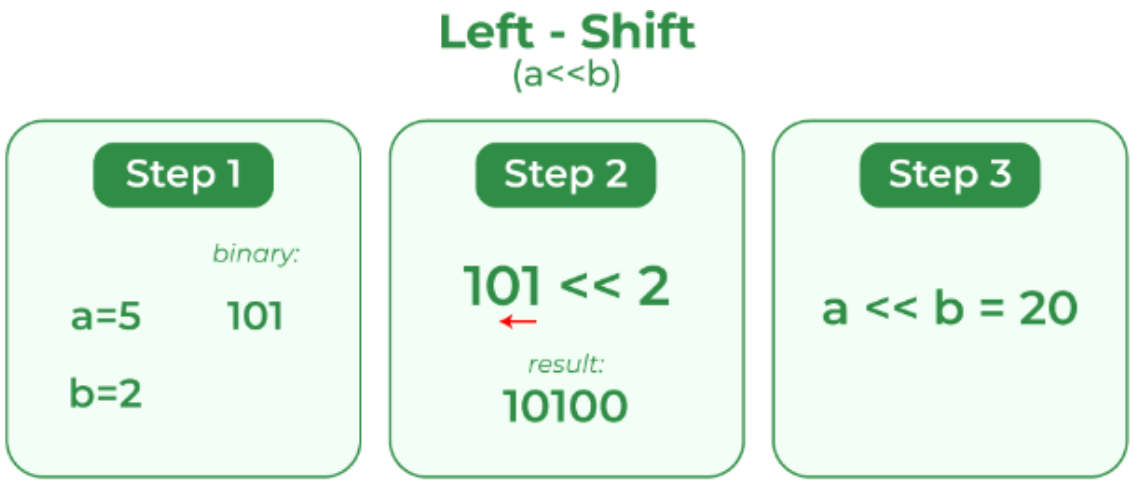
- Right shift operator

```
212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```



Shift Operators

- Left shift operator shifts all bits towards left by a certain number of specified bits.
- The bit positions that have been vacated by the left shift operator are filled with **0**. The symbol of the left shift operator is <<
- Left shift operator




```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 =11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)
```

Example : Shift Operators (Left & Right)

```c
#include <stdio.h>

int main() {

    int num=212, i;

    for (i = 0; i <= 2; ++i) {
        printf("Right shift by %d: %d\n", i, num >> i);
    }
    printf("\n");

    for (i = 0; i <= 2; ++i) {
        printf("Left shift by %d: %d\n", i, num << i);
    }

    return 0;
}
```

Output:

Right shift by 0: 212
Right shift by 1: 106
Right shift by 2: 53

Left shift by 0: 212
Left shift by 1: 424
Left shift by 2: 848

Other Operators

- Comma Operator
  - Comma operators are used to link related expressions together.
  - For example:

```
int x = 10, y = 5, z;
```

- The sizeof operator
  - The *sizeof* is an unary operator which returns the size of data (constant, variables, array, structure etc).

- C Ternary Operator (?:)

Exercise 1-Write down the output if one executes the following program.

```
#include <stdio.h>
int main()
{
unsigned char 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;
}
```

- **Output**

Exercise 1-Write down the output if one executes the following program.

```
#include <stdio.h>
int main()
{
unsigned char 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;
}
```

- **Output**

```
a = 5, b = 9
a & b = 1
a | b = 13
a ^ b = 12
~a = -6
b << 1 = 18
b >> 1 = 4
```

Exercise 2

a) Write a C program to swap two numbers without using temporary variable
b) Write a C program to swap two numbers using Bitwise operators.

Exercise 2

a)Write a C program to swap two numbers without using temporary variable

```
#include <stdio.h>
int main()
{
int x = 10, y = 5;
// Code to swap 'x' and 'y'
x = x * y; // x now becomes 50
y = x / y; // y becomes 10
x = x / y; // x becomes 5
printf("After Swapping: x = %d, y = %d", x, y);
return 0;
}
```

- **Output**

```
After Swapping: x =5, y=10
```

Exercise 2

(b) **Write a C program to swap two numbers using Bitwise operators.**

Most suitable operator is :Bitwise XOR

For example,

XOR of 10 (In Binary 1010) and

5 (In Binary 0101) is 1111, and

XOR of 7 (0111) and 5 (0101) is (0010)

C code for swapping

```c
#include <stdio.h>
int main()
{
    int x = 10, y = 5;
      // Code to swap 'x' (1010) and 'y' (0101)
    x = x ^ y; // x now becomes 15 (1111)
    y = x ^ y; // y becomes 10 (1010)
    x = x ^ y; // x becomes 5 (0101)

    printf("After Swapping: x = %d, y = %d", x, y);

    return 0;
}
```

Output:

```
After Swapping: x =5, y=10
```