



KEMENTERIAN PENGAJIAN TINGGI



# C PROGRAMMING DJM20032

**MECHANICAL**

**ENGINEERING DEPARTMENT**

VERSION 1.0

**E-BOOK**

**NORMAWAR BINTI ALI**



KEMENTERIAN PENGAJIAN TINGGI



# AUTHOR:



**NORMAWAR BINTI ALI**

Lecturer

Mechanical Engineering Department

Politeknik Tuanku Sultanah Bahiyah

All right reserve. No parts of this publication may be reproduce or transmitted in any forms or any means, electronic or mechanical including photocopy, recording or any information storage and retrieval system, without permission in writing from Library of Pol iteknik Tuanku Sultanah Bahiyah, Kul im Hi-Tech Park, 09090 Kulim, Kedah.

Copyright © 2021

eISBN 978-967-0855-94-3

Mechanical Engineering Department

Politeknik Tuanku Sultanah Bahiyah

Kulim Hi-Tech Park,

09090 Kulim,

Kedah

[normawar@ptsb.edu.my](mailto:normawar@ptsb.edu.my)





KEMENTERIAN PENGAJIAN TINGGI



# PREFACE

Alhamdulillah to Allah SWT With His grace and mercy, the first e-book module of DJM20032 C PROGRAMMING has finally completed. Thank you to my family and friends especially PTSB e-book team for being support me to complete this e-book. I also would like to thank you to my Head of Mechanical Engineering Department, En. Azhar Bin Fikri for giving this opportunity. As educator, I feel the need to prepare a e-book module that consist of notes ,step by step practical works and some exercises that can assist the student in the learning process.

The content of this e-book has been constructed to meet the polytechnic's syllabus requirement. Some chapter contains illustrative screenshots to match the main practical work, enhancing student comprehending ability. This e-book contains 5 chapter such as Introduction to Programming Concepts, Structure of C Programmes, Data Input Ouput, COnrol System, Function and also variety of exercises and examples

Any positive feedback from lecturer and student are mostly welcome and appreciated. Hopefully this e-book module is one of the step that start the long journey of road to excellent.



KEMENTERIAN PENGAJIAN TINGGI

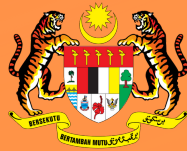


# SYNOPSIS

C PROGRAMMING course provides an instruction to program design and development. Student will learn to design, code, debug, test and document well structured programs based on technical and engineering problem. Topic covered : software development principle, programming language basic, data types, input and output operation, the use of selection, loops, arrays and function structure.

Upon completion of this course, student should be able to:

1. Explain knowledge of basic concepts of C Programming to solve given problem using an appropriate data type. (C2, PLO1)
2. Constructs a high level programming language in solving variety engineering and scientific problems. (P3, PLO3)
3. Present a solution for assigned project based on programming which relates to current or upcoming technologies and peripherals. (A2, plo12)



KEMENTERIAN PENGAJIAN TINGGI

# TABLE OF CONTENTS

## TOPICS

## PAGES

**1**

Introduction to  
Programming Concepts

**1 - 27**

**2**

Structure of  
C Programmes

**28 - 42**

**3**

Data Input  
Output

**43 - 58**

**4**

Control  
Statement

**59 - 89**

**5**

Function

**90- 107**

**6**

Question and Exercise

**108 - 140**



# CHAPTER 1

# INTRODUCTION

TO

# PROGRAMMING CONCEPTS

# Types of Programming

# Problem Solving

# Constants & Variables

# Data Types

# Operators & Expressions



# INTRODUCTION TO COMPUTERS

In a layman language, a computer is a fast calculating device which can perform arithmetic operations.

Although the computer was originally invented mainly for performing high speed and accurate calculations, it is not just a calculating device.

## Important terms used:

**Data** - A set of basic facts and entities that itself has no meaning.

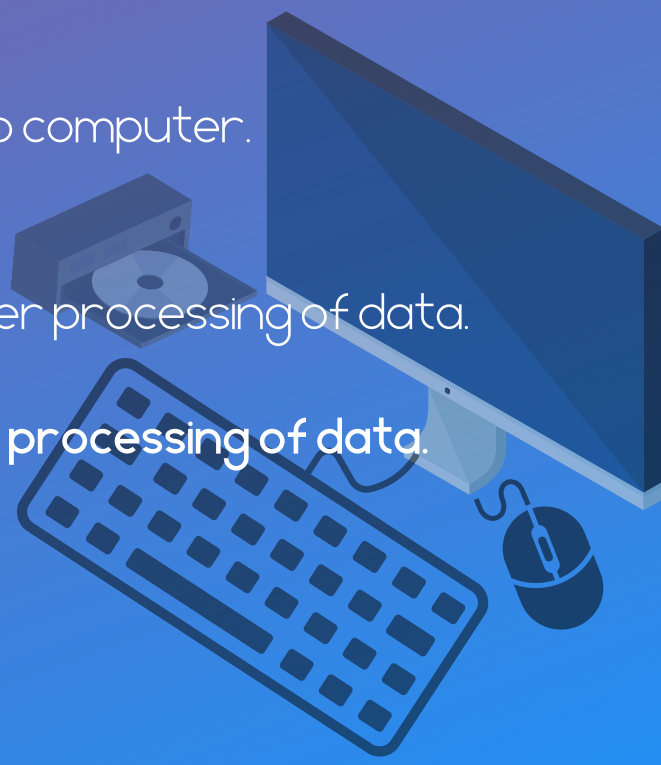
**Information** - Data which has some meaning or value for the user.

**Instruction** - Data which has some meaning or value for the user.

**Input** - Data and instructions given to computer.

**Process** - Manipulation obtained after processing of data.

**Output** - Information obtained after processing of data.



# WHO IS A PROGRAMMER?

A computer programmer writes, tests, and maintains software and programs that instruct the computer as to what it should do.



Computer programmers convert what needs to happen into computer language, so that the computer can “understand” it.

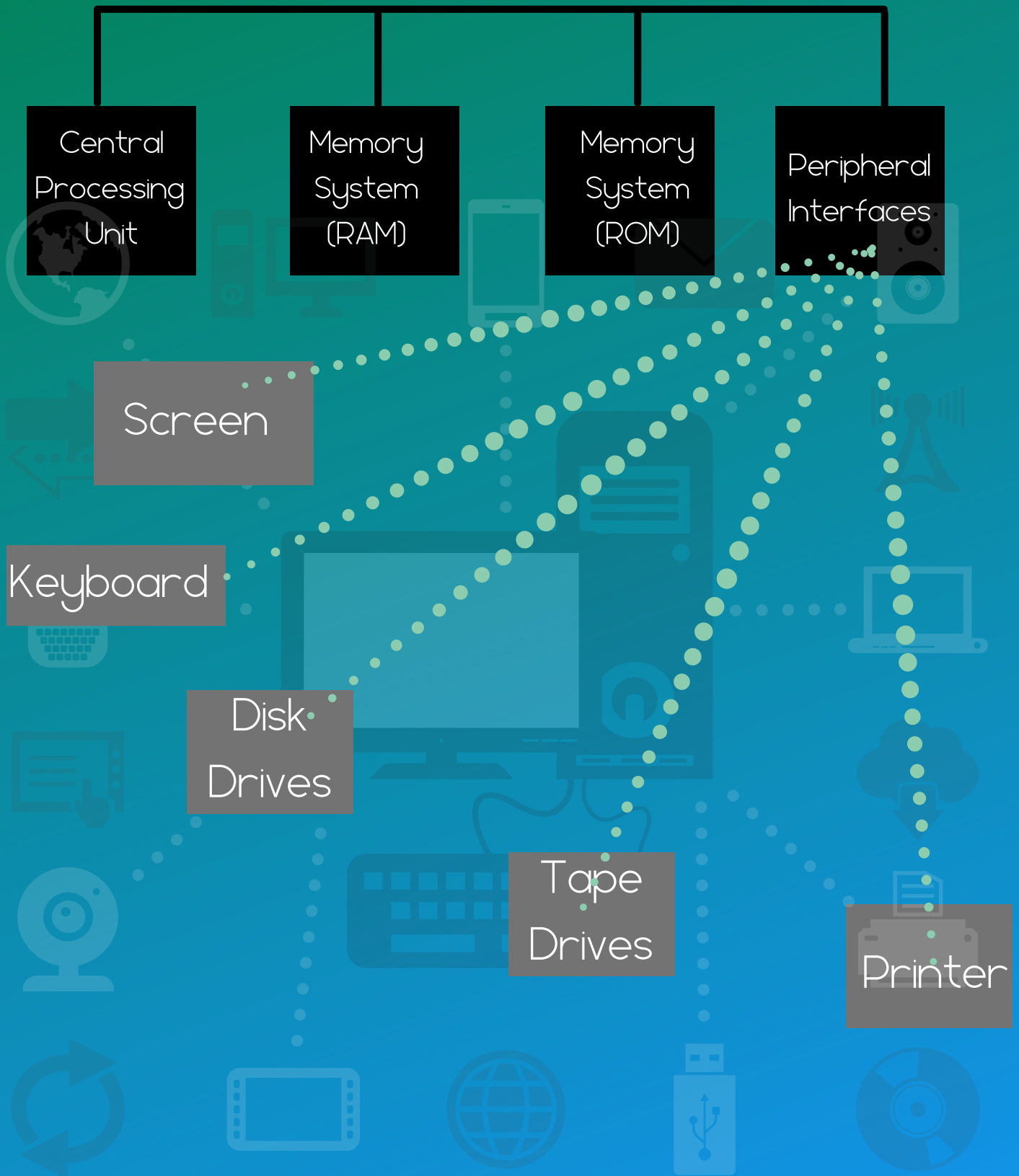
A computer programmer writes and develops computer programs to accomplish certain tasks, to store data or documents, and locate and retrieve that data or those documents.



Computer programmers code instructions for the computer into one of the many computer languages in existence.



# BASICS OF COMPUTER AND ITS OPERATIONS



# BASICS OF COMPUTER AND ITS OPERATIONS

The different parts are linked together in some way by the bus, so that information can pass between them.

To be able to write programs, we need a "model" of the computer system.

## Central Processing Unit

It contains the hardware resources required to execute instructions.

## Memory System (RAM)

Simply a mechanism in which information can be stored. The only operations that can be performed on memory are reading information from it or writing information to it.

## Peripheral Interfaces

Devices external to the computer system. They are connected to the system through gateways called interfaces.

## Microprocessor

Simply a CPU on a small number of integrated circuit package.

## Single Chip Microcomputer

Complete computer system on one integrated circuit package.

# BASICS OF COMPUTER AND ITS OPERATIONS

Regardless of type and size, all computer follow same four main operations.



Input Operation



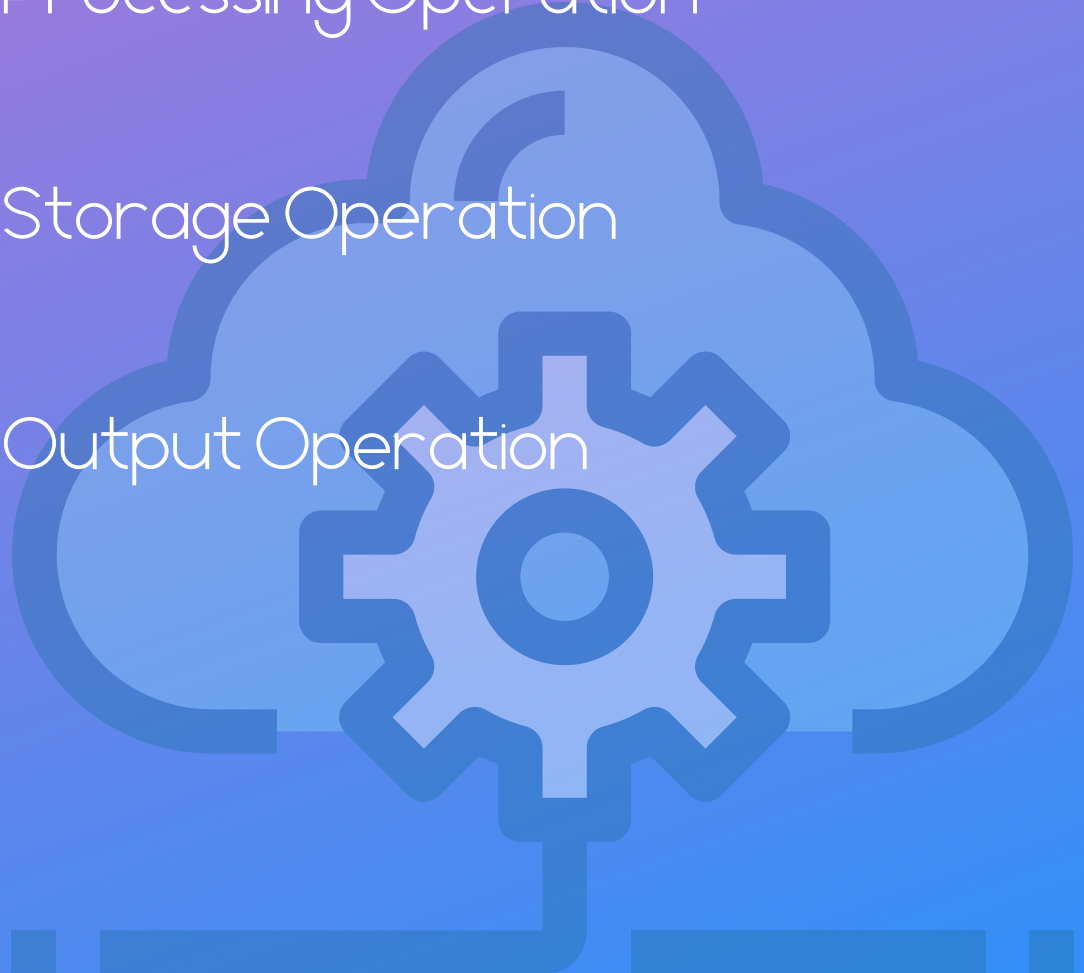
Processing Operation



Storage Operation



Output Operation





# BASICS OF COMPUTER AND ITS OPERATIONS



## Input Operation

Input is whatever is put in (input) to a computer system. Input can be nearly any kind of data.. Which is letters, numbers, symbols, shapes, colours, sound or whatever raw material needs processing. When user types some words or numbers on a keyboard, these words are considered input data.



## Processing Operation

Processing is the manipulation, a computer does to transform data into information. The processing is done by the Central Processing Unit(CPU).

# BASICS OF COMPUTER AND ITS OPERATIONS

## 3 Storage Operation

Storage is of two types which primary storage and permanent storage. Primary storage or memory is the computer circuitry that temporarily hold data waiting to be processed.

Secondary storage is the area in the computer where data or information is held permanently. A hard disk is an example of this kind storage.

## 4 Output Operation

Output is whatever is output from the computer system, the result of processing usually information. For example, numbers or pictures displayed on the monitor, words printed out on paper using a printer.

# LIST THE VARIOUS TYPES OF PROGRAMMING LANGUAGE

1

## Machine code or machine languages

A sequence of 0's and 1's giving machine specific instructions

Example: 00011001

2

## Assembly language

Using meaningful symbols to represent machine code.

Example: addhl, de

Assembler : Assembly code à machine code

Disassembler : machine code à assembly code

3

## High-level languages

Similar to everyday English and use mathematical notations (processed by compilers or interpreters)

Example of a C statement:

a = a + 8;

## List of Programming Language

C++  
COMIT

CobolScript  
Combined Programming Language



# ADVANTAGES & DISADVANTAGE OF C PROGRAMMING

"C" is the language's entire name and it does not stand for anything. Developed at Bell Laboratories in the early 1970's by Dennis Ritchie, C is a general purpose, compiled language that works well for microcomputers and is portable among many computers.

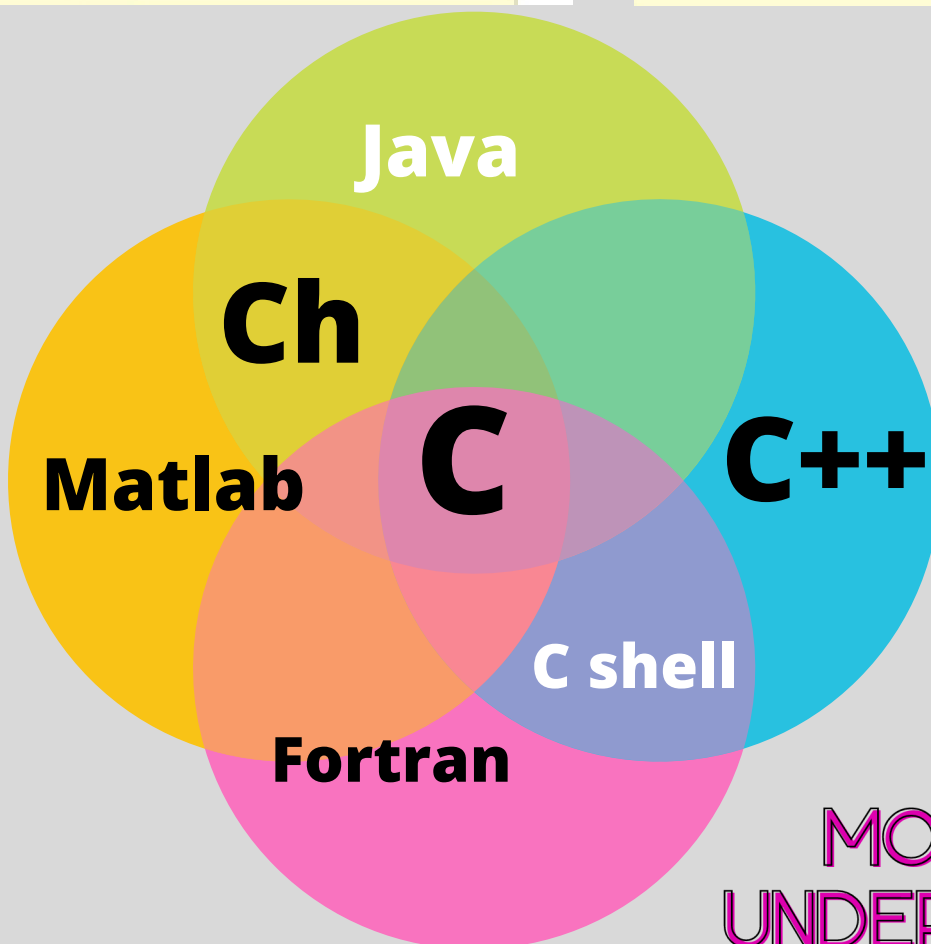
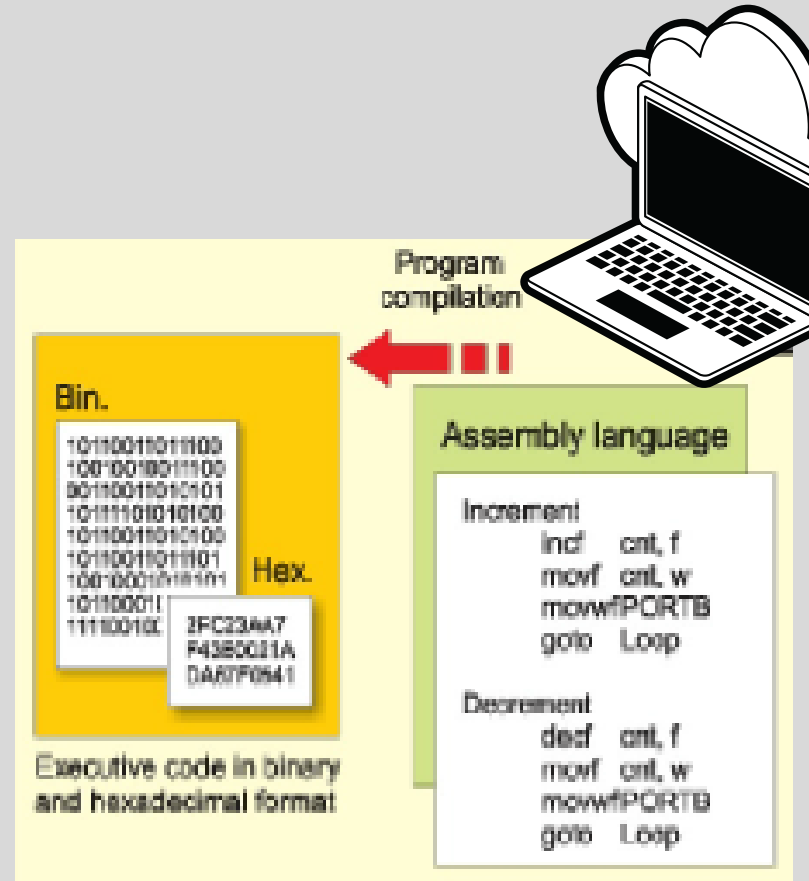
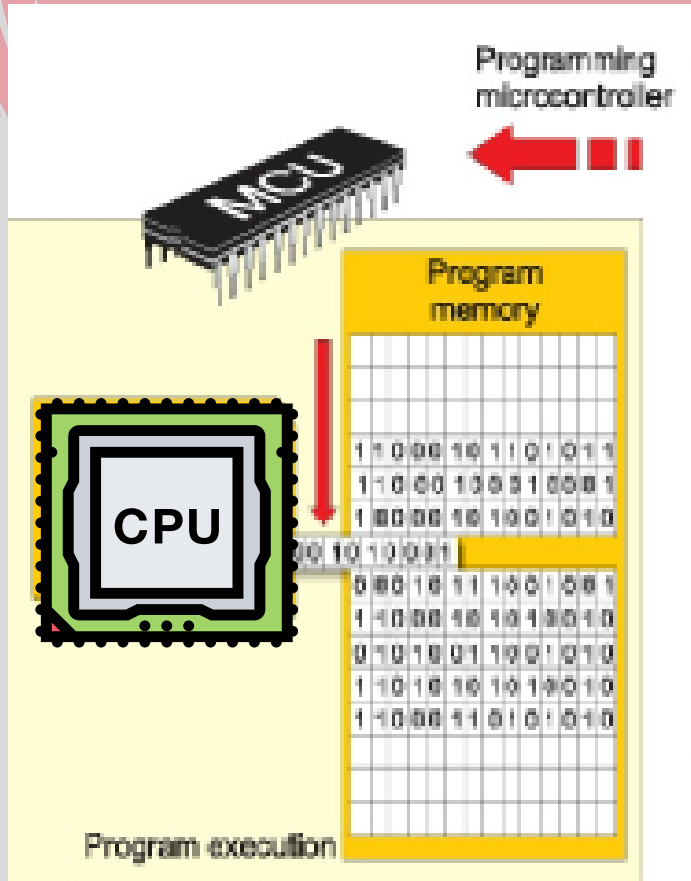
## Advantages

- C is flexible, high level, structured programming language.
- C includes many low level features that are normally available only in assembly or machine language.
- C programs are very concise, due to the large number of operators within the language.
- C is weakly typed language and the programs written in it compile into small object programs that run or execute efficiently.

## Disadvantages

- C is considered difficult to learn.
- Because of its conciseness, the code can be difficult to follow.
- It is not suited to application that require a lot of report formatting and data file manipulation.

# LEVELS OF PROGRAMMING LANGUAGES



MORE HUMAN UNDERSTANDABLE

# LEVELS OF PROGRAMMING LANGUAGES

## Machine Language

written in either  
binary numbers  
(0 and 1)  
or hexadecimal (0 to F)

## Assembly Language

Symbolic language  
that uses symbols to  
represent computer  
instructions.

## High Level Language

Written in English & easy  
to understand

## Assembler

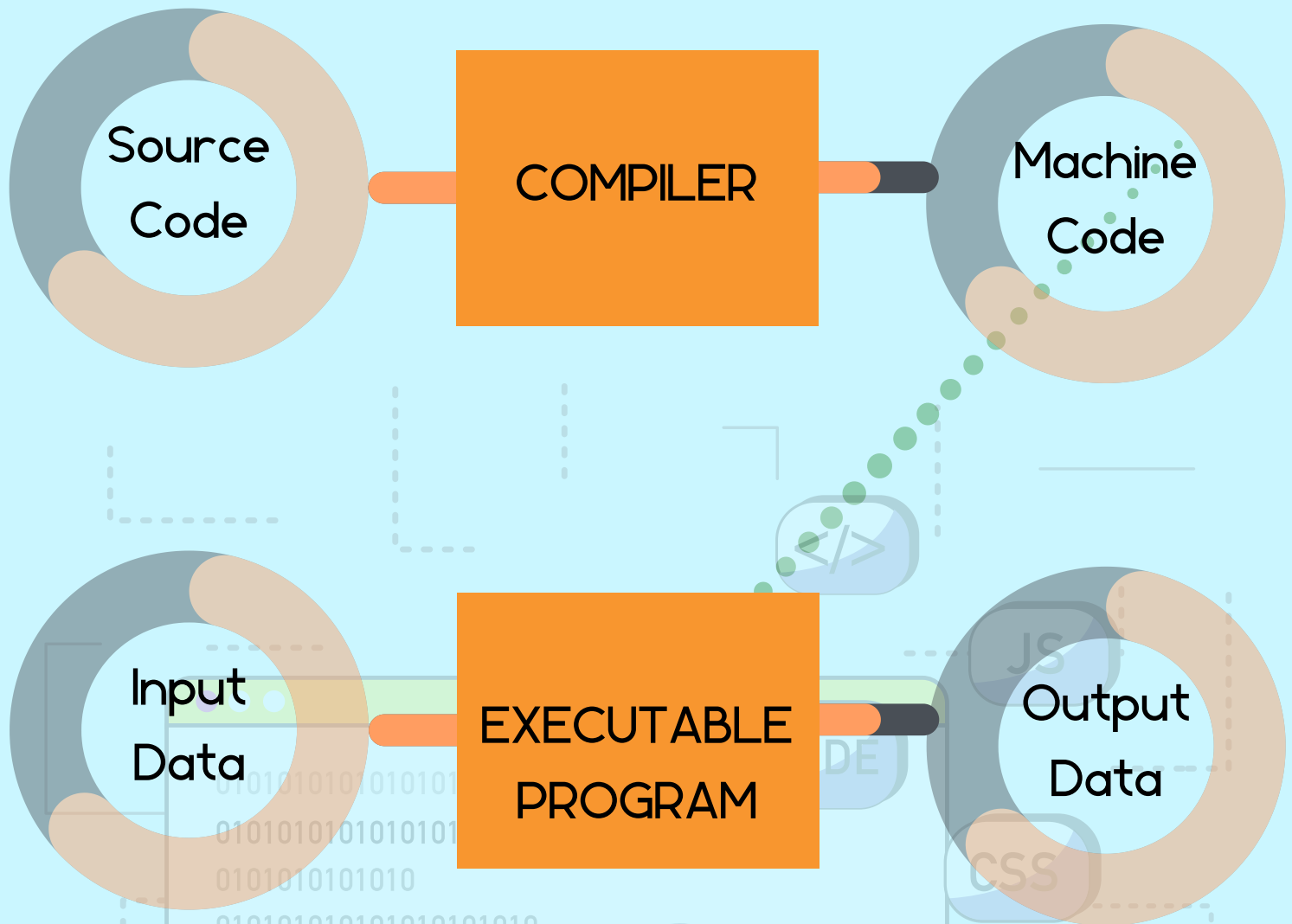
A software package used to convert assembly  
language into machine language.

Translation of highlevel language into machine  
instruction done by special computer program  
which **compilers** or **interpreters**.





# COMPILER AND INTERPRETER



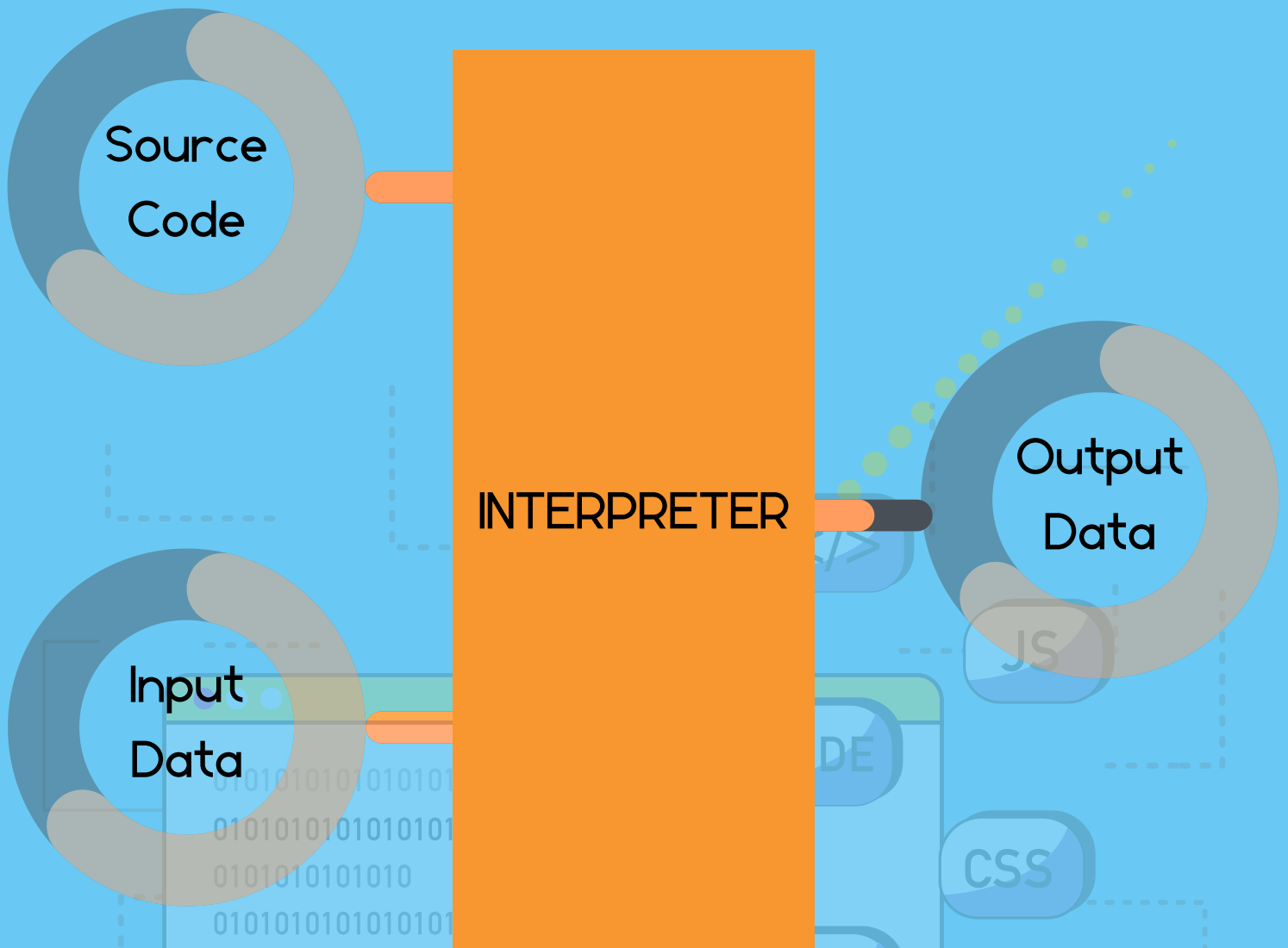
## COMPILER

Analyzes program and translates it into machine language.

## EXECUTABLE PROGRAM

Can be run independently from compiler as many times => fast execution.

# COMPILER AND INTERPRETER



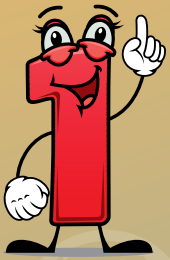
## INTERPRETER

Analyzes and executes program statements at the same time.

## EXECUTABLE PROGRAM

Execution is slower and easier to debug program.

# TYPES OF PROGRAMMING



## Structured Programming

Use control structures with only one entry point and one exit point

The most important of these structures are sequence, selection(if and if..else)and repetition (while).

**Example:**

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

```
int main()
```

```
{
```

```
printf("Hello,world\n");
```

```
return 0;
```

```
}
```



# TYPES OF PROGRAMMING



## Modular Programming

Modular programming is subdividing your program into separate subprograms such as functions and subroutines.

Then if some one else wants to compute a different solution using your program, Only these subroutines need to be changed.

This is a lot easier than having to read through a program line by line, trying to figure out what each line is supposed to do and whether it needs to be changed.

Control Module

Module

Module

Module

Module

SubModule

LibraryModule

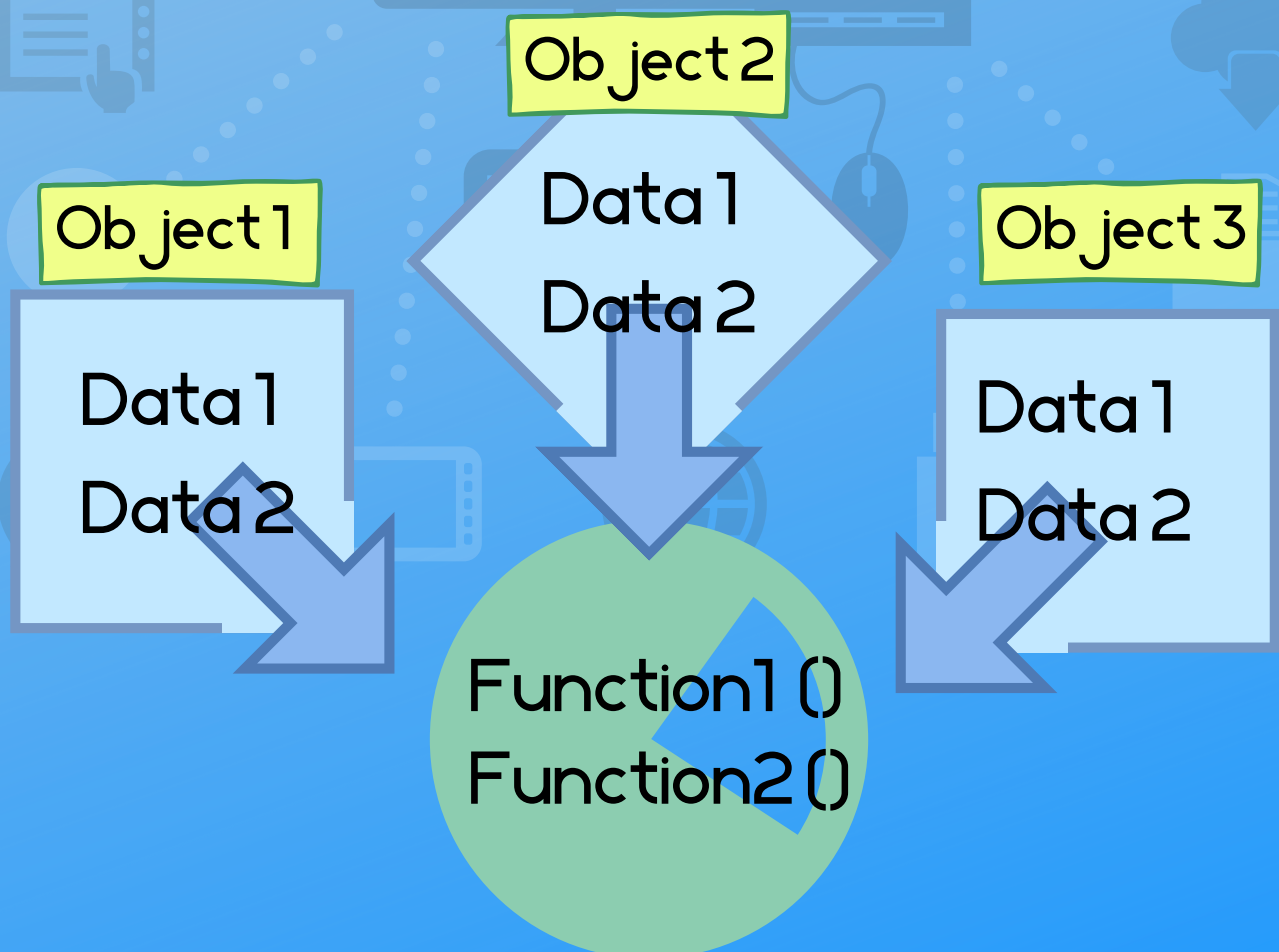
# TYPES OF PROGRAMMING



## Object Oriented Programming

Object Orientation is a set of tools and methods that enable software engineers to build reliable, user friendly, maintainable, well documented, reusable software systems that fulfill the requirements of its users.

In the older styles of programming, a programmer who is faced with some problem must identify a computing task that needs to be performed in order to solve the problem.

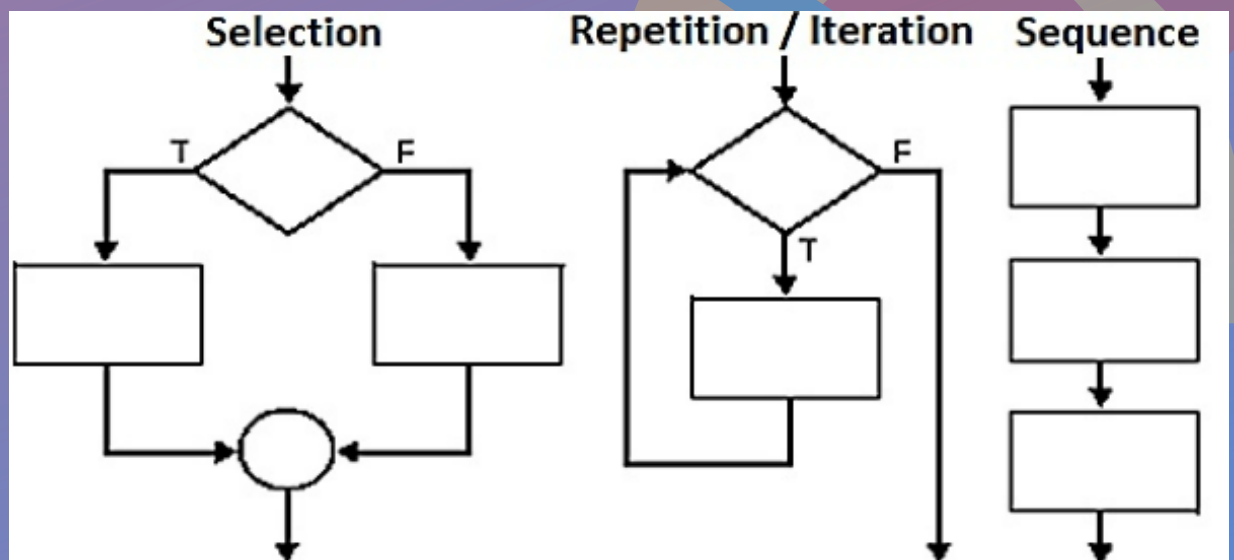


# STRUCTURED PROGRAMMING

Structured Programming is easier than unstructured program to understand, test, debug and modify programs.

## Rules for Structured Programming

- Rules developed by programming community
- Only single entry/single exit control structures are used
- Rules:
  1. Begin with the simplest flowchart
  2. Any rectangle (action) can be replaced by two rectangles (actions) in sequence.
  3. Any rectangle (action) can be replaced by any control structure ( sequence, if, if/else, switch, while, do/while or for)
  4. Rules 2 and 3 can be applied in any order and multiple times.





# PROBLEM SOLVING

**Solving problems** is the core of computer science. Programmers must first understand how a human solves a problem, then understand how to translate this "algorithm" into something a computer can do and finally how to "write" the specific syntax to get job done.

## Problem Solving Phase

1. Analysis & Specification
  - a. Define problem & what solution
2. General Solution (Algorithm)
  - a. Develop logical sequence of step to solve problem
3. Verify



## Problem Solving Phase

1. Specific Solution (Program)
  - a. Translate algorithm to code
2. Test - Check result manually

## Maintenance Phase

1. Use the program
2. Maintain
  - a. Modify to meet changed requirement or to correct errors.

# PROBLEM SOLVING STAGES

Developing a program involves steps similar to any problem solving task. This phase requires 3 stages:

1. Analyzing the problem - Problem Analysis Chart (PAC)

2. Creating general solutions

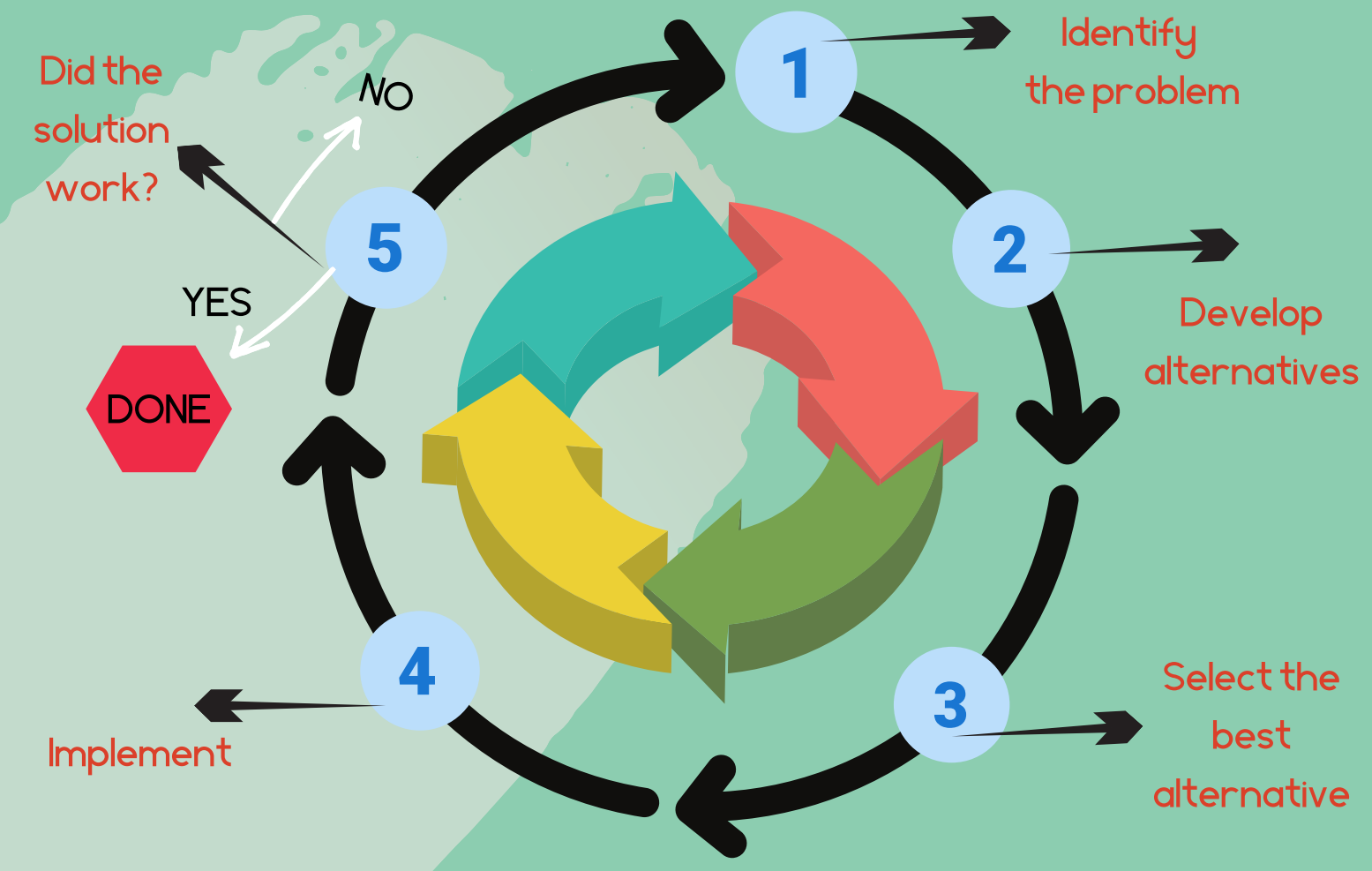
a. Developing the Hierarchy Input Process Output (HIPO) chart or Interactivity Chart (IC)

b. Developing the Input Process Output (IPO) Chart

c. Drawing the Program Flowcharts

d. Writing the Algorithms

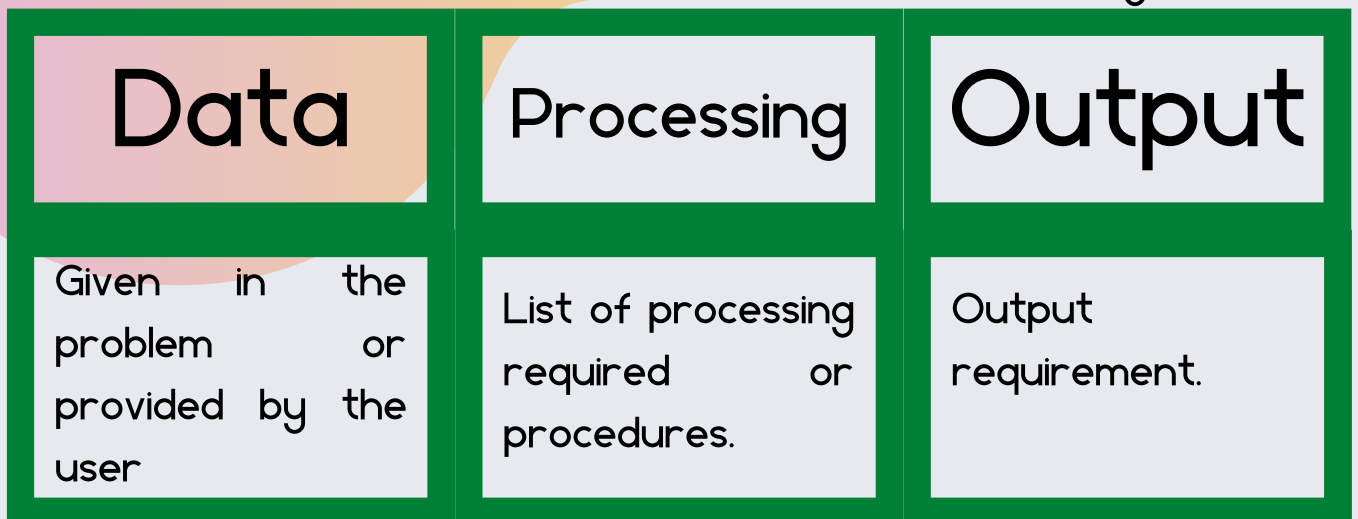
3. Verify that your solution really solves the problem



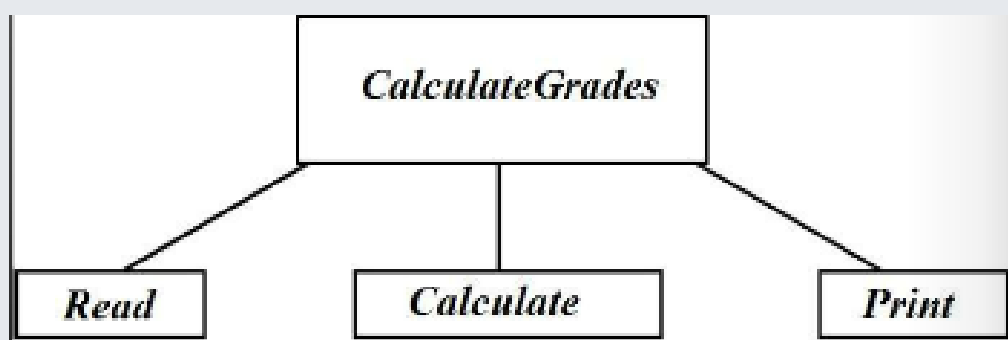
# Problem Analysis Chart

- Understand and analyze the problem to determine whether it can be solved by a computer.
- Analyze the requirement of the problem.
- Identify the following:
  - Data requirement
  - Processing requirement or procedures that will be needed to solve the problem.
  - The output.
- All these requirement can be presented in:

Problem Analysis Chart



Hierarchy Chart





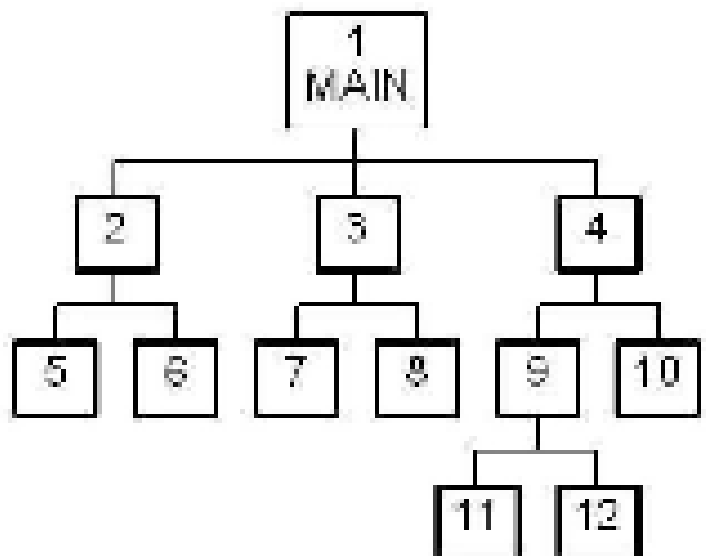
# Hierarchy Output Process Output (HIPO) / Interactivity Chart (IC)

- The problem is normally big and complex and requires big program.
- The processing can be divided into subtasks called modules. Each module accomplishes one function.
- These modules are connected to each other to show the interaction of processing between the modules.
- Main module controls the flow all other modules.
- The (IC) is developed using top down method: top to down left to right order.
- Modules are numbers, marked for duplication, repetition or decision.

## Structure diagrams HIPO diagrams

Contents:

1.  
2.  
3.  
4.  
5.  
6.  
7.  
8.  
9.  
10.  
11.  
12.



# Input Process Output (IPO) Chart

- Extends and organizes the information in the Problem Analysis Chart.
- It shows in more detail what data items are input, what are the processing or modules on that data and what will be the result or output.
- It combines information from PAC and HIPO Chart.






IPO Chart

Input	Processing	Module	Output
All input data from PAC	All processing steps from HIPO/IC	Module reference from the IC	All output data from PAC

# Flow Charts & Pseudo Code

## Drawing the Program Flowchart

- Flowchart is the graphic representations of the individual steps or actions to implements a particular module.
- Flowchart is independent of any programming language.
- Flowchart is the logical design of a program.
- Flowchart serves as documentation for computer program.
- The flowchart must be drawn according to definite rules and utilizes standard symbols adopted internationally.

Symbol	Name	Function
	Start/end	An oval represents a start or end point.
	Arrows	A line is a connector that shows relationships between the representative shapes.
	Input/Output	A parallelogram represents input or output.
	Process	A rectangle represents a process.
	Decision	A diamond indicates a decision.



# Flow Charts & Pseudo Code

## Writing the Algorithm (Pseudo Code)

- Pseudo code means an imitation computer code.
- It is used in place of symbols or a flowchart to describe the logic of a program. Thus it is a set of instructions to describe the logic of a program.
- Pseudo code is close to the actual programming language.
- Using the pseudo code, the programmer can start to write the actual code.

Start

Read price, quantity

Sale = price x quantity

Print Sale

End

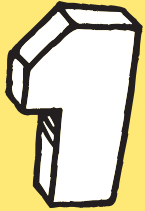
### Pseudocode

```
PROGRAM PrintDoubleNumber:  
    Read A;  
    B = A*2;  
    Print B;  
END .
```



# Various Types of ERROR

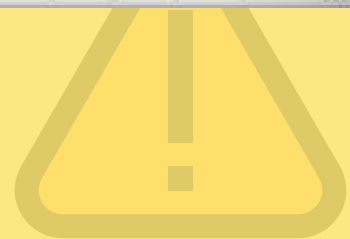
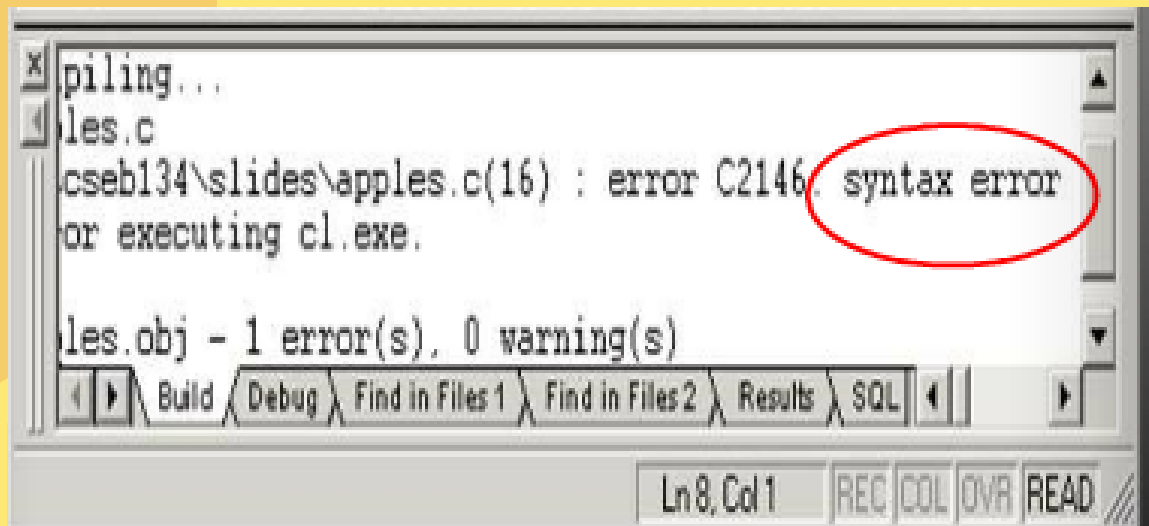
THREE (3) kinds of errors:



## SYNTAX ERROR

- A violation of the C grammar rules, detected during program transition.
- Statement cannot be translated and program cannot be executed

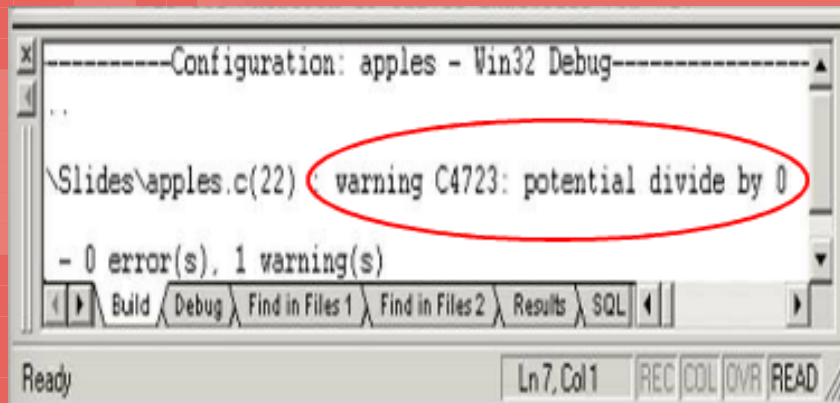
```
#include <stdio.h> ;  
printf('I like programming.');
```



# 2

## RUN TIME ERROR

- An attempt to perform invalid operation, detected during program execution.
- Occurs when the program directs computer to perform an illegal operation such as dividing a number by zero.
- The computer will stop executing the program and displays a diagnostic message indicates the line where the error was detected.



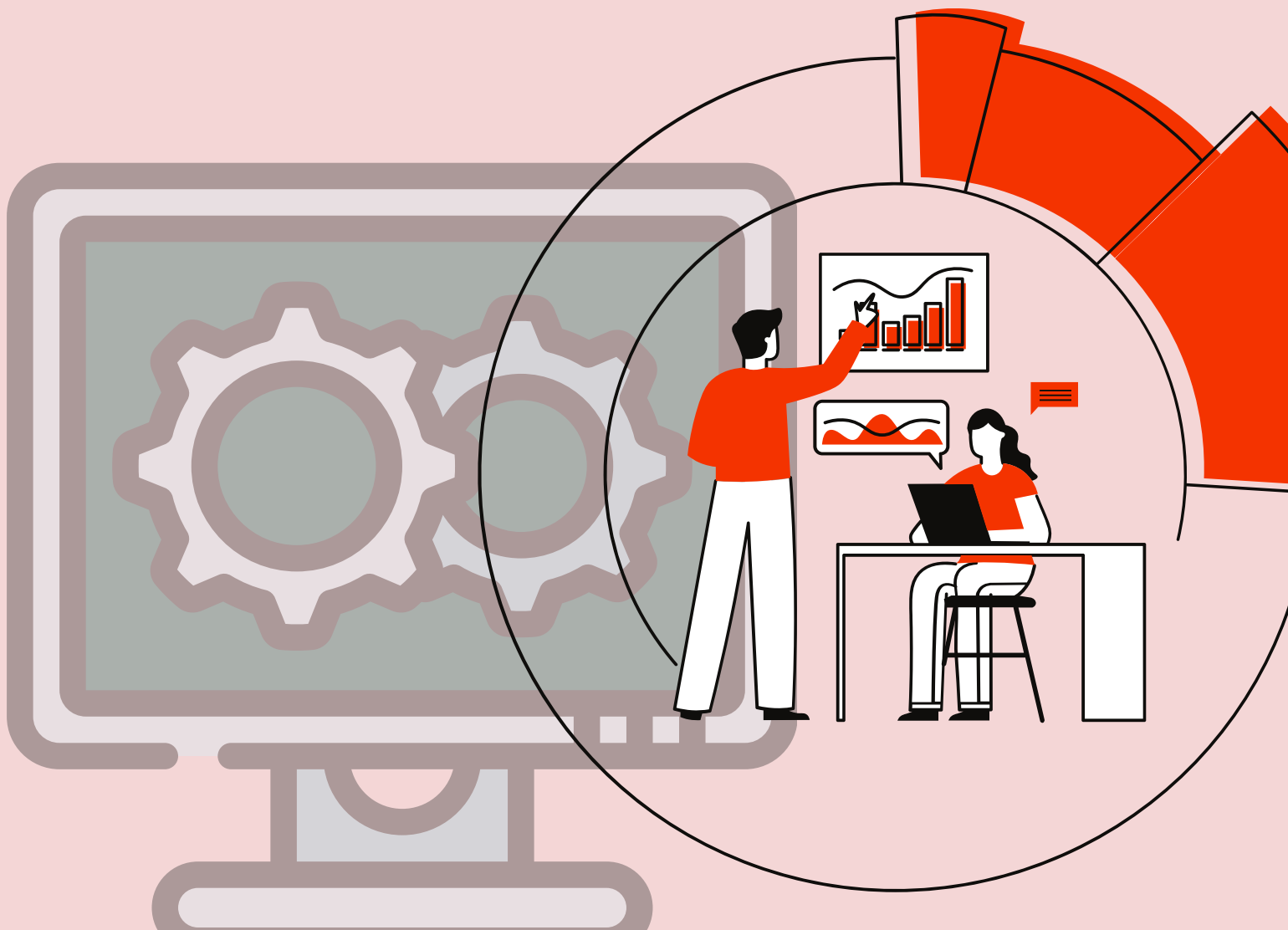
# 3

## LOGIC ERROR / DESIGN ERROR

- An error caused by following an incorrect algorithm.
- Very difficult to detect.
- The only sign of logic error.
- Can be detected by testing the program

# CHAPTER 2

## STRUCTURE OF C PROGRAMMES



# General Form of C Programmes

```
/*
 * Converts distances from miles to kilometers.
 */
#include <stdio.h>          /* printf, scanf definitions */
#define KMS_PER_MILE 1.609 /* conversion constant */

int
main(void)
{
    double miles, /* distance in miles */
           kms;   /* equivalent distance in kilometers */

    /* Get the distance in miles. */
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    /* Convert the distance to kilometers. */
    kms = KMS_PER_MILE * miles;

    /* Display the distance in kilometers. */
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Diagram labels and arrows:

- preprocessor directive** points to `#include <stdio.h>` and `#define KMS_PER_MILE 1.609`.
- constant** points to `1.609`.
- reserved word** points to `int` and `main(void)`.
- variable** points to `miles` and `kms`.
- comment** points to `/* Converts distances from miles to kilometers. */` and `/* Get the distance in miles. */`.
- standard identifier** points to `printf` and `scanf`.
- special symbol** points to `*` in `KMS_PER_MILE * miles` and `\n` in `printf("That equals %f kilometers.\n", kms);`.
- punctuation** points to `(0)` in `return (0);` and `;` in `scanf("%lf", &miles);` and `printf("That equals %f kilometers.\n", kms);`.
- reserved word** points to `return` in `return (0);`.





# Preprocessor Directives



Commands that gives instructions to the C preprocessor



C preprocessor modifies the text of C program before it is compiled



Begins with #



Two most common ones are #include and #define



Contains collections of useful functions and symbols called the libraries

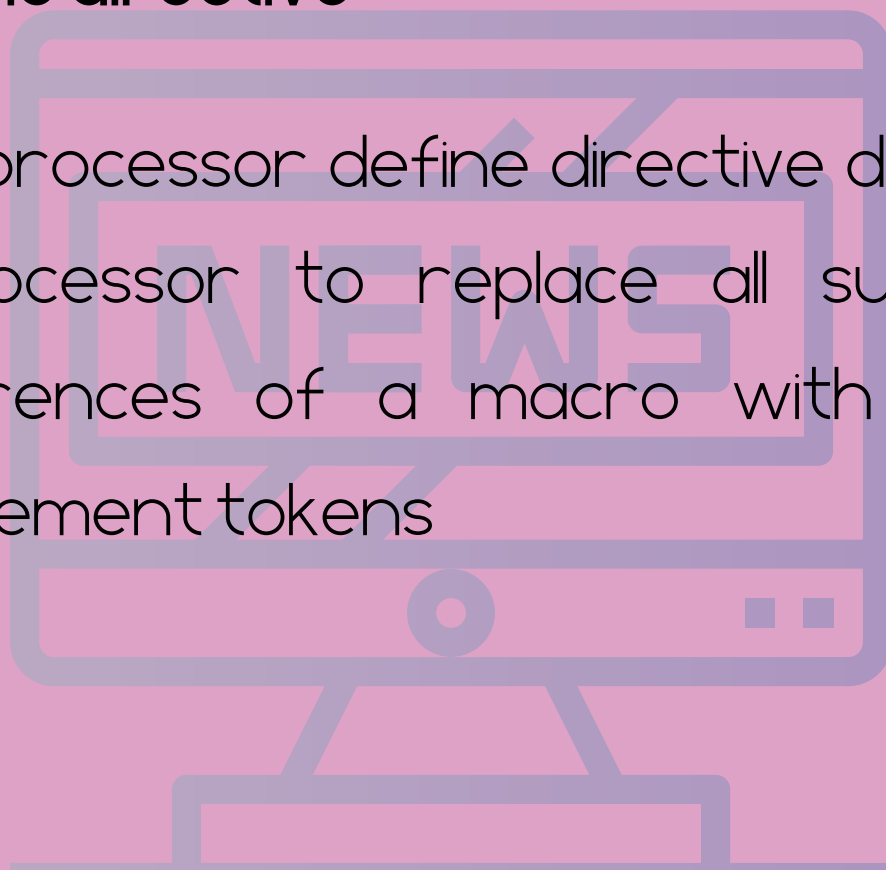
# Preprocessor Directives

## **#include <file>**

This variant is used for system header files. It searches for a file named file in a list of directories specified by you, then in a standard list of system directories

## **#define directive**

A preprocessor define directive directs the preprocessor to replace all subsequent occurrences of a macro with specified replacement tokens



# Preprocessor Directives



Each library has a standard header file whose name ends **with .h**



**#include** gives a program access to a library



**#include** <stdio.h>



**#define** KMS\_PER\_MILE 1.609



# Header Files

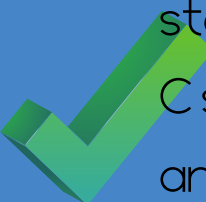
- 1 These files allow programmers to separate certain elements of a program's source code into reusable files.
- 2 With a header file the related declarations appear in only one place.
- 3 If they need to be changed, they can be changed in one place and programs that include the header file will automatically use the new version when next recompiled.

```
/*Save this file as myfunction.h*/  
  
#include <stdio.h>  
#include <conio.h>  
  
void display()  
- {  
    printf("Hi..this is from my function\n");  
}
```

```
/*Save this as testingmyfunction.c*/  
  
#include <myfunction.h>  
  
void main()  
- {  
    display();  
}
```




# stdio.h



stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations.



All library functions in C (and its many derivatives) are declared in header files.



Thus, programmers have to include the `stdio.h` header in the source code in order to use the functions declared in it.

```
/* Copyright (c) 2001-2009 by SoftIntegration, Inc. A  
- /* a sample program that writes a string  
to stdout. */  
#include <stdio.h>  
#include <stdlib.h>  
  
- int main() {  
    char str[] = "This is a test.\n";  
    char *p;  
  
    p = str;  
    for (; *p; p++) putchar(*p);  
}
```

# conio.h

1 conio.h header contains functions for console input/output.

2 Some of the most commonly used functions of conio.h are clrscr, getch, getche, kbhit etc.

3 Functions of conio.h can be used to clear screen, change color of text and background, move text, check if a key is pressed or not and many more.

```
/* Copyright (c) 2001-2009 by SoftIntegration, Inc. All rights reserved.
#include <conio.h>

- int main() {
    int i;
    unsigned short us;
    unsigned long ul;
    char msg1[] = "Please type a letter: ";

    _cputs(msg1);
    i = _getch();
    _cputs("Your input is ");
    _putch(i);
    _putch('\n');

    return 0;
}
```

# math.h

math.h contains functions to perform mathematical operations often required in c programs such as calculating absolute value of a number,


calculating logarithms and using trigonometric functions to calculate sin, cos of a number.

```
/* Copyright (c) 2001-2009 by SoftIntegration, Inc. A
#include <math.h>
#include <stdio.h>

- int main () {
    double z = 16;
    printf("sqrt(z) = %f\n", sqrt(z));
}
```

# Ctype.h

This header declares a set of functions to classify and transform individual characters. There are two sets of functions:



Set of classifying functions that check whether the character passed as parameter belongs to a certain category.



Two functions to convert between letter cases.

```
/* Copyright (c) 2001-2009 by SoftIntegration, Inc. All Rights Reserved */
/* a sample program testing islower() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("islower('%c') returns %d\n", c, islower(c));
    if(islower(c) > 0)
        printf("'c' is a lower case character.\n", c);
    else
        printf("'c' is not a lower case character.\n", c);
    c = 'a';
    printf("islower('%c') returns %d\n", c, islower(c));
    if(islower(c) > 0)
        printf("'c' is a lower case character.\n", c);
    else
        printf("'c' is not a lower case character.\n", c);
}
```

```
/* Copyright (c) 2001-2009 by SoftIntegration, Inc. All Rights Reserved */
/* a sample program that displays uppercase of the letter. */
#include <stdio.h>
#include <ctype.h>

int main() {
    char c = 'a';
    printf("toupper('%c') = %c\n", c, toupper(c));
}
```

# string.h

Standard C library to manipulate C strings

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

- int main() {
    char s[]="123456789";
    char *result;
    char c;

    c = '5';
    result = index(s,c);
    printf("the string behind 5 is %s \n",result);
}
```

## Comments

Text beginning with /\* and \*/

Provides supplementary information about the program

Ignored by the preprocessor and compiler

Use comments to describe the purpose of the program

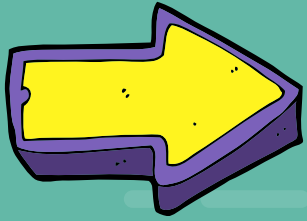
Part of the program documentation

Can appear in itself or at the end of a line

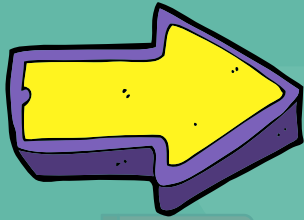
Each program should begin with a header section



# Function main



`int main (void)` marks the beginning of a program.



`{... }` is the body of the program

A function (main) has two parts which is:

**1**

**Declarations:**

tells what memories are required

**2**

**Executable statements:**

tells the computer what to do

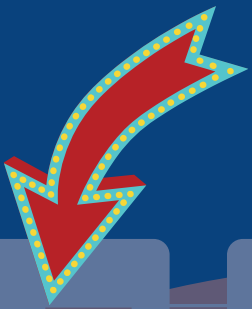
Example Program



```
int main (void)
{
    int a;
    int b;
    int c;
    a = 10;
    b = 20;
    c = a + b;
    printf("%d\n", c);
    return(0);
}
```

# Return Statement

`return(0)`



Transfers control from the program to the operating system

0 means program is executed without error



# REVIEW QUESTION & EXERCISES

1. Give some advantages and disadvantages of C language.
2. What does the C character set consists of?
3. What are the identifiers in any programming language?
4. How the identifiers can be formed in C?
5. Explain the structure of a C program.
6. What are various data types used in C language?
7. Name and describe any four basic data types in C.
8. Name and describe the four data type qualifiers. To which data types can each qualifier be applied?
9. What the different types of integer constants?
10. What is variable?

# CHAPTER 3

## DATA

### INPUT OUTPUT

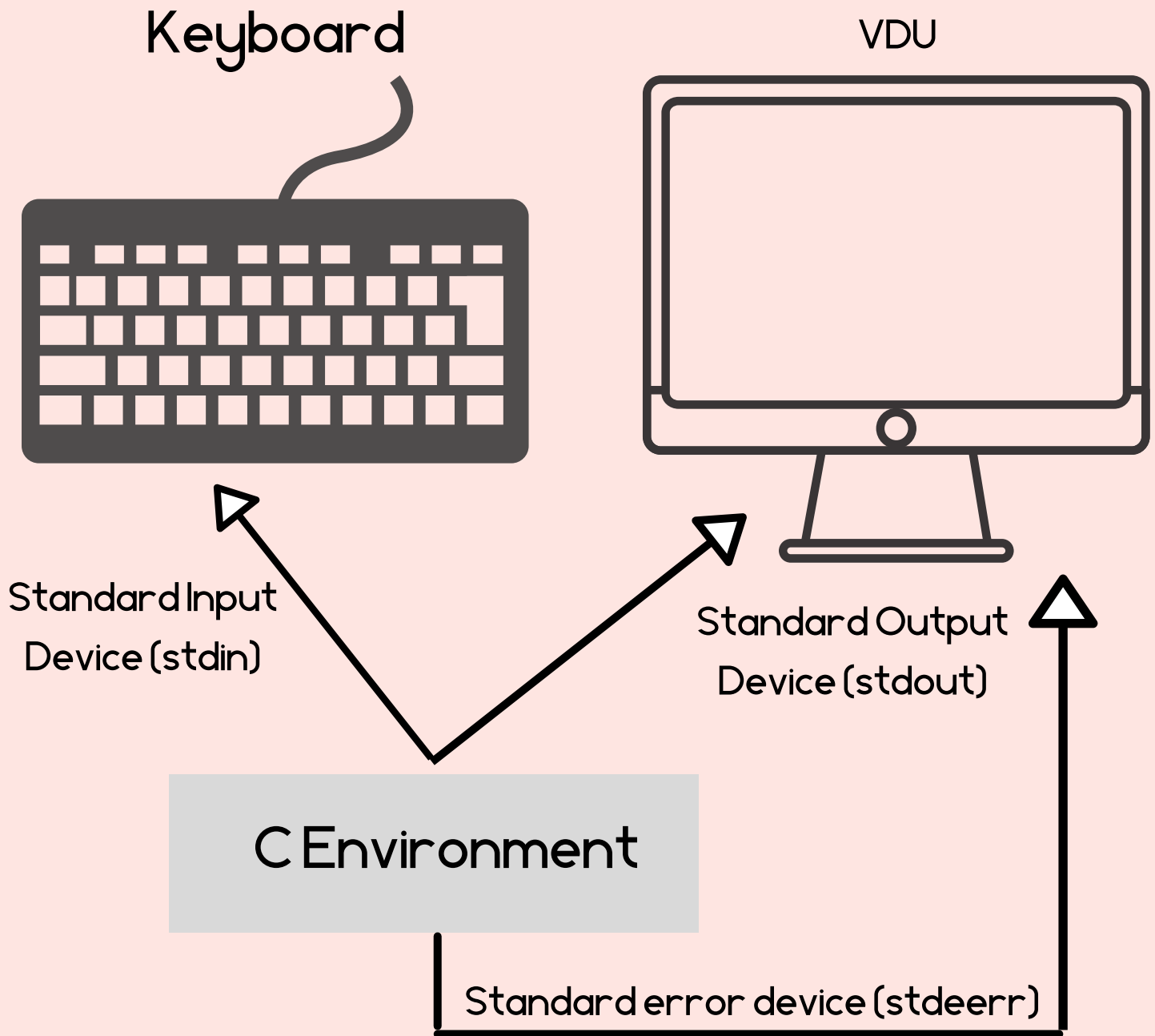
Understand Selection Statement

Understand Function of Input Output Statement

Input Output Operation



# Introduction



The C environment consists of the standard input device (keyboard), the standard output device (VDU) and the standard error device and they linked together.



The `stdin`, `stdout` and the `stderr` are used as references for accessing the devices by the C environments for the standard input output operations.

The input output operations are not part of C language as such rather they exist as functions as part of the language.

The standard input output function may be considered as character based input output functions and string based input output functions.

The statement **`#include <stdio.h>`** includes the contents of standard input output file. **`stdio.h`** at compile time which contains definitions of **`stdin`**, **`stdout`** and **`stderr`**.

C language provides the functions

`getc()`, `getchar()`, `getche()`

`putch()`, `putchar()`,

`scanf()`, `printf()`

for character input output and

`gets()`, `puts()`

`scanf()`, `printf()` for string based input output

# Character Input/Output

In C, we have functions for performing input/output of single character at a time known as **character I/O functions**.

The `getchar ( )`, `getche ( )` and `getch ( )` are used for character input and `putchar ( )` is used for character output.

## `getchar ( )` Function

It return a character just entered from the standard input unit that is keyboard.

The enter character can be either assigned to a character variable or to the computer screen

The function `getchar ( )` has the following form:

```
var_name = getchar ( );
```

## `getche ( )` Function

It return a character just entered from the standard input unit. The entered character is displayed to the computer screen.

It read a single character the moment it is typed without waiting for the ENTER key to be hit.

The function `getche ( )` has the following form:

```
var_name = getche ( );
```

## getch() Function

It return a character just entered from the standard input unit. The entered is not displayed on the screen.

It read a single character the moment it is typed without waiting for the Enter Key to be hit.

The function getch() has the following form:

**getche();**

## putchar() Function

It prints the character constant or the character variable to the standard output device.

The function putchar() has the following form:

**putchar (var\_name );**

# Formatted Input / Output

## The `scanf()` Function for Input Operation

It is used for formatted input from standard input device that is keyboard.

The format specification string along the data to be input are the parameters to the `scanf()` function.

The syntax of the function `scanf()`:

`scanf("format specification string", list address of variable);`

Example:

`char ch;`

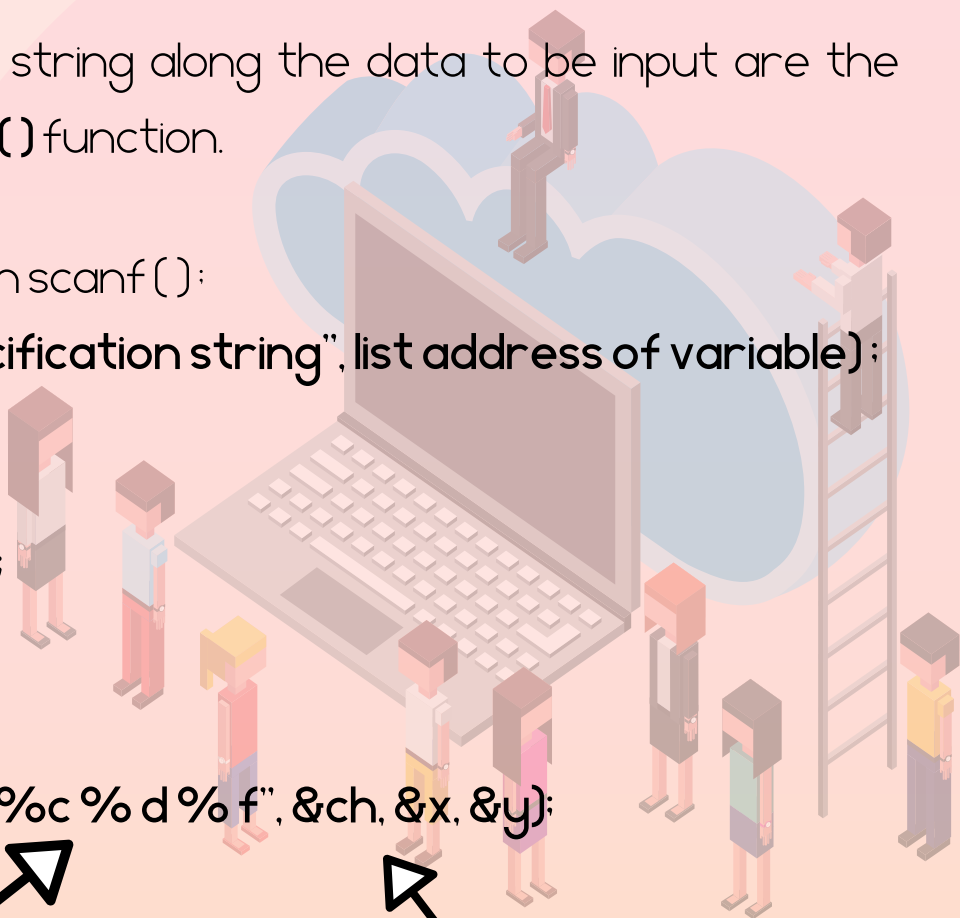
`int x;`

`float y;`

`scanf("%c %d %f", &ch, &x, &y);`

Format specification  
string

List of addresses of  
variable



# Formatted Input / Output

→ The data items entered from the keyboard must correspond to the arguments in the **scanf()** function in number, in type and in order.

→ The **scanf()** function reads and converts characters from the standard input according to the specified format string and stores the input in the memory locations represented by the list of the variables.

→ **NOTED THAT**, the list of variables is listed as **&ch**, **&x** and **&y**. This is the way of specifying list of addresses of variables in a **scanf()** function.

→ Conversion Specification having % an optional number for width specification and a conversion character.

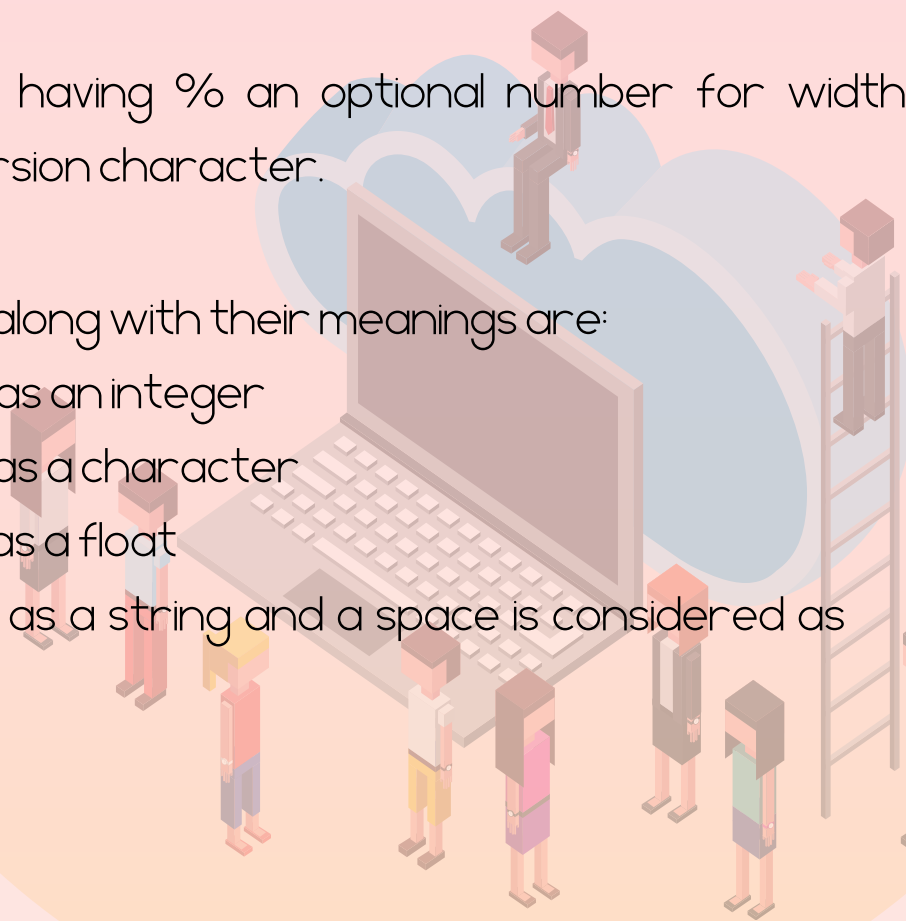
The conversion characters along with their meanings are:

**d** the variable accepted as an integer

**c** the variable accepted as a character

**f** the variable accepted as a float

**s** the variable accepted as a string and a space is considered as string terminator.





# Input of Integer Values

The following specification is provided for integer input:

`%wd`

Here `%` used for conversion specification

`w` denotes the field width

`d` used for integer data type

For example:

```
scanf ("%3d %4d", &val1, &val2);
```

# Input of Real Values Float and Double Type

In C, the `scanf ( )` function accepts real values in both forms fractional form and exponential form.

The specifications for float and double are `%f` and `%lf`.

For example:

```
scanf ("%5f %8lf", &x, &y);
```

# Input / Output of Character Type Data

A single character can be read from the keyboard by using the `scanf ( )` function with `%c` format specification.

For example:

```
char ch;
```

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

# Input / Output of String Type Data

A string is an array of characters terminated by a `'\0'` character.

The maximum number of characters that can be stored in an array of characters is 1 less than its size as the last character is always a NULL character (`\0`).

For example:

`char string[21];`

**it will store at maximum 20 characters**

The two types of console Input/Output functions for string type data are:

1

Unformatted functions

2

Formatted functions

# 1

## Unformatted Functions

For input the function is **gets()** and for output **puts()**

The **gets()** function overcomes the drawback of the **scanf()** function for receiving a multiword string.

The **scanf()** function which is a formatted function for receiving a string cannot receive multiword string.

The **gets()** receives a string from the keyboard when ENTER KEY is hit after typing the string.

The **puts()** function displays only one string at a time on the screen.

# 2

## Formatted Functions

For input the function is **scanf()** and for output **printf()**

The syntax for **scanf()**:

```
scanf ("%s", string_variable);
```

But **scanf()** cannot receive a multiword string. The array name do not begin with ampersand in **scanf()** function.

The syntax for **printf()**:

```
printf ("%s", string_variable);
```

# INPUT OF MIXED DATA TYPES



Allows input of mixed data type using one **scanf()** statement.

But one must be careful in such case as the **order** and **type** of data entered must match with that specified in the format specification string.

Example:

```
#include<stdio.h>
main ()
{
    char ch;
    int x;
    float y;
    clrscr ();
    printf ("Enter the value of char int and float types \n");
    scanf ("%c%d%f", &ch, &x, &y);
    printf ("\nThe entered values are\n");
    printf ("\n%c %d %f \n", ch, x, y);
}
```

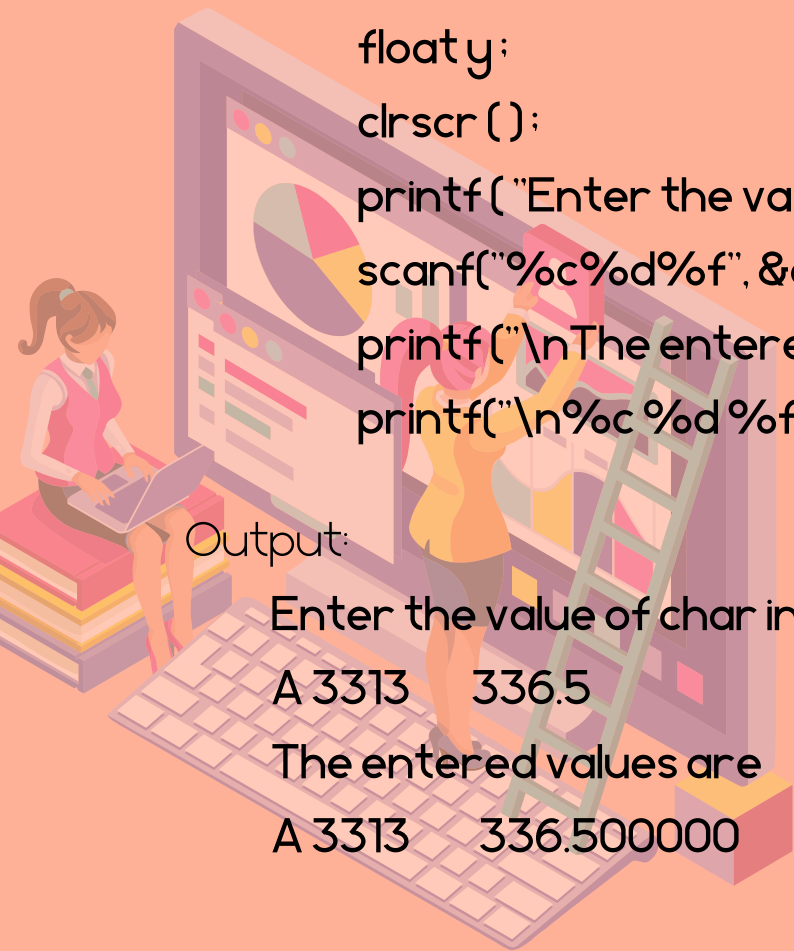
Output:

Enter the value of char int and float types

A 3313 336.5

The entered values are

A 3313 336.500000



# OUTPUT OF INTEGER VALUES

In C, the format specification for printing integer value is given as:

**%wd**

Here, **w** is minimum field width. It is optional. In case the width of the number to be printed is greater than the specified width, it will be printed correctly, over riding the width specified by the user.

**d** is used for integer specification. Using the above format the number is right justified and blanks will appears on the left side of it.

Example, if we want to print 3313 on the monitor using `printf()`, the it can be achieved in any one of the following ways.

## Specified Format

## Output Produced

`printf( "%8d", 3313);`

				3	3	1	3
--	--	--	--	---	---	---	---

`printf( "%d", 3313);`

3	3	1	3
---	---	---	---

`printf( "%-8d", 3313);`

3	3	1	3				
---	---	---	---	--	--	--	--

`printf( "%-08d, 3313);`

0	0	0	0	3	3	1	3
---	---	---	---	---	---	---	---

`printf( "%2d", 3313);`

3	3	1	3
---	---	---	---

**note: We use %ld for outputting long integers values.**



# OUTPUT OF REAL VALUES

A real number can be displayed in any one of following forms:

1

Fractional form

`% w.pf`

w denotes the minimum width including the decimal point.

p denotes the number of digits after decimal point. f is used for float specifications.

The output will be right justified.

2

Exponent form

`% w.pe`

w denotes the field width.

p the number of digits after decimal point.

The value is rounded off and printed right justified in the specified width of w column.

## OUTPUT OF SINGLE CHARACTER

In C, the format specification for printing a single character value is given as `%wc`

w specifies the field width and the output will be right justified.

# REVIEW QUESTION & EXERCISE

1. What is the purpose of `scanf()` and `printf()` functions?
2. Describe formatted Input/Output functions in C language.
3. What is meant by a control string used in I/O statement?  
Discuss it.
4. Explain `getchar()`, `putchar()`, `gets()` and `puts()`
5. When printing a float or double with `%f`, how many digits after decimal does `printf()` output?
6. What is the purpose of the following in `printf()`:
  - a. `%lf`
  - b. `%6.2f`
  - c. `\a`
  - d. `\t`
7. Using one `printf()` statement only, print following message:  
Remainder can be found using /  
and `"%"` isn't a special symbol.

# REVIEW QUESTION & EXERCISE

1. Find syntax error in following program and write equivalent corrected code:

```
#include<stdio.h>

main
{
    int a; b =20;
    10 =a;
    int c;
    c = a+b
    printf('the sum='c);
}
```

2. What will be the output of the following program?

```
#include<stdio.h>

main ( )
{
    char a,b;
    a = 'b'
    b=a;
    printf("b=%c\n",b);
}
```

# CHAPTER 4

## Control Statement

Understand Selection Statements  
Understand Looping Statement  
Understand Array Data Structure  
Understand Multi-Dimensional Array



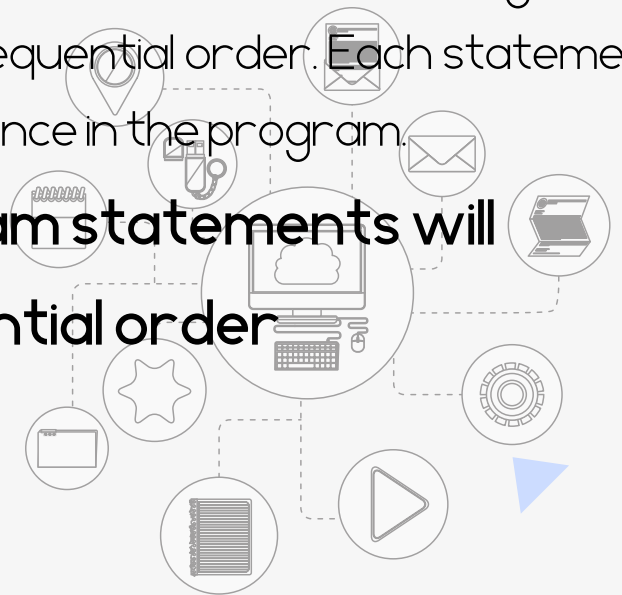


# INTRODUCTION



The normal flow of execution of statements in a high level language program is also in the sequential order. Each statement is executed in its order of occurrence in the program.

**For example the c program statements will be executed in the sequential order**



.....

.....

**a = 50;**

**b = 10;**

**q = a/b;**

.....

.....


First, a = 50 will be executed followed by the statement b = 10 and the statement q = a/b **strictly** in the sequential order.

- Some problems oftenly require that the normal flow of control be altered depending on the requirement.
- It means that we may wish **selective** and/or **repetitive** execution of any portion of the program
- C supports a number of control structures to perform the processing.



# Control Structure

- ⇒ A function is a set of statements to perform a specific task.
- ⇒ For solving specific task, we may have many algorithms, some simple and other complex.
- ⇒ The program should be written in a user friendly way so that it may be modified later on by anyone who wishes to change it.
- ⇒ Debugging and maintenance would be easy if the program is coded with a proper format.
- ⇒ For attaining the objective of a good program, we use one or any combination of the following three control structures:

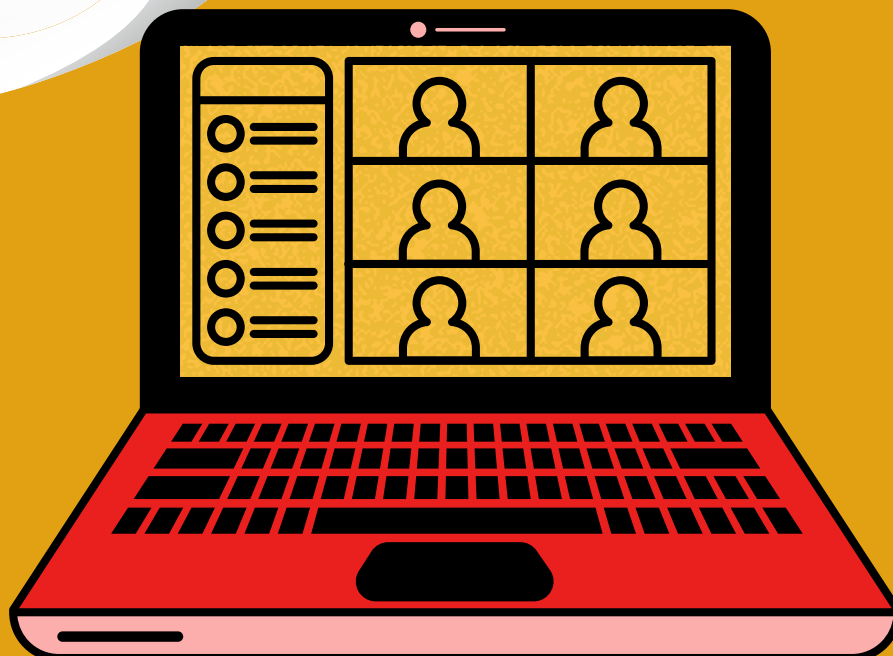


3

Sequence control structure

Selection control structure (branching)

Looping control structure (repetition or iteration)



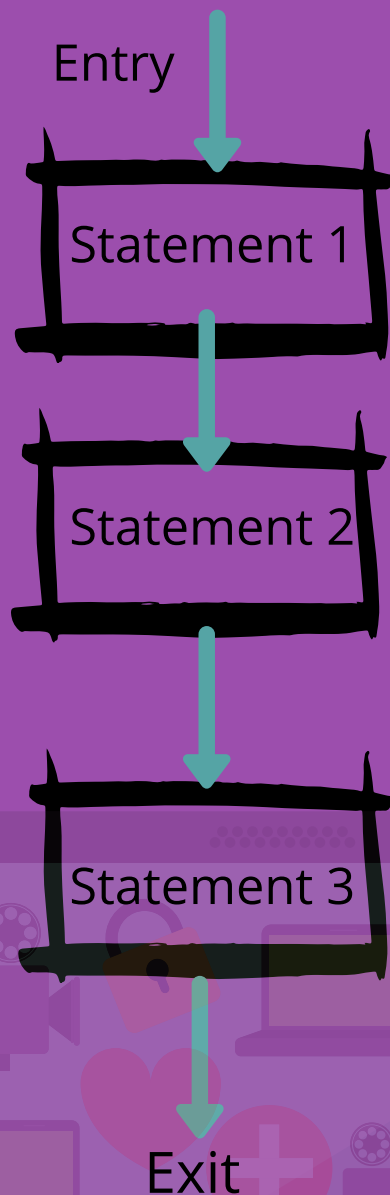


# The Sequence Control Structure



The sequence control structure may consist of a single statement or a sequence of statement with a single entry and single exit.

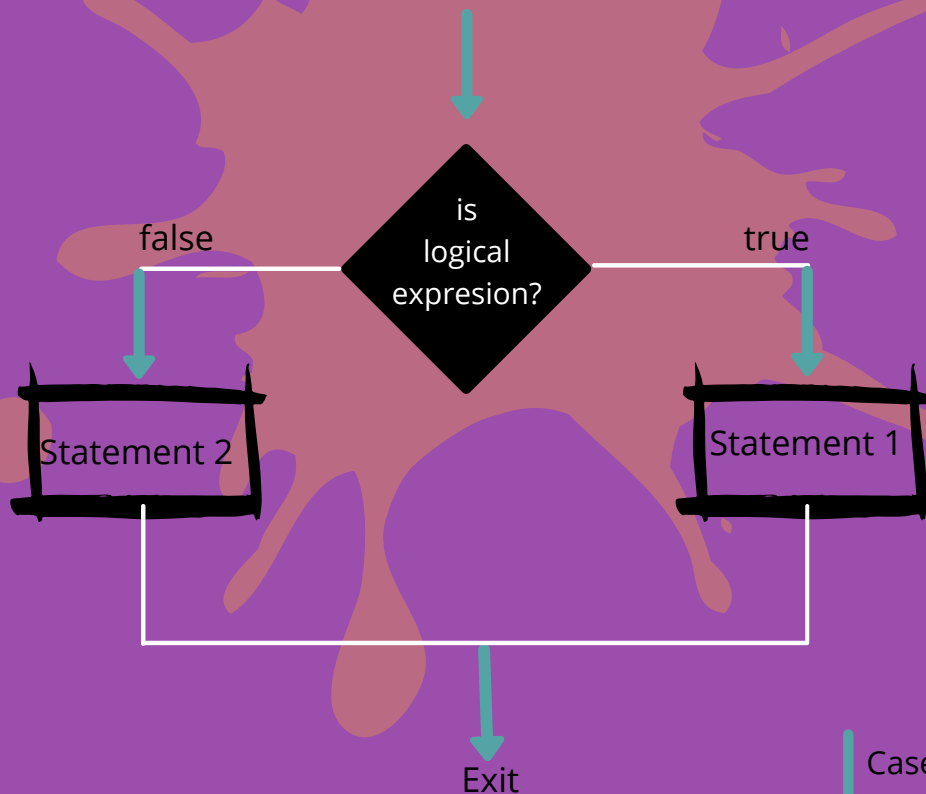
Example statement or function call.



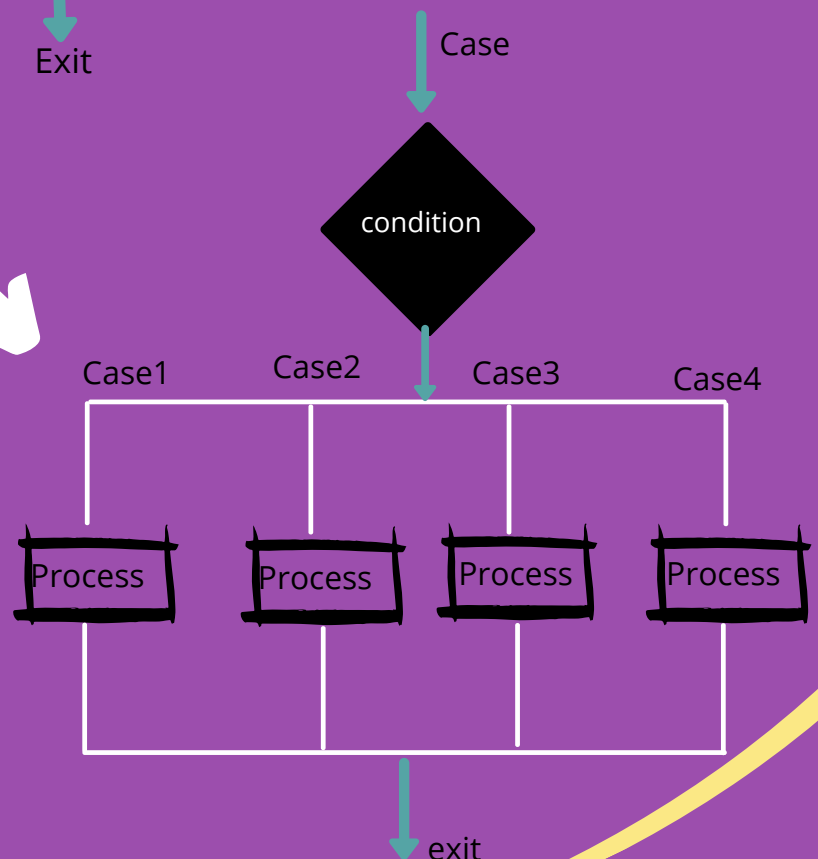
# Selection Control Structure

Selection control structure performs one out of two or more statements depending upon the condition.

The selection control structure (IF-THEN-ELSE) is shown in figure below.



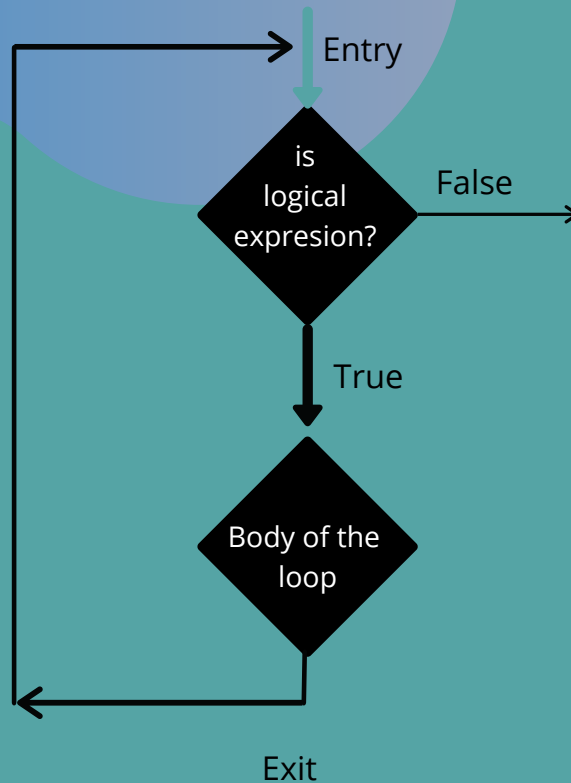
Case is an important version of selection (more than a single yes or no decision). The figure beside illustrates variation on selection for the case control structure.



# Looping Control Structure



Has 1 condition (simple or compound) and a sequence structure which is executed a number of times depending on the condition (logical expression) being true or false.



We can code any program using the three structure. Any program coded using the all shown control structures is called a **Structured Program** and the approach is known as **Structured Programming**.



The main objective of structured programming are readability and clarity of program, maintenance and reduced testing problems.

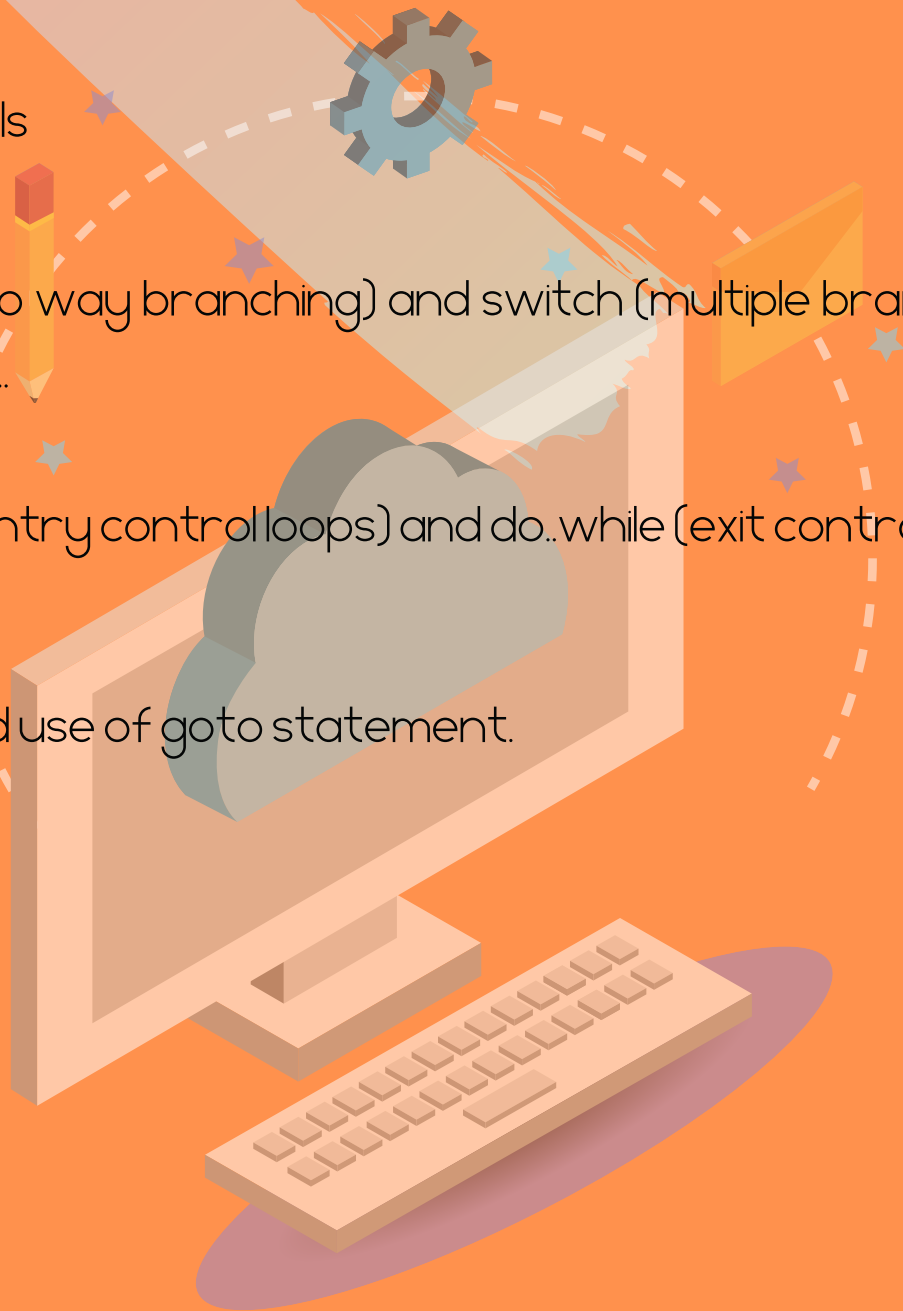


Structured programming eliminates the use of goto (unconditional control).



C Implements the three control structures and uses the following language constructs:

- 1 All straight line statements such as assignment statement, input and output statements.
- 2 Function calls
- 3 if if-else (two way branching) and switch (multiple branching) statements..
- 4 while, for (entry control loops) and do..while (exit control loop)
- 5 A restricted use of goto statement.



# Statements

These are the instructions given, to the computer to perform some action and form the smallest executable unit within a C Program. A semicolon (;) terminates a statement. The empty or null statement is written as,

;

*/\*a null statement\*/*

Is it useful in the situations where the syntax of language need the presence of a statement but the program logic does not.

It will be used in loops and their bodies.

## ● Simple Statement

single statement terminated by a semicolon


```
{  
    statement1;  
    statement2;  
    :  
    statementn;  
}
```

## ● Compound Statement

Formed by two or more statements enclosed by a pair of braces { } also known as a block

```
{  
    .....  
    .....  
    {  
        .....  
        .....  
    }  
    .....  
    .....  
}
```

# CONDITIONAL STATEMENT (Selection)



There are situations when we wish to execute some part of the program based on some conditions being true or false.



In c, **if** and **switch** statements are used for selective execution of a program segment.





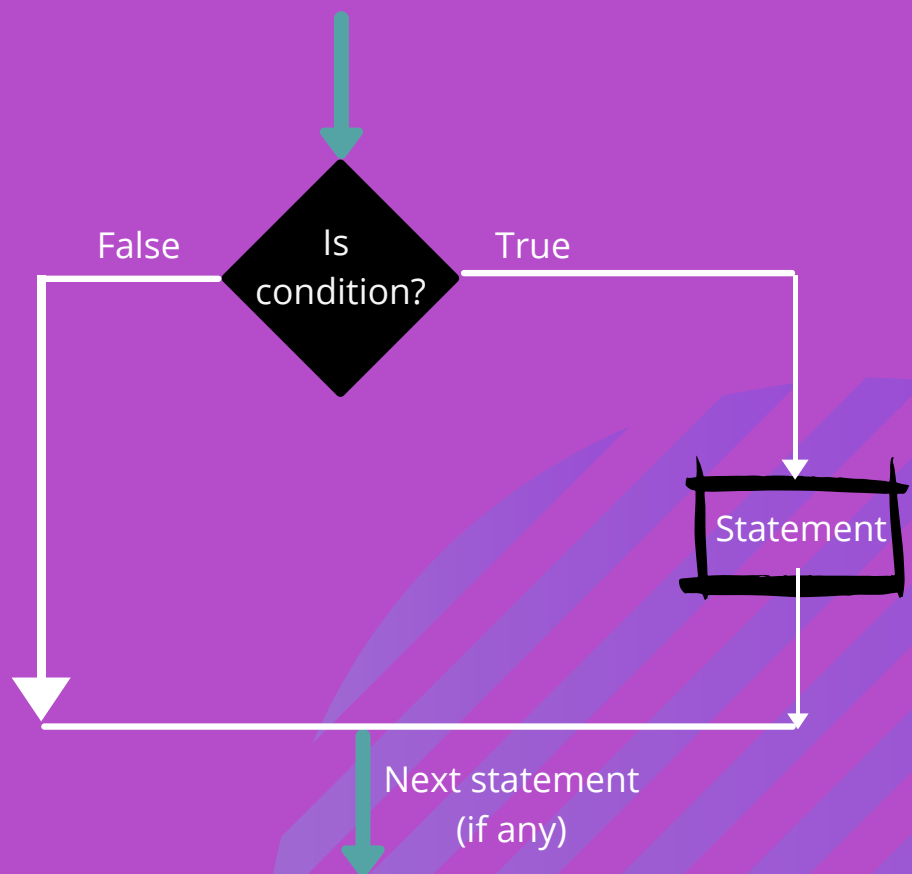
# if statement

It tests a condition.

The statement(s) associated with if is/are executed when the condition is true, otherwise the statement(s) is/are not executed at all.

The statement can be either simple or compound.

Generally, it is a compound (two or more statement) which may have other control statements.



For example:

```
int x,y;
printf("Enter the two intergers\n");
scanf("%d%d",&x,&y);
if(x>y)
printf("%d",x);
```

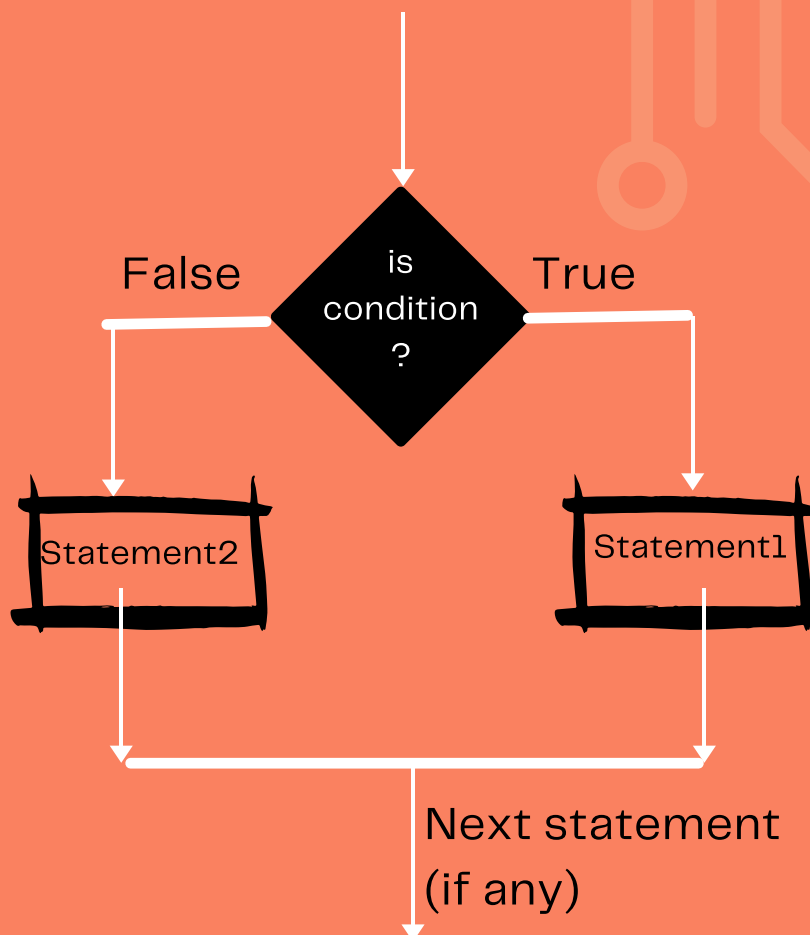
# If...else statement(selector)

It tests a condition. The statement 1 is executed when the condition is true otherwise statement 2 is executed.

The statements may themselves be compound statements.

The syntax of if-else statement is given below:

```
if (condition)
    statement1;
else
    statement2;
```



For example:

```
int x,y;
printf("Enter the two intergers\n");
scanf("%d%d",&x,&y);
if (x>y)
    printf("%d",x);
else
    printf("%d",y);
```

# NESTED if-else statement



In C, it is possible to nest if-else statement within one another.



One or more if statement embedded within the if statement are called nested ifs.

The following if-else statement is a nested if statement nested to level three:

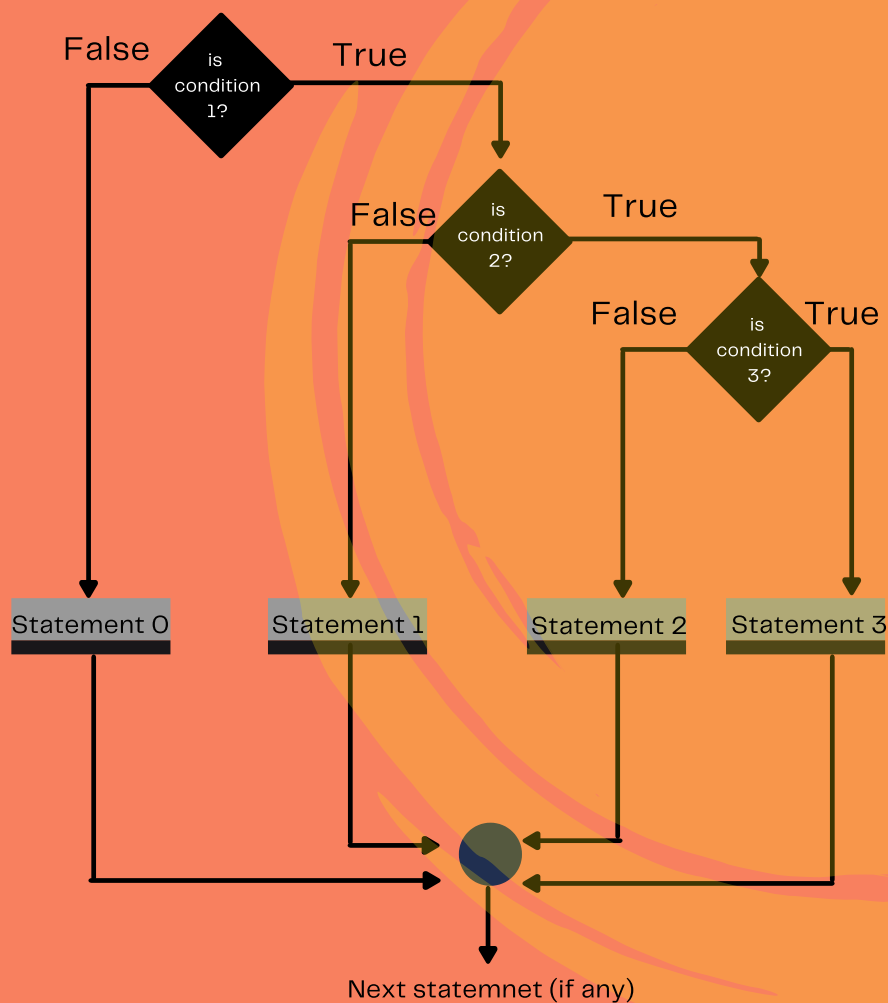
```
if (condition)
{
    if (condition 2)
    {
        if(condition3)
            statement3;
        else
            statement2;
    }
    else
        statement1;
}
else
    statement0;
```

The logic of execution is shown in figure below:

**1** If the condition 1 is false, the statement 0 will be executed. Otherwise it continues to perform the second test.

**2** If the condition 2 is false, the statement 1 will be executed. Otherwise it continues to perform the third test.

**3** If the condition 3 is true, the statement 3 will be evaluated. Otherwise the statement 2 will be evaluated and then the control transferred to the statement following the statement 0.(if any)



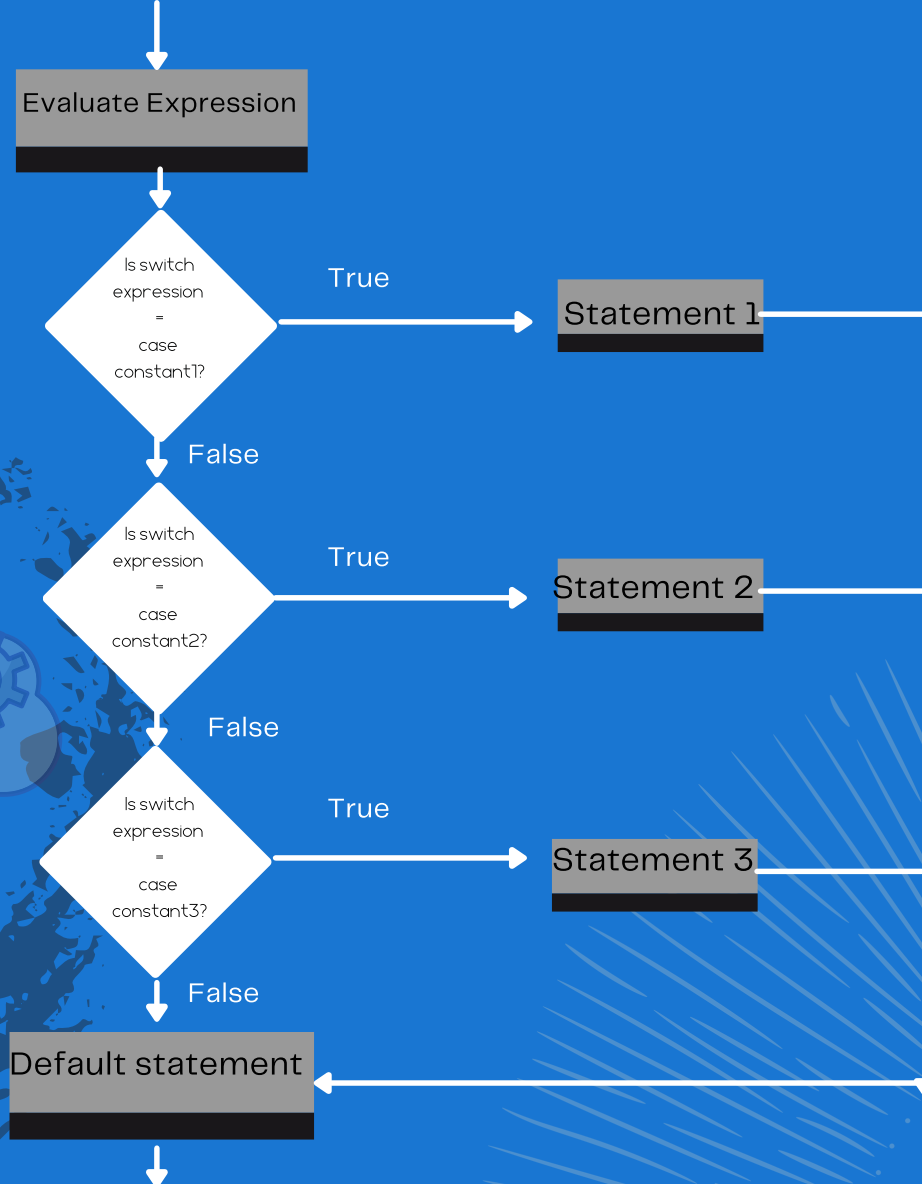
**4** The nested if statements should be written with proper indentation. Use braces { } generously.

# Switch (Multiselector)






- ① It test a control expression (condition). The control is transferred to one of the several alternatives.
- ② The value of the expression may be type of **int** or **char** but not of type **float** or **double**.
- ③ The syntax of switch statement is as follows:

```
switch (control expression)
{
    case constant 1 : statement(s);
                    break;
    case constant2 : statement(s);
                    break;

    case constant n : statement(s);
                    break;
    default :      statement(s);
                    break;
}
```



The following rule should be used for switch statement:

-  The expression value must be an integer, hence the type can be `int` or `char`.
-  Case should always be followed by an integer constant, character constant or constant expression.
-  All the cases should be distinct.
-  The block of statement under default is executed when none of the cases match the value of expression.
-  Default can optionally be present.



# Break

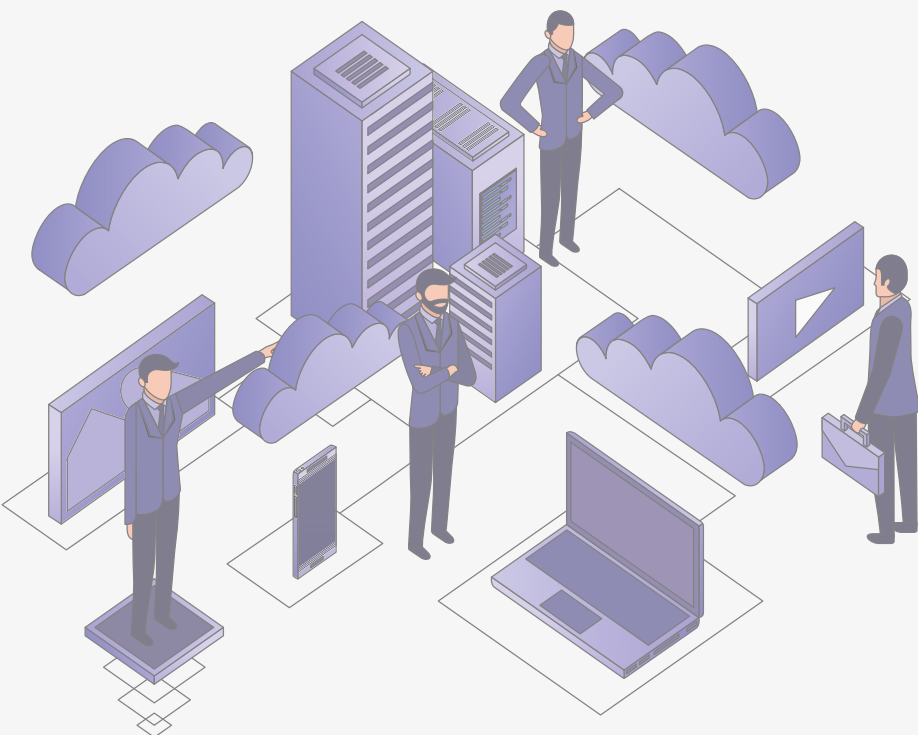
Statement ( to be used in switch..case only

It causes an exit from the switch body. Control goes to the first statement following the end of the switch statement.

If the **break** statement is not used, the control passes to the next case constant, and the remaining statements in the switch construct will also executed.

# Default Keyword

The default keyword (if present) in the switch construct gives a way to take action id the value of the switch expression does not match with any of the case constants.



# Loops/ Repetitions / Iterations



Many times its require that a group of instructions is to be executed until some logical condition is satisfied. It is known as looping.



In the some situations the number of repetitions is known in advance.



Situations also arise, however in which the required number of repetitions is not known. The execution of statements is repeated until the logical condition becomes true.

C provides three (3) statements for repeatedly executing a sequence of statements.

# While

# do-while

# for

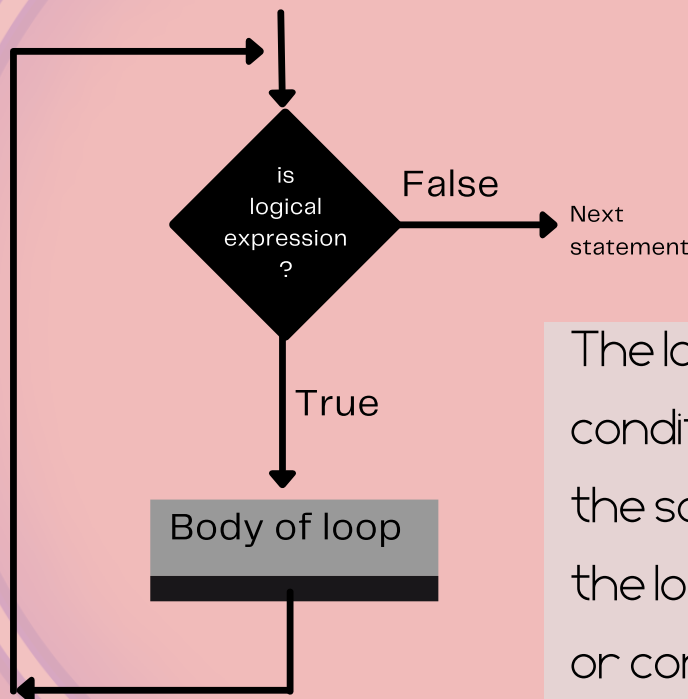


# While Statement

The syntax of the while statement is as follows:

```
while(condition)
{
    body of loop
}
```

It can be shown with the help of the flowchart.



The logical expression or condition means one and the same and the body of the loop may have a simple or compound statement.

The point should be remembered while using the while statement:

1

It may not be executed even once if the condition is false initially.

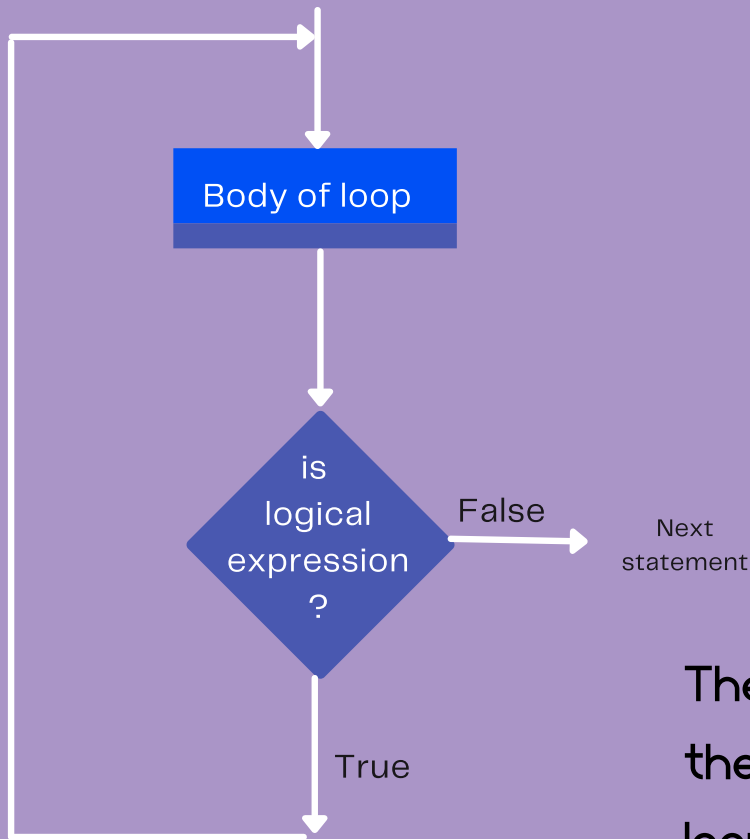
2

It is executed till the condition remains true and the control comes out of the loop when the condition becomes false.

3

there must be some loop terminating condition inside the body of the loop to avoid infinite looping.

# do-while Statement



The syntax of the while statement is as follows:

```
do
{
    body of loop
} while (condition);
```

The braces are not required when there is only single statement in the loop.

The point should be remembered while using the do-while statement:

1

It is executed at least one.

2

It is executed till the condition remains true and the control comes out of the loop when the condition becomes false.

3

there must be some loop terminating condition inside the body of the loop to avoid infinite looping.

# For Statement



The while and do-while loops are used when the number of iterations is not known.



Iterations = the number of times the body of the loop is executed.



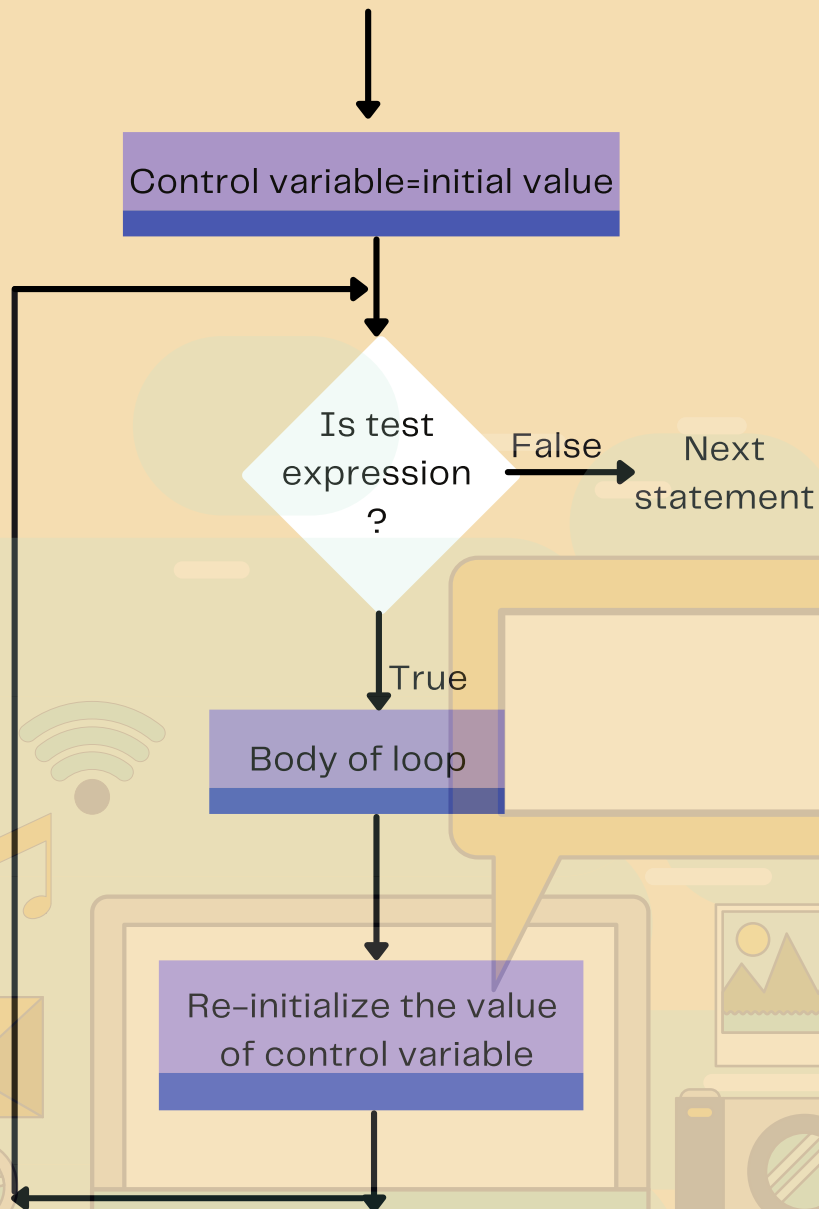
The for loop is used when the number of iterations is known in advance.

The syntax of the for loop is as follows:

```
for ( initialization ; test expression ; re-initialization )  
{  
    body of loop  
}
```



The syntax of the for loop is as follows:



**Initialization Expression** : It is executed only once when the loop first starts. It provides the loop variable (control variable) an initial value.

**Test Expression** : It involves relational operators. It is executed every time through the loop before the body of the loop is executed. If the test expression is true, the body of the loop is executed and if false the control comes out of the loop.

**Increment/Decrement (Re-initialization) Expression** : It is always executed at the end of the loop after the body of the loop.



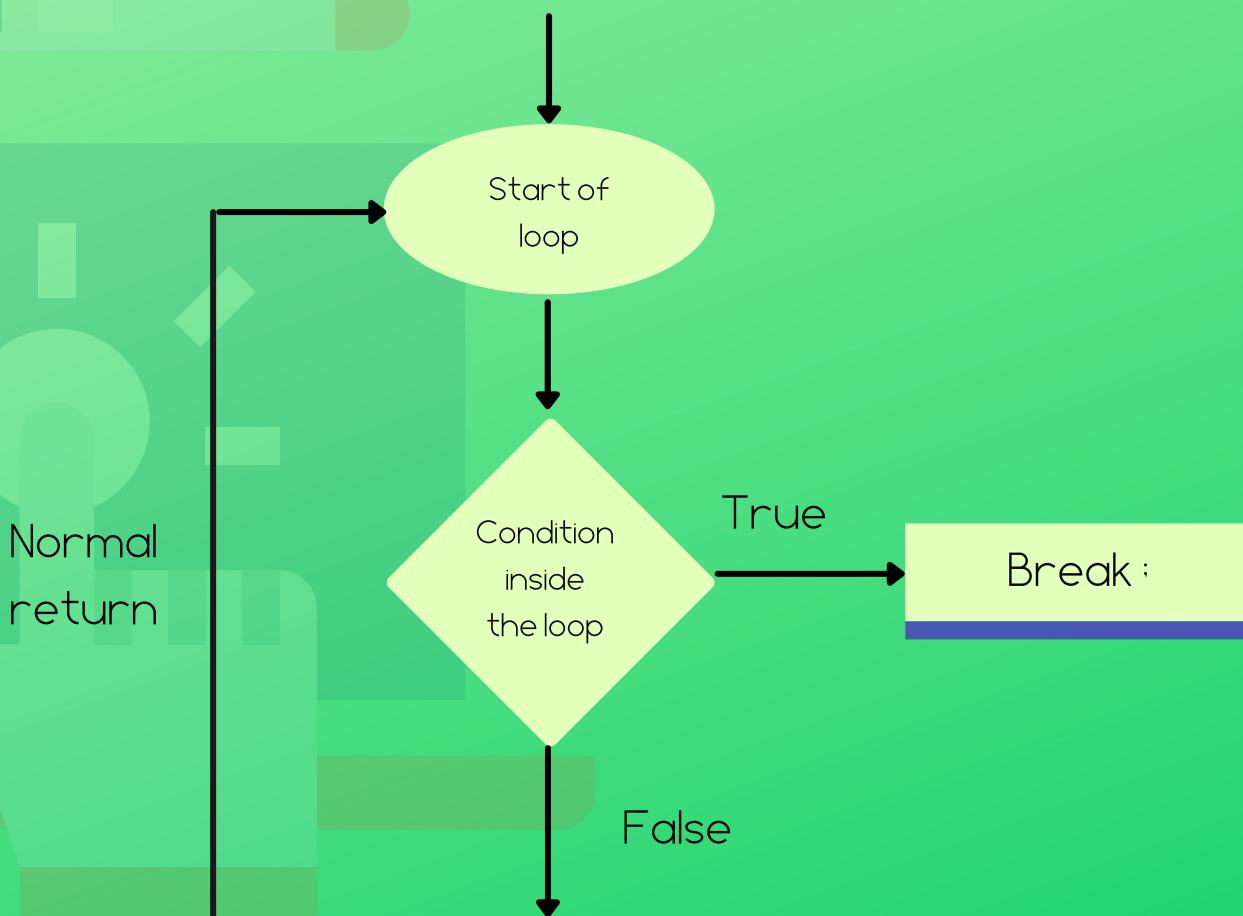
# Break Statement

It transfers control out of a loop, by passing the normal loop condition test.

We have already used the **break** statement for separating case labels in switch statements.

When a **break** is encountered inside a loop, the loop is terminated and the control passes to the statement following the body of the loop.

It can be shown with the help of the flowchart.





# Continue Statement



It forces the next iteration of the loop to take place, skipping any statements following the continue statement in the body of the loop.

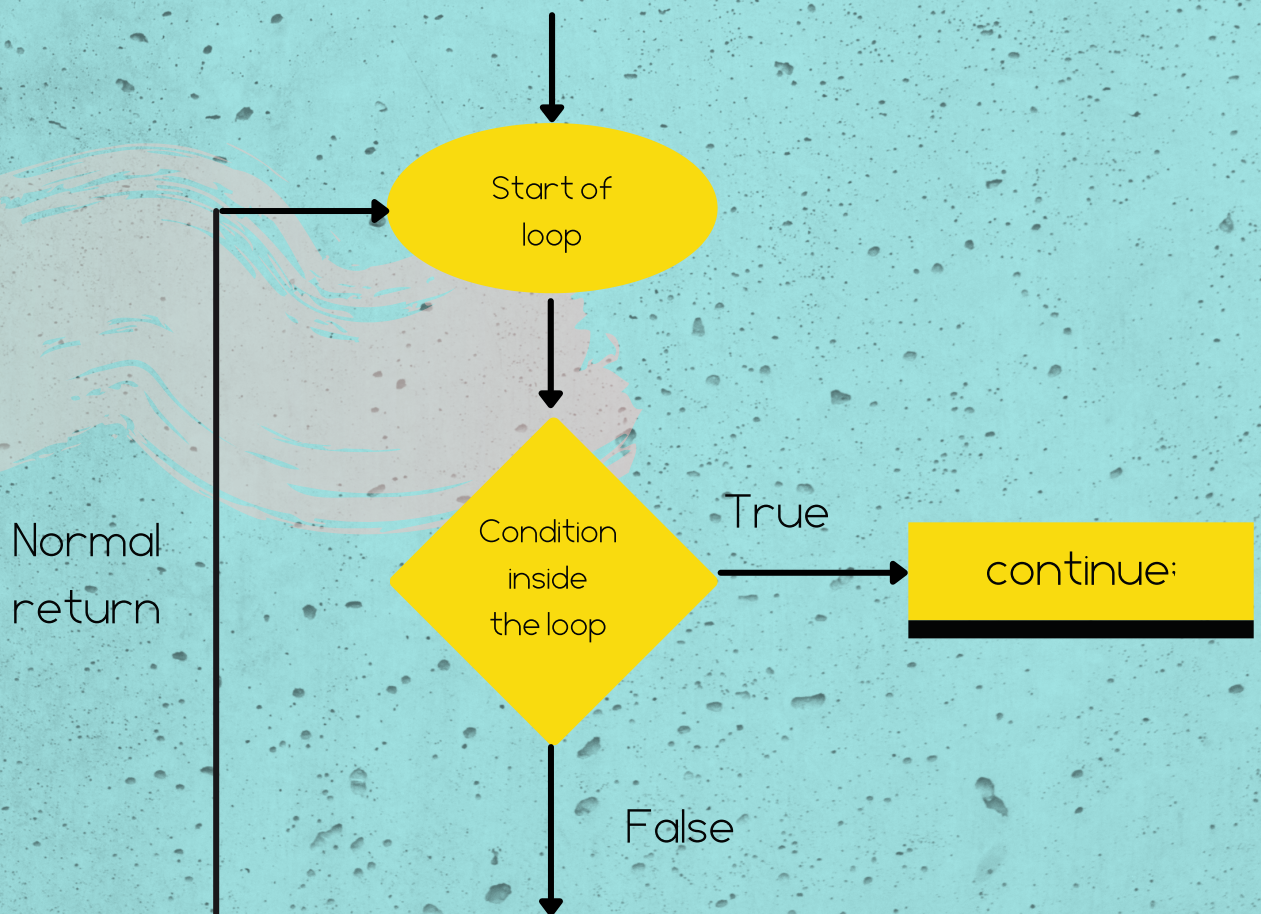


In the while and do-while loops, the control passes to the conditional test.



In the for loop, the continue statement causes the conditional test and then the re-initialization part of the loop is executed.

The following flowchart shows the operation of the continue statement:





# goto Statement

(Unconditional Branching Statement)

It is given here only for the sake of completeness, because **use of a goto statement makes a program difficult to understand and debug.**

The syntax for goto is:

**goto** label;

The label is a valid C identifier followed by a colon. You may have any statement after the label in the form:

**label:** statement;

The following program illustrates this concept:

```
#include<stdio.h>
main ()
{
    int limit, num, sum=0;
    clrscr ();
    printf("Enter the limit for addition of natural number\n");
    scanf("%d",&limit);
    num=1;
    target : sum+num : /*target used as label */
    if (num<limit)
    {
        num++;
        goto target;
    }
    printf("\n sum of first %d natural number is %d\n\n",
    limit, sum);
}
```



# Array

1

Many application require the processing of multiple data items that share common properties.

2

The individual data items can be characters, integers, floating point number.

3

C provides the derived data types also, which are built from the basic integer and floating data types.

4

An array is a C derived type that can store several values of one type.

5

An array is a collection of homogeneous elements that are referred by a common name.

6

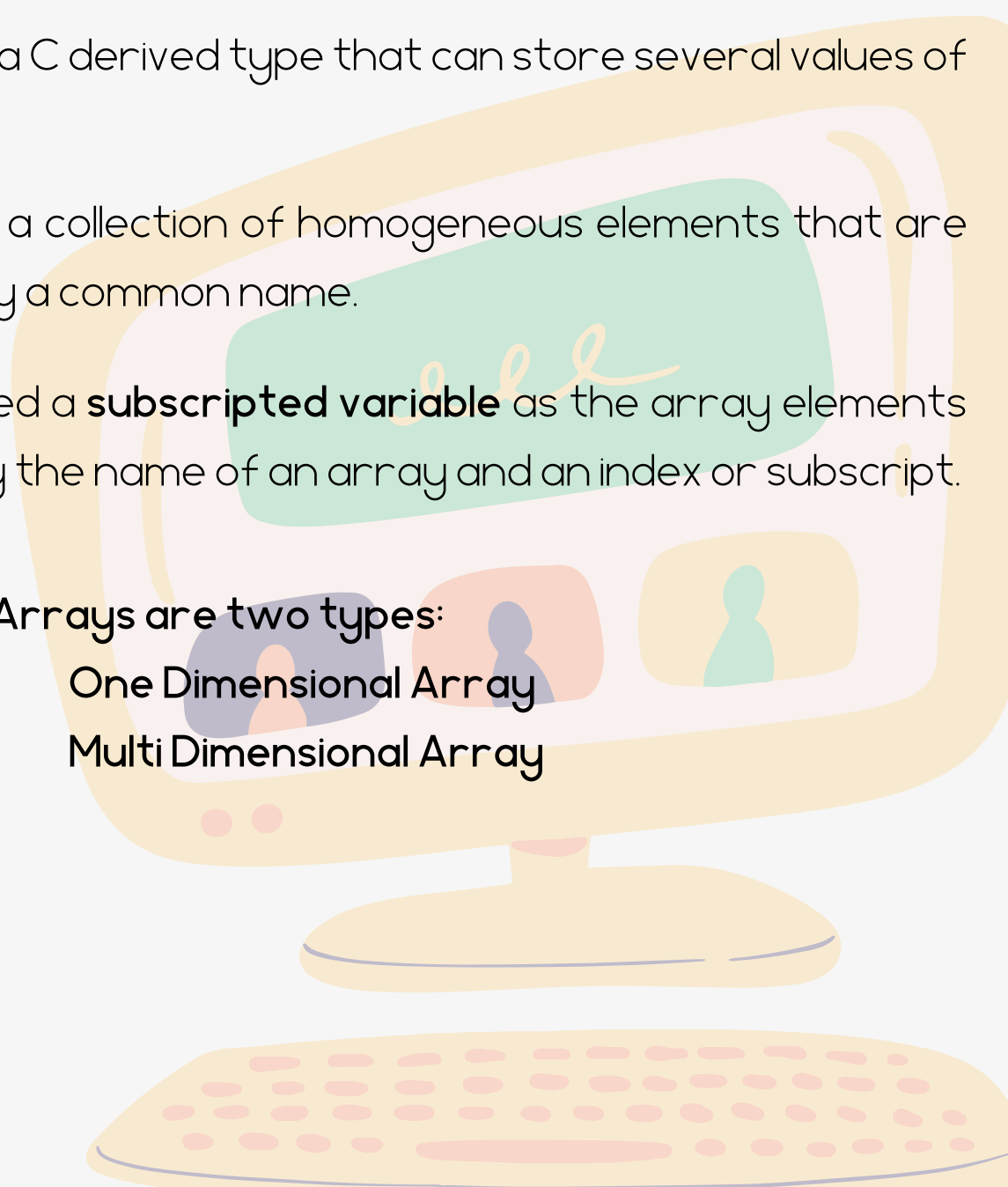
It is also called a **subscripted variable** as the array elements are used by the name of an array and an index or subscript.

7

Arrays are two types:

One Dimensional Array

Multi Dimensional Array



# Declaration/Initialisation of One-Dimensional Array

The syntax of declaring a one dimensional array in C :

**type array\_name[size];**

Here **type** declare the **base type** of the array which is the type of each element of the array.

The **array\_name** specifies the array name by which the array will be referenced and **size** defines the number of elements the array will store.

**Example:** `int a[5];`

In C, the array index always begin with 0. So a [5] would refer to the third element in the array **a** where **5** is the array index or subscript.

The entire array having elements 55, 90, 17, 88 and 36.

Memory

Name of array **a**

Array index  
from 0 to 4

0	55	← a[0]
1	90	← a[1]
2	17	← a[2]
3	88	← a[3]
4	36	← a[4]



Since each element in **a** is an integer, it occupies two bytes. Notice that the first element has the index 0.



Thus, since there are five elements, the last one is number 4.



Name of an array without subscripts also refer to the address of the first element.

The General syntax of initialization of arrays is given:  
**type array\_name = {list of values comma separated};**

Initialization at the time of declaration is known as

**Compile Time Initialization**

Example : `int a[5] = { 55, 90, 17, 88, 36};`



## Inputting Array Elements



For reading the array elements (input operation) we must declare the array first along with the index to be used on the array.



Then the array elements can be inputted as given below:

```
int a[5], i;  
for (i=0, i<5, i++)  
scanf ("%d", &a[i];
```

An Explicit initialization at run time is known as **Run Time Initialization**

# Passing Array Elements to Function

In C, array elements can be passed to a function in two ways.

1

Calling the function by value

2

Calling the function by reference

```
#include <stdio.h>
#define SIZE 10
main()
```

```
{
    void show (int) ;
    int a [size], i , n ;
    clrscr();
    printf("Enter number of elements in the array<= %d\n\n", SIZE) ;
    scanf("%d",&n);
    printf("\nEnter %d elements\n\n", n);
    for (i=0; i<n , i++)
        scanf("%d" , &a[i]);
    printf("\nEntered elements of array are\n\n");
    for (i=0; i<n;i++)
        show(a[i]) ;
    getch() ;
}

void show (int value)
{
    printf ("%8d", value);
}
```

The Different

Calling the function by value


**The program output:**

```
Enter number of elements in the array <= 10
5
Enter 5 elements
18 27 15 38 91
Entered elements of array are
18 27 15 38 91
```



# Calling the function by reference

```
#include <stdio.h>
#define SIZE 10
main()
{
    void show (int) ;
    int a [size], i , n ;
    clrscr();
    printf("Enter number of elements in the array<=10", SIZE) ;
    scanf("%d",&n);
    printf("\nEnter %d elements\n\n", n);
    for (i=0; i<n , i++)
        scanf("%d", &a[i]);
    printf("\nEnter elements of array are\n\n");
    for (i=0; i<n;i++)
        show(&a[i]) ;
    getch() ;
}
void show (int *value)
{
    printf ("%8d", *value);
}
```



The Different

## The program output:

```
Enter number of elements in the array <= 10
5
Enter 5 elements
18 27 15 38 91
Entered elements of array are
18 27 15 38 91
```



# Multi Dimensional Array

1

The array we used in the last example was a one dimensional array.

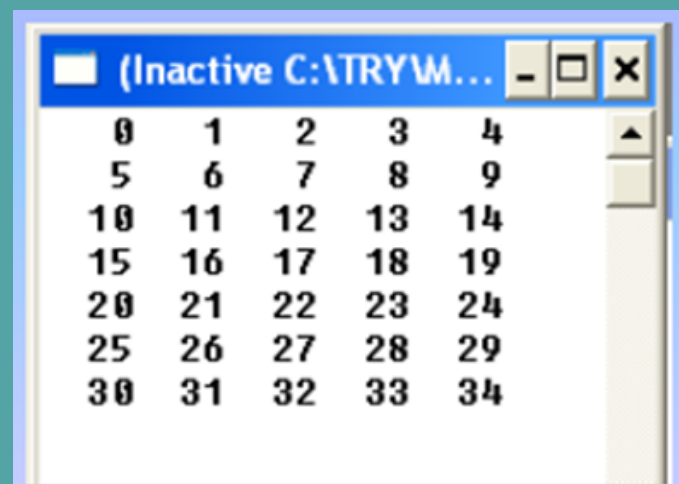
2

Arrays can have more than one dimension, these arrays-of-arrays are called multidimensional arrays.

3

They are very similar to standard arrays with the exception that they have multiple sets of square brackets after the array identifier.

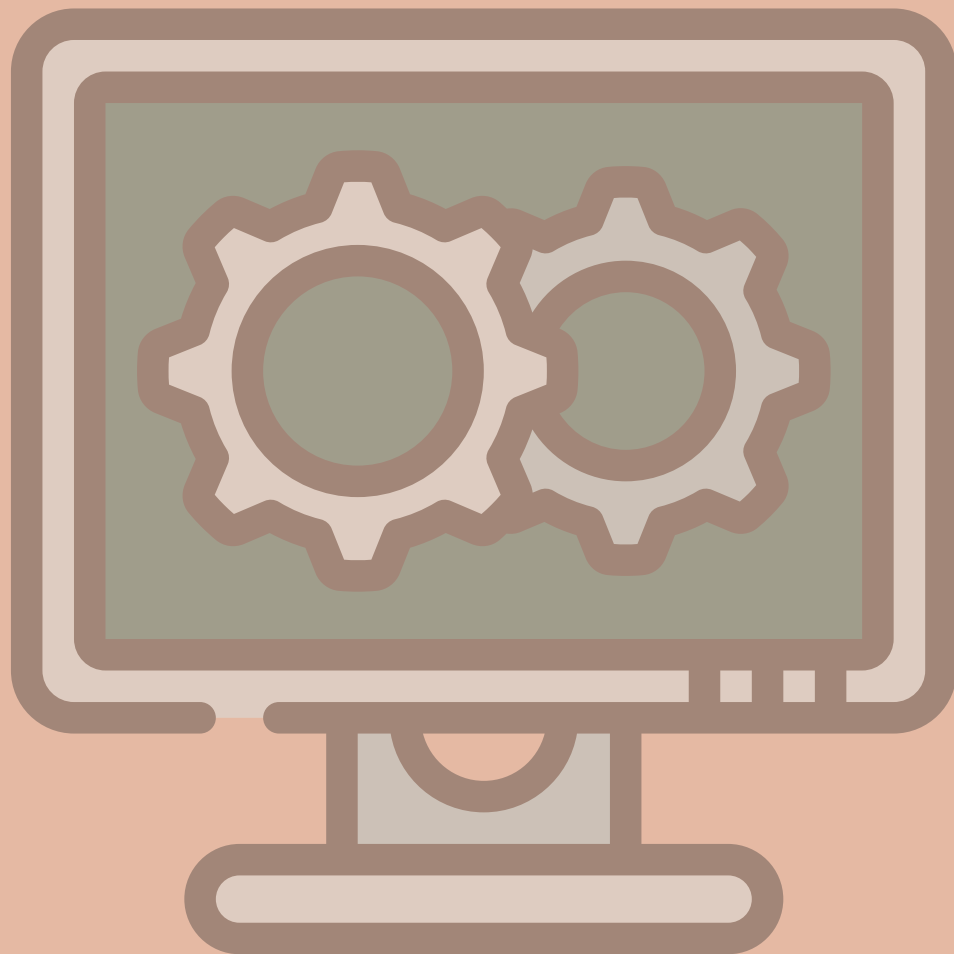
```
#include <stdio.h>
const int num_rows= 7;
const int num_columns= 5;
int main()
{
    int box[num_rows][num_columns];
    int row, column;
    for(row = 0; row < num_rows; row++)
        for(column = 0; column < num_columns; column++)
            box[row][column] = column + (row * num_columns);
    for(row = 0; row < num_rows; row++)
    {
        for(column = 0; column < num_columns; column++)
        {
            printf("%4d", box[row][column]);
        }
        printf("\n");
    }
    return 0;
}
```



0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24
25	26	27	28	29
30	31	32	33	34

# CHAPTER 5

## FUNCTION



# UNDERSTAND FUNCTIONS



The best way to develop and maintain a large program is to construct it from smaller pieces or modules.

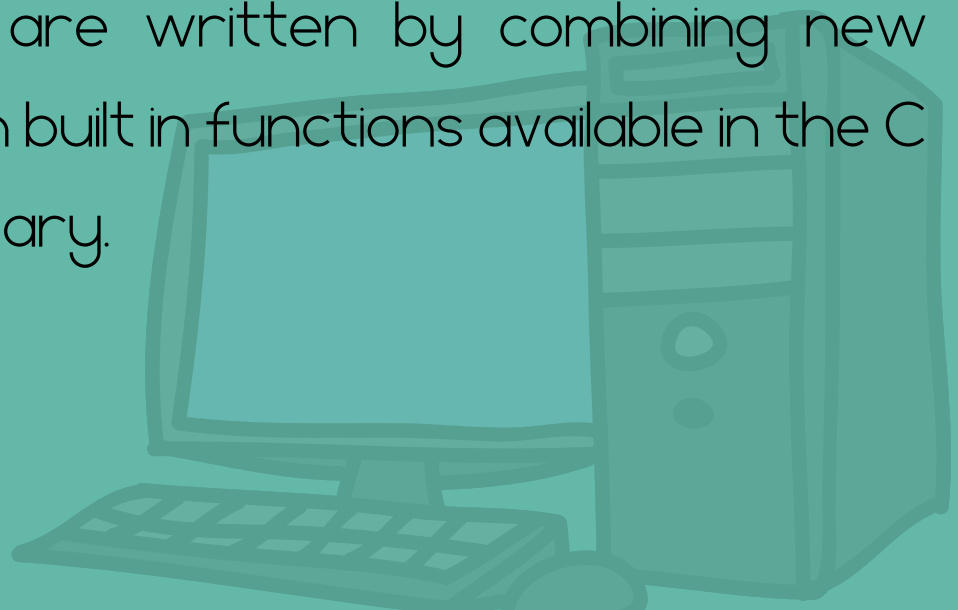


Modules in C are called Functions.

- A function is a block of code used to perform a specific task.
- A function is reusable and therefore prevents programmers from having to unnecessarily rewrite what are written before.
- A function cannot be contained in another function.



C programs are written by combining new functions with built in functions available in the C Standard Library.





Two categories of functions:

**BUILT IN** or **PREDEFINED** or **STANDARD FUNCTIONS** and **USER DEFINED FUNCTIONS**.

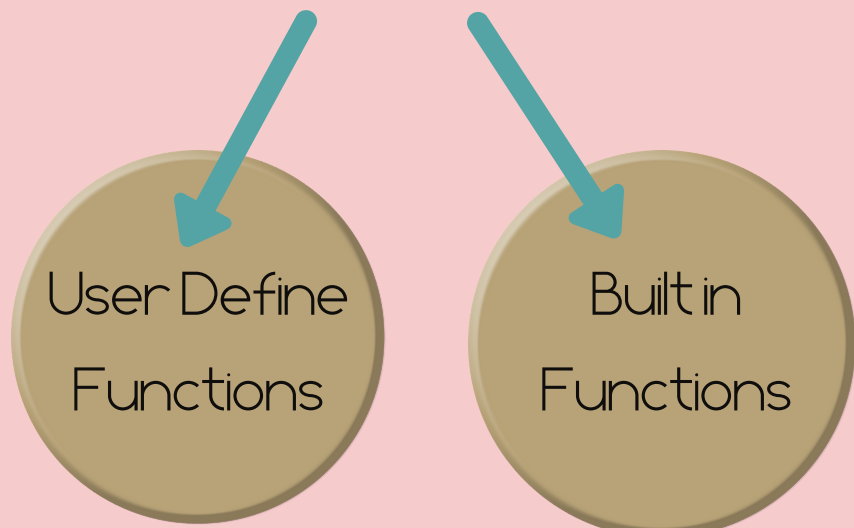


To work with **USER DEFINED FUNCTIONS**, it is necessary to declare functions prototypes call functions and define functions.



A function definition is where the actual code for the function is written. This code will determine what the function will do when it is called.

## Types of Function



# DEFINE TERM OF FUNCTION

## How Do I Define a Function?

Functions can be defined as having a head & a body.

To define a function, identify:

1. The **return\_type** - 4 return type data: char, int, float & double. Either void or same as type of value returned to caller.
2. The **function\_name** - given by programmer.
3. An optional **parameter\_list** - made up of variable declaration of all the values to be inserted into the function and enclosed in parentheses.
4. A compound statement (Function body) - use braces {}

Function definition header - must be the same as function prototype

```
return_type  function_name  (parameter_list)  
{  
    declaration-statement;  
    executable-statement;  
}
```

} Head

} Body

# USE OF FUNCTIONS IN C PROGRAMME



One of the advantages of using a function is that the programmer can easily read and identify all the functions that are used in the program.

EXAMPLE: Function prototype usage

```
#include <stdio.h>
void square (int x);      // function prototype
void main(void)
{
    int num = 5;
    square (num);         // function call
}
void square (int x)       // function definition
{
    printf("Square of %d is %d", x , x * x);
}
```

**Output**

**Square of 5 is 25**





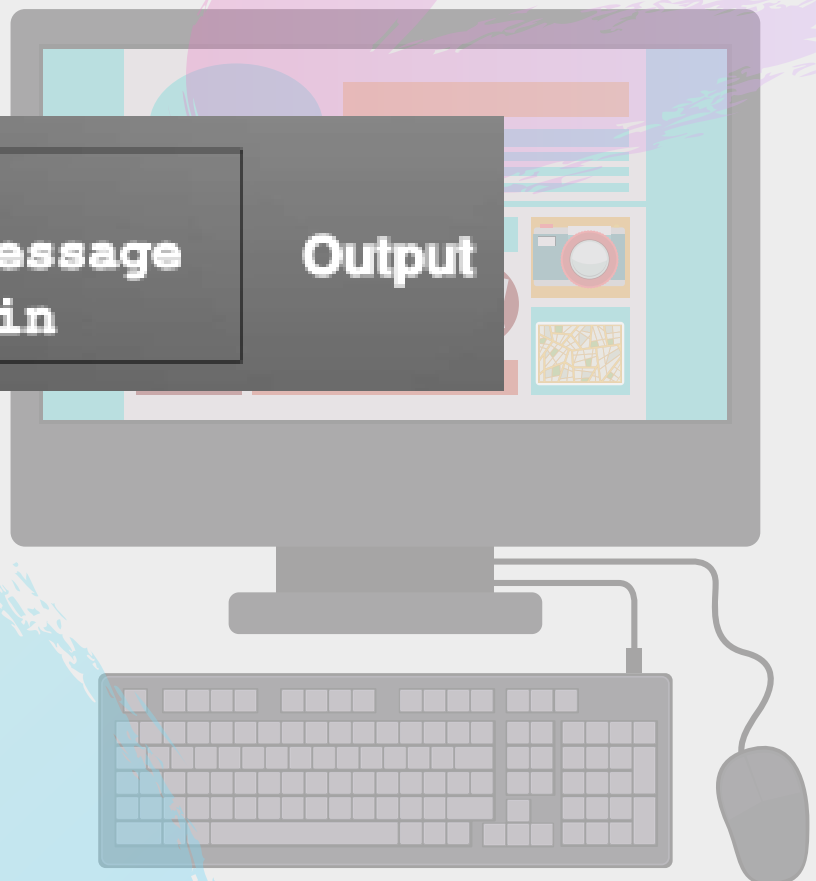


EXAMPLE: A complete program showing the use of functions in C program.

```
#include <stdio.h>
void display_message (void);    // function prototype
void main (void)
{
    printf("In function main\n");
    display_message();          // function call
    printf("Return to function main\n");
}
void display_message(void)      // function definition
{
    printf("In function display_message\n");
}
```

```
In function main
In function display_message
Return to function main
```

Output



# IDENTIFY THE TYPES OF FUNCTIONS

## Pre Defined Functions

Built in functions provided by C

Input / Output Functions  
`printf()`  
`scanf()`  
`getchar()`  
`putchar()`

## User Defined Functions

Functions that are written by users/programmers themselves to carry out various individual tasks.

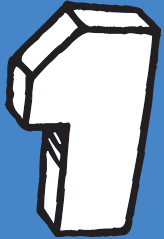
Functions  
without input parameter  
and do not return any value

Functions  
without input parameter  
and return a value

Functions  
without input parameter (1-more)  
and do not return any value

Functions  
without input parameter (1-more)  
and return a value

# THE DIFFERENCES BETWEEN TYPES OF FUNCTION



## PRE DEFINED FUNCTION

Pre -defined at the Standard C Libraries

Example : `printf()`, `scanf()`,

What we need to do is to use them is to include the appropriate header files.

Example: `#include <stdio.h>`

`#include <math.h>`

What contained in the header files are the prototypes of the standard functions. The functions definitions (the body of the functions) has been compiled and put into a Standard C Library which will be linked by the compiler during compilation.

# 2

## USER DEFINED FUNCTION

The four types of programmer defined functions discussed are categorized based on a value returned and input parameters from each function.

Can be determined from the function prototypes.

Type	Returned Value	Input Parameter	Function Prototype - Examples
1	No	No	<code>void functionX (void);</code>
2	Yes	No	<code>int functionX (void);</code> <code>char functionY (void);</code> <code>float functionZ (void);</code>
3	No	Yes (One or More)	<code>void functionX (int a);</code> <code>void functionY (float p, int r);</code>
4	Yes	Yes (One or More)	<code>int functionX (int a);</code> <code>float functionY (float p, int r);</code> <code>char functionZ (char c, char d);</code>

# COMPONENTS IN THE USER DEFINED FUNCTION

1

## Function Prototypes

It is a function declaration or function head which is defined by the programmer, before the function main.

2

## Function Definition

Is known as function implementation which define the function somewhere in the program.

3

## Return Statement

A must appear anywhere in the body of the function definition.

4

## Call Statement

Call the function whenever it needs to be used.

In order to write and use your own function, you need to

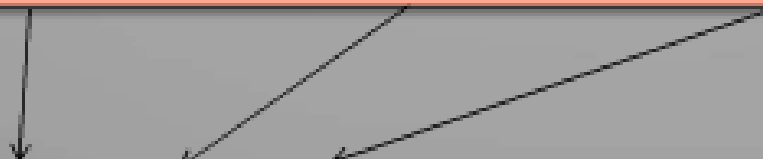
# 1 Declare Function Prototypes

Also called function declaration / prototypes.

The format:

*type-of-returned-value* *function-name* (*parameter-list*);

**Example:**



```
void roof(void);  
int getScore();  
Void calculateTotal(int);  
char testing(char, char);
```

# 2 Call Statement

Call the function whenever it needs to be used.

The format:

*function-name* (*parameter-list*);  
*variable-to hold-returned-value*  
*= function-name* (*parameter-list*);

**Example:**

```
roof();  
printMenu(3);  
printMenu(x+y, sum);  
val = printMenu(x);
```

# 3

## Define Function

Also known as function definition / implementation, which define the function somewhere in the program.

### The format:

Function definition header - must be the same as function prototype

```
type-of-returned-value function-name  
(parameter-list)  
{  
    declaration-statement;  
    executable-statement;  
}
```





# THE DIFFERENCES BETWEEN FUNCTION CALL BY VALUE & REFERENCES

Used when invoking functions and the differences are:



## Call by value

- Copy of argument passed to function
- Changes in function do not effect original
- Use when function does not need to modify argument
- Avoids accidental changes



## Call by reference

- The reference ( memory address) of the variable is passed to the function.
- Changes in function effect original
- Only used with trusted functions

# WRITE C PROGRAMME USING USER DEFINED FUNCTION

function type - void means the function does not return any value

```
#include <stdio.h>
void printRect(void); //declaring a function prototype
void main(void)
{
    printRect(); //calling a function
    printRect();
    printRect();
}
void printRect(void)
{
    printf ("-----\n"
           "|                               | \n"
           "-----");
}
```

list of parameter type - void means NO input parameter

() blank means function being called does not receive parameter

a function can be called repeatedly

defining a function - usually after the end of main () function

Output

```
-----
|                               | 
-----
```

# EXAMPLE OF FUNCTION

```
return    function
type      name    parameter(s)
  ↙       ↓       ↘
int factorial(int n) {
    int result = 1;
    ...
    return result;
}
```

body of function

return the result

The diagram shows a C++ function definition for factorial. Annotations with arrows point to parts of the code: 'return type' points to 'int', 'function name' points to 'factorial', 'parameter(s)' points to '(int n)', and 'body of function' points to the curly braces. An arrow labeled 'return the result' points to the 'return' statement.

Define the function `factorial ( )`, which take a single parameter "int n" and also returns a value of type `int`. Then call this function multiple times from `main ( )` to calculate the factorials of the integers between 1 and 10.

# EXAMPLE OF FUNCTION

```
#include <stdio.h>

int factorial(int n) {
    int result = 1;
    int ctr;
    for ( ctr = 2; ctr <= n; ctr++) {
        result *= ctr;
    }
    return result;
}

main()
{
    int i;
    int fact_i;
    for (i = 1; i <= 10 ; i++) {
        fact_i = factorial(i);
        printf("factorial(%d) is %d\n", i, fact_i);
    }
}
```

C program

Output

```
factorial(1) is 1
factorial(2) is 2
factorial(3) is 6
factorial(4) is 24
factorial(5) is 120
factorial(6) is 720
factorial(7) is 5040
factorial(8) is 40320
factorial(9) is 362880
factorial(10) is 3628800
```

# **R**EVIEW **Q**UESTION **E**XERCISES

# QUESTIONS & EXERCISES

1. What is the purpose of `scanf()` and `printf()` functions?
2. Describe formatted Input/Output functions in C language.
3. What is meant by a control string used in I/O statement? Discuss it.
4. Explain `getchar()`, `putchar()`, `gets()` and `puts()`
5. When printing a float or double with `%f`, how many digits after decimal does `printf()` output?
6. What is the purpose of the following in `printf()`:
  - a. `%lf`
  - b. `%6.2f`
  - c. `\a`
  - d. `\t`
7. Using one `printf()` statement only, print following message:  
Remainder can be found using /  
and `"%"` isn't a special symbol.



# QUESTION & EXERCISE

8. Find syntax error in following program and write equivalent corrected code:

```
#include<stdio.h>
main
{
int a; b=20;
10=a;
int c;
c = a+b
printf ('the sum='c);
}
```

9. What will be the output of the following program?

```
#include<stdio.h>
main ( )
{
char a,b;
a = 'b'
b=a;
printf("b=%c\n",b);
}
```

# QUESTIONS & EXERCISES

10. Write a C program to compute the sum of the two given integer values. If the two values are the same, then return triple their sum.

Expected Output:

3

12

11. Write a C program to get the absolute difference between n and 51. If n is greater than 51 return triple the absolute difference.

Expected Output:

6

21

0

12. Write a C program to check two given integers, and return true if one of them is 30 or if their sum is 30.

Expected Output:

1

1

0

# QUESTIONS & EXERCISES

13. Write a C program to check a given integer and return true if it is within 10 of 100 or 200.

Expected Output:

1

1

0

14. Write a C program to check whether a given positive number is a multiple of 3 or a multiple of 7.

Expected Output:

1

1

1

0

15. Write a C program to check whether a given temperatures is less than 0 and the other is greater than 100

Expected Output:

1

1

0

# QUESTIONS & EXERCISES

16. What is variable initialization and explain why is it important.
17. Explain details THREE (3) types of programming.
18. Illustrate with a suitable diagram to explain the different between compiler and interpreter.
19. Developing a program involves steps similar to any problem solving task. List the stage involved in a problem solving.
20. Elaborate the keyword which is used to transfer the controls back to a calling function from a function and write a sample program to justify the answer.
21. In understanding the structure of the C Programs, explain in details using of comments in programs.
22. Explain the debugged in C Programs in details.
23. List the types of control statement.

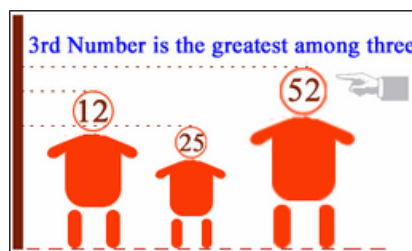
# QUESTIONS & EXERCISES

24. Write the output of the following C code.

```
#include<stdio.h>
void main()
{
    if(0xA)
    if(052)
    if('xeb')
    if('12')
    printf("Our Edu");
    else:
    else:
    else:
    else:

}
```

25. With help of diagram below, write a proper C program which is find the largest three number



# QUESTIONS & EXERCISES

26. Write a C program that accepts an employee's ID, total worked hours of a month and the amount he received per hour. Print the employee's ID and salary (with two decimal places) of a particular month.

Test Data:

Input the Employees ID (Max. 10 chars): 0342

Input the working hrs: 8

Salary amount/hr: 15000

Expected Output:

Employees ID = 0342

Salary = RM 120000.00

27. Write a C program to display following variables.

$a + c$ ,  $x + c$ ,  $dx + x$ ,  $((int) dx) + ax$ ,  $a + x$ ,  $s + b$ ,  $ax + b$ ,  $s + c$ ,  $ax + c$ ,  $ax + ux$

Variable declaration :

`int a = 125, b = 12345;`

`long ax = 1234567890;`

`short s = 4043;`

`float x = 2.13459;`

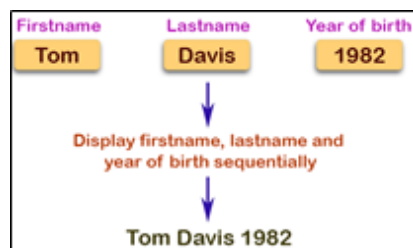
`double dx = 1.1415927;`

`char c = 'W';`

`unsigned long ux = 2541567890;`

# QUESTIONS & EXERCISES

28. Write a C program to read a password until it is correct. For wrong password print "Incorrect password" and for correct password print "Correct password" and quit the program. The correct password is 1234.
29. Write a program in C that reads a first name, last name and year of birth and display the names and the year one after another sequentially.

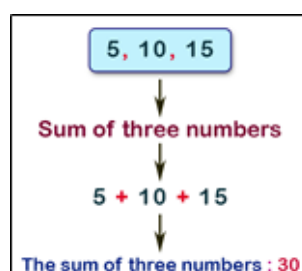


30. Write a program in C to calculate the sum of three numbers with getting input in one line separated by a comma.

Expected Output:

Input three numbers separated by comma: 5,10,15

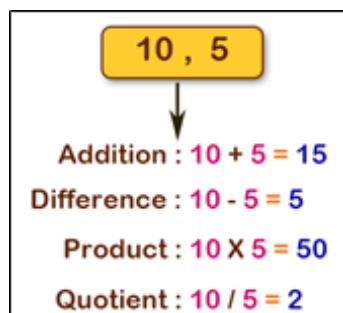
The sum of three numbers: 30





# QUESTIONS & EXERCISES

31. Write a C program to perform addition, subtraction, multiplication and division of two numbers.



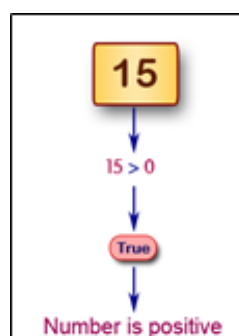
32. What are some ways information is input into the computer?

33. Which of these are examples of information being processed in a computer?

34. What does an algorithm have to do with processing?

35. A precise sequence of instructions for processes that can be executed by a computer is called a(n) ...

36. Write a C program to check whether a given number is positive or negative.



# QUESTIONS & EXERCISES

37. Write a C program to find the eligibility of admission for a professional course based on the following criteria:

Marks in Maths  $\geq 65$

Marks in Phy  $\geq 55$

Marks in Chem  $\geq 50$

Total in all three sub ject  $\geq 190$  or

Total in Math and Physics  $\geq 140$

38. What a syntax for C Ternary Operator.

39. What is the output of the C statement below and explain it.

```
int main()
{
    int a=0;
    a = 5<2 ? 4 : 3;
    printf("%d",a);

    return 0;
}
```

40. What is the output of the C Program below.

```
int main()
{
    int a=0;
    a = printf("4");
    printf("%d",a);

    return 0;
}
```

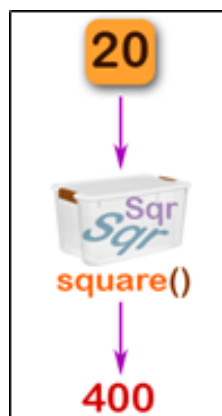
# QUESTIONS & EXERCISES

41. Write a program in C to find the sum of the series  $1!/1 + 2!/2 + 3!/3 + 4!/4 + 5!/5$  using the function.

$$\frac{1!}{1} + \frac{2!}{2} + \frac{3!}{3} + \frac{4!}{4} + \frac{5!}{5}$$
$$= 1 + 1 + 2 + 6 + 24$$

The sum of the series is : 34

42. Write a program in C to find the square of any number using the function.



43. Explain what is a static function.

44. It is possible to execute code even after the program exits the main () function.

45. Explain the use of flush () function.

# QUESTIONS & EXERCISES

46. Write a C program to perform addition, subtraction, multiplication and division of two numbers.

Expected Output:

Input any two numbers separated by comma: 10,5

The sum of the given numbers: 15

The difference of the given numbers: 5

The product of the given numbers: 50

The quotient of the given numbers: 2.000000

47. Write a program in C to store elements in an array and print it.

Test Data:

Input 10 elements in the array:

element - 0: 1

element - 1: 1

element - 2: 2

48. Write a program in C to read n number of values in an array and display it in reverse order

Test Data:

Input the number of elements to store in the array: 3

Input 3 number of elements in the array:

element - 0: 2

element - 1: 5

element - 2: 7

# QUESTIONS & EXERCISES

48. Write a program in C to read n number of values in an array and display it in reverse order

Test Data:

Input the number of elements to store in the array:3

Input 3 number of elements in the array:

element - 0:2

element - 1:5

element - 2:7

49. Write a program in C to copy the elements of one array into another array.

Test Data:

Input the number of elements to be stored in the array:3

Input 3 elements in the array:

element - 0:15

element - 1:10

element - 2:12

50. Write a program in C to show the simple structure of a function.

Expected Output:

The total is: 11

# QUESTIONS & EXERCISES

51. Write a program in C to find the square of any number using the function.

Test Data:

Input any number for square : 20

52. Write a program in C to swap two numbers using function.

Test Data:

Input 1st number : 2

Input 2nd number : 4

53. Write a program in C to check a given number is even or odd using the function.

Test Data:

Input any number : 5

54. What are static variables and functions?

55. What are the valid places where the programmer can apply Break Control Statement?

56. How can we store a negative integer?

# QUESTIONS & EXERCISES

57. Differentiate between Actual Parameters and Formal Parameters.
58. Can a C program be compiled or executed in the absence of a main()?
59. What do you mean by a Nested Structure?
60. Mention the features of C Programming Language.
61. What is an array?
62. What is /0 character?
63. How is a Function declared in C Language?
64. What is Dynamic Memory allocation? Mention the syntax.
65. Differentiate between call by value and call by reference.
66. Can a programmer create a customized Head File in C language?
67. Explain Local Static Variables and what is their use?
68. What is the difference between declaring a header file with <> and ""?



# QUESTIONS & EXERCISES

69. When should we use the register storage specifier?
70. Which statement is efficient and why? `x=x+1;` or `x++;`?
71. Can I declare the same variable name to the variables which have different scopes?
72. What are the different storage class specifiers in C?
73. Write a program to swap two numbers without using the third variable.
74. How can you print a string with the symbol % in it?
75. How can a programmer remove duplicates in an array?
76. Which structure is used to link the program and the operating system?
77. What are the limitations of `scanf()` and how can it be avoided?
78. Differentiate between the macros and the functions.
79. Suppose a global variable and local variable have the same name. Is it possible to access a global variable from a block where local variables are defined?

# QUESTIONS & EXERCISES

80. Write a program in C to check whether a number is a prime number or not using the function.

Test Data:

Input a positive number : 5

81. Write a program in C to get the largest element of an array using the function.

Test Data:

Input the number of elements to be stored in the array : 5

Input 5 elements in the array :

element - 0 : 1

element - 1 : 2

element - 2 : 3

element - 3 : 4

element - 4 : 5

82. Write a program in C to check armstrong and perfect numbers using the function.

Test Data:

Input any number : 371

# QUESTIONS & EXERCISES

83. Write a program in C to print all perfect numbers in given range using the function.

Test Data:

Input lowest search limit of perfect numbers: 1

Input highest search limit of perfect numbers: 100

84. Write a C programming to find out maximum and minimum of some values using function which will return an array.

Test Data:

Input 5 values

25

11

35

65

20

85. Write A C Program For Check Number Is Even Or Odd.

86. Write A C Program To Check Year Is Leap Year Or Not Means leap Year Has 366 Days in Year .

# QUESTIONS & EXERCISES

87. What is the output of the following program?

```
#include<stdio.h>

main()
{
    char s[] = "Fine";
    *s = 'N';

    printf("%s", s);
}
```

88. In C, what is the correct hierarchy of arithmetic operations?

89. What is the output of the following statement?

```
#include<stdio.h>

main()
{
    printf("%d", !0<2);
}
```

90. The maximum combined length of the command-line arguments as well as the spaces between adjacent arguments is -

a) 120 characters, b) 56 characters, c) Vary from one OS to another

# QUESTIONS & EXERCISES

91. According to ANSI specification, how to declare main () function with command-line arguments?

92. In the given below code, if a short int value is 5 byte long, then how many times the while loop will get executed?

```
#include<stdio.h>

int main ()
{
    int j = 1;
    while(j <= 300)
    {
        printf("%c %d\n", j, j);
        j++;
    }
    return 0;
}
```

93. What is x in the following program?

```
#include<stdio.h>

int main ()
{
    typedef char (*(arrfptr[3]))[10];
    arrfptr x;
    return 0;
}
```

94. What function can be used to free the memory allocated by calloc()?

95. A Variable name in C includes which special symbols?

# QUESTIONS & EXERCISES

96. What actually get pass when you pass an array as a function argument?

97. In the given below code, what will be return by the function get ()?

```
#include<stdio.h>

int get();

int main()
{
    const int x = get();

    printf("%d", x);
    return 0;
}

int get()
{
    return 40;
}
```

98. To print a float value which format specifier can be used?

99. What is the output of the following program?

```
#include<stdio.h>

main()
{
    char *s = "Hello";

    while(*s!=NULL)
        printf("%c", *s++);
}
```

100. In Decimal system a programmer can convert the binary number 1011011111000101 very easily.

# ANSWERS

Write a C program that accepts an employee's ID, total worked hours of a month and the amount he received per hour. Print the employee's ID and salary (with two decimal places) of a particular month.

Test Data:

Input the Employees ID (Max. 10 chars): 0342

Input the working hrs: 8

Salary amount/hr: 15000

Expected Output:

Employees ID = 0342

Salary = RM 120000.00

```
#include <stdio.h>
int main()
{
    char id[10];
    int hour;
    double value, salary;
    printf("Input the Employees ID(Max. 10 chars): ");
    scanf("%s", &id);
    printf("\nInput the working hrs: ");
    scanf("%d", &hour);
    printf("\nSalary amount/hr: ");
    scanf("%lf", &value);
    salary = value * hour;
    printf("\nEmployees ID = %s", id);
    printf("\nSalary=RM.2lf", salary);
    return 0;
}
```



# ANSWERS

Write a C program to display following variables.  $a + c$ ,  $x + c$ ,  $dx + x$ ,  $((int) dx) + ax$ ,  $a + x$ ,  $s + b$ ,  $ax + b$ ,  $s + c$ ,  $ax + c$ ,  $ax + ux$

Variable declaration:

`int a = 125, b = 12345;`

`long ax = 1234567890;`

`short s = 4043;`

`float x = 2.13459;`

`double dx = 1.1415927;`

`char c = 'W';`

`unsigned long ux = 2541567890;`

---

```
#include <stdio.h>
int main()
{
    int a = 125, b = 12345;
    long ax = 1234567890;
    short s = 4043;
    float x = 2.13459;
    double dx = 1.1415927;
    char c = 'W';
    unsigned long ux = 2541567890;

    printf("a+c=%d\n", a+c);
    printf("x+c=%f\n", x+c);
    printf("dx+x=%f\n", dx+x);
    printf("((int)dx)+ax=%d\n", ((int)dx)+ax);
    printf("a+x=%f\n", a+x);
    printf("s+b=%d\n", s+b);
    printf("ax+b=%d\n", ax+b);
    printf("s+c=%hd\n", s+c);
    printf("ax+c=%d\n", ax+c);
    printf("ax+ux=%lu\n", ax+ux);

    return 0;
}
```

# ANSWERS

Write a C program to read a password until it is correct. For wrong password print "Incorrect password" and for correct password print "Correct password" and quit the program. The correct password is 1234.

```
#include <stdio.h>

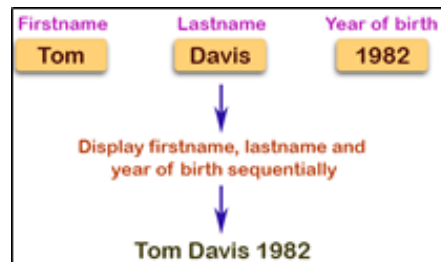
int main() {
    int pass, x=10;

    while (x!=0)
    {
        printf("\nInput the password: ");
        scanf("%d",&pass);

        if (pass==1234)
        {
            printf("Correct password");
            x=0;
        }
        else
        {
            printf("Wrong password, try another");
        }
        printf("\n");
    }
    return 0;
}
```

# ANSWERS

Write a program in C that reads a first name, last name and year of birth and display the names and the year one after another sequentially.



```
#include <stdio.h>

int main()
{
    char firstname[20], lastname[20];
    int bir_year;
    printf("Input your firstname: ");
    scanf("%s", firstname);
    printf("Input your lastname: ");
    scanf("%s", lastname);
    printf("Input your year of birth: ");
    scanf("%d", &bir_year);
    printf("%s %s %d\n", firstname, lastname, bir_year);
    return 0;
}
```

# Answers

Write a program in C to calculate the sum of three numbers with getting input in one line separated by a comma.

Expected Output:

Input three numbers separated by comma: 5,10,15

The sum of three numbers: 30

```
#include <stdio.h>

int num1,num2,num3;
int sum;
char line_text[50];

int main()

{
    printf("Input three numbers separated by comma: ");
    fgets(line_text, sizeof(line_text), stdin);
    sscanf(line_text, "%d, %d, %d", &num1, &num2,
&num3);

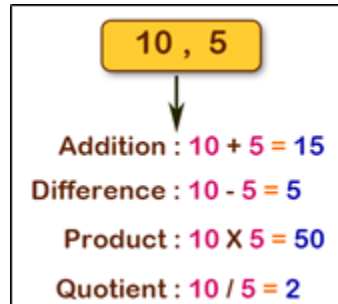
    sum = num1+num2+num3;

    printf("The sum of three numbers: %d\n", sum);

    return(0);
}
```

# ANSWERS

Write a C program to perform addition, subtraction, multiplication and division of two numbers.

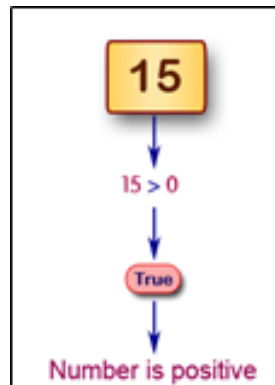


```
#include <stdio.h>
int main()
{
    int num1, num2;
    int sum, sub, mult, mod;
    float div;
    /*
     * Read two numbers from user separated by comma
     */
    printf("Input any two numbers separated by comma : ");
    scanf("%d,%d", &num1, &num2);

    /*
     * Performs all arithmetic operations
     */
    sum = num1 + num2;
    sub = num1 - num2;
    mult = num1 * num2;
    div = (float)num1 / num2;
    mod = num1 % num2;
    /*
     * Prints the result of all arithmetic operations
     */
    printf("The sum of the given numbers : %d\n", sum);
    printf("The difference of the given numbers : %d\n", sub);
    printf("The product of the given numbers : %d\n", mult);
    printf("The quotient of the given numbers : %f\n", div);
    printf("MODULUS = %d\n", mod);
    return 0;
}
```

# ANSWERS

Write a C program to check whether a given number is positive or negative.



```
#include <stdio.h>
void main()
{
    int num;
    printf("Input a number :");
    scanf("%d", &num);
    if (num >= 0)
        printf("%d is a positive number \n", num);
    else
        printf("%d is a negative number \n", num);
    return 0;
}
```

# Answers

Write a C program to find the eligibility of admission for a professional course based on the following criteria:

Marks in Maths  $\geq 65$

Marks in Phy  $\geq 55$

Marks in Chem  $\geq 50$

Total in all three sub ject  $\geq 190$  or

Total in Math and Physics  $\geq 140$

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

```
void main()
```

```
{ int p,c,m,tmp;
```

```
printf("Eligibility Criteria:\n");
```

```
printf("Marks in Maths  $\geq 65$ \n");
```

```
printf("and Marks in Phy  $\geq 55$ \n");
```

```
printf("and Marks in Chem  $\geq 50$ \n");
```

```
printf("and Total in all three sub ject  $\geq 190$ \n");
```

```
printf("or Total in Maths and Physics  $\geq 140$ \n");
```

```
printf("-----\n");
```

```
printf("Input the marks obtained in Physics :");
```

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

```
printf("Input the marks obtained in Chemistry :");
```

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

```
printf("Input the marks obtained in Mathematics :");
```

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

```
printf("Total marks of Maths, Physics and Chemistry : %d\n",m+p+c);
```

```
printf("Total marks of Maths and Physics : %d\n",m+p);
```



# ANSWERS

```
if (m>=65)
    if(p>=55)
        if(c>=50)
            if((m+p+c)>=190||(m+p)>=140)
                printf("The candidate is eligible for admission.\n");
            else
                printf("The candidate is not eligible.\n");
        else
            printf("The candidate is not eligible.\n");
    else
        printf("The candidate is not eligible.\n");
else
    printf("The candidate is not eligible.\n");
}
```

# ANSWERS

Write a program in C to find the sum of the series  $1!/1+2!/2+3!/3+4!/4+5!/5$  using the function.

$$\begin{array}{c} \frac{1!}{1} + \frac{2!}{2} + \frac{3!}{3} + \frac{4!}{4} + \frac{5!}{5} \\ \downarrow \\ = 1 + 1 + 2 + 6 + 24 \\ \downarrow \\ \text{The sum of the series is : 34} \end{array}$$

```
#include <stdio.h>

int fact(int);

void main()
{
    int sum;
    sum=fact(1)/1+fact(2)/2+fact(3)/3+fact(4)/4+fact(5)/5;
    printf("\n\nFunction : find the sum of 1!/1+2!/2+3!/3+4!/4+5!/5 :\n");
    printf("-----\n");
    printf("The sum of the series is : %d\n\n",sum);
}

int fact(int n)
{
    int num=0,f=1;
    while(num<=n-1)
    {
        f=f*f*num;
        num++;
    }
    return f;
}
```

# ANSWERS

Write a program in C to find the square of any number using the function.

```
#include <stdio.h>

double square(double num)
{
    return (num * num);
}

int main()
{
    int num;
    double n;
    printf("\n\nFunction : find square of any number :\n");
    printf("-----\n");

    printf("Input any number for square : ");
    scanf("%d", &num);
    n = square(num);
    printf("The square of %d is : %.2f\n", num, n);
    return 0;
}
```

# REFERENCES

Miller, G..P (2014). C Programming Absolute Beginner's Guide : (C Progr Absol Begin Guide 3rd Edition ed.) UK: Kindle Edition

Ba\_jpai, S. (. ((2007)). Introduction to Computer and C Programming ( (3rd Edition). ed.). NY: New Age international.

Gookiin, D. ((2004)). C For Dummies. India: Wiley Publishing International.

Kernighan, B. W. (2012). C Programming Language. UK: PRENTICE HALL SOFTWARE SERIES.

King, K. ((2008)). C Programming: A Modern Approach. UK: W.W.Norton & Company.

Michael, V. ((2007)). C Programming For The Absolute Beginner ((2nd Edition). ed.). UK: Course Technology PTR.

Publisher



**POLITEKNIK TUANKU SULTANAH BAHYIAH**

e ISBN 978-967-0855-94-3



9 7 8 9 6 7 0 8 5 5 9 4 3