

Difference between Python and C

- C programs – Compiled
- Python programs – Interpreted

Compiler	Interpreter
Takes entire program as input and generate a output file with object code	Takes instruction by instruction as input and gives an output. But does not generate a file
Errors are displayed after entire program is checked	Errors are displayed for every instruction interpreted (if any)

Variable Declaration in C

- In C, it is mandatory to do variable **declaration**
- We say variable's **type**, whether it is an integer (**int**), floating-point number (**float**), character (**char**) etc
- The Syntax for variable declaration is –
type of variable, white space, name of variable semicolon
- Eg: int number;

White spaces and Intendation

- No problem of difference between white space and tab in C (Happy!)
- Block of code in C need not be intended as in Python
- In C, Curly braces are used for giving a block of code

Eg: Block of code in 'C'

```
{
```

```
-----
```

```
} continuation
```

Layout of a C program

pre-processor directives – Preceded by a '#'

global declarations – Optional and not a good programming practice

int main() - standard start for all C programs

{

local variables to function main ; - all variables used in the function must be declared in the beginning

statements associated with function main ;

}

void f1()

{

local variables to function 1 ;

statements associated with function 1 ;

}

Components of a C program

A C program consists of the following parts:

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

Comments in C

Two types of comments

Single line and multi line comment

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

Multiline Comments in C

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

Types of C Variables

- Variable names - Names given to locations in memory
- Locations can contain integer, real (float) or character constants
- Because a particular type of variable can hold only same type of constant
- Integer variable - integer constant, real variable - real constant and a character variable - a character constant

Rules for Constructing Variable Names

- Combination of alphabets, digits or underscores
- Some compilers allow up to 247 characters but safer to 31 characters
- First character must be an alphabet or underscore
- No commas or blanks are allowed
- No special symbol other than an underscore
- Eg: `si_int` `m_hra` `pop_e_89`

Valid and Invalid variable names

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

Data types in C

very common data types used in C:

char: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

int: As the name suggests, an int variable is used to store an integer.

float: It is used to store decimal numbers (numbers with floating point value) with single precision.

double: It is used to store decimal numbers (numbers with floating point value) with double precision.

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d

long int	4	-2,147,483,648 to 2,147,483,647	%ld
----------	---	------------------------------------	-----

unsigned long int	4	0 to 4,294,967,295	%lu
----------------------	---	--------------------	-----

long long int	8	-(2 ⁶³) to (2 ⁶³)-1	%lld
---------------	---	---	------

unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
---------------------------	---	------------------------------------	------

signed char	1	-128 to 127	%c
-------------	---	-------------	----

signed char	1	-128 to 127	%c
-------------	---	-------------	----

unsigned char	1	0 to 255	%c
---------------	---	----------	----

float	4		%f
-------	---	--	----

double	8		%lf
--------	---	--	-----

long double	16		%Lf
-------------	----	--	-----

Keywords/ Reserved words

- 32 keywords available in C

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

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

I/O in C

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

I/O in C

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

Example

- ```
#include<stdio.h>
void main()
{
 int a;
 int b;
 scanf("%d",&a);
 scanf("%d",&b);
 int c = a - b;
 printf("%d",c);
}
```

# printf and scanf format codes

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

# printf and scanf format codes

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

# Address of a variable

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

# Operator in C

| Common operators                                                                                                                                                                                                                                                           |                                                                              |                                                                                                                                                                                                                                                                                                        |                                                                       |                                                                                                                                                  |                                                                                                         |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|--|
| <u>assignment</u>                                                                                                                                                                                                                                                          | <u>increment<br/>decrement</u>                                               | <u>arithmetic</u>                                                                                                                                                                                                                                                                                      | <u>logical</u>                                                        | <u>comparison</u>                                                                                                                                | <u>member<br/>access</u>                                                                                |  |
| <code>a = b</code><br><code>a += b</code><br><code>a -= b</code><br><code>a *= b</code><br><code>a /= b</code><br><code>a %= b</code><br><code>a &amp;= b</code><br><code>a  = b</code><br><code>a ^= b</code><br><code>a &lt;&lt;= b</code><br><code>a &gt;&gt;= b</code> | <code>++a</code><br><code>--a</code><br><code>a++</code><br><code>a--</code> | <code>+a</code><br><code>-a</code><br><code>a + b</code><br><code>a - b</code><br><code>a * b</code><br><code>a / b</code><br><code>a % b</code><br><code>a ^ b</code><br><code>a &lt;&lt; b</code><br><code>a &gt;&gt; b</code><br><code>a &amp; b</code><br><code>a   b</code><br><code>a ^ b</code> | <code>!a</code><br><code>a &amp;&amp; b</code><br><code>a    b</code> | <code>a == b</code><br><code>a != b</code><br><code>a &lt; b</code><br><code>a &gt; b</code><br><code>a &lt;= b</code><br><code>a &gt;= b</code> | <code>a[b]</code><br><code>*a</code><br><code>&amp;a</code><br><code>a-&gt;b</code><br><code>a.b</code> |  |

# Arithmetic instructions in C

To perform arithmetic operations between constants and variables

- Operands shall be either constant or variables
- Variables can be of any type of integer or floating point value or character except for modulus operator
- Modulus operator cannot be applied for floating point values but can be applied for integer and character types
- If you try to use the modulo operator with floating-point constants or variables, the compiler will produce an error

# Example 1

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

```
void main()
```

```
{
```

```
int a = 27;
```

```
int b = 25;
```

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

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

```
}
```



# Output 1

2

# Precedence of Operators in C

| Category       | Operator                          | Associativity |
|----------------|-----------------------------------|---------------|
| Postfix        | () [] -> . ++ --                  | Left to right |
| Unary          | + - ! ~ ++ -- (type)* & sizeof    | Right to left |
| Multiplicative | * / %                             | Left to right |
| Additive       | + -                               | Left to right |
| Shift          | << >>                             | Left to right |
| Relational     | < <= > >=                         | Left to right |
| Equality       | == !=                             | Left to right |
| Bitwise AND    | &                                 | Left to right |
| Bitwise XOR    | ^                                 | Left to right |
| Bitwise OR     |                                   | Left to right |
| Logical AND    | &&                                | Left to right |
| Logical OR     |                                   | Left to right |
| Conditional    | ?:                                | Right to left |
| Assignment     | = += -= *= /= %= >>= <<= &= ^=  = | Right to left |
| Comma          | ,                                 | Left to right |

```
#include <stdio.h>
void main() {
 int a = 20;
 int b = 10;
 int c = 15;
 int d = 5;
 int e;
 e = (a + b) * c / d; // (30 * 15) / 5
 printf("Value of (a + b) * c / d is : %d\n", e);
 e = ((a + b) * c) / d; // (30 * 15) / 5
 printf("Value of ((a + b) * c) / d is : %d\n", e);
 e = (a + b) * (c / d); // (30) * (15/5)
 printf("Value of (a + b) * (c / d) is : %d\n", e);
 e = a + (b * c) / d; // 20 + (150/5)
 printf("Value of a + (b * c) / d is : %d\n", e);
}
```

## Postfix / Prefix operator

++

--

a = 5;

++a;      // a becomes 6

a++;      // a becomes 7

--a;      // a becomes 6

a--;      // a becomes 5

```
#include <stdio.h>
void main() {
 int i=5;
 int j;
 j=i++;
 printf ("\nafter postfix increment i=%d j=%d", i,j);
 i=5;
 j=++i;
 printf ("\n after prefix increment i=%d j=%d",i,j);
}
```

The output is

after postfix increment i=6 j=5

after prefix increment i=6 j=6

# Example

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

```
main()
```

```
{
```

```
int a, b,c;
```

```
a = 4;
```

```
b = 2;
```

```
c = -a+--b;
```

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

```
}
```

# Output 4

$$c = -3$$

# Example 5

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

```
main()
```

```
{
```

```
int a, b,c;
```

```
a = 4;
```

```
b = 2;
```

```
c = -a+ b--;
```

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

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

```
}
```



# Output 5

$c = -2$

$b = 1$

# Car Loan Problem

- You have saved some amount to use as a down payment on a car. Before beginning your car shopping, you decide to write a program to help you figure out what your monthly payment will be, given the car's purchase price, the monthly interest rate, and the time period over which you will pay back the loan. The formula for calculating your payment is

$$\text{payment} = iP / (1 - (1 + i)^{-n})$$

where  $P$  = principal (the amount you borrow)

$i$  = monthly interest rate (1/12 of the annual rate)

$n$  = total number of payments

Total number of payments is usually 36, 48, or 60 (months). Program should then display the amount borrowed and the monthly payment rounded to two decimal places.

# Car Loan problem

| Input                                                                                          | Output                        | Logic Involved                                               |
|------------------------------------------------------------------------------------------------|-------------------------------|--------------------------------------------------------------|
| Purchase price,<br>down payment,<br>annual interest<br>rate and total<br>number of<br>payments | Amount<br>borrowed and<br>EMI | Compute Monthly<br>payment<br>$= i * P / (1 - (1 + i)^{-n})$ |

# Algorithm for Car Loan Problem

- Read input such as Purchase price, down payment, annual interest rate and total number of payments from user
- $\text{amount\_Borrowed} = \text{purchase\_Price} - \text{down\_Payment}$
- $\text{EMI} = i * P / (1 - (1 + i)^{-n})$
- Print amount\_Borrowed and EMI rounded to two decimal places

# Implementation in C

- Already learnt to read input and print values
- Learnt about arithmetic operators
- The expression has to find power of a value
- Similarly some problems may need to find square root or log values, trigonometric values etc...
- Have to learn to format output as it is specified in problem to print two digits after decimal point

# Library Functions

- Predefined Functions and Code Reuse
- A primary goal of software engineering is to write error-free code.
- Code reuse - reusing program fragments that have already been written and tested whenever possible, is one way to accomplish this goal.

## Some Mathematical Library Functions

| Function              | Standard Header File          | Purpose: Example                                                                                                                                                     | Argument(s)                      | Result              |
|-----------------------|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|---------------------|
| <code>abs(x)</code>   | <code>&lt;stdlib.h&gt;</code> | Returns the absolute value of its integer argument:<br>if <code>x</code> is <code>-5</code> , <code>abs(x)</code> is <code>5</code>                                  | <code>int</code>                 | <code>int</code>    |
| <code>ceil(x)</code>  | <code>&lt;math.h&gt;</code>   | Returns the smallest integral value that is not less than <code>x</code> :<br>if <code>x</code> is <code>45.23</code> , <code>ceil(x)</code> is <code>46.0</code>    | <code>double</code>              | <code>double</code> |
| <code>cos(x)</code>   | <code>&lt;math.h&gt;</code>   | Returns the cosine of angle <code>x</code> :<br>if <code>x</code> is <code>0.0</code> , <code>cos(x)</code> is <code>1.0</code>                                      | <code>double</code><br>(radians) | <code>double</code> |
| <code>exp(x)</code>   | <code>&lt;math.h&gt;</code>   | Returns $e^x$ where $e = 2.71828\dots$ :<br>if <code>x</code> is <code>1.0</code> , <code>exp(x)</code> is <code>2.71828</code>                                      | <code>double</code>              | <code>double</code> |
| <code>fabs(x)</code>  | <code>&lt;math.h&gt;</code>   | Returns the absolute value of its type <code>double</code> argument:<br>if <code>x</code> is <code>-8.432</code> , <code>fabs(x)</code> is <code>8.432</code>        | <code>double</code>              | <code>double</code> |
| <code>floor(x)</code> | <code>&lt;math.h&gt;</code>   | Returns the largest integral value that is not greater than <code>x</code> :<br>if <code>x</code> is <code>45.23</code> , <code>floor(x)</code> is <code>45.0</code> | <code>double</code>              | <code>double</code> |
| <code>log(x)</code>   | <code>&lt;math.h&gt;</code>   | Returns the natural logarithm of <code>x</code> for <code>x &gt; 0.0</code> :<br>if <code>x</code> is <code>2.71828</code> , <code>log(x)</code> is <code>1.0</code> | <code>double</code>              | <code>double</code> |

|                        |                             |                                                                                                                                                                                                                                |                                              |                     |
|------------------------|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|---------------------|
| <code>log10(x)</code>  | <code>&lt;math.h&gt;</code> | Returns the base-10 logarithm of <code>x</code> for <code>x &gt; 0.0</code> :<br>if <code>x</code> is <code>100.0</code> , <code>log10(x)</code> is <code>2.0</code>                                                           | <code>double</code>                          | <code>double</code> |
| <code>pow(x, y)</code> | <code>&lt;math.h&gt;</code> | Returns <code>x<sup>y</sup></code> . If <code>x</code> is negative, <code>y</code> must be integral: if <code>x</code> is <code>0.16</code> and <code>y</code> is <code>0.5</code> , <code>pow(x,y)</code> is <code>0.4</code> | <code>double</code> ,<br><code>double</code> | <code>double</code> |
| <code>sin(x)</code>    | <code>&lt;math.h&gt;</code> | Returns the sine of angle <code>x</code> :<br>if <code>x</code> is <code>1.5708</code> , <code>sin(x)</code> is <code>1.0</code>                                                                                               | <code>double</code><br>(radians)             | <code>double</code> |
| <code>sqrt(x)</code>   | <code>&lt;math.h&gt;</code> | Returns the nonnegative square root of <code>x</code> ( $\sqrt{x}$ ) for <code>x ≥ 0.0</code> :<br>if <code>x</code> is <code>2.25</code> , <code>sqrt(x)</code> is <code>1.5</code>                                           | <code>double</code>                          | <code>double</code> |
| <code>tan(x)</code>    | <code>&lt;math.h&gt;</code> | Returns the tangent of angle <code>x</code> :<br>if <code>x</code> is <code>0.0</code> , <code>tan(x)</code> is <code>0.0</code>                                                                                               | <code>double</code><br>(radians)             | <code>double</code> |

---



# Scanf and printf

- scanf and printf are also predefined functions that are defined in the header file `stdio.h`
- **scanf** - returns the number of items successfully read
- **printf()** - returns the number of **characters** successfully written on the output

# Formatting Output

- Additional arguments in printf to format as requested by user:
- **%[flags][width][.precision][length]specifier**  
**“ % f”**  
**% d**
- **10**
- **float a;**
- **a=10.1;**
- **printf(“The value of a: %.5d”,a); 10.10000**

# Formatting Output

| Flags         | Description                                                                                                                                                |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -<br>%-f<br>4 | Left-justify within the given field width; Right justification is the default                                                                              |
| +<br>%+d      | Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a -ve sign. |
| (space)       | If no sign is going to be written, a blank space is inserted before the value                                                                              |
| 0<br>0003     | Left-pads the number with zeroes (0) instead of spaces, where padding is specified                                                                         |

# Formatting Output

| Width             | Description                                                                                                                                                      |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (number)          | Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces.                      |
| <b>.precision</b> | <b>Description</b>                                                                                                                                               |
| .number           | precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. |

# Formatting Output

| length | Description                                                                                                                                                                       |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| h      | Argument is interpreted as a short int or unsigned short int (only applies to integer specifiers: i, d, o, u, x and X).                                                           |
| l      | argument is interpreted as a long int or unsigned long int for integer specifiers (i, d, o, u, x and X), and as a wide character or wide character string for specifiers c and s. |
| L      | argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g and G)                                                                            |

```
int i=10;
int i1=-10;
float j=10.1;
printf("\nThe value of i:%d",i); :10
printf("\nThe value of i:%5d",i) : 10
printf("\nThe value of i:%-5d",i) :10
printf("\nThe value of i:%+d",i) :+10
printf("\nThe value of i:%+d",i1) :-10
printf("\nThe value of i:% d",i1) :-10
printf("\nThe vaue of i:% d",i1) : 10
printf("\nThe vaue of i:%-05d",i) :10
printf("\nThe vaue of i:%f",j) :10.100000
printf("\nThe vaue of i:%.5f",j) :10.10000
```

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

```
main()
```

```
{
```

```
 int a,b;
```

```
 float c,d;
```

```
 a = 15;
```

```
 b = a / 2;
```

```
 printf("%d\n",b);
```

```
 printf("%3d\n",b);
```

```
 printf("%03d\n",b);
```

```
 c = 15.3;
```

```
 d = c / 3;
```

```
 printf("%3.2f\n",d);
```

```
}
```

# Output

```
7
 7
007
5.10
```



# Example for Formatting

```
#include<stdio.h>

main()
{
 printf("The color: %s\n", "blue");
 printf("First number: %d\n", 12345);
 printf("Second number: %04d\n", 25);
 printf("Third number: %i\n", 1234);
 printf("Float number: %3.2f\n", 3.14159);
 printf("Hexadecimal: %x\n", 255);
 printf("Octal: %o\n", 255);
 printf("Unsigned value: %u\n", 150);
 printf("Just print the percentage sign %%\n", 10);
}
```

# Output

```
The color: blue
First number: 12345
Second number: 0025
Third number: 1234
Float number: 3.14
Hexadecimal: ff
Octal: 377
Unsigned value: 150
Just print the percentage sign %
```

# Example for Formatting

```
#include<stdio.h>

main()
{
 printf(":%s:\n", "Hello, world!");
 printf(":%15s:\n", "Hello, world!");
 printf(":%.10s:\n", "Hello, world!");
 printf(":%-10s:\n", "Hello, world!");
 printf(":%-15s:\n", "Hello, world!");
 printf(":%.15s:\n", "Hello, world!");
 printf(":%15.10s:\n", "Hello, world!");
 printf(":%-15.10s:\n", "Hello, world!");
}
```

# Output

```
:Hello, world!:
: Hello, world!:
:Hello, wor:
:Hello, world!:
:Hello, world! :
:Hello, world!:
: Hello, wor:
:Hello, wor :
```

`printf(":%s:\n", "Hello, world!");` - nothing special happens

`printf(":%15s:\n", "Hello, world!");` print 15 characters. If string is smaller then "empty" positions will be filled with "whitespace."

`printf(":%.10s:\n", "Hello, world!");` statement prints the string, but print only 10 characters of the string.

`printf(":%-10s:\n", "Hello, world!");` statement prints the string, but prints at least 10 characters. If the string is smaller "whitespace" is added at the end.

`printf(":%-15s:\n", "Hello, world!");` statement prints the string, but prints at least 15 characters. The string in this case is shorter than the defined 15 character, thus "whitespace" is added at the end (defined by the minus sign.)

`printf(":%.15s:\n", "Hello, world!");` statement prints the string, but print only 15 characters of the string. In this case the string is shorter than 15, thus the whole string is printed.

```
#include<stdio.h>
#include<math.h>
void main()
{
double purchase_Price, down_Payment, annual_Interest,l;
int num_Of_Payment;

double principal,EMI,monthly_Interest,dr;
//Read purcahse price of car
scanf("%lf",&purchase_Price);
//Read the value of down payment
scanf("%lf",&down_Payment);
//Read annual interest
scanf("%lf",&annual_Interest);
//read num_Of_Payment
scanf("%d",&num_Of_Payment);
monthly_Interest = annual_Interest/(12*100);
principal = purchase_Price - down_Payment;
dr = 1-pow((1+monthly_Interest),(-1*num_Of_Payment));
EMI = (monthly_Interest*principal)/dr;
printf("Loan Amount %.2lf\n",principal);
printf("Monthly installment %.2lf",EMI);

}
```

To compile – gcc carloan.c –lm

./a.out

When input is:

400000

100000

10

36

Output is:

Loan Amount 300000.00

Monthly installment 9680.16



# Type of Instructions

- Instructions are organized into three kinds of control structures to control execution flow:
  - Sequence
  - Selection
  - Repetition
- A selection control structure chooses which alternative to execute

# Compound Statements

- Until now we have been using only sequential flow
- A compound statement, written as a group of statements bracketed by { and }, is used to specify sequential flow.
- {  
statement 1 ;  
statement 2 ;  
...  
statement n ;  
}

# Data Types

- No boolean and no String data type in C as in Python
- C uses 0 and 1 values to indicate true and false

# Defining Constants

- Similar like a variable declaration except the value cannot be changed
- *In declaration, const* keyword is used before or after type
- `int const a = 1; const int a = 2;`
- It is usual to initialise a *const* with a value as it cannot get a value *any other way*.

# Preprocessor definition

- *#define* is another more flexible method to define *constants* in a program
- `#define TRUE 1`
- `#define FALSE 0`
- `#define NAME_SIZE 20`
- Here TRUE, FALSE and NAME\_SIZE are constant

# Relational and Equality Operators

- Variable relational-operator variable
- Variable relational-operator constant
- Variable equality-operator variable
- Variable equality-operator constant

# Relational and Equality Operators

| Operator | Meaning                  | Type       |
|----------|--------------------------|------------|
| <        | less than                | relational |
| >        | greater than             | relational |
| <=       | less than or equal to    | relational |
| >=       | greater than or equal to | relational |
| ==       | equal to                 | equality   |
| !=       | not equal to             | equality   |

# Examples

Memory with Values

|    |       |         |   |      |          |            |     |          |
|----|-------|---------|---|------|----------|------------|-----|----------|
| x  | power | MAX_POW | y | item | MIN_ITEM | mom_or_dad | num | SENTINEL |
| -5 | 1024  | 1024    | 7 | 1.5  | -999.0   | 'M'        | 999 | 999      |

## Sample Conditions

| Operator | Condition         | English Meaning              | Value     |
|----------|-------------------|------------------------------|-----------|
| <=       | x <= 0            | x less than or equal to 0    | 1 (true)  |
| <        | power < MAX_POW   | power less than MAX_POW      | 0 (false) |
| >=       | x >= y            | x greater than or equal to y | 0 (false) |
| >        | item > MIN_ITEM   | item greater than MIN_ITEM   | 1 (true)  |
| ==       | mom_or_dad == 'M' | mom_or_dad equal to 'M'      | 1 (true)  |
| !=       | num != SENTINEL   | num not equal to SENTINEL    | 0 (false) |



# Logical Operators

- To form more complicated conditions or logical expressions
- Three operators:
  - And (&&)
  - Or (||)
  - Not(!)

# Logical And

## The && Operator (and)

| operand1       | operand2       | operand1 && operand2 |
|----------------|----------------|----------------------|
| nonzero (true) | nonzero (true) | 1 (true)             |
| nonzero (true) | 0 (false)      | 0 (false)            |
| 0 (false)      | nonzero (true) | 0 (false)            |
| 0 (false)      | 0 (false)      | 0 (false)            |

# Logical Or

## The || Operator (or)

| operand1       | operand2       | operand1    operand2 |
|----------------|----------------|----------------------|
| nonzero (true) | nonzero (true) | 1 (true)             |
| nonzero (true) | 0 (false)      | 1 (true)             |
| 0 (false)      | nonzero (true) | 1 (true)             |
| 0 (false)      | 0 (false)      | 0 (false)            |

# Logical Not


## The ! Operator (not)

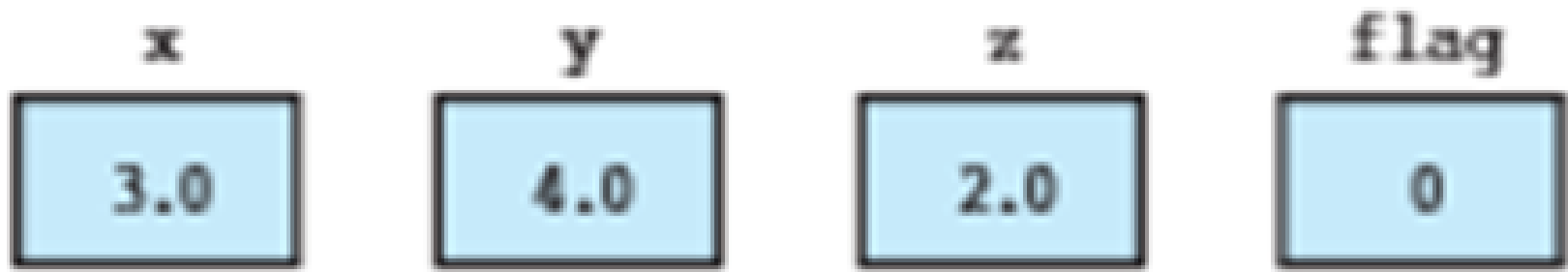
| operand1       | !operand1 |
|----------------|-----------|
| nonzero (true) | 0 (false) |
| 0 (false)      | 1 (true)  |

# True/False Values

- For numbers all values except 0 is true
- For characters all values except '/0' (Null Character) is true

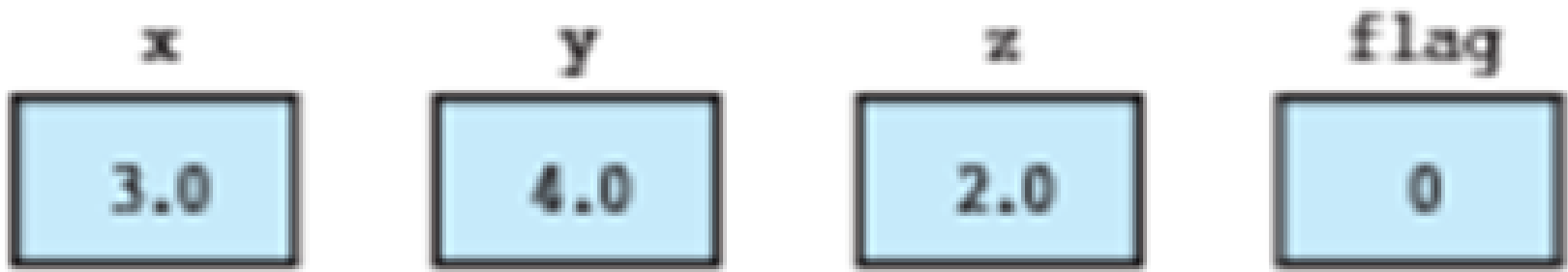
## Operator Precedence

| Operator                  | Precedence                                                                           |
|---------------------------|--------------------------------------------------------------------------------------|
| function calls            | highest                                                                              |
| ! + - & (unary operators) |  |
| * / %                     |                                                                                      |
| + -                       |                                                                                      |
| < <= >= >                 |                                                                                      |
| -- !-                     |                                                                                      |
| &&                        |                                                                                      |
|                           |                                                                                      |
| =                         | lowest                                                                               |



!flag

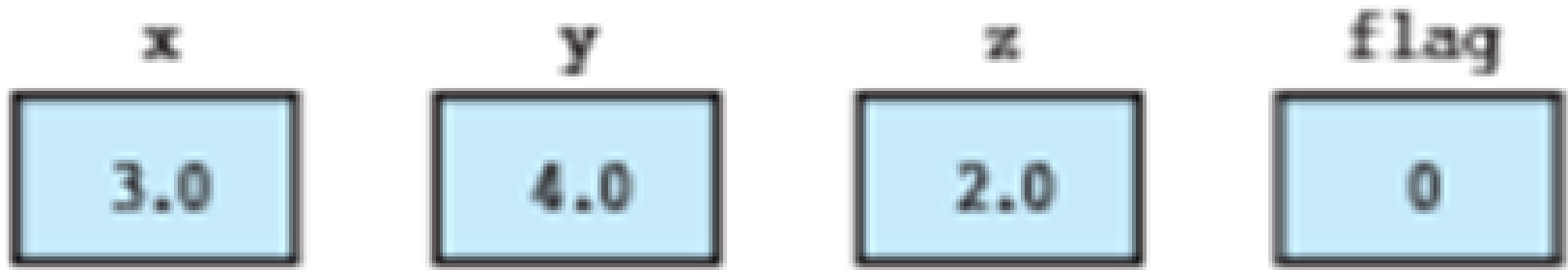
1 (true)



$$x + y / z \leq 3.5$$

5.0  $\leq$  3.5 is 0 (False)





`!flag || (y + z >= x - z)`

`1 || 1 = 1`

# Logical Assignment

- `even = (n % 2 == 0);`
- `in_range = (n > -10 && n < 10);`
- `is_letter = ('A' <= ch && ch <= 'Z') || ('a' <= ch && ch <= 'z');`
- Variable `in_range` gets 1 (true) if the value of `n` is between -10 and 10 excluding the endpoints;
- `is_letter` gets 1 (true) if `ch` is an uppercase or a lowercase letter.

When 'A' = 60 and 'B' =13

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

```
#include<stdio.h>
void main()
{
 int a = 60, b=13;
 printf("%d\n",a&b);
 printf("%d\n",a|b);
 printf("%d\n",a^b);
 printf("%d\n",~a);
 printf("%d\n",a<<2);
 printf("%d\n",a>>2);
}
```

# #define statements

- These are preprocessor directives that are used to define constants
- When a value is defined using #define, if there is a change in value it is easier to make modification

1. Write a C program using the switch case to display the following,

Dress Code for the different academic Lab's in a College:

- Physics Lab
- Chemistry Lab
- Computer Science Lab
- Engg. Drawing
- Mechanical Lab
- Electrical Lab

Keep a warning message about the wrong choice of lab, in the default case.

2. Extend the same program of question 1 as following,

- Keep different dress code for Boy's and Girl's (use if statement).

`clrscr() -> #include conio.h;`

Escape sequences are:

- \n (newline)
- \t (tab)
- \v (vertical tab)
- \f (new page)
- \b (backspace)
- \r (carriage return)
- \n (newline)

- `%d` (print as a decimal integer)
- `%6d` (print as a decimal integer with a width of at least 6 wide)
- `%f` (print as a floating point)
- `%4f` (print as a floating point with a width of at least 4 wide)
- `%.4f` (print as a floating point with a precision of four characters after the decimal point)
- `%3.2f` (print as a floating point at least 3 wide and a precision of 2)

- `%d`
- `%f`



```

#include<stdio.h>
main()
{
 printf(":%s:\n", "Hello, world!");
 printf(":%15s:\n", "Hello, world!");
 printf(":%.10s:\n", "Hello, world!");
 printf(":%-10s:\n", "Hello, world!");
 printf(":%-15s:\n", "Hello, world!");
 printf(":%.15s:\n", "Hello, world!");
 printf(":%15.10s:\n", "Hello, world!");
 printf(":%-15.10s:\n", "Hello, world!");
}

```

The output of the example above:

```

:Hello, world!:
: Hello, world!:
:Hello, wor:
:Hello, world!:
:Hello, world! :
:Hello, world!:
: Hello, wor:
:Hello, wor :

```

# C if else Statement

The syntax of the if statement in C programming is:

```
int age;
```

```
age = 50;
```

```
if (age>=18)
```

```
{
```

```
 printf("you are eligible");
```

```
 if(age>=60)
```

```
 {
```

```
 printf("Senior");
```

```
 }
```

```
 else
```

```
 {
```

```
 printf("Not a Senior");
```

```
 }
```

```
}
```

```
else if(age<=10)
```

```
{
```

```
 printf("you are not eligible");
```

```
}
```

```
else
```

```
{
```

Expression is true.

```
int test = 5;
```

```
if (test < 10)
```

```
{
```

```
 // codes
```

```
}
```

```
// codes after if
```

Expression is false.

```
int test = 5;
```

```
if (test > 10)
```

```
{
```

```
 // codes
```

```
}
```

```
// codes after if
```

```
// Program to display a number if it is negative
```

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

```
int main() {
```

```
 int number;
```

```
 printf("Enter an integer: ");
```

```
 scanf("%d", &number);
```

```
 // true if number is less than 0
```

```
 if (number < 0) {
```

```
 printf("You entered %d.\n", number);
```

```
 }
```

```
 printf("The if statement is easy.");
```

```
 return 0;
```

```
}
```

# C if...else Statement


The if statement may have an optional else block. The syntax of the if..else statement is:

```
if (test expression)
{
 // run code if test expression is true
}
else
{
 // run code if test expression is false
}
```

Expression is true.

```
int test = 5;

if (test < 10)
{
 // body of if
}
else
{
 // body of else
}
```




A flowchart illustrating the execution of the code when the expression is true. It starts with an arrow pointing to the opening curly brace of the 'if' block. Another arrow points to the opening curly brace of the 'else' block. A third arrow points to the closing curly brace of the 'if' block, indicating that the 'else' block is not executed.

Expression is false.

```
int test = 5;

if (test > 10)
{
 // body of if
}
else
{
 // body of else
}
```



A flowchart illustrating the execution of the code when the expression is false. It starts with an arrow pointing to the opening curly brace of the 'if' block. Another arrow points to the opening curly brace of the 'else' block. A third arrow points to the closing curly brace of the 'else' block, indicating that the 'if' block is not executed.

# Example 1

- ```
#include<stdio.h>
void main()
{
float i=0.0;
if(i)
printf("Yes");
else
printf("No");
}
```

Output 1

- No

Example 2

- ```
#include<stdio.h>
void main()
{
int i=-3;
if(i)
printf("Yes");
else
printf("No");
}
```

# Output 2

- Yes

# Example 3

- ```
#include<stdio.h>
void main()
{
char i ='a';
if(i)
printf("Yes");
else
printf("No");
}
```

Output 3

- Yes

Example 4

- ```
#include<stdio.h>
void main()
{
int a = 2 , b=5;
if ((a==0)&&(b=1))
printf("Hi");
printf("a is %d and b is %d", a, b);
}
```

# Output 4

- a is 2 and b = 5

# Example 5

- ```
#include<stdio.h>
void main()
{
int a = 2 , b=5;
if ((a==2)&&(b=1))
printf("Hi");
printf("a is %d and b is %d", a, b);
}
```

Output 5

- a is 2 and b = 1

C if...else Ladder

Syntax of if...else Ladder

```
if (test expression1) {  
    // statement(s)  
}  
else if(test expression2) {  
    // statement(s)  
}  
else if (test expression3) {  
    // statement(s)  
}  
.  
.  
else {  
    // statement(s)  
}
```

// Program to relate two integers using =, > or < symbol

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

```
void main() {
```

```
    int number1, number2;
```

```
    printf("Enter two integers: ");
```

```
    scanf("%d %d", &number1, &number2);
```

```
    //checks if the two integers are equal.
```

```
    if(number1 == number2) {
```

```
        printf("Result: %d = %d",number1,number2);
```

```
    }
```

```
    //checks if number1 is greater than number2.
```

```
    else if (number1 > number2) {
```

```
        printf("Result: %d > %d", number1, number2);
```

```
    }
```

```
    //checks if both test expressions are false
```

```
    else {
```

```
        printf("Result: %d < %d",number1, number2);
```

```
    }
```

```
}
```

Nested if...else

```
#include <stdio.h>
int main() {
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    if (number1 >= number2) {
        if (number1 == number2) {
            printf("Result: %d = %d", number1, number2);
        }
        else {
            printf("Result: %d > %d", number1, number2);
        }
    }
    else {
        printf("Result: %d < %d", number1, number2);
    }

    return 0;
}
```

If the body of an if...else statement has only one statement, you do not need to use brackets {}.

For example, this code

```
if (a > b) {  
    printf("Hello");  
}
```

```
printf("Hi");  
is equivalent to
```

```
if (a > b)  
    printf("Hello");  
printf("Hi");
```

Example

```
/* increment num_pos, num_neg, or num_zero depending on  
   x */
```

```
if (x > 0)
```

```
    num_pos = num_pos + 1;
```

```
else if (x < 0)
```

```
    num_neg = num_neg + 1;
```

```
else /* x equals 0 */
```

```
    num_zero = num_zero + 1;
```

Class of the Ship

- Each ship serial number begins with a letter indicating the class of the ship. Write a program that reads a ship's first character of serial number and displays the class of the ship.

Class ID	Ship Class
B or b	Battleship
C or c	Cruiser
D or d	Destroyer
F or f	Frigate

Nested if Statements

```
#include<stdio.h>
void main()
{
    char c;
    scanf("%c",&c);
    if ((c=='b')||(c=='B'))
        printf("Battleship");
    else if ((c=='c')||(c=='C'))
        printf("Cruiser");
    else if ((c=='d')||(c=='D'))
        printf("Destroyer");
    else if ((c=='f')||(c=='F'))
        printf("Frigate");
}
```

switch statement

- A **switch** statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each **switch case**.

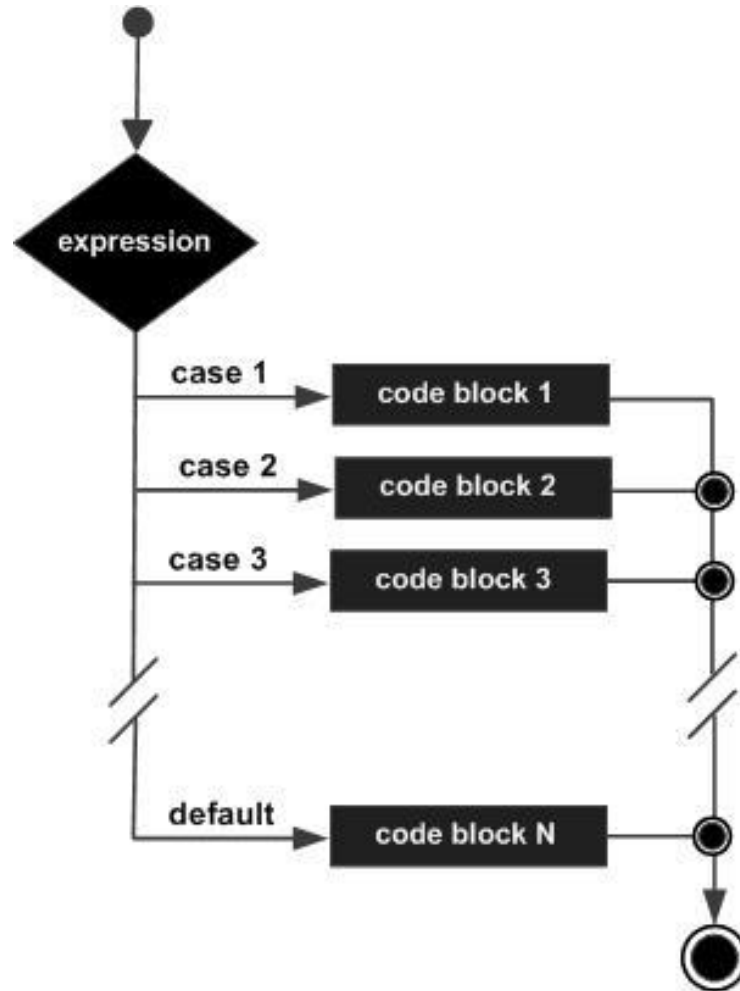
syntax of switch statement

```
switch(Expression) {  
    case 1:  
        //code to be executed;  
        break; //optional  
    case 'b':  
        //code to be executed;  
        //optional  
    case 3:  
        //code to be executed;  
        break; //optional  
    .....  
    default:  
    {  
        code to be executed if all cases are not matched;  
    }  
}
```

Rules for Switch case

- The *switch expression* must be of an **integer or character** type.
- The *case value* must be an integer or character constant.
- You can have any number of case statements within a switch.
- The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case.
- The default clause inside the switch statement is optional. If there is no match, the default statements are executed.

Flow diagram for Switch case



```
int x,y,z;
```

```
char a,b;
```

```
float f;
```

Valid Switch	Invalid Switch	Valid Case	Invalid Case
switch(x)	switch(f)	case 3;	case 2.5;
switch(x>y)	switch(x+2.5)	case 'a';	case x;
switch(a+b-2)		case 1+2;	case x+2;
switch(func(x,y))		case 'x'>'y';	case 1,2,3;

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

```
void main () {  
    char grade = 'B';  
    switch(grade) {  
        case 'A' :  
            printf("Excellent!\n" );  
            break;  
        case 'B' :  
        case 'C' :  
            printf("Well done\n" );  
            break;  
        case 'D' :  
            printf("You passed\n" );  
            break;  
        case 'F' :  
            printf("Better try again\n" );  
            break;  
        default:  
            printf("Invalid grade\n" );  
    }  
    printf("Your grade is  %c\n", grade );  
}
```

```
// Program to create a simple calculator
#include <stdio.h>
void main() {
    char operation;
    double n1, n2;
    printf("Enter an operator (+, -, *, /): ");
    scanf("%c", &operation);
    printf("Enter two operands: ");
    scanf("%lf %lf", &n1, &n2);
    switch(operation)
    {
        case '+':
            printf("%.1lf + %.1lf = %.1lf", n1, n2, n1+n2);
            break;
        case '-':
            printf("%.1lf - %.1lf = %.1lf", n1, n2, n1-n2);
            break;
        case '*':
            printf("%.1lf * %.1lf = %.1lf", n1, n2, n1*n2);
            break;
```

```
    case '/':  
        printf("%.1lf / %.1lf = %.1lf",n1, n2,  
n1/n2);  
        break;  
  
    // operator doesn't match any case  
constant +, -, *, /  
    default:  
        printf("Error! operator is not correct");  
    }  
  
    return 0;  
}
```

```
#include<stdio.h>
void main()
{
char c;
scanf("%c",&c);
switch(c)
{
case 'b':
case 'B':
printf("Battleship");
break;
case 'c':
case 'C':
printf("Cruiser");
break;
case 'd':
case 'D':
printf("Destroyer");
break;
case 'f':
case 'F':
printf("Frigate");
}
}
```



```
#include<stdio.h>
void main()
{
char c;
scanf("%c",&c);
switch(c)
{
case 'b':
case 'B':
printf("Battleship");
case 'c':
case 'C':
printf("Cruiser");
case 'd':
case 'D':
printf("Destroyer");
case 'f':
case 'F':
printf("Frigate");
default:
printf("No match");
}
}
```

Output

BattleshipCruiserDestroyerFrigateNo match

When input is b

break statement

- The break statement ends the loop immediately when it is encountered.
- Its syntax is:

```
break;
```

Conditional Operator (Ternary Operator)

Syntax:

`expression 1 ? expression 2 : expression 3`

“if expression 1 is true (that is, if its value is non-zero), then the value returned will be expression 2, otherwise the value returned will be expression 3”.

Conditional Operator (Ternary Operator)

Example:

```
char a ;
```

```
int y ;
```

```
scanf ( "%c", &a ) ;
```

```
y = ( a >= 65 && a <= 90 ? 1 : 0 ) ;
```

Here 1 would be assigned to y if `a >= 65 && a <= 90` evaluates to true, otherwise 0 would be assigned

Conditional Operator (Ternary Operator)

- Not necessary that conditional operators should be used only in arithmetic statements

Eg1:

```
int i ;  
scanf ( "%d", &i ) ;  
( i == 1 ? printf ( "Amit" ) : printf ( "All and  
sundry" ) )
```

Eg2:

```
char a = 'z' ;  
printf ( "%c" , ( a >= 'a' ? a : '!' ) ) ;
```

Nested

- Conditional operators can be nested :

```
int  big, a, b, c ;
```

```
big = ( a > b ? ( a > c ? 3: 4 ) : ( b > c ? 6: 8 ) ) ;
```

```
a > b ? g = a : g = b ;
```

Compiler Error

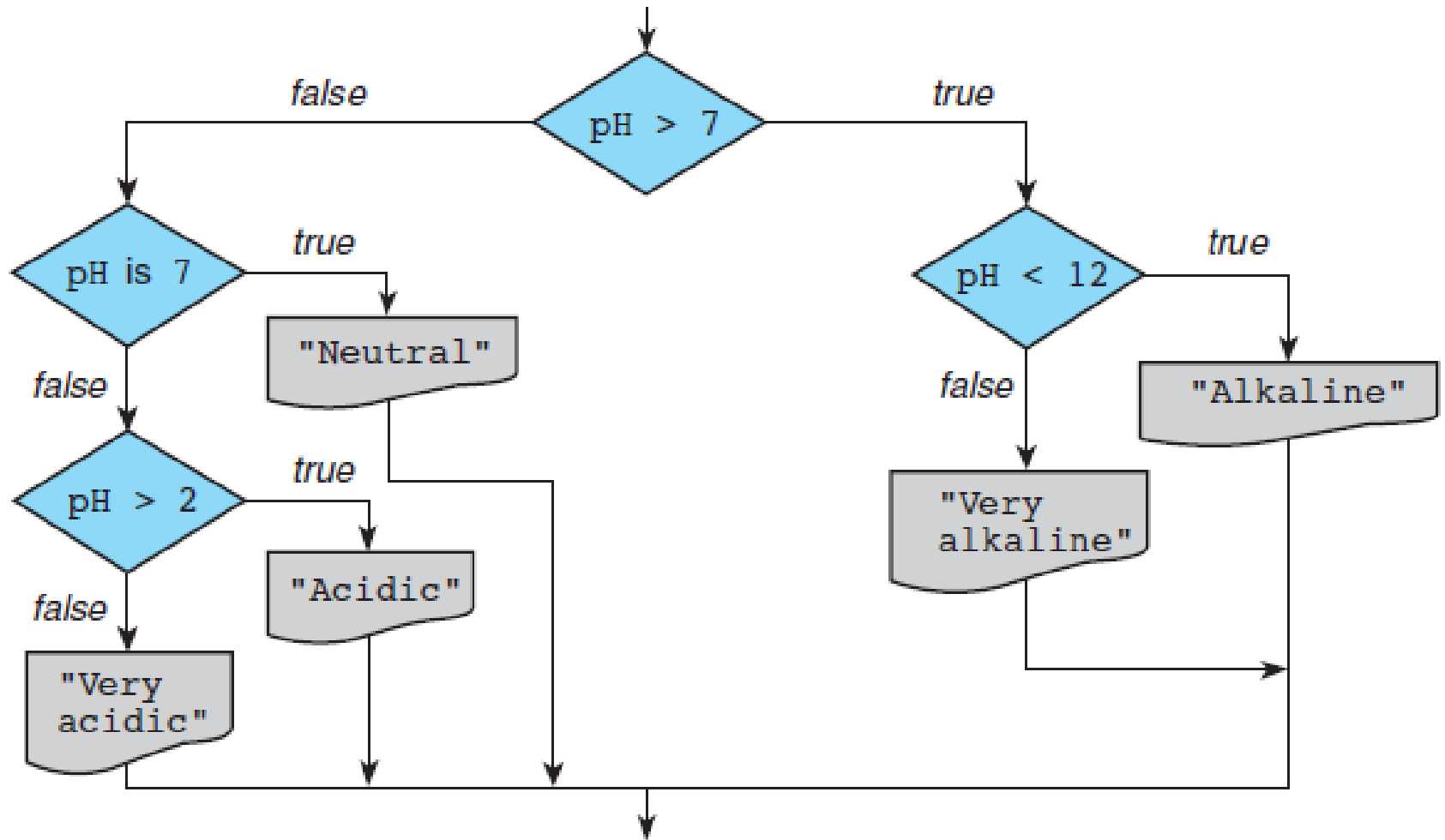
```
a > b ? g = a : ( g = b ) ;
```

Ex: Nature of a Solution

- A solution may be classified into acidic, very acidic, neutral, alkaline or very alkaline based on its pH value. The nature of the solution is given by the following table and determined as shown in the figure:

pH value	Nature of Solution
0 to 2	Very acidic
Greater than 2 and less than 7	Acidic
Equal to 7	Neutral
Greater than 7 and less than 12	Alkaline
Greater than 12	Very Alkaline

Nature of a Solution



pH problem

Input	Output	Alternate Ways for Solution
pH value of solution	Nature of solution	Compare value and make decision

Algorithm

- Get the pH value of the solution from the user
- Write instructions that will make decision for nature of solution as per details in the table
- Print the nature of the solution

Partial C code

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

```
void main()
```

```
{
```

```
float ph_Value; // Declare necessary variables
```

```
//Get the ph_Value from the user
```

```
scanf("%d",&ph_Value);
```

```
// Based on ph_Value make decision and print
```

```
}
```

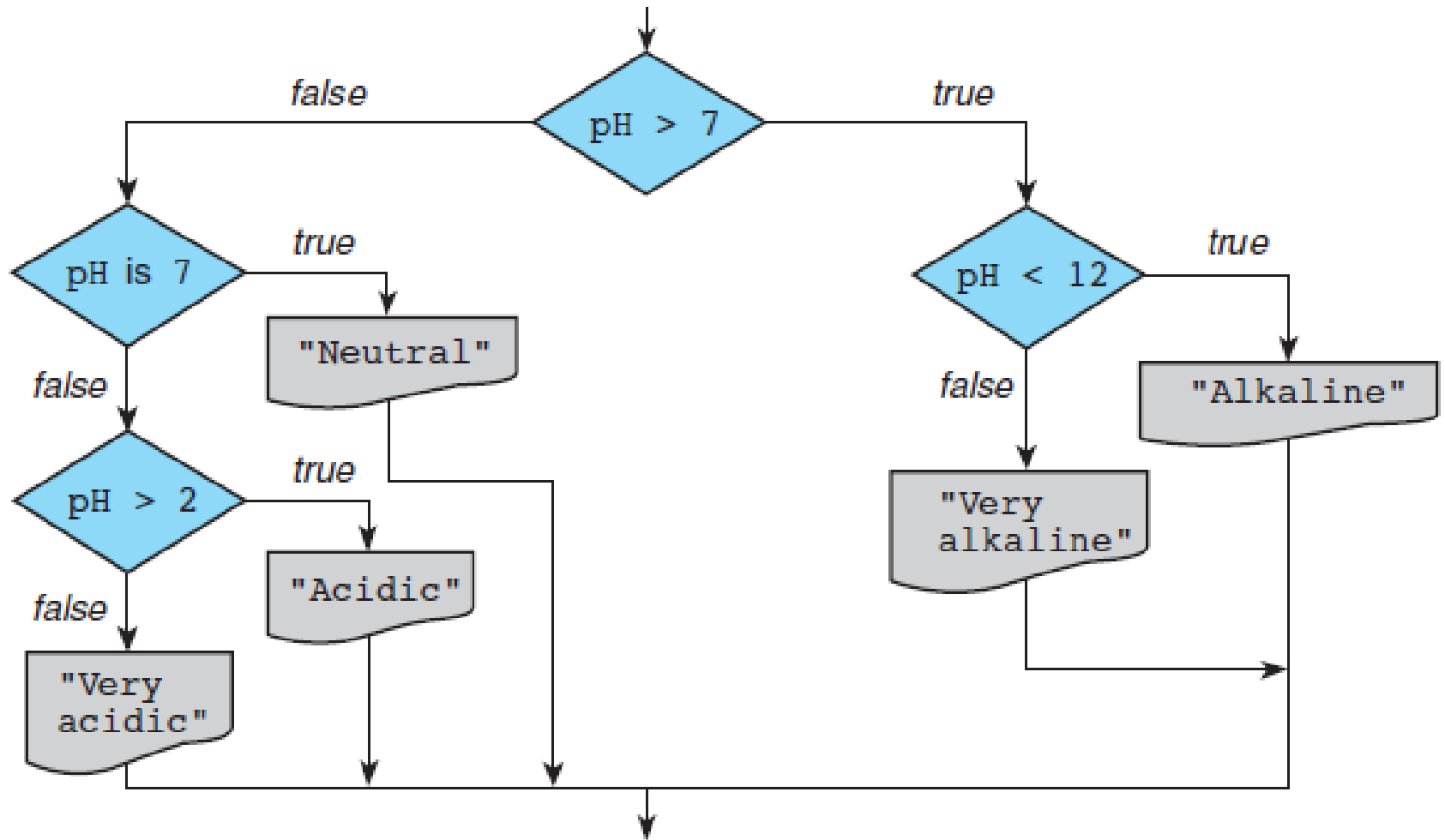
```
#include<stdio.h>
void main()
{
//Declare a variable to store ph value
float ph;
//Read the value of ph
scanf("%f",&ph);
//if ph is from 0 to 2 then print very acidic
if ((ph>=0)&&(ph<=2))
printf("Very acidic");
//if ph is greater than 2 and less than 7 then print acidic
else if((ph>2) &&(ph<7))
printf("Acidic");
//if ph is equal to 7 then print neutral
else if (ph==7)
printf("Neutral");
//if ph is greater than 7 and less than or equal to 12 then print alkaline
else if(ph>7&&ph<=12)
printf("Alkaline");
//if ph is greater than 12 then print Very alkaline
else
printf("Very alkaline");
}
```

Nature of a Solution

- A solution may be classified into acidic, very acidic, neutral, alkaline or very alkaline based on its pH value. The nature of the solution is given by the following table and determined as shown in the figure:

pH value	Nature of Solution
0 to 2	Very acidic
Greater than 2 and less than 7	Acidic
Equal to 7	Neutral
Greater than 7 and less than 12	Alkaline
Greater than 12	Very Alkaline

Nature of a Solution



pH problem

Input	Output	Alternate Ways for Solution
pH value of solution	Nature of solution	Compare value and make decision

Algorithm

- Get the pH value of the solution from the user
- Write instructions that will make decision for nature of solution as per details in the table
- Print the nature of the solution

Partial C code

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

```
void main()
```

```
{
```

```
float ph_Value; // Declare necessary variables
```

```
//Get the ph_Value from the user
```

```
scanf("%d",&ph_Value);
```

```
// Based on ph_Value make decision and print
```

```
}
```

```
#include<stdio.h>
void main()
{
//Declare a variable to store ph value
float ph;
//Read the value of ph
scanf("%f",&ph);
//if ph is from 0 to 2 then print very acidic
if ((ph>=0)&&(ph<=2))
printf("Very acidic");
//if ph is greater than 2 and less than 7 then print acidic
else if((ph>2) &&(ph<7))
printf("Acidic");
//if ph is equal to 7 then print neutral
else if (ph==7)
printf("Neutral");
//if ph is greater than 7 and less than or equal to 12 then print alkaline
else if(ph>7&&ph<=12)
printf("Alkaline");
//if ph is greater than 12 then print Very alkaline
else
printf("Very alkaline");
}
```