-----Syllabus-----

==========

1.Introductiontodifferentprogramminglanguages

2.BasicsofProgramminglanguages

3.BasicofCprogramming

4.StartingCprogramming

5.Decisionmakingstatements(if,if-else,else-if,conditionaloperator)

6.Loops(Iterativestatements)

7.Switch-case

8.FuntionsandPointers

9.ArraysandString

10.StructureandUnion

11.FileHandling

==========

==========

Whatisapplications/useofaprogramminglanguage?

-Weuseaprogramminglangaugetodevelopdifferenttypesofapplications/softwares.

-Theseapplications/softwaresaretocommunicatewiththesystem.

-Wecanpreparesoftwaresforcomputers,electroniccircuits,mechanicalmachines(for automation),smart-phones,smart-watches,consumerelectronicdevices,etc.

-Wehavedifferentprogramminglanguagestodoabovejobs.Becausedifferentprogramming languagesaretopreparedifferenttypesofsoftwares(fordifferenttechnologies)

-ArduinoandRaspberry-PIarealsoveryfamouselectroniccircuits.Toprogram forsuch circuits---->Python,R,Assemblylanguage,etc..

-TheArduinoandRaspberry-PIareusedforautomation.

==========

==========

*Andeverysoftwarethatwearecurrentlyusing,isanoutputofaprogramwritteninspecific programminglanguage.

*Forexample:MSoffice,Anti-Virus,Games,Notepad,OperatingSystem,Web-sites,Web Servers,Mailingservices,SocialMediaapplications,etc...

==========

==========

***Everyprogramminglanguage,hasitsownfeaturesandcapabilities.

***Thereforeasingleprogramminglanguagecannotperformeverypossible/requiredoperation.

***Thereforeasingleprogramminglanguagecannotbeusedtodevelopalltypesofsoftwares.

==========

==========

WhatisaComputer?

-Acomputerisanelectronicmachinethattakesinput,processesonitandreturnsoutput.

-Andtodosuchoperations,incomputer,therearemanyunits;likeALU,CU,PU,I/Oetc..C Commonly

O Operated

M Machinethat

P Perfectly

U Uses

T Technicaland

E Electrical

R Resources.

-Here,Technicalresources--->Softwares

Electricalresources--->Hardwares

*Weknowthat,acomputerismainlymade-upoftwocomponents:

-Hardware

-Software

==========*Programminglanguageisusedtodevelopdifferenttypesofsoftwares.

*Differentprogramminglanguagehasdifferentfeaturesandcapabilities.

*Thereforesingleprogramminglanguagecannotdo/perform/createalltypesofsoftwares.

*Everysoftwarethatwearecurrentlyusingareultimatelyoutputofaprogram writtenin

particularprogramminglanguage.

*Acomputerismainlymade-upoftwocomponents:

-Hardwares

-Softwares

==========

===

Areaofapplicationsoffamouslanguages:

C,C++ ---->consoleapplications,small-scalegames

VisualBasic(VB),

VB.net,C#, ---->WindowsbasedGUIapplications,OperatingSystem,

VisualC++ Networkbasedapplications

XML,ASP.net --->Dynamicweb-pages

HTML --->Staticweb-pages

GUI--->GraphicalUserInterface

XML--->eXtensibleMarkupLanguage

*OnlyfirstversionofAndroid(in2001)wasfullypreparedinJAVA.

*Currentlyalmosteverysoftware,OS---->arepreparedfromhybridlanguages.

*Dynamicweb-pagesmeans--->theweb-pageswhosecontentschangesasperuser'slogin

(user'srequest)

Therearetwotypesofweb-pages:

-staticweb-pages--->sameforeveryone

-dynamicweb-pages----
>differentfordifferentusers*Weusecomputerprogramminglanguages,todevelopcomputersoftwares/
applications.

*Wehavemanymanycomputerprogramminglanguages.Alltheseprogramminglanguagesare

usedtodevelopdifferenttypesofsoftwares.

*Thereasonis,differentprogramminglanguageshavedifferentfeaturesandcapabilities.

*Thereisnosinglelanguagethatcanalonedevelopeverytypesofsoftwares.

IMP

Whenaprogramminglanguagewasgettinginvented/developed,atthattimeonlyitsinventor(s)

decidesthefeatures,purposeandapplicationofthatprogramminglanguage.

*TherearetwomaincomponentsinComputersystem:

-Hardwares

-Softwares

==========

==========*Weuseprogramminglanguage,todevelopdifferenttypesofsoftwares.

*Wehavedifferentprogramminglanguagestodevelopdifferenttypesofsoftwares.

*Differentprogramminglanguageshavedifferentfeaturesandarchitectureandprogram

skeletonandcapabilities.

Areaofapplicationsoffamouslanguages:

C,C++ ----> consoleapplications,smallscalegames

VB,VB.net

VisualC++, ----> WindowsbasedGUIapplications,OperatingSystem,

C# networkbasedapplications,largescalegames

HTML,ASP ---->staticweb-pages

XML,PHP, ----> dynamicweb-pages

ASP.net

Assemblylanguage ---->circuitprogramming,compilerdesigning,drivers

==========

staticweb-pages--->aretheweb-pageswhosecontentsremainssameforallusers.

[eg.collegewebsite,newswebsite,cityinfowebsite,

mahanagarpalikawebsite,govtdepartmentwebsite.]

dynamicweb-pages--->aretheweb-pageswhosecontentschangesasuser'srequestand

login.eg.instagram,mailbox,facebook,amazon-flipkart,any

ticketreservation,etc]

==========

Fullforms:

HTML HypterTextMarkupLanguage

ASPActiveServerPages

XMLeXtensibleMarkupLanguage

PHPPersonalHomePage[HypertextPre-Procesing]

==========*Weuseprogramminglanguages,todevelopdifferenttypesofsoftwares/applications.

*Therearedifferentprogramminglanguages.

*Differentprogramminglanguagehasdifferentfeaturesandcapabilities.

* Different programming languages are used to develop different types of softwares/applications.

Areaofapplicationoffamousprogramminglanguages:

C,C++ ---->consoleapplications,smallscalegames

VB,VB.net ---->WindowsbasedGUIapplications,networkbasedapplications

C#,VisualC++ Operatingsystems,large-scalegames

HTML,ASP ---->staticwebsites

XML,ASP.net, ---->dynamicwebsites

PHP

Assemblylanguage ---->circuitprogramming,drivers,compilerdesign

JavaScript,VBScript ----> toaddscriptsinweb-pages

(Scriptinglanguages)

==========

*Thelanguageswhicharepreferredtodevelopwebpages,arecalledas"markuplanguages"

*Thestatementsofmarkuplanguagesarewritteninangularbraces<>.

*Thesearecalledas"TAG"

*ThemarkuplanguagesonlyhaveTAG.Theydon'tsupportanylogicalstatements.

*Toperformanylogicaloperationsinawebpage,weneedtoaddascriptinit.

==========

*FirstversionofC-languagewasinitiallyinventedin1972byDennisRitchie.

*SecondversionofC-languagewasin1983

*ThirdversionofC-languagewasin1999 ---->TurboC3 or C99

*FourthversionofC-languagewasin2003-04

==========

BCPL ----> BasicCombinedProgrammingLanguage

CPL---->CombinedProgrammingLanguage

PASCAL---->

COBOL ---->COmmonBusinessOrientedLanguage

SIMULA

ALGOL

ASSEMBLYLANGUAGE

==========Areaofapplicationoffamousprogramminglanguages:

C,C++ ---->consoleapplications,smallscalegames

VB,VB.net ---->WindowsbasedGUIapplications,networkbasedapplications

C#,VisualC++ Operatingsystems,large-scalegames

HTML,ASP ---->staticwebsites

XML,ASP.net, ---->dynamicwebsites

PHP

Assemblylanguage ---->circuitprogramming,drivers,compilerdesign

JavaScript,VBScript ----> toaddscriptsinweb-pages

(Scriptinglanguages)

PERL ----> biologicalapplications,Drugdiscoveryapplications

(Pharmacompanies)

SQL ---->DatabaseManagementapplications

==========

SQL-StructuredQueryLanguage

==========*Don'tthink,thatyourprogram'sfileisdirectlyapplied/feedtoprocessor.

*Therearemanyoptions/waysthatarefollowedinprogrammingtools.

*Platformmeans--->theenvironmentinwhichthatprogramisexecuting.

ouroperatingsystem,hardwares

----Typesofprogramminglanguages----

==========

*Therearemanytypesofprogramminglanguages,basedondifferentcriteria.

1.Typesofprogramminglanguages(basedonplatform)

----------Typesofprogramminglanguages----

==========

*Therearemanytypesofprogramminglanguages,basedondifferentcriteria.

1.Typesofprogramminglanguages(basedonplatform)

---

*Theplatform istheO.S.orHardwareconfigurationonwhichanapplication/program will execute.

*Therearetwotypesofprogramminglanguagesbasedonplatform.

-platformdependentlanguages

-platformindependentlanguages

a.platformdependentlanguages

Aplatformdependentlanguageisthelanguagewhichadoptssomefeaturesfromtheplatform onwhichitisexecuting.[Thereforesomeoutputofplatform dependentlanguagesvaries platformtoplatform]

E.g.---->C,C++,VisualBasic

b.platformindependentlanguages

Aplatformindependentlanguageisthelanguagewhichdoesnotadoptanyofthefeaturefrom theplatform onwhichitisexecuting.[Thereforeoutputofplatform independentlanguage remainssameoneveryplatform]

E.g.---->Java,Python,PHP,XML,PERL

Thereasonis,everyplatform independentlanguagehasit'sownexecutionenvironment.That environmentiscalledas"virtualenvironment".

==========

*TheEditorsandIDEsarelikeintermediatorbetweenprogrammerandcompiler.

*Asaprogrammer,wedon'twanttowritethecommandsandprocessestousecompiler.

ThereforeasafacilityweprefertouseEditorsandIDEs.

IDE--->IntegratedDevelopmentEnvironment

==========

*AnIDEhasmuchmorefacilityforprogrammer,ascomparedtoEditor.

==========1.Typesofprogramminglanguages(basedonplatform)

---

a.Platformdependentlanguages

b.Platformindependentlanguages

Platformdependentlanguageisthelanguagewhichadoptssomefeaturesfromtheplatformon whichitisexecuting.

Thereforesomeoftheoutputofplatformdependentlanguageschangesasplatformchanges.

e.g.---->C,C++andVB

Platformindependentlanguageisthelanguagewhichdoesnotadoptanythinganyfeaturefrom platformonwhichitisexecuting.

e.g.--->Java,Python,PHP,XML,PERL,etc...

Thereforeoutputofeveryprogramwillremainsameoneveryplatform.

==========

==========

IMP

Tocompileandrunaprogramweneedtousesometools.Thesetoolsare:

-compiler

-translator

-linker

-assembler

-interpreter

-loader

*Programcompile---->meanstocheckforerrors(checkforvalidstatements)

The"compiler"isthetoolwhichcompilestheprogram.

*Programexecute/run---->meansexecutingthatprogramandgettheoutput/result.

The"interpreter"isthetoolwhichexecutes/runstheprogram.

WhatisIDEandEditor?

---

*TheIDEsandEditorsarethesoftwares/applicationsinwhichwecantype/prepareour program.

*TheIDEsandEditorswilltakeyourprogramtocompiler-interpreter.AndadditionallytheIDEs andEditorswillalsoprovidemorefacilitiestoprogrammer.

*TheIDE--->IntegeratedDevelopementEnvironment

*TheIDEprovidesmuchmorefacilitytoprogrammer,ascomparedtoEditors.

Students,trainee,learning---->editor

developer,employee---->IDEs

==========

=====WhatisIDEandEditor?

---

*TheIDEsandEditorsarethesoftwares/applicationsinwhichwecantype/prepareour program.

*TheIDEsandEditorswilltakeyourprogramtocompiler-interpreter.AndadditionallytheIDEs andEditorswillalsoprovidemorefacilitiestoprogrammer.

*TheIDE--->IntegeratedDevelopementEnvironment

*TheIDEprovidesmuchmorefacilitytoprogrammer,ascomparedtoEditors.

Students,trainee,learning---->editor

developer,employee---->IDEs

*TherearedifferentIDEsandEditorsfordifferentprogramminglanguages.

*WhetheryouuseanyIDEorEditor,butyoumusthavetoinstallcompiler,interpreterandother toolsofthatprogramminglanguage.

programminglanguage IDE Editor

C,C++ Codeblocks,Notepad++ TurboC,Notepad,

Cppdroid,Clion, Wordpad

VSCode

Python Pycharm,WINGIDE, Notepad,TextPad

VSCode,Notebook

Java NetBeans,Eclipse, BlueJ,TextPad,Notepad

JStudio

==========

==========

1.Typesofprogramminglanguages(basedonplatform)

-platformdependentlanguages

-platformindependentlanguages

==========

==========

2.Typesofprogramminglanguages(basedoncase-sensitivity)

---

*Therearetwotypesofprogramminglanguages,basedoncase-sensitivity.

-casesensitivelanguages

-caseinsensitivelanguages

Incasesensitivelanguage,thecompilerconsidersthedifferencebetweenupper-caseandlower□case.i.e.ifsomethingisoriginallyexpectedinlower-casethenitmustbewritteninlower-case

only.Andifsomethingisoriginallyexpectedinupper-casethenitmustbewritteninupper-case

only.

e.g.--->C,C++,Java,Python,C#,PERL

Incaseinsensitivelanguage,thecompilerdoesnotconsidersthedifferencebetweenupper-case

andlower-case.

e.g.---->SQL(fordatabasemanagement),everyMarkupLanguages(XML,HTML,etc..)

==========***Theexecutionspeedof C-languageisfasterthananyotherlanguage.

[Cis----->ProcedureOrientedProgramming(POP)]

==========Typesofprogramminglanguages:

1.Typesofprogramminglanguages(basedonplatform)

-Platformdependentlanguages[e.g.--->C,C++,VB]

-Platformindependentlanguages[e.g.--->Java,Python,PERL,PHP,HTML,C#]

2.Typeofprogramminglanguages(basedoncase-sensitivity)

-casesensitivelanguages[e.g.C,C++,Java,Python,PERL,C#]

-caseinsensitivelanguages[MarkuplanguagesandSQL]

==========

WhatisERRORandBUG?

---

*AnERRORismainlybecauseofprogrammer'stypingmistakesanddeclrationerrors.

*ThecompilerwillfindandshoweveryERROR.

==========-Weuseprogramminglanguagetodevelopdifferenttypesofsoftwares/applications.

-Therearemanyprogramminglanguagesandeveryprogramminglanguagehasdifferent

featuresandcapabilities.

-Thereisnosinglelanguage,thatcanperformeverything.

-Becauseeveryprogramminglanguagehasdifferentareaofapplications.

-Weknowthat,totypeanyprogramweshouldseproperIDEorEditor.

*Typesofprogramminglanguages:

1.Typesofprogramminglanguages(basedonplatform)

-Platformdependentlanguages[C,C++,VB]

-Platformindependentlanguages[Java,Python,PERL,PHP]

2.Typesofprogramminglanguages(basedoncase-sensitivity)

-Casesensitivielanguages[C,C++,Java,Python,VB,PHP,PERL]

-Caseinsensitivielanguages[SQL,QBEandMarkuplanguages]

==========

*Weknowthat:

compile---->Tocheckforerrors.Whethereverystatementiscorrectlywrittenornot.

whethereverymemoryblockiscorrectlydeclaredornot.Whetherproper

operatorsareusedornot.

*The"compiler"isresponsibletocompileourprogram.

interprete---> Toruntheprogram.Toexecutetheprogram.Toseetheoutputofprogram.

*The"interpreter"isresponsibletointerprete/runourprogram.

step1:Wewriteourprogramandsendtocompilertocheckforerrors.

step2:Ifourprogramiserrorfree,thencompilertranslatesourprogramintoanew

separatemachine-languagefile.

[Ifourprogramhasanyerror,thenitwillnotcreatethattranslatedfile]

step3:Thisnewmachinelanguagefileisprovidedtointerpretertorun.

***YouroriginalCProgramfilenevergetsexecuted.

***Theinterpreterisnotabletounderstnad/run/executeouroriginalprogram'sfile.

==========1.Typesofprogramminglanguages(basedonplatform)

-Platformdependentlanguages

-Platformindependentlanguages

2.Typesofprogramminglanguage(basedoncase-sensitivity)

-casesensitivelanguages

-caseinsensitivelanguages

==========

Thecompilation--->theprocessofcheckingforerrors.

Theinterpretation--->theprocessofexecutingtheprogram.

Thetranslation--->theprocessoftranslatingourCprogramintomachine-languagefile.

==========

*OnlyinC/C++ourprogramfilegetstranslatedintoM/Clanguage.

*Butinotherlanguages,therearedifferentmechanismoftranslation.

==========

3.Typesofprogramminglanguages(basedoncompilationandinterpretationprocess)

*Therearetwotypesoflanguagesbasedoncompilationandinterpretationprocesses.

-Compiledlanguages

-Interpretedlanguages

Acompiledlanguageisthelanguageinwhichthecompleteprogram/blockfirstgetscompiled

thengetsexecuted.

[e.g.--->C,C++,Java]

Aninterpretedlangaugeisthelanguageinwhichcompilationandexecutionhappensline-by-line.

[e.g.--->Python,PHP,PERL,HTML,VB.net,C#,XML]

==========

========*Wecanpreparedifferenttypesofsoftwares/applicationsbyusingprogramminglanguages.

*Everyprogramminglanguagehasdifferentareaofapplications.Becauseeveryprogramming

languagehasdifferentfeaturesandcapabilities.

Typesofprogramminglanguages:

1.Typesofprogramminglanguages(basedonplatform)

-platformdependentlanguages

-platformindependentlanguages

2.Typesofprogramminglanguages(basedoncase-sensitivity)

-casesensitivelanguages

-caseinsensitivelangauges

3.Typesofprogramminglanguages(basedoncompilationandinterpretation)

-compiledlanguages

-interpretedlanguages

Thecompiledlanguagesarethelanguageswhichcompilescompleteprogram/blockatatime.

Andtheninterpretes/executescompleteprogram.

[e.g.--->C,C++,Java]

Theinterpretedlanguagesarethelanguageswhichcompilesandinterpretesline-by-line.

[e.g.--->Python,VB.net,C#,PHP,PERL,HTML,JavaScript]

==========

=====

==========

=====

Areaofapplicationsoffamousprogramminglanguages

C,C++ ---->consoleapplications,smallscalegames

VB,VB.net, ----> WindowsbasedGUIapplications,networkingapplications,

C#,VisualC++ OperatingSystem,large-scalegames

HTML,ASP ---->staticwebpages

ASP.net,PHP, ---->dynamicwebpages

XML

Assemblylanguage ----> circuitprogramming,programminglanguagetools,

driver

JavaScript,VBScript ----> Toaddscriptsinweb-pages.

SQL ----> DatabasemanagementapplicationsPERL ----
>Drugdiscoveryapplications,Biologicalapplications

SWIFT ---->Todevelopi-phoneandi-OSapplications.

Python ---->consoleapplications,static&dynamicweb-pages,games,

automationrelatedapplications,IOT,Datasciencerelated

applications,etc...

Java ---->consoleapplications,static&dynamicweb-pages,

consumerelectronicdeviceapplications,web-servers,

linuxapplications,Frameworks,etc...

==========

=====

==========

=====

*Therearetwoimportanttoolsineverylanguage:

-compiler --->checksforerrors

-interpreter---> executestheprogram

*Thetranslator--->converts/translatestheCprogramfileintoM/Clanguagefile.

==========

*Wecansayfollowingthingsabout"C-Language":

-Cisaplatformdependentlanguage

-Cisacase-sensitivelanguage

-Cisacompiledlanguage

==========

4.Typesofprogramminglanguages(basedonprogramskeleton)

---

*Therearetwotypeslangaugesbasedonprogramskeleton:

-structuredprogramminglanguage

-unstructuredprogramminglanguage

A structured programming langauge meansa language inwhich we cannotwrite open statements.Thestatementsmustbeinsideafunction/block.

[e.g.--->C,C++,Java,C#,VC++,VB.net]

Anunstructured programming language meansa languageinwhichwe canwriteopen statements.

[e.g.--->Python,PHP,PERL,SQL,Markuplanguages,QBE]

*Everyprogramminglangaugeshasindividualsyntax.

==========

==*Therearetwotypesofsoftwares:

-Systemsoftwares

-Applicationsoftwares

*WeknowthatanOperatingSystem isa"System Software".Anditisaplatform onwhich softwaresandhardwarescollaboratesandworkstogather.

*TherearetwotypesofOS:

-CommandbasedOS

-GUIbasedOS

*ThecommandbasedOShasonlycommandstorun.Theuserhastotypedifferentcommands toperformrequiedoperations.[NoGUI,NoDesktop,Noright-click,Nopop-up,Noicons]

e.g.UNIX,MSDOS

*TheGUIbasedOShasgraphicalappearance.Theuserhasdifferenticons,mouse-cursor, desktop,my-computer,etc..foreasyusage.[moreuser-friendly,reducesuser'sefforts,noneed toremembereverycommand,attractive,etc..]

e.g.MicrosoftWindows,Linux,MACOS,UBUNTU,ANDROID,IOS

****InMicrosoftWindows,wearealsoprovidedwithMS-DOS.

****InLINUX,wearealsoprovidedwithUNIX.

==========

*WhenOSstartedinaround1965,theinitialOSwereonlycommandbasedOS.

*TherewasnothinglikeGUI.Nothinglikecolorfulmonitors,nothinglikeaudioclips.

*Thesecommand-basedOSwereonlytostorethedata/files/information.

*Atthattime,thereweretwofamouscommand-basedOS---->DOSandUNIX

*Inaround1980-82,verysmallGUIbasedO.S.waslaunched--->MINIX(MUNIX).ThisMINIXwas onlyabletodobasic4-5operations.

*In1991,anEngineeringstudent,"LinusTrovaldus"hasinventedanewGUIbasedO.S.onthebase onUNIX.

Linus+UNIX====>Linux

*ThisnewO.S.wasthefirststeptowardsGUIbasedO.S.

UNIX ka GUI====>LINUX

DOSka GUI====>WINDOWS

GUI--->GraphicalUserInterface

[TheinterfacetouserisGraphical]

==========

=====*EvenifweareusingWINDOWS(GUIbased)thenalsotheyprovideDOSasaninternal

application.IfanylogicalrequirementarrivesforusingDOS,thenwecaneasilyuseit.

DOS--->DiskOperatingSystem

*EvenifweareusingLINUX(GUIbased)thenalsotheyprovidesUNIXasinternalapplication.If youwanttofire/runanycommandthenwecandoitinUNIX.

*Ifyouhavecompilerandinterpreterofanyprogramminglanguage,thenyoucanfire/runthe commandtousethatcompilerandinterpreter.

*JIT--->JustInTimecompiler

==========

Windows--->airfilledballoon[weareusingit'soutput]

Linux--->unfilledballoon[weareprovidedwithsourcecodeandwhenwestartourPC thenthatsourcecodegetsexecuted.AndasoutputwegetLINUXOS]

==========*Therearetwoparadigms/approaches/aspectstodevelopaprogram.

-POP --->ProcedureOrientedProgramming

-OOP --->ObjectOrientedProgramming

*Basedonmemoryallocationtherearetwotypesofprogramminglanguages:

-Topdownapproach(formemoryallocation)

-Bottomupapproach(formemoryallocation)

==========

*TherearetwotypesofO.S.

-CommandbasedO.S.

-GUIbasedO.S.

*ThesecommandbasedO.S.startedinaround1965

*TheseGUIbasedO.S.startedinaround1991-93

==========

*Beforeinvensionof"C-language"everyoneusesUNIX.Andthattime,therewere:

-BCPL[BasicCombinedProgrammingLanguage][alsocalledas"Blanguage"]

-CPL[CombinedProgrammingLanguage]

-PASCAL

-COBOL[COmmonBusinessOrientedLanguage]

-SIMULA

-ALGOL[ALGOrithmicLanguage]

*TheIBMwasthecomputerhardwaremanufacturingcompany.

[IBM--->InternationalBusinessMachines]

*EverycomputerOWNERhastousespecific/compatibleversionofUNIXfortheircomputer model.

IBM123--->UNIXA

IBM3344--->UNIXB

IBM556--->UNIXC

*In1969,theDennisRitchiewasgivenatasktoinventaUNIXversionwhichshouldbecompatible oneverycomputermodel.

*Inintialdays,hestartedtocreateaUNIXversionbyusingBCPL/CPL.Butcouldnotsucceeded.

*Finally,in1970hedecidedtofirstprepareanew programminglanguage[withallrequired features/facilities]andthenfromthatnewlanguagewewillpreparecompatibleUNIXversion.

*Asresult,in1972hecompetedonenewprogramminglanguage.Andthislanguagewascalledas "NewB"language(orNBlangauge).

*Andin1973,hesuccessfullycompletedacompatibleUNIXversionfromthatNBlanguage.

*Asaresultofeveryefforts,thisUNIXversionbecameveryfamous.

*Totellthisconcepttoeveryotherprogrammer,hedecidedtowriteabookaboutthe conceptsofhisnewlanguage"NBlanguage".

*Finally,in1976hecompletedabook[withco-autor"BrianKarningham"].

*Whilecompletingthisbook,hegaveitanewname"C-language".

[becauseitisinventedafterB-language]

*After1979,the"BjarneStroustrup"ADDEDsomeOBJECTORIENTEDconceptsinC-languageandtheninventedanewlanguage"CWC" [CWithClasses].

*In1983,hewroteabookandrename"CWC"with"C++".

==========

======

*Compilerandinterpreterandtranslatorareultimatelysoftware/tools.

*Theyarealsowritteninspecificprogramminglanguage. [mainlyinAssemblyLanguage]

*TheAssemblyLanguageis"neartomachinelangauge"

==========

======*Everyinventorultimatelypreparesthelanguagetools.Like:

-compiler

-interpreter

-translator

-linker

-assembler

-loader

*Whyanewlanguagegetsinvented?

-Becausecurrentlangaugesisunabletosolvesomesoftware'sneed/situation/demand.So inventorsinventsnewlanguages.Suchdevelopmentofnewlanguages,takesmuchmoretime.

-Aspercurrentscenario,nooneinventsnewlanguage.Butinstead,everydeveloperprepares somedynamic/hybridframeworks/platforms

==========

1.Typesoflangauges(basedonplatform)

-platformdependent

-platformindependent

2.Typesoflanguages(basedoncasesensitivity)

-casesensitivelanguages

-caseinsensitivelanguages

3.Typesoflanguages(basedoncompilationandinterpretationprocesses)

-compiledlanguages

-interpretedlanguages

4.Typesoflangauges(basedonprogramskeleton/structure)

-structuredlanguages

-unstructuredlanguages

==========

5.Typesoflangauges(basedonsourcecodeavailability)

-opensourcelangauges

-notopensourcelanguages(closedsource)(properitorylanguages)

6.Typesoflanguages(basedonmemoryallocation)

-topdownapproachofmemoryallocation

-bottomupapproachofmemoryallocation

7.Typesofprogramminglanguages(basedonfeatures)

-highlevellanguages

-middlelevellanguages

-lowlevellanguages

8.Typesofprogramminglangages(basedonapproach)

-POP

-OOP

==========

=======1.Typesoflangauges(basedonplatform)

-platformdependent

-platformindependent

2.Typesoflanguages(basedoncasesensitivity)

-casesensitivelanguages

-caseinsensitivelanguages

3.Typesoflanguages(basedoncompilationandinterpretationprocesses)

-compiledlanguages

-interpretedlanguages

4.Typesoflangauges(basedonprogramskeleton/structure)

-structuredlanguages

-unstructuredlanguages

==========

5.Typesoflangauges(basedonsourcecodeavailability)

-opensourcelangauges

-notopensourcelanguages(closedsource)(properitorylanguages)

*Everyprogramminglanguagehasalibrary.

*Suchlibrarycontainssomebuilt-infunctions/modules.

*Anopensourcelanguageisthelanguageinwhichtheoriginalsource-codeofit'slibrariesis

available.

e.g.Java,Python,PERL

*Aclosedsource(properitory)langaugeisthelangaugeinwhichtheoriginalsource-codeofit's

librariesisnotavailable.

e.g.C,C++,VB,VB.net,C#

==========

*Everycommand/functionhassomebackgroundinstructions.

*Whenweusethatcommnad--->thenit'sbackgroundinstructionsruns.

*Thesebackgroundinstructionsarepresentinseparatefile;calledas"supprotingfile"or "headerfile" [.hwalifiles]

*Iftheoriginalfileisavailableinoriginalform ---->thatlanguageiscalledasopensource language.

*Iftheoriginalfileisnotavailableinoriginalform(maybetranslate/hidden)--->thatlangaugeis calledasaproperitorylangauge.

******Theoutputisnotatallpossiblewithoutincluding/attachingtheseinstructionfiles.

Theheaderfilesof"C"arenotoriginallyavailabletostudy/read.

*theyaretranslatedintoassemblylanguage.

==========

==========5.Typesofprogramminglangauges(basedonsouce-codeavailability)

-opensourcelanguages

-notopensourcelanguages(closedsourcelanguages)(properitorylanguages)

*Inopen-sourcelanguages,thesource-codeofeverylibraryisopenlyavailable.

e.g.--->Java,Python,PERL

*Inproperitorylanguages,thesource-codeofeverylibraryisnotopenlyavailable.

e.g.--->C,C++,VB,VB.net,C#

==========

Forexample,Ifwecheck'C'header-files,thenwecanobservethattheyarenotprovidedwith originalsourcecode.

Therefore,if you want to check/study the originalconcept/logic/code behind any command/functionthenitisnotpossibleinC/C++.

==========

Forexample,IfwecheckJavapakcages,thenwecanobservethattheyareoriginallyprovided inoriginalsourcecode.[astheyarewritten]

Therefore,ifyouwanttocheck/studytheoriginalconcept/logic/codeofanycommand/function

thenitpossible.Wecanopenthatfileinnotepadandvieworiginalcode.

==========

*InC,thereareatotal24header-files.

*Wehavetoinclude/attachrequiredheader-file,touseanyfunctionfromit.

==========

=====

*Nolanguageshares/providesthesource-codeofcompilerandinterpreterandanothertools.

==========

==========

----MemoryBlock----

==========

*Memoryblockistheareawhereruntimevaluesarestored.

*Amemoryblockhas4properties:

-name

-value

-datatype

-address

*Everyinputandcalculatedvaluegetsstoredinthosememoryblock.

*Theprogrammerhastothinkanddeclare/createrequirednumbersofmemoryblocks.

*Thesememoryblocksaretemporary.Whenpragramoutputends,thenthosememoryblocks

getsdeleted(automatically)

==========

6.Typesoflanguages(basedonmemoryallocation)

-topdownapproachofmemoryallocation

-bottomupapproachofmemoryallocation

==========

=====*Thesemi-colonindicatesendofstatement.

printf("Hello

kunal,how

are

youtoday

?

");

Itisindicationtocompilerandinterpreterthatmystatementendshere,youcancompilerand runuptothissemi-colon.

==========

----MemoryBlock----

==========

*Whateverweinputfromtheuser,willbestored/holdinamemory-block.

*Similarly,whateverwecalculate(result),willbestored/holdinamemoryblock.

*Theprogrammerhastologicallyguess(think)aboutrequirednumbersofmemoryblocks.And theprogrammerhastodeclare/createtherequirednumbersofmemoryblocks.

*Forexample:Iwanttowriteaprogram toaccept2numbersfrom theuserandprinttheir addition.Inthiscase,Ihavetodeclaretotal3memoryblocks.

*Forexample:IwanttowriteaprogramtoacceptageofKunalandcheckwhetherHeiseligible forDLornot.Inthiscase,Ihavetodeclare1memoryblockandstoreageofKunalinthat memory-block.

*****Whateverydataweinputorcalculate,mustbekept/storedinmemory-block.

*Inaprogram,weandeclare/createanynumbersofmemoryblocks.

==========

*Finally,memoryblockisthearea/blockwhereruntimevaluesarestored.

*Thememoryblockswillbecreatedduringruntime(notduringcompile-time).

==========

*Amemoryblockhas4attributes:

-name

-value

-datatype

-address

*Asaprogrammer,wemustprovide--->nameanddatatype.

*Whereas,thevalueisinputbytheuser(orcalculated)

*Andtheaddressisautomaticallyassigned/generated.

*Thememoryblocksandit'svaluesaretemporary.Whenoutputends,thememoryblocksgets destroyed.

***Asaprogramme,wecangivename,datatypeandvalue.Butaddressisalwaysautomatically generated.

***Therearetwonature/typesofamemoryblock:-variable

-constant

-Variablemeans,whosevaluecanbechangedduringruntime.(over-write)

-Constantmeans,whosevaluecannotbechangedduringruntime.

==========

=====*Thememory-blockisthearea/blockwhereruntimevaluesarestored.

*Whateverwetakeinputfromtheuserandwhateverwecalculate,everythinggetsstoredin memoryblock.

*Wecanstoreonlyonevalueinamemory-block.

*Therearetwotypes/naturesofamemory-block.

-variable

-constant

*Amemory-blockisvariable--->meanswecanchangeit'svalueduringruntime.

*Amemory-blockisconstant--->meanswecannotchangeit'svalueduringruntime.

*Tomakeaconstantmemoryblock---->usekeyword"const"

==========

*Keywordsarethewordswhosemeaningsarealredyknowntocompiler.

*InC-language,wehavetotal32keywords.

*Keywordscannotbeusedasidentifiersandthereforekeywordsarealsocalledas"reserved words".Aprogrammercannotusethemas"identifier".

*Languagecompilerknowsonlytwothings:

-keywords

-operators

*Andeverythingelsemustbedeclaredordefined.

==========

*Togiveanametomemory-block,thereare5rules:

-shouldbeofmax31characterandmin1characterlong

-specialsymbolsarenotallowed.Onlyunderscore(_)isallowed.

-spaceisnotallowed.

-muststartwithalphabetorunderscore(_).[mustnotstartwithdigit][internallyok]

-keywordsarenotallowedasitis.

==========

**Followingaresomeofthecommonlyexperiencedvalid+invalidvariable-names.

intfor; <---isnotvalid

intFor; <---isvalid

intFOR; <---isvalid

int_for; <---isvalid

intfor1; <---isvalid

int_for1; <---isvalid

int_for1234567890; <---isvalid

intFor_; <---isvalid

int_1for; <---isvalid

intmy_acc_num;

intmy_____ac___num;

int_____1____;

==========

==========

*Variableisanatureofmemory-blockwhosevaluecanbechanged/updatedduringruntime.

*Todeclarevariable,generalsyntaxis:

datatype variable_name;intnum;

floatavg;

chargen;

longac_num;

OR

datatype variable_name=value;

int num=5;

floatb=67.88;

inta=-877;

OR

datatype variable_names_list;

intn1,n2,n3;

floatavg,wt,pr;

intsroll,sage;

***Abovestatementsareorders/instructionstointerpretertocreatevariablememory-blocks.

***Wearedeclaringabovevariablesinourprogram,becausewewanttostorevaluesinthem.

***Theremustmustmustbespacebetweendatatypeandvariable'sname[whiledeclaring

memoryblocks]

==========**Memory-block:

---

*Amemory-blockistheareawhereruntimevaluesarestored.

*Andcalculatedvaluesarestored.

*Amemory-blockhas4properties/attributes:

-name(identifier)

-value

-datatype

-address

*Asaprogrammercangivename,datatypeandvalue.Whereas,theaddressisautomatically

assigned.

*Weknowthe5rulesofgivinganidentifier.

*Therearetwonaturesofamemory-block:

-variable

-constant

*Thegeneralsyntaxtodeclareavariable:

datatype vari_name;

OR

datatype vari_name=value;

OR

datatype variable_list;

*Todeclareconstant,simplyaddkeyword"const".

*Thekeywordsarethewordswhosemeaningsarealreadyknowntocompiler.

*Thecompilerknowsonlytwothings:

-keywords

-operators

*Itisprogrammer'sresponsibilitytodeclarerequirednumbersofmemory-blocks.[either variableorconstant]

==========

----Datatype----

==========

*Data/value ka type.

*"Thedatatypespecifieswhichtypeofvaluewearegoingtostoreinamemoryblock."

*Thememory-blockwillbepreparedasper"datatype".

*Inanyprogramminglanguage,thereareonlytwopossiblevaluesthatwecanstore.

-numericvalue[storenumbers]

-alphabeticvalue[storealphabetsandspecialsymbols]

*Thesenumericvaluesaredividedintotwotypes:

-numericvaluewithoutdecimalpoint(withoutfloatingpoint)

-numericvaluewithdecimalpoint(withfloatingpoint)

****Ifwecombineabovetwopoints,thentherearethreepossibilitiesofvalue:

1.Numericvaluewithoutdecimalpoint ----> INTEGER

2.Numericvaluewithdecimalpoint ----> REAL

3.Alphabeticvalues ----> CHARACTER****Eachofabovedatatype'scategorieshaskeywordsunderit.

INTEGER--->int,long,short

REAL--->float,double,longdouble

CHARACTER--->char

****Whiledecalringamemory-blockwewillusethesekeywords.Wewillnotusetheoriginal datatypecategories.

==========TherearetwoveryfamousCcompilers.

-BorlandC --->TurboC

-GCC --->Codeblocks

-MinGW --->Codeblocks

*EveryonepreferesTurboC++.

Becauseitincludesboth--->CandC++

*ProgrammingconceptsandoperatorsandkeywordswillbesameineveryC-compiler.

*The"int"and"short"bothdatatypesaregivenforsamepurpose.

*Whenwedolow-levelcircuitprogramming[kieluprogramming]thenyoushoulduse"short"

*Whenwedogeneralcomputerprogrammingthenyoushoulduse"int".

Inlow-levelcircuitprogramming:

short

long

Incomputerprogramming:

int

long

*Frominitialdaysofcomputerprogramming,theuseof"int"ispreferred.

*Stillifyouwant,youcanuse"short".

==========

*Theprogrammerhastothinkaboutexpectedvalueandaccordingtoithehastousethe

datatype.

Forexample:

Ifwewanttostoreageofastudent---->int

Ifwewanttostoreweightofastudent--->float

Ifwewanttostoreweightofaloadedtruck--->double (longdouble)

Ifwewanttostoregenderofastudent--->char

Ifwewanttostoretotalmarksofstudent--->int

IfwewanttostorepostalcodeofAuangabad--->long

Ifwewanttostorepetrolpriceof1Ltr--->float

==========*Thedatatypeistos
pecifythetypeofvalueofamemoryblock.

*Becauseofdatatype,theinterpreterwillget

-typeofvalue

-memoryallocation

-valuerange

*Itisprogrammer'sresponsibilitytothinkanddecidethedatatypeofmemory-blockwhile declaration.

==========

IMP

Ifwewritethestatement:

intnum=15;

Thiswillcreateavariablememoryblock

-withname'num'

-withdatatype'int'

-initialvaluewillbe15

==========

==========

Ifwewritethestatement:

constintnum=15;

Thiswilcreateaconstantmemoryblock

-withname'num'

-withdatatype'int'

-initialconstantvaluewillbe15

==========

Ifwewritethestatement:

intnum;

Thiswillcreateavariablememory-block

-withname'num'

-withdatatype'int'

-withgarbagevalue.

==========

*Ifwedeclareamemoryblockanddoesnotassignanyvaluetoit,thenthatmemoryblockwill notgenerateempty,butitwillhaveanauto-generatevalueinit... Thatauto-generatevalueis

calledas"Garbage Value".

*Whenwegiveinputorcalculatesomething,thenthatinputvalueorcalculatedvaluewillover□writethegarbagevalue.

*Thisgarbagevalueisauto-generatevalue.Thereisnoalgorithm/logicwhichcanguessthe

garbagevalue.

*Thisgarbagevaluecanbepositiveornegativeintegervalue.

==========

TherearemanycompilersforC/C++.Inallthesecompilersthelogicalthingsaretotallysame.[yoursyllabusissameinall

compilers].

***Inthesedifferentcompilers:

-datatyperanges

-datatype'smemoryallocation(bytes)

-operatorprecedence

BorlandCcompiler

GCCcompiler

MinGWcompiler

==========

==========*TherearemanyprogramminglangaugesthatdirectlysupportsC/C++.

*ThereasonisfeaturesofC/C++.

*IfwemergeCwithJava,thentheproblemistherangesofdatatype.

datatype C Java

---

int---> 32767 2147483647

short 32767 32767

long 2147483647 9223372036854775807

*TheC/C++aremostcompatiblelanguages.Manyoftheprogramminglanguages,framworks,

technologiesusesC/C++intheircenter/core.

*Buttheonlyproblemwas,theoriginaldatatypes(range+memory-allocations)andsomebasic

featuresarenotuptothemark.

*Thereforemanycompanieshavebuilttheirown C-compileranduseitintheirframeworksand

technologies.

*TheGCC compiler,MinGW compiler,KeilCompiler,IBM compiler,etcareexamplesof personalizedC-compilers.

==========

========memoryblock

name5rules

keywords

keywordsarereservedwords--->becausewecannotuseitasouridentifier

variable-constants(usekeyword"const")

garbagevalue[auto-generatedvalue]

datatype-->range,memoryallocation,formatspecifier

==========

---Operators----

==========

*Operatorsarespecialsymbolswhichhasoperationalmeanings.

*Compilerknowsoperators.

Forexample:

Inexpression:A+B

+isoperator

AandBareoperands

**Basedonoperands,thereare3typesofoperators:

-UnaryOperators

-BinaryOperators

-TernaryOperators

UnaryOperators:

aretheoperatorswhichrequiresonlyoneoperandtoworkwith.

BinaryOperators:

aretheoperatorswhichrequirestwooperandstoworkwith.

TernaryOperator:

aretheoperatorwhichrequiresthreeoperandstoworkwith.

***Chastotal8groups/categoriesofoperators.

(T)C CONDITIONALOPERATOR ?:

(U)IINCREMENTOPERATOR ++

(U)D DECREMENTOPERATOR --

(B)L LOGICALOPERATOR &&(LOGICALAND),||(LOGICALOR),!(LOGICALNOT)

(B)A ARITHMETICOPERATOR +,-,*,/,%(MODULUS/MOD/MODULO)

(B)R RELATIONALOPERATOR >,<,>=,<=,!=,==

(B)A ASSIGNMENTOPERATOR =

(B)B BITWISEOPERATOR <<(LEFTSHIFT),>>(RIGHTSHIFT),&(BITWISEAND),

|(BITWISEOR),^(EXCLUSIVEOR),~(BITWISENOT)

==========

==========1.ArithmeticOperators:

---

*Thearithemticoperatorsaretoperformarithmeticoperations.

operator description

+ toobtainaddition

- toobtainsubtraction

* toobtainmultiplication

/ toobtaindivision

% toobtainremainder

Forexample:

---

a=10%2;

a--->0

b=10/2;

b=5;

c=10%3;

c--->1

***Inshort,ifnumbersaredivisiblethenremainderis0.

==========

==========----Operators----

==========

*Operatorsarespecialsymbolsthathasoperationalmeaning.

*Compileralreadyknoweveryoperatorandit'spurpose.

*Thereare3typesofoperators:

-Unaryoperators

-Binaryoperators

-Ternaryoperators

*Thereare8groups/categoriesofoperators:

C CONDITIONALOPERATOR

I INCREMENTOPERATOR

D DECREMENTOPERATOR

L LOGICALOPERATOR

A ARITHMETICOPERATOR

R RELATIONALOPERATOR

A ASSIGNMENTOPERATOR

B BITWISEOPERATOR

==========

1.ArithmeticOperators:

---

*Thearithemticoperatorsaretoperformarithmeticoperations.

operator description

+ toobtainaddition

- toobtainsubtraction

* toobtainmultiplication

/ toobtaindivision

% toobtainremainder

Forexample:

---

a=10%2;

a--->0

b=10/2;

b--->5

c=15%2;

c--->1

d=11%3;

d--->2

e=1572%10;

e--->2

f=1572%100;

f--->72

g=1572%1000;

g--->572h=3%15;

h=3

i=1%10;

i--->1

***ModulusofREALvaluesisnotpossible.ItwillshowCTerror.

***i.e.bothoperandsmustbeINTEGER.

==========

==========

IMP

-InProgramming,thereareTwotypesofmemory.

-HEAPMEMORY

-BUFFERMEMORY

-TheHEAPMEMORYispermanentmemory.Itwillstayuptoprogram'sexecution.Everymemory blockdeclaredbytheprogrammer,goestoHEAPMEMORY.

-TheBUFFERMEMORYistemporarymemory.Itwillbeonlyuptoafewstatements.

==========

==========*ASCII-AmericanStandardCodeforInformationInterchange.

*ASCIIvaluesareuniquepre-definedvaluesofcharacter.

*ASCIIvaluesareuniqueserialnumbersofcharacter.

*ASCIIvaluesrepresentsanycharacterinmemory.

*Anycharactervalueisassignedandcomparedinsingleinvertedcomma.

*Ifweperformanyoperationoncharacters,thenitwillbeultimatelyonit'sASCIIvalue.

*AnycharactermemoryblockalsostoresASCIIvaluesofcharacter.

inta='A'+'B'; --->65+66--->131

intb=2+3;--->2+3--->5

intc='2'+'3'--->50+51--->101

'X'>'x'--->false

15>'X' --->false

'AB'>'CD'--->CTerror

'10'>'11'--->CTerror

==========

====

==========

====

characters ASCIIvalue

A-Z 65-90

a-z 97-122

0-9 48-57

specialsymbols 0-47,58-64,91-96,123-127

==========

=====

==========

=====

1.Arithmeticoperators:

+ toobtainaddition

- toobtainsubtaction

* toobtainmultiplication

/ toobtaindivision

% toobtainremainder[modulus/mod/modulo]

2.Incrementoperator(++):

---

*IncrementoperatorisaUNARYoperator.Itworksononlyoneoperand.

*Incrementoperatoristoincreasethevalueofanyvariableby1.

*Incrementoperatorworksoneverydatatype.

Forexample:

---

inta=12;

a++;<---willmakea=13*Wecanalsowriteitas:

a++;

++a;

a=a+1;

a+=1 <---short-handexpression

**Ifyouwanttoincrementby2,thenwecanuseincrementoperatortwice,indifferent statements.

a++;

a++;

OR

a=a+2;

OR

a+=2

OR

++a;

++a;

**Ifyouwanttoincrementby15,thenbettertousearithmeticexpressioninsteadofusing incrementoperator15times.

a=a+15;

OR

a+=15;

**Thestatementa+=10;--->isshort-handof---> a=a+10;

==========

==========

3.Decrementoperator(--):

---

*DecrementoperatorisaUNARYoperator.Itrequiresonlyoneoperandtoworkwith.

*Decrementoperatoristodecreasethevalueofanyvariableby1.

*Decrementoperatorworksonanydatatype.

Forexample:

---

inta=12;

a--; <----willmakea=11

*Wecanalsowriteitas:

a--;

--a;

a=a-1;

a-=1;

*Ifwewanttodecreaseby2,thenusedecrementoperatortwiceintwodifferentstatements.

Orwecanusearithmeticexpressionorshort-handexpression.

a--;

a--;

OR

a=a-2;OR

a-=2;

OR

--a;

--a;

*Ifwewanttodecreasevalueofanyvaraibleby15,thenbettertousearithmeticexpresionor

short-handexpression.

a=a-15;

a-=15;

==========

==========

IMP

-TheUNARYoperatorsworksonlyonvariables.

-Willnotworkonconstantsandconsolevalues.

15++; <---isCTerror

4.5++; <---isCTerror

'A'++ <---isCTerror

==========

==========

==========

==========***IMP***

==========

*Incrementoperator(++)andDecrementoperator(--)areUNARYoperators.

*Wehave2optionsinincrementoperator:

a++--->POSTINCREMENT

++a--->PREINCREMENT

*Wehave2optionsindecrementoperator:

a-- --->POSTDECREMENT

--a --->PREDECREMENT

*Whensuchpreand postoperatorsareused inexpression,thenthey havedifferent

role/explanation.

POSTINCREMENT--->firstuse/assignasitis,thenincreasethevalueby1.

PREINCREMENT--->firstincreasethevalueby1,thenuse/assignnewincrementedvalue.

POSTDECREMENT--->firstuse/assignasitis,thendecreasethevalueby1.

PREDECREMENT--->firstdecreasethevalueby1,thenuse/assignnewdecrementedvalue.

==========

=====

inta,b,c;

a=7;

b=3;

c=(a++)+(++b);

c--->11

a--->8

b--->4

==========

inta,b,c,d,e;

a=6;

b=2;

c=4;

d=1;

e=(--a)+(b--)+(c++)+(--d);

e--->11

a--->5

b--->1

c--->5

d--->0

==========

voidmain()

{

intx=7,y,z,p,q;

y=x++;

z=++x;<---considerupdatedvalue(x=8)

p=x--;<---considerupdatedvalue(x=9)

q=--x;}

x--->

y--->

z--->

p--->

q--->

==========*Whatwillbetheoutputoffollowingprograms?

```
voidmain()
{
intx=9,y,z,p,q;
y=++x;
z=x++;
p=x--;
q=--x;
}
x--->9
y--->10
z--->10
p--->11
q--->9
```

==========

*Whatwillbetheoutputoffollowingprogram?

```
voidmain()
{
intx=9,y,z,p,q;
y=x++;
z=--x;
x++;
++x;
p=y--;
q=++z;
}
x--->11
y--->8
z--->10
p--->9
q--->10
```

==========

**Whatwillbetheoutputoffollowingprogram?

```
voidmain()
{
intx=7,y,z,p,q,r,s;
y=x++;
z=(y++)+(--x);
y++;
z++;
p=++y;
q=p++;r=(x++)-(--y);
s=r++;
}
```

x--->8

y--->9

z--->15

p--->11

q--->10

r--->-1

s--->-2

==========

```
voidmain()
{
intx=10,y,z,p,q,r;
y=x++;
z=--x;
p=(x++)-(y--)-(++z);
q=(--p)*2;
r=q++;
s=(--r)*(x++);
}
```

==========1.ArithmeticOperators:

+ toobtainaddition

- toobtainsubtraction

* toobtainmultiplication

/ toobtaindivision

% toobtainremainder

2.Incrementoperator(++):

istoincreasethevalueofanyvariableby1.

a++(Postincrement)

++a(Preincrement)

3.Decrementoperator(--):

istodecreasethevalueofanyvaraibleby1.

a--(Postdecrement)

--a(Predecrement)

==========

==

4.Assignmentoperator(=):

---

*Assignmentoperatorisabinaryoperator[requires2operands]

*Assignmentoperator(=)istoassignthevalue/resultofexpressiontovariable/constant.

*TheassignmentoperatorhasLEASTpriorityamongallotheroperators.

***Rememberthat,whileusingassignmentoperator:

-theleftoperandmustbeavariable/constant

-therightoperandmustbeanexpressionorconsole-valueorvariable

inta=10;<---isvalid

intb=a; <---isvalid

constinta=6;<---isvalid

constintb=al <---isvalid

10=a<---iserror

a+b=c+d<---iserror

a+b=c<---iserror

a+b=10<---iserror

==========

==

==========

==

5.Bitwiseoperators:

---

*Bitwiseoperatosarebinaryoperators.[requires2operands]

*TheBitwiseoperatorsworkson"BinaryBitsofoperands"

a.LeftShiftoperator(<<):

==========

ThisoperatorLEFTshiftsthebinarybitsofLeftoperandaspervalueofright-operand.x=85<<2;

x--->340

y=135<<3;

y--->1080

z=280<<3;

z--->2240

b.RightShiftoperator(>>):

==========

ThisoperatorRIGHTshiftsthebinarybitsofLeftoperandaspervalueofright-operand.

x=85>>2;

x--->21

y=135>>3;

y--->16

z=280>>3;

z--->35

==========

=====

Homework:

---

voidmain()

```
{
inta=10,b=15,c=154,d=60,e,f,g,h;
e=(a<<3)+(b>>2);
f=++e;
g=(f<<3)*(b++);
h=(g*++f);
}
```

==========SolvedHomework:

```
voidmain()
{
inta=10,b=15,c=154,d=60,e,f,g,h;
e=(a<<3)+(b>>2);
f=++e;
g=(f<<3)*(b++);
h=(g*++f);
}
```

Answer:

a=10

b=16

c=154

d=60

e=84

f=85

g=10080

h=856800

==========

1.Arithmeticoperators

2.Incrementoperator(++)

3.Decrementoperator(--)

4.Assignmentoperator(=)

5.Bitwiseoperators

==========

a.LeftShiftoperator(<<)

---

b.RightShiftoperator(>>)

---

c.BitwiseANDoperator(&)

Thisoperatorcomparesbinarybitsofbothoperandsandresultsbit'1'ifbothbitoperandsare'1';

otherwiseresultsbit'0'.

TruthTableofBitwiseAND(&)

==========

Left-bit Right-bitResult-bit

---

1 1 1

1 0 0

0 0 0

0 1 0

Forexample:

---

x=45&122;

x--->40y=200&17;

y--->0

z=250&250;

z--->250

d.BitwiseORoperator(|)

Thisoperatorcomparesbinarybitsofbothoperandsandresultsbit'1'ifanyoneoperandbitis'1';

otherwiseresultsbit'0'.

TruthTableofBitwiseOR(|)

==========

Left-bit Right-bitResult-bit

---

1 1 1

1 0 1

0 0 0

0 1 1

Forexample:

---

x=45|122;

x--->127

y=200|17;

y--->217

z=250|250;

z--->250

==========

==========----Bitwiseoperators----

==========

a.Leftshiftoperator(<<)

b.Rightshiftoperator(>>)

c.BitwiseANDoperator(&)

d.BitwiseORoperator(|)

e.BitwiseXORoperator(exclusiveOR)(^)

---

*Thisoperatorworksonbinarybitsofbothoperands.

*TheBitwiseXORoperatorresultsbit'1'ifbothoperandbitsareopposite;otherwiseresultsbit

'0'.

TruthTableofBitwiseXOR(^)

==========

Left-bit Right-bitResult-bit

---

1 1 0

1 0 1

0 1 1

0 0 0

Forexample:

---

x=12^25;

x--->21

y=45^122;

y--->87

z=120^220;

z--->164

==========

=======

f.Bitwisecomplementoperator(~):

**TheBitwiseComplementoperatorisaUNARYoperator.[worksononeoperand]

**Itdoesnotrequiretwooperands.

*Thisoperatorperformsinverse/reverseofbinarybit.

operatorbit resultbit

---

1 0

0 1

Forexample:

---

x=~85;

x--->42

==========

=

***The"sizeof"iskeywordoperator.Itreturnsthesize(inbytes)ofspecifiedvariable/value.

Forexample:

---

inta;

floatb;

charc;

sizeof(a)--->2

sizeof(b)--->4

sizeof(c)--->1

*Wearealreadyfamiliarwithdatatypesandtheirvalue-rangeandmemory-allocationand

format-specifier.

==========

========

*Whatwillbetheresultoffollowingprogram?

voidmain()

{

inta=5,b=6,c=7,d=8;

inte,f,g,h,i;

e=++a;

f=(a<<3)+(b++)+(c<<2);

g=(++f)+2;

h=f&a;

i=f|a;

}

a--->

b--->

c--->

d--->

e--->

f--->

g--->

h--->

i--->

==========

*Whatwillbetheresultoffollowingprogram?

voidmain()

{

inta=-10,b,c,d,e;

b=a++;

c=a--;

d=--a;

f=++a;

}a --->

b--->

c--->

d--->

f--->

==========

*Whatwillbetheresultoffollowingprogram?

voidmain()

{

inta,b,c,d,e;

a=(155<<2)+(85|25);

b=(++a)<<2;

c=a^b;

d=(--a)+(--b)+(--c);

}

a--->

b--->

c--->

d--->

==========----InputOutputstatements----

==========

*Everyprogramminglanguagehasinput-outputstatements.

*Theinput-outputstatementsaretocommunicatewithend-user.

*Iftheend-userwantstoprovide/enteranyvalue--->useinputstatement

*Iftheprogramwantstoshowanymessage/result/decision--->useoutputstatement

*Cusesfunctiontoperforminput-outputoperations.

==========

=======

outputstatement

---

*Anoutputstatementistodisplaymessages/results/decisions/resulting-values/greetingsto

theend-user.

*Chasmanyfunctionasoutputstatement.Forexample:

printf()--->toprintinconsolescreen

puts()--->toprintstring

fprintf()--->toprintintofile

vfprintf()--->lowlevelprinting

putchar()--->singlecharacterprint

fputc()--->towritesinglecharacterinfile

*Allabovefunctionsareusedforoutput.Butallofthemhasdifferent/specialapplications.

*Buttheprintf()isthemostcommonlyusedinconsoleprinting.

==========

=========

==========

=========

intputstatement

---

*Aninputstatementistotakeinput/values/datafrom theend-
userandfillitintomemory blocks(variables&constants).

*Chasmanyfunctionsasinputstatements.Forexample:

scanf()--->totakeinputfromconsole

gets()--->toinputstring

fscanf() --->toreadfromfile

vfscanf()--->toreadfromlowlevel

getch()--->toreadsinglecharacter[echoisOFF]

getche()--->toreadsinglecharacter[echoisON]

fgetc()--->toreadsinglecharacterfromfile

*Allabovefunctionsareusedforinput.Butallofthemhasdifferent/specialapplications.

*Butthescanf()isthemostcommonlyusedfunctiontotakeinputfromconsole.

==========

==========

=========**Anywordfollowedbyparenthesis()isafunction.

----OutputStatement----

==========

*Theoutputstatementistodisplay/showoutputmessages/resultingvalues/decisions.

*Weknowthat,wehavemanyfunctiontobeusedasoutputstatement.

*Theprintf()isthemostcommonlyusedoutputfunction.

*Byusingprintf()wecanshowoutputmessages/resultingvalues/decsionsinconsolescreen.

*Touseprintf()generalsyntaxis:

printf("MessageHere");

*Themessagewithin""willbedisplayedasitisinconsoleoutput.

Forexample:

Part-I--->printingnormalmessages

---

printf("Helloall");---->Helloall

printf("Whatisyourname?");--->Whatisyourname?

printf("Therewasaking.Hewasverybrave.HewenttoJungle.");

--->Therewasaking.Hewasverybrave.HewenttoJungle.

printf("Mynameis"Hatim""); <---isCTERROR

printf("c=a+b"); --->c=a+b

Thiswillnotperformadditionoperation.

Becausewewriteitinsideprintf()

Itwillbedisplayedasmessage.

printf("Hello(all,how?abc@xyz{36253&$2@262'sdga23[dsffs");

--->Hello(all,how?abc@xyz{36253&$2@262'sdga23[dsffs

Part-II--->printingvaluesofvariablesandconstants

---

youperformedaddition

inta,b,c;

c=a+b; --->additionisstoredinvariable'c'

step1:preparemessagewithin

step2:useproperformat-specifierinplaceofvalue

AdditionisX

printf("Additionis%d",c);

Xisaddition

printf("%disaddition",c);

resultingvalueisX

printf("resultingvalueis%d",c);

addition=Xprintf("addition=%d",c);

TheadditionofinputvaluesisX

printf("additionofinputvaluesis%d",c);

theadditionisXofinputvalues

printf("theadditionis%dofinputvalues",c);

X

printf("%d",c);

Part-III--->printingmultiplevalues

Toprintmulitplevalues,simplyreplacevalueswithformat-specifiers.

inta=10,b=20;

floatavg=56.77;

chargen='M';

valueofaisXandbisX

printf("valueofais%dandbis%d",a,b);

a=Xandb=X

printf("a=%dandb=%d",a,b);

valueofaisXandaverageisX

printf("valueofais%dandaverageis%f",a,avg);

Student'saverageisXandGenderisX

printf("Student'saverageis%fandGenderis%c",avg,gen);

Part-IV--->printingdoublequotes

----

*Toprintdoublequotesinmessage,usesingle\beforedoublequote.

Forexample:

Mynameis"Hatim"fromGPA.

printf("Mynameis\"Hatim\"fromGPA.");

Thiswillnotprint\butitwillprint"

This\iscalledas"EscapeSequence".

==========

===

*InC++/Java/Pythonwehavetodividethemessageandvariables.

*InC,wewriteitcontinue.

==========

===----inputstatement----

==========

*Theinputstatementistoreceive/acceptinputfromtheend-userandfillitintovariable.

*Weknowthat,therearemanyinputrelatedfunctions.Butweusescanf()forourconsoleinput

purposes.

*Thegeneralsyntaxofusingscanf()is:

scanf("format_specifier",&variable_name);

&--->Ampersand

Weknowthat,theformat-specifierdependsonvariable'sdatatype.

Forexample:

Part-I:--->takingsingleinput

---

intnum;

scanf("%d",&num); <---takesinputfor'num'

floatavg;

scanf("%f",&avg); <---takesinputfor'avg'

chargen;

scanf("%c",&gen); <---takesinputfor'gen'

Part-II--->takingmultipleinputs

---

intage,roll;

scanf("%d%d",&age,&roll);

//receivesfirstinputfor'age'andsecondinputfor'roll'

floatavg,wt;

scanf("%f%f",&avg,&wt);

//receivesfirstinputfor'avg'andsecondinputfor'wt'

intage;

floatavg;

scanf("%d%f",&age,&avg);

//receivesfirstinputfor'age'andsecondinputfor'avg'

intage;

floatavg;

scanf("%f%d",&avg,&age);

//receivesfirstinputfor'avg'andsecondinputfor'age'

ints1,s2,s3,s4,s5;

scanf("%d%d%d%d%d",&s1,&s2,&s3,&s4,&s5);

//receives5'int'typeinputs==========

==========

======

***Itisnotnecessarytotakeinputforalldeclaredvariables.

==========

======

Forexample:

---

intn1,n2,ad,su,mu;

floatdv;

scanf("%d%d",&n1,&n2); <-----STEP1:acceptvaluestoperformoperation

ad=n1+n2; <-----STEP2:calculateandfillrelatedvariableblocks

su=n1-n2;

mu=n1*n2;

dv=n1/n2;

printf("Additionis%d",ad); <-----STEP3:showingcalculatedresult

printf("Subtractionis%d",su);

printf("Multiplicationis%d",mu);

printf("Divisionis%f",dv);

==========

========

intn1,n2,ad,su,mu;

floatdv;

ad=n1+n2;

su=n1-n2;

mu=n1*n2;

dv=n1/n2;

scanf("%d%d",&n1,&n2);

printf("Additionis%d",ad);

printf("Subtractionis%d",su);

printf("Multiplicationis%d",mu);

printf("Divisionis%f",dv);

==========

==========*Intechnicalprogramming,awordiscalledas"token".

*Anytokenfollowedbyparenthesis()isafunction.

*Theinput-outputstatementsaretocommunicatebetweenprogram&end-user.

*Theinput-statementistoreceiveinput/valuefrom theend-userandfillitintovariable.[we cannottakeinputforconstant]

*Theoutput-statementistodisplaymessages/instructions/resultingvalues/decisionstoend□user.

Generalsyntax:

printf()--->printf("Messagehere");

input()---->scanf("format_specifier",&variable_name);

==========

Ageneralprogramhas3steps:

1.Acceptvalues

2.Performcalculations[useexpressionsandoperators]

3.Displayresults

==========

***Itisalwayspreferredtogiverelatednametomemoryblocks.

==========

**Whatyouwriteiswhatyouget..

==========

IMP

1.Theprintf()statementprintsthemessagesinsameline.Ifwewanttobreakthelineinoutput, thenuse"\n".

Example-I:

---

printf("Hello1");

printf("Hello2");

Outputwillbe:

Hello1Hello2

Example-II:

---

printf("Hello1\n");

printf("Hello2");

Outputwillbe:

Hello1

Hello2

Example-III:----------

printf("Hello1\nHello2");

Outputwillbe:

Hello1

Hello2

Example-IV:

----

printf("Hello1Hello2");

Outputwillbe:

Hello1Hello2

Example-V:

----

printf("Hello 1 .....He llo 2");

Theoutputwillbe:

Hello 1 .....He llo 2

==========

==========

FormattingCharacters:[useinprintf()statement]

\n--->fornewline

\t--->fortab

\b--->forbackspace

\a--->forbeep[currentlynoeffect]

==========

==========*Weknowthat,Cisa"structuredprogramminglanguage".

*Therefore,inCwemustwriteeverystatementinsidefunction'sbody.[Openstatementsare

notallowed.]

*InC,weareprovidedwithafixnamedfunction.Itis"main".Becausetheinterpreterisapre□definedtool.An dinterpreterispreparedtocall/run/execute"main"only.

*Inotherwords,theinterpreterisdesignedtocall/run/executethefunctionwhichisnamedas

"main".Thereforeprogramexecutionstartsfrommainonly.

#include<...>

#include<...>

voidmain()

{

writestatementshere

.....

.....

}

==========

***Aprogramminglanguageisfullydependentonnames/identifiers.

==========

**Aprogramwithoutmain()methodispossible.Suchprogramwillgetcompiled.Noerrorifwe

don'twritemain()method.

**Aprogramwithoutmain()methodcannotexeucte/run.

==========

***Everyfunctionmustendwithparenthesis.Itisageneralruleineveryprogramminglanguage.

==========

IMP

Ageneralskeleton/structureofprogram:

---

includerequiredheaderfiles

voidmain()

{

declarevariablesandconstants;

receiveinput(s)

performrequiredoperation(s)oninputs[canuseexpressionsandoperators]

displayresultingvalues/messages

}

==========

Example-1:

Writeaprogramtoaccept2integernumbersandperformtheiraddition.

```
#include<stdio.h>
voidmain()
{
intn1,n2,ad; <----declaring3varabies
scanf("%d%d",&n1,&n2); <----takinginputfor'n1'and'n2'
ad=n1+n2; <----performingadditionandassigningitinto'ad'
printf("Additionis:%d",ad);<----todisplayresultingvaluewithmessage
}
```

==========

****IMP****

-Theuseof---->intmain()

voidmain()

dependsoncompiler.[GCC,MinGW,Borland,ANSIC,etc....]

==========

*Weknowthat,theinterpreterispre-definedtocall/runtomain()method.Thereforeprogram executionstartsfrommain()only.

*Thereforewealwaysdefinemain()methodandwritelogicalstatementsinit.

==========

*Weknowthat,ifourprogramcompilessuccessfullywith0errors,thenthecompilercreatesa machine-languagefile(.objextension).

*Thismachine-languagefileexecutes.[interpeterwillusethisM/Clanguagefile]

==========

========

Program-1:

```
#include<stdio.h> <---toincluderequiredlibraryfunction'sbody
voidmain()
{
intn1,n2,ad; <----tocreate3'int'typevariablememory-blocks
printf("Enter2values\n"); <----asinstructiontoend-user
scanf("%d%d",&n1,&n2); <----toreceiveinputsandfillinto'n1'and'n2'
ad=n1+n2; <----tocalculateadditionof'n1'&'n2'andassigninginto'ad'
printf("Additionis%d",ad);<---todisplayresultingmessage
}
```

==========

Program-2:

Writeaprogramtoacceptoneintegervaluefromtheuserandprintit'ssquareandcube.

```
#include<stdio.h>
voidmain()
{
intnum,sq,cu;
printf("Enteroneintergervalue\n");
scanf("%d",&num);
sq=num*num;
cu=num*num*num;
printf("Squareis%d",sq);
printf("Cubeis%d",cu);
}
```

==========Homework

Program-1:

Writeaprogramtoaccept2integervaluesandperformallarithmeticoperationsonthem.

Program-2:

Writeaprogramtoprintyournameandyourcity.Bothindifferentlines.

[noinput&variablesneeded][useonlyprintf()]

==========

=========stdio.h --->printf()

scanf()

gets()

puts()

fflush()

fgetc()

fputc()

fprintf()

fscanf()

conio.h---> clrscr()**Noteverycompilersupportsconio.h

getch()

getche()

math.h---> sqrt()

pow()

sin()

cos()

log()

lon10()

==========

========

voidmain()

{

intx,y,z;

printf("Enter2values\n");

scanf("%d%d",&x,&y);

```
z=x+y;
printf("\nAdditionis:%d",z);
z=x-y;
printf("\nSubtractionis:%d",z);
z=x*y;
printf("\nMultiplicationis:%d",z);
z=x/y;
printf("\nDivisionis:%d",z);
}
==============================
voidmain()
{
printf("ShreyanshSoni\n");
printf("\nAurangabad");
}
================================================================
```

Program-1:

----------Writeaprogramtoacceptmarksof5subjectsofastudent.Calculateandprinttotal-marksand averagemarks.

```
#include<stdio.h>
voidmain()
{
inta,b,c,d,e;
inttot;
floatav;
printf("Entermarksof5subjects\n");
scanf("%d%d%d%d%d",&a,&b,&c,&d,&e);
tot=a+b+c+d+e;
av=tot/5.0;
printf("\nTotalmarks=%d",tot);
printf("\nAveragemarks=%f",av);
```

}

==========

Homework:

Program-1:

Writeaprogramtoaccepttotalbillamountfromtheuser.Give15%discountontotalbillamount.

Printoriginalbillamountanddiscountedbillamount.

Forexample:

totalbillamount---->1000

15%oftotalbill--->150

Aftergivingdiscount--->1000-150--->850

Program-2:

Writeaprogramtoprintyourname,cityandemailaddress.Allinnewlines.

==========

==========SolvedHomework:

Program-1:

Writeaprogramtoaccepttotalbillamountfromtheuser.Give15%discountontotalbillamount.

Printoriginalbillamountanddiscountedbillamount.

```
#include<stdio.h>
voidmain()
{
intamt,new_amt;
floatdis;
printf("Entertotalbillamount\n");
scanf("%d",&amt);
dis=(amt/100)*15;
new_amt=amt-dis;
```

printf("Originalbillamount:%d",amt);

printf("Afterdiscount,newbillamount:%d",new_amt);

}

==========

Program-2:

Writeaprogramtoprintyourname,cityandemailaddress.Allinnewlines.

#

#

voidmain()

{

printf("Pranali\n");

printf("Aurangabad\n");

printf("pranali@gmail.com\n");

}

OR

voidmain()

{

printf("Pranali\nAurangabad\npranali@gmail.com\n");

}

==========

==========

*Weknowthat,toprintdoubleinvertedcommainsideprintf'smessage,weusesingle\before doubleinvertedcommatobeprinted.Thisuseofsingle\willletitconsiderasnormalspecial symbolwithinprintf'smessage.

#include<stdio.h>

voidmain(){

printf("Mynameis\"Patil\"andIamfromGPA");

}

==========

*Nowifwewanttoprintthe%signinsideprintf'smessage,thenwehavetousesingle%before

your%signtobeprinted.Thisuseofsingle%willletitconsiderasnormalspecialsymbolwithin printf'smessage.

```
#include<stdio.h>
voidmain()
{
printf("After15%%discount");
printf("\n\n\n\n");
}
```

Thefirst%ismakingthesecond%asspecialsymbol.Notasformatspecifier.

==========

*Similarly,ifwewanttoprintaformat-specifierwithinprintf'smessage,thenalsowe canuse %beforeanyformatspecifier.

```
#include<stdio.h>
voidmain()
{
printf("%%d");
printf("\n\n\n\n");
}
```

Thefirst%ismakingthesecond%asspecialsymbol.Notasformatspecifier.

==========

*Similarly,ifwewanttoprintanyformattingcharacters(like\n,\t,\a,\b,etc)thenwemust usesingle\beforeanyformattingcharacter.

```
#include<stdio.h>
voidmain()
{
printf("Iwanttoprint\\ninprintf");
printf("\n\n\n\n");
}
```

==========

****Aboveadditionaleffortsareneededonlyforthosecharacterwhichhasspecialmeaning insideprintf'smessage.Allotherspecialsymbols(whichdon'thaveanymeaninginsideprintf's

message)willgetprintedeasily.Noadditionaleffortsarerequiredforthem.

```
#include<stdio.h>
voidmain(){
printf("Amey-+*(fdafaf)[sdfs#%^&**<>dfdf]asdf{sdfas}Akash");
printf("\n\n\n\n");
}
```

==========

#

#

```
voidmain()
{ <---function'sbodyopens
intx,y,z;
printf("Enteranumber:");
scanf("%d",&x);
y=x*x;
z=x*x*x;
printf("Squareis%d", y);
printf("Cubeis%d",z);
} <----function'sbodyends
printf("valueofxis%d",x); <---error.Can'twriteanythingoutsidebody
```

==========

Homework:

1.Writeaprogramtoacceptoneintegernumberfromtheuserandprintit'slastdigit.

Forexample:

input--->3467

output--->lastdigitis7

input--->1235

output--->lastdigitis5

input--->12

output--->lastdigitis2

==========

========SolvedHomework:

Program-1:

Writeaprogramtoacceptoneintegernumberfromtheuserandprintit'slastdigit.

[logic--->anynumber%10resultslastdigit]

```
#include<stdio.h>
voidmain()
{
intnum,r;
printf("Enteranumber\n");
scanf("%d",&num);
r=num%10;
printf("Lastdigitis:%d", r);
}
```

==========

Program-2:

Writeaprogramtoaccepttwonumbersfromtheuserandprintthesumoftheirlastdigits.

Forexample:

num1--->3642

num2--->3579

output--->Thesumoflastdigitsis11

```
#include<stdio.h>
voidmain()
{
intnum1,num2;
intr1,r2,ad;
```

```
printf("Enterfirstnumber\n");
scanf("%d",&num1);
printf("Entersecondnumber\n");
scanf("%d",&num2);
r1=num1%10;
r2=num2%10;
ad=r1+r2;
printf("Sumoflastdigitsis%d",ad);
}
```

----OR----

```
#include<stdio.h>
voidmain()
{intnum1,num2,ad;
printf("Enterfirstnumber\n");
scanf("%d",&num1);
printf("Entersecondnumber\n");
scanf("%d",&num2);
ad=(num1%10)+(num2%10);
printf("Sumoflastdigitsis%d",ad);
}
```

==========

Program-3:

Writeaprogram toacceptradiusofcircleandprintareaandcircumferenceofthatcircle.

[preferfloatvariables]

```
area=3.14*radius*radius
circumference=2*3.14*radius
#include<stdio.h>
voidmain()
{
```

```
floatrad;
floatar,cir;
printf("Enterradiusofcircle\n");
scanf("%f",&rad);
ar=3.14*rad*rad;
cir=2*3.14*rad;
OR
ar=(22/7.0)*rad*rad;
cir=(22/7.0)*2*rad;
printf("\nAreais%f",ar);
printf("\nCircumferenceis%f",cir);
}
```

==========

***Itisneverrecommandedtoprepare'C'constantforanyglobal/scientificconstant.

***Itisvalid,ifweprepareavariable/constantforit.

```
#include<stdio.h>
voidmain()
{
floatrad;
floatar,cir;
floatPI=3.14; OR floatPI=22/7.0; OR constfloatPI=22/7.0;
printf("Enterradiusofcircle\n");scanf("%f",&rad);
ar=PI*rad*rad;
cir=PI*2*rad;
printf("\nAreais%f",ar);
printf("\nCircumferenceis%f",cir);
}
```

==========

Program-5:

Writeaprogramtoacceptnumbersofminutesfromtheuserandprintit'sequivalentseconds.

Forexample:

input-->4

ouput-->Equivalentsecondsare240

```
#include<stdio.h>
voidmain()
{
intmin;
intsec;
printf("Enterminutes:");
scanf("%d",&min);
sec=min*60;
printf("Equivalentsecondsare%d",sec);
}
```

==========

***Variablenamedependsonprogrammer.Prefertogiverelatednames.

***Anymathematicalconstants,scientificconstants,universalconstantsarenotpre-builtinany programminglanguage.

***Anymathematicalformuleoranyotheruniversalformulearenotpre-built.

***Youhavetowriteeveryformulaasstatement.

==========

==========

Homework

1.Writeaprogram toacceptbasicsalaryofanemployeeandgive20%bonus.Printupdated salaryaftergivingbonus.

CHALLANGE:

2.Writeaprogramtoaccepta3digitnumberandprintthesumoflast2digits.

Forexample:

input--->731

output--->sumoflasttwodigitsis4input--->710

output--->sumoflasttwodigitsis1

==========

==========SolvedHomework:

1.Writeaprogram toacceptbasicsalaryofanemployeeandgive20%bonus.Printupdated salaryaftergivingbonus.

```
#include<stdio.h>
voidmain()
{
longbs,bon,new_sal;
printf("Enterbasicsalaryofemployee-");
scanf("%ld",&bs);
bon=(bs/100)*20;
new_sal=bs+bon;
printf("Newupdatedsalaryafter20%%bonusis:%ld",new_sal);
printf("\n\n\n\n");
}
```

==========

2.Writeaprogramtoaccepta3digitnumberandprintthesumoflast2digits.

```
#include<stdio.h>
voidmain()
{
intnum,a,b,ad;
printf("Entera3digitnumber\n");
scanf("%d",&num);
a=num%10;
num=num/10;
b=num%10;
ad=a+b;
printf("Sumoflasttwodigitsis:%d",ad);
}
```

***Ifwedivideanyintegernumberby10thenitwillautomaticallydeleteit'slastdigit.

==========

*Whatwillbetheoutputoffollowingprogram?

#

#

voidmain()

{

intnum=1942;inta,b,c,d;

a=num%10;

num=num/10;

b=num%10;

c=num/10;

d=a+b+c;

printf("Sumis:%d", d);

}

Outputwillbe:

---

sumis:25

==========

Program-3:

Writeaprogramtoaccepta4digitnumberandprintsumoflast3digits.

#

#

voidmain()

{

intnum,a,b,c,ad;

printf("Entera4digitnumber:");

scanf("%d",&num); <----num=1942

a=num%10; <---a=2

num=num/10; <---num=194

b=num%10; <---b=4

num=num/10; <---num=19

c=num%10; <---9

ad=a+b+c; <---ad=2+4+9=15

printf("Sumoflast3digitsis:%d",ad);

}

==========

Program-4:

Writeaprogramtoaccepta3digitnumberandprintthesumofit'sfirstandlastdigit.

#

#

voidmain()

{

intnumprintf("Entera3digitnumber\n");

scanf("%d",&num);<---num=195

a=num%10;<---a=5

b=num/100; <---b=1

ad=a+b; <---ad=5+1=6

printf("Sumoffirstandlastdigitis:%d",ad);

}

----OR----

voidmain()

{

intnum,a,b,ad;

printf("Entera3digitnumber\n");

scanf("%d",&num);

a=num%10;

num=num/10;

b=num%10;

num=num/10;

ad=a+num;

printf("Sumoffirstandlastdigitis:%d",ad);

}

==========

=======

Program-5:

Writeaprogramtoaccepta4digitnumberandprintsumofallit'sdigits.

#

#

voidmain()

{

intnum

printf("Entera4digitnumber:");

scanf("%d",&num); <---num=5327

a=num%10; <---a=7

num=num/10; <---num=532

b=num%10; <---b=2

num=num/10; <---num=53

c=num%10; <---c=3

num=num/10; <----num=5

ad=a+b+c+num; <---ad=7+2+3+5=17printf("Sumofalldigitsis:%d",ad);

}

==========

=======

Homework:

Program-1:

Writeaprogramtoacceptbasicsalaryofanemployeeandcalculategross-salary(finalsalary)

afteradding10%TA,15%DAand20%HRAinbasicsalary.

==========

========SolvedHomework:

1.Writeaprogramtoaccepta5digitnumberfromtheuserandprintthesumoffirst,thirdand fifthdigit.

```
#include<stdio.h>
#include<conio.h>
voidmain()
{
longnum;
inta,b,c,ad;
printf("Entera5digitnumber:");
scanf("%ld",&num);
a=num%10; <---willretrievelastdigit
b=num/10000; <---willretrievefirstdigit
c=(num/100)%10; <---willretrievethirddigit
OR
num=num/100;
c=num%10;
ad=a+b+c;
printf("Additionis:%d",ad);
}
```

==========

2.Writeaprogramtoaccepta4digitnumberfrom the userandprintit'sreverse.Strictly, takeinputsinglenumberandprintalsosinglenumber.

```
#include<stdio.h>
#include<conio.h>
voidmain()
{
intnum,a,b,c,sum;
printf("Entera4digitnumber:");
scanf("%d",&num); <---num=7293
```

a=num%10; <---a=3

num=num/10;<---num=729

b=num%10; <---b=9

num=num/10;<---num=72

c=num%10; <---c=2

num=num/10; <---num=7

sum=(a*1000)+(b*100)+(c*10)+num;

printf("Reverseis:%d",sum);

}

==========Program-1:

Writeaprogramtoacceptlengthandbreadthofrectangleandcalculatetheareaofrectangle.

[area=length*breadth]

#include<stdio.h>

voidmain()

{

intlen,br,ar;

printf("Enterlengthofrectangle:");

scanf("%d",&len);

printf("Enterbreadthofrectangle:");

scanf("%d",&br);

ar=len*br;

printf("Areaofrectangleis:%d",ar);

}

==========

Homework:

Program-1:

Writeaprogram toaccepttemperatureofacityindegreecelciusandconvertthesame

temperatureintodegreefarenhit.[prefer'float'variables]

CHALLANGE:

Program-2:

Writeaprogramtoacceptnumbersofsecondsandprintit'sequivalentnumbersofminutes.

Forexample:

input--->240

output--->0:4:0

input--->110

output--->0:1:50

input--->246

ouput--->0:4:6

input--->3789

output--->1:3:9

Program-3:

Writeaprogramtoacceptthedistancebetweentwocitiesinkm;andconvertthesamedistance intometersandfeets.

**CHALLANGE:----------

Program-4:

Writeaprogram toaccepta4digitnumberfrom theuserandprinttheresultingnumberby adding1ineachdigit.

Forexample:

---

input--->4728

output--->5839

input--->1405

ouput--->2516

input--->7293

output--->8304

**Appearanceofdigit'9'isnotconfirm.Andcountofdigit'9'isnotconfirm.

**Youhavetowriteauniversalprogram,thatworksforeverydigit.

==========

========SolvedHomework:

Program-1:

Writeaprogram toaccepttemperatureofacityindegreecelciusandconvertthesame

temperatureintodegreefarenhit.[prefer'float'variables]

Program-2:

Writeaprogramtoacceptnumbersofsecondsandprintit'sequivalentnumbersofminutes.

Forexample:

input--->240

output--->0:4:0

input--->110

output--->0:1:50

input--->246

ouput--->0:4:6

input--->3789

output--->1:3:9

```
#include<stdio.h>
voidmain()
{
longsec;
inth,m;
printf("Enternumbersofseconds:");
scanf("%ld",&sec);
h=sec/3600;
sec=sec-(h*3600); //OR sec=sec%3600;
m=sec/60;
```

```
sec=sec-(m*60); //OR sec=sec%60;
printf("Equivalenttimeis:%d:%d:%ld",h,m,sec);
printf("\n\n\n");
}
```

==========

Program-3:

Writeaprogramtoacceptthedistancebetweentwocitiesinkm;andconvertthesamedistance intometersandfeets.

```
#include<stdio.h>
voidmain()
{
intkm;
longm;longdoubleft;
printf("EnterdistanceinKM:");
scanf("%d",&km);
m=km*1000;
ft=m*3.28;
printf("Equivalentdistanceinmeteris:%ld",m);
printf("Equivalentdistanceinfeetis:%Lf",ft);
}
```

==========

Homework:

**CHALLANGE:

Program-1:

Writeaprogram toaccepta4digitnumberfrom theuserandprinttheresultingnumberby adding1ineachdigit.

Forexample:

---

input--->4728

output--->5839

input--->1405

ouput--->2516

input--->7293

output--->8304

**Appearanceofdigit'9'isnotconfirm.Andcountofdigit'9'isnotconfirm.

**Youhavetowriteauniversalprogram,thatworksforeverydigit.

Program-2:

Acashierhascurrencynotesof10,50and100.Writeaprogram toaccepttheamounttobe withdrawnandprinthowmanynotesofeachdenominationthecashiershouldgive.Givepriority of100then50then10.

Forexample:

---

amount--->580

output--->100x5

50x1

10x3

amount--->700

output--->100x7

50x0

10x0

amount--->1250output--->100x12

50x1

10x0

==========

==========SolvedHomework:

Program-1:

Writeaprogram toaccepta4digitnumberfrom theuserandprinttheresultingnumberby adding1ineachdigit.

**Appearanceofdigit'9'isnotconfirm.Andcountofdigit'9'isnotconfirm.

**Youhavetowriteauniversalprogram,thatworksforeverydigit.

```
#include<stdio.h>
voidmain()
{
intnum,sum,second,third,fourth;
printf("Enteranumber:");
scanf("%d",&num);
fourth=num%10;
num=num/10;
fourth++;
fourth=fourth%10;
third=num%10;
num=num/10;
third++;
third=third%10;
second=num%10;
num=num/10;
second++;
num++;
second=second%10;
num=num%10;
sum=(num*1000)+(second*100)+(third*10)+fourth;
printf("Resultingnumberis%d",sum);
printf("\n\n\n\n");
}
```

==========

Program-2:

Acashierhascurrencynotesof10,50and100.Writeaprogram toaccepttheamounttobe withdrawnandprinthowmanynotesofeachdenominationthecashiershouldgive.Givepriority of100then50then10.

Forexample:

---

amount--->580

output--->100x5

50x110x3

amount--->700

output--->100x7

50x0

10x0

amount--->1250

output--->100x12

50x1

10x0

```
#include<stdio.h>
voidmain()
{
intamt,h,f,t;
printf("Enteramounttowithdraw:");
scanf("%d",&amt);
h=amt/100;
amt=amt%100;//OR amt=amt-(h*100);
f=amt/50;
amt=amt%50; //OR amt=amt-(f*50);
t=amt/10;
printf("\nThecashierhastogive\n");
printf("%dx100\n",h);
printf("%dx50\n",f);
```

```
printf("%dx10\n",t);
printf("\n\n\n\n");
}
```

==========

==========

Program-3:

Writeaprogramtoacceptvaluesof2variableandswapthem.

```
#include<stdio.h>
voidmain()
{
inta,b,c;
printf("Enter2values\n");
scanf("%d%d",&a,&b);
printf("\nIntiallya=%dandb=%d",a,b);
c=a;
a=b;
b=c;printf("\nAfterswapping,a=%dandb=%d",a,b);
printf("\n\n\n");
}
```

==========

Program-4:

Writeaprogramtoacceptvaluesof2variableandswapthemwithoutusingthirdvariable.

[Hint:useadditionandsubtractionlogic]

[ORusemultiplicationanddivisionlogic]

```
#include<stdio.h>
voidmain()
{
inta,b;
printf("Enter2values\n");
```

scanf("%d%d",&a,&b);

printf("\nIntiallya=%dandb=%d",a,b);

b=a+b;OR b=a*b;

a=b-a;OR a=b/a;

b=b-a;OR b=b/a;

printf("\nAfterswapping,a=%dandb=%d",a,b);

printf("\n\n\n");

}

==========[A]Whichofthefollowingareinvalidvariablenamesandwhy?

BASICSALARY _basic basic-hra

#MEAN group. 422

populationin2006 overtime mindovermatter

FLOAT hELLO queue.

team’svictory Plot#3 2015_DDay

-min1characterandmax31characterlong

-specialsymbolsarenotallowed.Onlyunderscoreisallowed

-spaceisnotallowed

-keywordsarenotallowedasname

-muststartwithalphabetorunderscore.

==========

[B]Writeany10invalidvariablenames.

==========Homework:

Program-1:

Ifthetotalsellingpriceof15itemsandthetotalprofitearnedonthem isinputthroughthe keyboard,writeaprogramtofindthecostpriceofoneitem.

Program-2:

Inatown,thepercentageofmenis52.Thepercentageoftotalliteracyis48.Iftotalpercentage

ofliteratemenis35ofthetotalpopulation,writeaprogramtofindthetotalnumberofilliterate menandwomenifthepopulationofthetownis80,000.

==========----Decisionmakingstatements----

==========

*Inallpreviousprograms,wewritethelogicalstatementswithinmain().Andallthese statementsexecutesfromtoptobottom.

*Iflogicallyrequired,thenwecanalsowritesomedecisionbasedstatementsbyusing"Decision makingstatements".

*Wecanuse:

-ifstatement

-ifelsestatement

-elseifstatement

-conditionaloperator

*AlltheseDecisionmakingstatements,totallydependsoncorrectnessofacondition.

*Wepreparetheseconditionsbyusing"relationaloperators".

==========

1.ifstatement:

---

*Thegeneralsyntaxofif-statementis:

```
if(condition)
{
statements;
.....
.....
}
```

-IfspecifiedconditionisTRUE,thenonlyblockofstatementsexecutes.

-IfspecifiedconditionisFALSE,thenblockofstatementsgetsskipped.

*The"if"isakeyword,thereforenoneedtoincludeanyheader-fileforit.

*Theif'sconditionhasnoconcernwithprogram'sexecutionortermination.

*Theif'sconditiononlyaffectsit'sownblock.

Forexample:

#

#

voidmain()

{

intnum=5;

printf("Hello1");

if(num>10)

{

printf("Hello2");

}

printf("Hello3");

}output-1:[num=15]

Hello1

Hello2

Hello3

==========

output-2:[num=5]

Hello1

Hello3

==========

*From aboveoutputs,wecansaythatbyusingif-statementwecanmakesomeintelligent

programsthatcantakegooddecisions.

*Wecanwriteanynumbersofstatementinif-block:input-output,expression,anything.

==========

Program-1:

Writeaprogramtoacceptoneintegervaluefromtheuserandprint"HelloWorld"ifthatvalueis

greaterthan25.

voidmain()

{

intnum;

if(num>25)

{

printf("HELLOWORLD");

}

}

==========*Weknowthat,wecanuseDecisionmakingstatementswithspecificcondition.

*Wepreparetheseconditionsbyusing"relationaloperators".Becausetheresultofrelational operatorisin"True"or"False".

*Weknowthat,generalsyntaxofif-statementis:

if(condition)

{

statements;

.....

.....

}

*IfspecifiedconditionisTRUEthenonlyif-blockexecutes.

*IfspecifiedconditionisFALSEthenif-blockgetsskipped.

==========

*Basedondifferentlogicalrequirements,thereare5possibilitiesofwritingif-statement.[also knownastypesofif-statement]

simpleif-statement

multipleif-statements

nestedif-statements

nestedmultipleif-statements

multiplenestedif-statements

==========

==========*Nowwearefamiliarwithlogicalpossibilitiesofif-statement.

*Wealsoknowthat,inmultipleif-statementsallif-statementsareindependent.

*Wealsoknowthat,innestedif-statementtheinnerif-statementsaredependentonouterif statements.

*ifspecifiedconditionisTRUE--->executetheif-block

*ifspecifiedconditionisFALSE--->skiptheif-block

*Weknowthat,wecanwriteanynumbersofstatementsandanytypeofstatementinif-block.

==========

===

==========

===

Program-1:

Writeaprogram toacceptanumberfrom theuserandprintwhetherthatnumberisgreater

than10ornot.

```
#include<stdio.h>
voidmain()
{
intnum;
printf("Enteranumber:");
scanf("%d",&num);
if(num>10)
{
printf("Thisnumberisgreaterthan10");
}
if(num<10)
{
printf("Thisnumberislessthan10");
}
if(num==10)
{
printf("Thisnumberis10");
```

```
}
}
```

==========

==

Program-2:

Writeaprogramtoacceptanumberfromtheuserandcheckwhethersquareofthatnumberis morethan100ornot.Ifsquareismorethan100thenprintYESotherwiseprintNO.

```
voidmain()
{
intnum,sq;
printf("Enteranumber:");scanf("%d",&num);
sq=num*num;
if(sq>100)
{
printf("YES");
}
if(sq<=100)
{
printf("NO");
}
}
```

==========

=========

Program-3:

Writeaprogramtoacceptanumberfromtheuserandprintit'ssquareifnumberisgreaterthan 10;otherwiesprintcubeofthatnumber.

```
voidmain()
{
intnum,temp;
```

```
printf("Enteranumber:");
scanf("%d",&num);
if(num>10)
{
temp=num*num;
printf("Squareof%dis%d",num,temp);
}
if(num<=10)
{
temp=num*num*num;
printf("Cubeof%dis%d",num,temp);
}
}
==========OR==========
voidmain()
{
intnum,sq,cu;
printf("Enteranumber:");
scanf("%d",&num);
sq=num*num;
cu=num*num*num;
if(num>10)
{
printf("Squareof%dis%d",num,sq);
}
if(num<=10)
{
printf("Cubeof%dis%d",num,cu);}
}
================================================================================
===
```

Program-4:

Writeaprogramtoaccept3anglesofatriangleandcheckwhetherthattriangleisvalidornot.

Atriangleisvalidifsumof3anglesis180.

```
voidmain()
{
inta1,a2,a3,sum;
printf("Enter3anglesofatrianlge\n");
scanf("%d%d%d",&a1,&a2,&a3);
sum=a1+a2+a3;
if(sum==180)
{
printf("Triangleisvalid");
}
if(sum!=180)
{
printf("Triangleisnotvalid");
}
}
```

==========OR==========

```
voidmain()
{
inta1,a2,a3;
printf("Enter3anglesofatrianlge\n");
scanf("%d%d%d",&a1,&a2,&a3);
if((a1+a2+a3)==180)
{
printf("Triangleisvalid");
}
if((a1+a2+a3)!=180)
{
```

```
printf("Triangleisinvalid");
}
}
==========OR==========
voidmain()
{
inta1,a2,a3;
printf("Enter3anglesofatrianlge\n");
scanf("%d%d%d",&a1,&a2,&a3);
if((a1+a2+a3)==180)
{printf("Triangleisvalid");
}
if((a1+a2+a3)>180)
{
printf("Triangleisinvalid");
}
if((a1+a2+a3)<180)
{
printf("Triangleisinvalid");
}
}
```

==========

====

Homework:

Program-1:

Writeaprogram toacceptcostpriceandsellingpriceofaproductandprintwhetherthe shopkeeperhasmadeprofitorloss.Andalsoprinttheamountheearnedorlost.[takefloat variables]

Program-2:

Writeaprogramtoaccepttotalbillamountfromtheuserandgive15%discountifbillamountis morethan1000Rs;otherwisenodiscount.

Program-3:

Writeaprogram toacceptaveragemarksofastudentandprintwhetherstudentispassor failed.Astudentispassifaveragemarksis>=40.00;otherwisefailed.

==========

========***IMP***

-Wecandeclarevariablesinif-blockalso.Thesevariablesarecalledas"blockvariables"

Forexample:

```
voidmain()
{
inta; <---isalocalvariable
.....
.....
if(condition)
{
intb; <---isablockvariable
.......
.......
}
......
......
}
```

-Theimportantthingis,scope/accessibility/availabilityofblockvariableiswithinthatblock only.

-Thereisnologicalrequirementwherewehavetogivesamenametolocalvariableandblock variable.

==========

==========

Homework:

Program-1:

Writeaprogram toacceptcostpriceandsellingpriceofaproductandprintwhetherthe shopkeeperhasmadeprofitorloss.Andalsoprinttheamountheearnedorlost.[takefloat variables]

```
voidmain()
{
floatcp,sp,temp
printf("Entercostpriceofproduct:");
scanf("%f",&cp);
printf("Entersellingpriceofproduct:");
scanf("%f",&sp);
if(sp>cp)
{
temp=sp-cp;
printf("ShopkeeperhasmadeprofitofRs%f",temp);
}
if(cp>sp)
{
temp=cp-sp;
printf("ShopkeeperhasmadelossofRs%f",temp);
}
if(cp==sp)
{printf("Shopkeeperhasmadenolossnoprofit");
}
}
```

==========

==========

Program-2:

Writeaprogramtoaccepttotalbillamountfromtheuserandgive15%discountifbillamountis morethan1000Rs;otherwisenodiscount.

```
voidmain()
{
intamt,dis,namt;
printf("Entertotalbillamount:");
scanf("%d",&amt);
if(amt>1000)
{
dis=(amt/100)*15;
namt=amt-dis;
printf("Afterdiscount,PayRs.%d",namt);
}
if(amt<=1000)
{
printf("Nodiscount.PayRs.%d",amt);
}
}
```

==========

==========

Program-3:

Writeaprogram toacceptaveragemarksofastudentandprintwhetherstudentispassor failed.Astudentispassifaveragemarksis>=40.00;otherwisefailed.

```
voidmain()
{
floatav;
printf("Enteraveragemarks:");
scanf("%f",&av);
```

```
if(av>=40.00)
{
printf("StudentisPASS");
}
if(av<40.00)
{
printf("StudentisFAILED");
}
}
```

==========

=

==========

=Program-4:

Writeaprogramtoacceptageofapersonandprintwhetherheiseligibleforl)rivingLicenceor not.Apersoniseligibleifhisageis>=18.

```
voidmain()
{
intage,temp;
printf("Enterageofperson:");
scanf("%d",&age);
if(age>=18)
{
printf("PersoniseligibleforDrivingLicence");
}
if(age<18)
{
temp=18-age;
printf("PersonisnoteligibleforDrivingLicence\n");
printf("Youcanapplyafter%dyears",temp);
}
```

}

==========

=

==========

=

Program-5:

Writeaprogramtoacceptanumberfromtheuserandprintwhetheritispositivenumberor

negativenumberorzero.

```
voidmain()
{
intnum;
printf("Enteranumber:");
scanf("%d",&num);
if(num>0)
{
printf("NumberisPOSITIVE");
}
if(num<0)
{
printf("NumberisNEGATIVE");
}
if(num==0)
{
printf("NumberisZERO");
}
}
```

==========

==========

=

Program-6:

Writeaprogramtoaccept2numbersfromtheuserandprintthelargestofthem.

```
voidmain()
{
intn1,n2;
printf("Enterfirstnumber:");
scanf("%",&n1);
printf("Entersecondnumber:");
scanf("%d",&n2);
if(n1>n2)
{
printf("%disgreater",n1);
}
if(n2>n1)
{
printf("%disgreater",n2);
}
}
```

==========OR==========

```
voidmain()
{
intn1,n2,temp;
printf("Enterfirstnumber:");
scanf("%d",&n1);
printf("Entersecondnumber:");
scanf("%d",&n2);
temp=n1-n2;
if(temp>0)
{
printf("%dislargest",n1);
}
```

```
if(temp<0)
{
printf("%dislargest",n2);
}
}
```

==========OR==========

```
voidmain()
{
intn1,n2,temp;
printf("Enterfirstnumber:");
scanf("%d",&n1);printf("Entersecondnumber:");
scanf("%d",&n2);
if(n1>n2)
{
temp=n1;
}
if(n2>n1)
{
temp=n2;
}
printf("%dislargest",temp);
}
```

==========

Homework:

CHALLANGE:

Program-1:

Writeaprogramtoacceptmarksof5subjectsofastudentandprintwhetherstudent'sresult fromfollowingcriteria.[studentispassinasubjectwhenmarksis>=40]

criteria result

---

passedinall5subjects studentisALL-CLEAR

passedinmorethan3subjects studentisPASS

passedinexact3subjects studenthasFATKT

passedinlessthan3subjects studentisFAILED.

Program-2:

Writeaprogramtoaccept3numbersfromtheuserandprintlargestofthem.[HINT:usenested if-statements]

Program-3:

Writeaprogramtoaccept4numbersfromtheuserandprintlargestofthem.[HINT:usenested if-statements]

==========

==========SolvedHomework:

Program-1:

Writeaprogramtoacceptmarksof5subjectsofastudentandprintwhetherstudent'sresult fromfollowingcriteria.[studentispassinasubjectwhenmarksis>=40]

criteria result

---

passedinall5subjects studentisALL-CLEAR

passedinmorethan3subjects studentisPASS

passedinexact3subjects studenthasFATKT

passedinlessthan3subjects studentisFAILED.

voidmain()

```
{
intm1,m2,m3,m4,m5;
intpass_count=0;
printf("Entermarksof5subjects\n");
scanf("%d%d%d%d%d",&m1,&m2,&m3,&m4,&m5);
if(m1>=40)
{
pass_count++;
}
if(m2>=40)
{
pass_count++;
}
if(m3>=40)
{
pass_count++;
}
if(m4>=40)
{
pass_count++;
}
if(m5>=40)
{
pass_count++;
}
//printingresults
if(pass_count==5)
{
printf("Student'sresultisALLCLEAR");
}
if(pass_count==4)
```

```
{
printf("Student'sresultisPASS");
}
if(pass_count==3)
{
printf("Student'sresultisFATKT");
}
if(pass_count<3)
{printf("Student'sresultisFAILED");
}
}
```

==========

======

==========

======

Program-2:

Writeaprogramtoaccept3numbersfromtheuserandprintlargestofthem.[HINT:usenested if-statements]

```
voidmain()
{
intn1,n2,n3;
printf("Enter3numbers:");
scanf("%d%d%d",&n1,&n2,&n3);
if(n1>n2)
{
if(n1>n3)
{
printf("%dislargest",n1);
}
}
```

```
if(n2>n1)
{
if(n2>n3)
{
printf("%dislargest",n2);
}
}
if(n3>n1)
{
if(n3>n2)
{
printf("%dislargest",n3);
}
}
}
```

==========

======

==========

======

Program-3:

Writeaprogramtoaccept4numbersfromtheuserandprintlargestofthem.[HINT:usenested if-statements]

```
voidmain()
{
intn1,n2,n3,n4;printf("Enter4numbers\n");
scanf("%d%d%d%d",&n1,&n2,&n3,&n4);
if(n1>n2)
{
if(n1>n3)
{
```

```
if(n1>n4)
{
printf("%dislargest",n1);
}
}
}
if(n2>n1)
{
if(n2>n3)
{
if(n2>n4)
{
printf("%dislargest",n2);
}
}
}
.....
.....
}
```

==========

======

==========

======

IMP

-IfanyconditionisTRUE----->theninterpreterresults1

-IfanyconditionisFALSE---->theninterpreterresults0

****Whenexecutionofstatementsdependsoncorrectnessofmultipleconditions,thenuse nestedif-statements.

==========

======

==========

======

Program-1:

Writeaprogramtoacceptpriceofaproductfromtheuseranddecidethecategoryofproduct

fromfollowingtable.

number message

<10 Firstcategory

11-25 Secondcategory

26-50 Thirdcategory

>50 Fourthcategoryvoidmain()

{

intprice;

printf("Enterpriceofproduct:");

scanf("%d",&price);

if(price<10)

{

printf("FirstCategory");

}

if(price>=11)

{

if(price<=25)

{

printf("SecondCategory");

}

}

if(price>=26)

{

if(price<=50)

{

printf("ThirdCategory");

}

}

if(price>50)

{

printf("FourthCategory");

}

}

==========

==========

Homework:

Program-1:

Alibrarychargesfineforeverybookreturnedlate.Writeaprogramtoacceptthenumbersof

days,apersonislatetoreturnthebook;andcalculate-printthefinefromfollowingtablewith

propermessage.

days fine

1-8 1Rs/Day

9-18 2Rs/Day

19-25 3Rs/Day

26-30 5Rs/Day

>30 Membershipiscancelled.

==========

Program-2:

Writeaprogramtoacceptaveragemarksofastudentandprintresultfromfollowingtable.averagemarks result

---

0.00- 39.99 FAILED

40.00-49.99 Cgrade

50.00-59.99 Bgrade

60.00-67.99 Agrade

68.00-100.00 Distinction

==========

==========SolvedHomework:

Program-1:

Alibrarychargesfineforeverybookreturnedlate.Writeaprogramtoacceptthenumbersof days,apersonislatetoreturnthebook;andcalculate-printthefinefromfollowingtablewith propermessage.

days fine

1-8 1Rs/Day

9-18 2Rs/Day

19-25 3Rs/Day

26-30 5Rs/Day

>30 Membershipiscancelled.

```
voidmain()
{
intdays,f;
printf("Enternumbersofdays,apersonislatetoreturnthebook:");
scanf("%d",&days);
if(days>=1)
{
if(days<=8)
{
f=days*1;
printf("FineisRs.%d",f);
}
}
```

```
if(days>=9)
{
if(days<=18)
{
f=days*2;
printf("FineisRs.%d",f);
}
}
if(days>=19)
{
if(days<=25)
{
f=days*3;
printf("FindisRs.%d",f);
}
}
if(days>=26)
{
if(days<=30)
{
f=days*5;
printf("FineisRs.%d",f);
}
}if(days>30)
{
printf("YourMembershipiscancelled");
}
}
================================================================================
=======
================================================================================
```

=======

Program-2:

Writeaprogramtoacceptaveragemarksofastudentandprintresultfromfollowingtable.

averagemarks result

0.00- 39.99 FAILED

40.00-49.99 Cgrade

50.00-59.99 Bgrade

60.00-67.99 Agrade

68.00-100.00 Distinction

```
voidmain()
{
floatav;
printf("Enteraveragemarks:");
scanf("%.2f",&av);
if(av>=0.0)
{
if(av<=39.99)
{
printf("StudentisFAILED");
}
}
if(av>=40.00)
{
if(av<=49.99)
{
printf("StudentgetsCGRADE");
}
}
if(av>=50.00)
```

{

if(av<=59.99)

{

printf("StudentgetsBGRADE");

}

}

if(av>=60.00)

{

if(av<=67.99)

{

printf("StudentgetsAGRADE");}

}

if(av>=68.00)

{

if(av<=100.00)

{

printf("StudentgetsDISTINCTION");

}

}

}

==========

=======

==========

=======

Logicaloperators:

---

*Thelogicaloperatorsaretocheckmultipleconditionsinsinglestatement.

Logicaloperator Description

&&(LogicalAND)resultsTRUEifbothoperandconditionsareTRUE;

otherwiseresultsFALSE.

||(LogicalOR) resultsTRUEifanyoneoperandconditionisTRUE;

otherwiseresultsFALSE.

!(LogicalNOT) istoinverttheresult.

***FirstprioritygoestoNOToperatorthenANDoperatorthenORoperator.

***FristpriotygoestoARITHMETICOPERATORSthenRELATIONALOPERATORSthenLOGICAL

OPERATORS.

==========

Whatwillbetheoutputoffollowingstatements?

inta=10,b=20,c=45,d=0;

a>5||b>10 --->T||T--->TRUE(1)

a!=10||b!=10 --->F||T--->TRUE(1)

a+b>35||c>20||d==0 --->F||T||T--->TRUE(1)

a!=b||b!=c||c!=d||d==0 --->T||T||T||T--->TRUE(1)

a==b||b==c||c<d||d!=0 --->F||F||F||F--->FALSE(0)

a>b&&b>c||c>d --->F&&F||T --->F||T --->TRUE(1)

a>b||b>c&&c>d --->F||F&&T --->F||F --->FALSE(0)

!(a==5) --->!(F)--->TRUE(1)

!(a>5) --->!(T)--->FALSE(0)

!(a>b)&&b>c&&!(c>d) ---> !(F)&&F&&!(T)--->T&&F&&F--->FALSE(0)!(a>b&&b>c||c>d) --->!(F&&F||T) --->!(F||T)--->!(T)--->FALSE(0)

a>5&&b>10 --->TRUE(1)

a!=10&&b!=10--->FALSE(0)

a+b>35&&c>20&&d==0 --->FALSE(0)

a!=b&&b!=c&&c!=d&&d==0 --->TRUE(1)

a!=b&&b!=c&&c<d&&d==0 --->FALSE(0)

==========

==========SolvedHomework:

Program-1:

Theefficiencyofanemployeeisdecidedbythetimetakenbyhimforanassignedtask.Writea

program toacceptthenumbersofhoursfrom theusertospecifythetimetakenbythe employeeandprintthepropermessagefromfollowingtable.[Uselogicaloperator]

Timetaken message

1-3hours"EfficincyofemployeeisHigh"

4-5hours "Efficincyofemployeeisacceptable"

6-8hours "Givewarningtoemployee..Improvetheefficiency"

9-10 "Trainingisprovidedtoemployee"

>10 "Employeehastoleavethecompany"

```
voidmain()
{
inth;
printf("Enternumbersofhours,takenbyemployeetocompletethejob:");
scanf("%d",&h);
if(h>=1&&h<=3)
{
printf("EfficincyofemployeeisHigh");
}
if(h==4||h==5)
{
printf("Efficincyofemployeeisacceptable");
}
if(h>=6&&h<=8)
{
printf("Givewarningtoemployee..Improvetheefficiency");
}
if(h==9||h==10)
{
printf("Trainingisprovidedtoemployee");
}
if(h>10)
```

```
{
printf("Employeehastoleavethecompany");
}
}
```

==========

======

CHALLANGE:

Writeaprogramtoacceptacharacterfromtheuserandprintwhetheritisanuppercaseletter orlowercaseletterordigitorspecialsymbol.[Uselogicaloperators][Hint:referASCIIvalues]

```
voidmain()
{
charvar;
printf("Enteranycharacter:");
scanf("%c",&var);
if(var>=65&&var<=90)
{
printf("Itisanuppercaseletter");
}
if(var>=97&&var<=122)
{
printf("Itisalowercaseletter");
}
if(var>=48&&var<=57)
{
printf("Itisadigit");
}
if((var>=0&&var<=47)||(var>=58&&var<=64)||(var>=91&&var<=96)
||(var>=123&&var<=127))
{
printf("Itisaspecialsymbol");
}
```

```
}
==========OR==========
voidmain()
{
charvar,temp=144;
printf("Enteranycharacter:");
scanf("%c",&var);
if(var>=65&&var<=90)
{
printf("Itisanuppercaseletter");
temp=145;
}
if(var>=97&&var<=122)
{
printf("Itisalowercaseletter");
temp=177;
}
if(var>=48&&var<=57)
{
printf("Itisadigit");
temp=225;
}
if(temp==144)
{
printf("Itisaspecialsymbol");
}}
==========OR==========
voidmain()
{
charvar,temp=144;
printf("Enteranycharacter:");
```

```
scanf("%c",&var);
if(var>=65&&var<=90)
{
printf("Itisanuppercaseletter");
temp=145;
}
if(var>=97&&var<=122)
{
printf("Itisalowercaseletter");
temp=177;
}
if(var>=48&&var<=57)
{
printf("Itisadigit");
temp=225;
}
if(!((var>=65&&var<=90)||(var>=97&&var<=122)||(var>=48&&var<=57)))
{
printf("Itisaspecialsymbol");
}
}
```

==========

**CHALLANGE:

Writeaprogramtoacceptacharacterfromtheuserandperfromswap-case.

Forexample:

---

input--->'m'

output--->EquivalentuppercaseletterisM

input--->'R'

output--->Equivalentlowercaseletterisr

==========

======

==========

======

Homework:

CHALLANGE-1

----------Acompanydecidessalaryofemployeefromfollowingtable.

AGE EXPERIENCE(inyrs) GENDER SALARY

19-25 0-3 M 25000

19-25 0-3 F 28000

26-31 4-10 M 38000

26-31 4-10 F 42000

>31 >10 M/F 50000

>45 >20 M/F 70000

Writeaprogramtoacceptage,experience(inyears),genderofanemployeeandprintthesalary fromabovetable.

Forexample:

---

age=28

exp=7

gender=m

output---->Salarywillbe38000

==========

CHALLANGE-2:

Writeaprogramtoaccept3sidesofatriangleandcheckwhetheritisanequileteraltriangleor not.[all3sidesareequal]

==========

CHALLANGE-3:

AuniversityhasthefollowingrulesforastudenttoqualifyforadegreewithAasthemain subjectandBasthesubsidiarysubject:

(a)Heshouldget55percentormoreinAand45percentormoreinB.

(b)IfhegetsthanS5percentinAheshouldget55percentormoreinB.

However,heshouldgetatleast45percentinA.

(c)IfhegetslessthanS45percentinBand65percentormoreinAheis allowedtoreappearinanexaminationinBtoqualify.

(d)Inallothercasesheisdeclaredtohavefailed.

WriteaprogramtoreceivemarksinAandBandOutputwhetherthestudenthaspassed,failed orisallowedtoreappearinB.

==========

=======SolvedHomework:

Program-1:

AuniversityhasthefollowingrulesforastudenttoqualifyforadegreewithAasthemain subjectandBasthesubsidiarysubject:

(a)HeshouldgetS5marksormoreinAand45marksormoreinB.

(b)IfhegetslessthanS5marksinAheshouldget55marksormoreinB.

However,heshouldgetatleast45marksinA.

(c)IfhegetslessthanS45marksinBand65marksormoreinAheis allowedtoreappearinanexaminationinBtoqualify.

(d)Inallothercasesheisdeclaredtohavefailed.

WriteaprogramtoreceivemarksinAandBandOutputwhetherthestudenthaspassed,failed orisallowedtoreappearinB.

```
voidmain()
{
intA,B,flag=1;
printf("EntermarksofsubjectA:");
```

```
scanf("%d",&A);
printf("EntermarksofsubjectB:");
scanf("%d",&B);
if(A>=55&&B>=45)
{
printf("StudentisPASS");
flag=2;
}
if(A>=45&&A<55&&B>=55)
{
printf("StudentisPASS");
flag=2;
}
if(B<45&&A>=65)
{
printf("Allowedtore-appearinexamofB");
flag=2;
}
if(flag==1)
{
printf("StudentisFAILED");
}
}
```

==========

==========

CHALLANGE:

Program-1:

----------Writeaprogramtoacceptmonthnumberandyearfromtheuserandprinthowmanydaysthat monthhas.

Program-2:

Writeaprogramtoaccept3sidesofatriangleandcheckwhetheritisanequileteraltriangleor isoscelecetriangleorscalenetriangleorright-angletriangle.

CHALLANGE**

Program-3:

Asteelproductmanufacturingcompanydecidesthegradeofsteelfrom followingstandard values.

1.carboncontentsmustbelessthan0.80

2.hardnessmustbemorethan11.2

3.tensilestrengthmustbemorethan14.4

Considerthat,thiscompanyhasmanufacturedalotofsteelrod.Andsendsasampleto laboratorytofindoutthevaluesofcarboncontents,hardnessandtensilestrength.

Writeaprogramtoacceptthevaluesofcarboncontents,hardnessandtensilestrengthfromthe useranddecidethegradeofsteelfromfollowingconditions.

-Gradeis'A'ifall3standardvaluesaremet.

-Gradeis'B'ifstandardvaluesof1and2aremet.

-Gradeis'C'ifstandardvaluesof2and3aremet.

-Gradeis'D'ifstandardvaluesof3and1aremet.

-Gradeis'E'ifanyonestandardvalueismet.

-Gradeis'F'ifnostandardvaluesaremet.

==========

====SolvedHomework:

Program-1:

Writeaprogramtoacceptmonthnumberandyearfromtheuserandprinthowmanydaysthat monthhas.

```
voidmain()
{
intm,y;
```

```
printf("Entermonthnumber:");
scanf("%d",&m);
printf("EnterYear:");
scanf("%d",&y);
if(m==1||m==3||m==5||m==7||m==8||m==10||m==12)
{
printf("Thismonthhas31days");
}
if(m==4||m==6||m==9||m==11)
{
printf("Thismonthhas30days");
}
if(m==2)
{
if(y%4==0)
{
printf("Thismonthhas29days");
}
if(y%4!=0)
{
printf("Thismonthhas28days");
}
}
}
```

==========

==

Program-2:

Writeaprogramtoaccept3sidesofatriangleandcheckwhetheritisanequileteraltriangleor isoscelecetriangleorscalenetriangleorright-angletriangle.

```
voidmain()
```

```
{
ints1,s2,s3;
printf("Enter3sidesoftriangle\n");
scanf("%d%d%d",&s1,&s2,&s3);
if(s1==s2&&s2==s3&&s3==s1)
{
printf("ItisanEquileteralTriangle");}
if(s1!=s2&&s2!=s3&&s3!=s1)
{
printf("ItisaScaleneTriangle");
}
if((s1==s2&&s3!=s1&&s3!=s2)||(s2==s3&&s1!=s2&&s1!=s3)||
(s3==s1&&s2!=s3&&s2!=s1))
{
printf("ItisanIsosceleceTriangle");
}
if(((s1*s1)+(s2*s2)==(s3*s3)) ||
((s2*s2)+(s3*s3)==(s1*s1)) ||
((s1*s1)+(s3*s3)==(s2*s2)))
{
printf("ItisaRightAngleTriangle");
}
}
```

==========

CHALLANGE**

Program-3:

Asteelproductmanufacturingcompanydecidesthegradeofsteelfrom followingstandard values.

1.carboncontentsmustbelessthan0.80

2.hardnessmustbemorethan11.2

3.tensilestrengthmustbemorethan14.4

Considerthat,thiscompanyhasmanufacturedalotofsteelrod.Andsendsasampleto laboratorytofindoutthevaluesofcarboncontents,hardnessandtensilestrength.

Writeaprogramtoacceptthevaluesofcarboncontents,hardnessandtensilestrengthfromthe useranddecidethegradeofsteelfromfollowingconditions.

-Gradeis'A'ifall3standardvaluesaremet.

-Gradeis'B'ifstandardvaluesof1and2aremet.

-Gradeis'C'ifstandardvaluesof2and3aremet.

-Gradeis'D'ifstandardvaluesof3and1aremet.

-Gradeis'E'ifanyonestandardvalueismet.

-Gradeis'F'ifnostandardvaluesaremet.

```
voidmain()
{
floatcc,h,ts;
intst1=0,st2=0,st3=0;
printf("EntervalueofCarbonContents:");
scanf("%f",&cc);
printf("EntervalueofHardness:");
scanf("%f",&h);
printf("EntervalueofTensileStrength:");
scanf("%f",&ts);
if(cc<0.80)
{st1=1;
}
if(h>11.2)
{
st2=1;
}
if(ts>14.4)
{
st3=1;
```

```
}
if(st1==1&&st2==1&&st3==1)
{
printf("GradeisA");
}
if(st1==1&&st2==1&&st3==0)
{
printf("GradeisB");
}
if(st1==0&&st2==1&&st3==1)
{
printf("GradeisC");
}
if(st1==1&&st2==0&&st3==1)
{
printf("GradeisD");
}
if((st1==1&&st2==0&&st3==0)||
(st2==1&&st1==0&&st3==0)||
(st3==1&&st1==0&&st2==0))
{
printf("GradeisE");
}
if(st1==0&&st2==0&&st3==0)
{
printf("GradeisF");
}
}
```

==========

======*Tilldate,wediscussedandpreparedexamplesonif-statement.

*Thegeneralsyntaxis:

if(condition)

{

statements;

.....

.....

}

*Weknowthat,

ifspecifiedconditionisTRUE,thenblockofstatementsexecutes

ifspecifiedconditionisFALSE,thenblockofstatementsgetsskipped

==========

====

----if-elsestatement----

==========

*Thegeneralsyntaxis:

if(condition)

{

statements;

.....

}

else

{

statements;

.....

}

*Tounderstandexecutionofif-elsestatement:

ifconditionisTRUE,thenif-blockexecutes

ifconditionisFALSE,thenelse-blockexecutes

*Importantthingsare:

1.Thereisonlyoneif-blockforelse-blockandonlyoneelse-blockforif-block.

2.Theremustnotbeanystatementbetweenif-blockandelse-block.

[mustappearimmediately]

3.Thereisnoconditionforelse-block.

==========

Example-1:

Writeaprogramtocheckwhetherenterednumberisevenorodd.

Byusingmultipleif-statements:

---

voidmain()

{

intnum;

.....

.....

if(num%2==0)

{

printf("NumberisEVEN");}

if(num%2!=0)

{

printf("NumberisODD");

}

}

Byusingif-elsestatement:

---

voidmain()

{

intnum;

.....

.....

if(num%2==0)

{

printf("NumberisEVEN");

}

```
else
{
printf("NumberisODD");
}
}
```

==========

*Writeaprogramtoaccept3sidesoftriangleandcheckwhetheritisanequileteraltriangleor not.

```
voidmain()
{
ints1,s2,s3;
printf("Enter3sidesoftriangle\n");
scanf("%d%d%d",&s1,&s2,&s3);
if(s1==s2&&s2==s3&&s3==s1)
{
printf("Itisanequileteraltriangle");
}
else
{
printf("Notequileteraltriangle");
}
}
```

==========

IMP

1.Executionofmultipleif-statementsandif-elseistotallydifferent.

2.Prefertheif-elseblockonlyforthesituationswherewehaveexact2options.[whichare exactoppositetoeachother]

3.Therearemanysituationswhereweneedtoprepareoppositeconditionsornotoperatoror useflags.Insuchsituations,if-elseisbestoption.

==========

=Homework: --->useif-elsestatement

Program-3:

Writeaprogram toacceptaveragemarksofastudentandprintwhetherstudentisPASSor FAILED.[studentispassifaverageis>=40.00]

Program-4:

Writeaprogramtocheckwhetherenterednumberisdivisibleby7ornot.[hint:usemodulus%]

Program-5:

Writeaprogramtoacceptageofapersonandcheckwhetherheiseligibleforlicence ornot.[personiseligibleofage>=18]

Program-6:

Writeaprogramtoaccept3anglesofatriangleandcheckwhetherthattriangleisvalidornot. [triangleisvalidifsum3anglesis180]

==========

===SolvedHomework:

Program-3:

Writeaprogram toacceptaveragemarksofastudentandprintwhetherstudentisPASSor FAILED.[studentispassifaverageis>=40.00]

```
voidmain()
{
floatav;
printf("Enteraveragemarks:");
scanf("%f",&av);
if(av>=40.00)
{
printf("StudentisPASS");
```

```
}
else
{
printf("StudentisFAILED");
}
}
```

==========

Program-4:

Writeaprogramtocheckwhetherenterednumberisdivisibleby7ornot.[hint:usemodulus%]

```
voidmain()
{
intnum;
printf("Enteranumber:");
scanf("%d",&num);
if(num%7==0)
{
printf("Numberisdivisibleby7");
}
else
{
printf("Numberisnotdivisibleby7");
}
}
```

==========

Program-5:

Writeaprogramtoacceptageofapersonandcheckwhetherheiseligibleforlicence ornot.[personiseligibleofage>=18]

```
voidmain()
{
```

```
intage;
printf("Enterageofperson:");scanf("%d",&age);
if(age>=18)
{
printf("StudentisEligibleforDL");
}
else
{
printf("StudentisnotEligibleforDL");
}
}
```

==========

Program-6:

Writeaprogramtoaccept3anglesofatriangleandcheckwhetherthattriangleisvalidornot.

[triangleisvalidifsum3anglesis180]

```
voidmain()
{
inta1,a2,a3,sum;
printf("Enterthreeanglesoftriangle\n");
scanf("%d%d%d",&a1,&a2,&a3);
sum=a1+a2+a3;
if(sum==180)
{
printf("Triangleisvalid");
}
else
{
printf("Triangleisnotvalid");
}
}
```

==========

====

***Inanycasetheinterpreterwillexecuteeitherif-blockofelse-block.

***Thereisnocondition/situationwhenif-blockandelse-blockbothexecutesorbothskips.

==========

====

****IMP****

Ifthereisonlystatementunderif-blockorelse-blockthencurlybraces{}areoptional.

Example-1:

---

voidmain()

{

intage,diff;

printf("Enterageofperson:");

scanf("%d",&age);

if(age>=18)

{printf("StudentisEligibleforDL");

}

if(age<18)

{

diff=18-age;

printf("StudentisnotEligibleforDL\n");

printf("Applyafter%dyears",diff);

}

}

***Inaboveexample,firstif-blockhasonlystatement.Sowecanavoidthecurlybraces{}for

it.Butsecondif-blockhas3statements.Soweshouldputcurlybraces{}.

==========

======Themodulusoperator(%)--->returnsremainder.

a=15%3;

a--->0

b=15/3;

b--->5

c=1567%10;

c--->7

d=1567%100;

d--->67

e=3%15;

e--->3

==========

floata=15/3;

a--->5.00

floatb=15/2;

b--->7.00

floatc=100/2;

c---->50.00

intd=15/2;

d--->7

floate=15/2.0;

e--->7.50

intf=15.78;

f--->15

floatg=15;

g--->15.00

inta=15;

intb=a;

b--->15

==========

***Charactervaluesareassignedandcomparedinsingle-inverted-comma.

inta=15;

float=25.66;

chargen='M'; <---charactervalueMmustbeassignedinsingle-quotescharopr='+'; <---charactervalue+mustbeassignedinsingle-quotes

10>20--->false

20>10--->true

'X'>'x'---->?

15>'X' ---->?

==========

*Everycharacterhasauniqueserial-number.ThisnumberiscalledasASCIIvalueofthat character.

*ASCII-->AmericanStandardCodeforInformationInterchange

*EVERYOPERATIONONCHARACTERVALUEISPERFORMEDBYREFERRINGIT'SASCIIVALUE.

*IFYOUPERFORMANYOPERATIONONANYCHARACTER,THENITWILLBEPERFORMEDONIT'S ASCIIVALUE.

Forexample:

'X'>'x' ---->willultimatelycompareASCIIvaluesof'X'and'x'

15>'X' ---->willultimatelycompare15withASCIIvalueof'X'

inta='A'+'B'; --->willultimatelyaddASCIIvaluesof'A'and'B'

chargen='M'; --->willultimatelystoreASCIIvalueof'M'

----ASCIIValues----

==========

characters ASCIIvalue

A-Z 65-90

a-z 97-122

0-9 48-57

specialsymbols 0-47,58-64,91-96,123-127

*TotalASCIIvalues--->128--->from0-127

'X'>'x'--->FALSE

15>'X' --->FALSE

inta=2+3; --->a=5

intb='2'+'3'; --->b=101

*Infirststatement,2and3aretreatedasnormalnumericintegervalues.

*Insecondstatement,'2'and'3'aretreatedascharactervalues.Thereforeinterpreterwilltake

it'sASCIIvaluesinexpression.

intx='AB'+'CD'; <---CTerror

inty='10'+'11'; <---CTerror

==========

==========

voidmain()

{

charvar='+';printf("%d",var);

}

==========

========

Solved Homework:

Program-3:

Write a program to accept average marks of a student and print whether student is PASS or FAILED.

[student is pass if average is >=40.00]

```
void main()
{
float av;
printf("Enter average marks : ");
scanf("%f", &av);
if(av>=40.00)
{
printf("Student is PASS");
}
else
```

```
{
printf("Student is FAILED");
}
}
```

==========

Program-4:

Write a program to check whether entered number is divisible by 7 or not. [hint: use modulus %]void main()

```
{
int num;
printf("Enter a number : ");
scanf("%d", &num);
if(num%7 == 0)
{
printf("Number is divisible by 7");
}
else
{
printf("Number is not divisible by 7");
}
}
```

==========

Program-5:

Write a program to accept age of a person and check whether he is eligible for Driving Licence or not. [person is eligible of age>=18]

```
void main()
{
int age;
printf("Enter age of person : ");
```

```
scanf("%d", &age);if(age>=18)
{
printf("Student is Eligible for DL");
}
else
{
printf("Student is not Eligible for DL");
}
}
```

==========

Program-6:

Write a program to accept 3 angles of a triangle and check whether that triangle is valid or not.

[triangle is valid if sum 3 angles is 180]

```
void main()
{
int a1,a2,a3,sum;
printf("Enter three angles of triangle\n");
scanf("%d%d%d", &a1, &a2, &a3);
sum = a1 + a2 + a3;
if(sum==180)
{
printf("Triangle is valid");
}
else
{printf("Triangle is not valid");
}
}
```

==========

*** In any case the interpreter will execute either if-block of else-block.

*** There is no condition/situation when if-block and else-block both executes or both skips.

==========

**** IMP ****

If there is only statement under if-block or else-block then curly braces { } are optional.

Example-1:

```
void main()
{
int age, diff;
printf("Enter age of person : ");
scanf("%d", &age);
if(age>=18)
{
printf("Student is Eligible for DL");
}
if(age<18)
{
diff = 18 - age;
printf("Student is not Eligible for DL\n");printf("Apply after %d years", diff);
}
}
```

*** In above example, first if-block has only statement. So we can avoid the curly braces { } for it.

But second if-block has 3 statements. So we should put curly braces { }.

==========

16-6-21

CHALLANGE:

Write a program to accept a date from the user and print the total numbers of days passed since 01/01/2001 upto that date. [HINT: multiplication and addition and also consider LEAP years]

```
void main()
{
int d,m,y;
```

printf("Enter date : ");

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

.....

.....

.....

}

For example: 16-6-2021

input :

d = 16

m = 6

y = 2021==========

==========

* We know that, the general syntax of if-statement is:

if(condition)

{

statements;

.....

.....

}

* And we also discussed different logical possibilities of using if-statements.

- single if statement

- multiple if statement

- nested if statements

- nested multiple if statements

- multiple nested if statements

==========

* We also discussed about if-else statement. The general syntax is:

if(condition)

{

statements;

.....

.....

}

else

{

statements;

.....

}

* We know that:

- if specified condition is TRUE, then if-block executes

- if specified condition is FALSE, then else-block executes

* Similar to if-statements, there are also some logical combinations of if-else statements. They are:

- simple if else statement

- multiple if else statements

- nested if else statements

==========

=

* What will be the output of following programs?

```
void main()
{
int a=10, b=20;
if(a>10)
{
if(b>10)
{
printf("Hello 1");
}
else
{
printf("Hello 2");
}}
```

```
else
{
printf("Hello 3");
}
}
```

The output will be:

Hello 3

==========

```
void main()
{
int a=15, b=20;
if(a>10) TRUE
{
if(b>10) TRUE
{
printf("Hello 1");
}
else
{
printf("Hello 2");
}
}
else
{
printf("Hello 3");
}}
```

The output will be:

Hello 1

==========

```
void main()
{
```

```
int a=15, b=10;
if(a>10)
{
if(b>10)
{
printf("Hello 1");
}
else
{
printf("Hello 2");
}
}
else
{
printf("Hello 3");
}
}
```

The output will be:

Hello 2==========

```
void main()
{
int a=15, b=20;
if(a>10)
{
if(b>10)
{
printf("Hello 1");
}
else
{
printf("Hello 2");
```

```
}
if(a!=b)
{
printf("Hello 4");
}
}
else
{
printf("Hello 3");
}
}
```

The output will be:

Hello 1

Hello 4==========

```
void main()
{
int a=15, b=20;
if(a>10)
{
if(b>10)
{
printf("Hello 1");
}
if(a!=b)
{
printf("Hello 4");
}
else
{
printf("Hello 2");
}
```

```
}
else
{
printf("Hello 3");
}
}
```

The output will be:

Hello 1

Hello 4==========

```
void main()
{
int a=15, b=20;
if(a>10)
{
if(b>10)
{
printf("Hello 1");
}
if(a==b)
{
printf("Hello 4");
}
else
{
printf("Hello 2");
}
}
else
{
printf("Hello 3");
}
```

}

The output will be:

Hello 1

Hello
2==========
=

=

17-6-21

*** IMP-I ***

- If a condition is TRUE then interpreter returns 1.

- if a condition is FALSE then interpreter returns 0.

Example-1:

#include<stdio.h>

void main()

{

int a=10, b=20, c, d;

c = a<b; <--- condition is TRUE

d = a>b; <--- condition is FALSE

printf("\nvalue in c is : %d", c);

printf("\nvalue in d is : %d", d);

printf("\n\n\n\n");

}

The output will be:

value in c is : 1

value in d is : 0

==========

==========
====

====

*** IMP-II ***

- In many logical situations, we manually need to specify a value in-place of if-condition.

- In such case:

- ZERO will be treated as FALSE

- any NON-ZERO value will be treated as TRUE.

Example-1:

```
if(1) <--- treated as TRUE
{
printf("Hello");
}
else
{
printf("Hi");
}
```

The output will be:

Hello

==========

Example-2:

```
if(0) <--- treated as FALSE
{
printf("Hello");
} else
{
printf("Hi");
}
```

The output will be:

Hi

==========

Example-3:

```
if(157) <--- treated as TRUE
```

```
{
printf("Hello");
}
else
{
printf("Hi");
}
```

The output will be:

Hello

==========

Example-3:

```
if(-157) <--- also treated as TRUE
{
printf("Hello"); }
else
{
printf("Hi");
}
```

The output will be:

Hello

==========

Example-4:

```
if(157.55)
{
printf("Hello");
}
else
{
printf("Hi");
```

}

The output will be:

Hello

==========

Example-5:

---

if('A') <--- considers ASCII value 65. Treated as TRUE

{ printf("Hello");

}

else

{

printf("Hi");

}

The output will be:

Hello

==========

Example-6:

---

if('0') <--- considers ASCII value 48. Treated as TRUE.

{

printf("Hello");

}

else

{

printf("Hi");

}

The output will be:

Hello

==========

* What will be the output of following statements? If find the values of the expressions in the following table:Expression Value

---

a!=6 && b>5 ---> 1

a==9 || b<3 ---> 0

! (a<10) ---> 1

! (a>5 && c) ---> 1

5 && c!=8 || !c ---> 1

==========

Program-1:

Write a program to check whether entered number is positive or negative or zero. [use if-else, single if-statement is not allowed]

```
void main()
{
int num;
printf("Enter a number : ");
scanf("%d", &num);
if(num>0)
{
printf("POSITIVE");
}
else
{
// if it is not positive, then 2 possibilities : ZERO or NEGATIVE
if(num<0)
{printf("NEGATIVE");
}
else
{
printf("ZERO");
}
}
```

```
}
========= OR ==========
void main()
{
int num;
printf("Enter a number : ");
scanf("%d", &num);
if(num==0)
{
printf("ZERO");
}
else
{
if(num>0)
{
printf("POSITIVE");
}
else
{
printf("NEGATIVE");}
}
}
========= OR ========
void main()
{
int num;
printf("Enter a number : ");
scanf("%d", &num);
if(num>=0)
{
if(num>0)
```

```
{
printf("POSITIVE");
}
else
{
printf("ZERO");
}
}
else
{
printf("NEGATIVE");
}
}
```

==========Homework:

Program-1:

Write a program to accept 2 numbers from the user and print the largest of them. Also check whether both values are equal. [A is largest, B is largest, BOTH are equal]

[use if-else only]

Program-2:

Write a program to accept 3 numbers from the user and print the largest of them. [use only if-else]

a. By using logical operators

b. Without using logical operators

Program-3:

Write a program to accept 4 numbers from the user and print the largest of them. [use only if-else]

a. By using logical operators

b. Without using logical operators

==========

=

18-6-21

Solved Homework:

----------Program-1:

Write a program to accept 2 numbers from the user and print the largest of them. Also check whether both values are equal. [A is largest, B is largest, BOTH are equal]

[use if-else only]

```
void main()
{
int n1, n2;
printf("Enter 2 numbers\n");
scanf("%d%d", &n1, &n2);
if(n1==n2)
{
printf("Both values are equal");
}
else
{
if(n1>n2)
{
printf("%d is largest", n1);
}
else
{
printf("%d is largest", n2);
}
}
}
```

Program-2:----------

Write a program to accept 3 numbers from the user and print the largest of them. [use only if-else]

a. By using logical operators

```
void main()
{
int n1, n2, n3;
printf("Enter 2 numbers\n");
scanf("%d%d%d", &n1, &n2, &n3);
if(n1>n2 && n1>n3)
{
printf("%d is largest", n1);
}
else
{
if(n2>n3)
{
printf("%d is largest", n2);
}
else
{
printf("%d is largest", n3);
}
}
}
```

b. Without using logical operators

----------void main()

```
{
int n1, n2, n3;
printf("Enter 3 numbers\n");
scanf("%d%d%d", &n1, &n2, &n3);
if(n1>n2)
{
```

```
// means 'n1' is winner
if(n1>n3)
{
printf("%d is largest", n1);
}
else
{
printf("%d is largest", n3);
}
}
else
{
// means 'n2' is winner
if(n2>n3)
{
printf("%d is largest", n2);
}
else
{
printf("%d is largest", n3);
}
}}
```

==========

Program-3:

Write a program to accept 4 numbers from the user and print the largest of them. [use only if-else]

a. By using logical operators

```
void main()
{
int n1, n2, n3, n4;
```

```
printf("Enter 4 numbers\n");
scanf("%d%d%d%d", &n1, &n2, &n3, $n4);
}
```

b. Without using logical operators

```
void main()
{
int n1, n2, n3, n4;
printf("Enter 4 numbers\n");
scanf("%d%d%d%d", &n1, &n2, &n3, $n4);}
```

** Special Logic **

```
void main()
{
int n1, n2, n3, n4, w1, w2;
printf("Enter 4 numbers\n");
scanf("%d%d%d%d", &n1, &n2, &n3, $n4);
// pair-I
if(n1>n2)
{
w1 = n1;
}
else
{
w1 = n2;
}
// pair-II
if(n3>n4)
{
w2 = n3;
}
else
```

```
{w2 = n4;
}
// pair-III
if(w1>w2)
{
printf("%d is largest", w1);
}
else
{
printf("%d is largest", w2);
}
}
```

==========

=

SOLVED CHALLANGE:

Write a program to accept a date from the user and print the total numbers of days passed since 01/01/2001 upto that date. [HINT: multiplication and addition and also consider LEAP years]

for example: d = 18, m=2, y=2021

```
void main()
{
int d,m,y;
long total_days;
long month_days, year_days, leap_days;
printf("Enter date : ");
scanf("%d%d%d", &d, &m, &y);year_days = ((y-1) - 2000) * 365;
if(m==1)
{
month_days = 0;
}
if(m==2)
```

```
{
month_days = 31;
}
if(m==3)
{
month_days = 31 + 28;
}
if(m==4)
{
month_days = 31 + 28 + 31;
}
if(m==5)
{
month_days = 31 + 28 + 31 + 30;
}
if(m==6)
{
month_days = 31 + 28 + 31 + 30 + 31;
}
.....
.....
.....
if(m==12){
month_days = 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30;
}
leap_days = ((y-1) - 2001) / 4;
if(y%4==0 && m>2)
{
leap_days++;
}
total_days = year_days + month_days + d + leap_days;
```

printf("Total numbers of days passed : %ld", total_days);

}

==========

01/01/2001 ---> Monday [1st day of our calculation]

Tuesday [2nd day of our calculation]

Wed [3rd day of our calculation]

Sunday [7th day of our calculation]

Monday [8th day of our calculation]

[8473th day of our calculation]

if(total_days % 7 == 0)

{

printf("SUNDAY");

}

if(total_days % 7 == 1)

{printf("MONDAY");

}

if(total_days % 7 == 2)

{

printf("TUESDAY");

}

.....

.....

if(total_days % 7 == 6)

{

printf("SATURDAY");

}

==========

====

21-6-21

* We know that, as per general syntax there is no semi-colon at the end of if-condition.

if(condition);

{

statements;

statements;

}

* If we provide a semi-colon at the end of if-condition then there is no syntax error. but that condition is treated as statement. Hence, the if-block executes whether the condition is TRUE or FALSE.

* And in case of if-else blocks:

- if we put semi-colon with if-block then else-block will show error. because that if-block is treated as a statement. And else-block will be like without if-block..

- if we put semi-colon with else-block then ok. That else-block will be trated as statement. It will execute whether if-condition is TRUE or FALSE.

- if we put semi-colon with if-block and else-block then still error. Because the else-block will be like without if-block.

* The same rules for nested if-blocks and nested if-else blocks.

==========

==========

---- the else-if block ----

==========

* We know that, in case of multiple if-statments where there is possibility of only one if-statement to be TRUE; then also every if-condition gets checked.

* This wastes execution time. In such case, we can use else-if blocks. The general syntax is:

if(condition-1)

{

statements;

statements;

}

else if(condition-2)

{

statements;

statements;

}

else if(condition-3){

statements;

statements;

}

else if(condition-n)

{

statements;

statements;

}

.....

.....

* If any one if-condition appears TRUE, then all remaining else-if blocks gets skipped.

It will save execution time.

==========

* Use the else-if blocks only when we feel that exact one condition will be satisified.

==========

==

Homework:

Program-1:

Write a program to find largest of 3 values [use else-if blocks]

Program-2:---

Write a program to find largest of 4 values [use else-if blocks]

==========

=

22-6-21

Solved Homework:

Program-1:

Write a program to find largest of 3 values [use else-if blocks]

```
void main()
{
int a,b,c;
printf("Enter 3 values\n");
scanf("%d%d%d", &a, &b, &c);
if(a>b && a>c)
{
printf("%d is largest", a);
}
else if(b>a && b>c)
{
printf("%d is largest", b);
}
else if(c>a && c>b)
{printf("%d is largest", c);
}
}
```

==========

Program-2:

Write a program to find largest of 4 values [use else-if blocks]

```
void main()
{
int a,b,c,d;
printf("Enter 4 values\n");
```

```
scanf("%d%d%d%d", &a, &b, &c, &d);
if(a>b && a>c && a>d)
{
printf("%d is largest", a);
}
else if(b>a && b>c && b>d)
{
printf("%d is largest", b);
}
else if(c>a && c>b && c>d)
{
printf("%d is largest", c);
}
else if(d>a && d>b && d>c){
printf("%d is largest", d);
}
}
```

==========

* Now we know that, we can use else-if statement in-place of multiple if-statements; if there is possibility of correctness of only one condition.
* Important thing is, when an if-condition appears TRUE then all following else-blocks gets skipped.
* This will save execution-time.

==========

==========

* The else-if block has another general-syntax also:

```
if(condition-1)
{
statements;
```

```
.....
}
else if(condition-2)
{
statements;
.....
}else if(condition-n)
{
statements;
.....
}
else
{
statements;
.....
}
```

* This last else-block is optinal.

* This last else-block executes when no-condition satisfies. [if all above conditions are false].

==========

Program-1:

Write a program to find largest of 4 values.

```
void main()
{
int a,b,c,d;
printf("Enter 4 values\n");
scanf("%d%d%d%d", &a, &b, &c, &d);
if(a>b && a>c && a>d)
{
printf("%d is largest", a);
}else if(b>a && b>c && b>d)
```

```
{
printf("%d is largest", b);
}
else if(c>a && c>b && c>d)
{
printf("%d is largest", c);
}
else
{
printf("%d is largest", d);
}
}
```

==========

Program-2:

Write a program to accept month number and year from the user and print how many days that month has. [use else-if blocks]

```
void main()
{
int m, y;
printf("Enter month number : ");
scanf("%d", &m);
printf("Enter Year : ");
scanf("%d", &y);if(m==4 || m==6 || m==9 || m==11)
{
printf("This month has 30 days");
}
else if(m==2)
{
if(y%4 == 0)
```

```
{
printf("This month has 29 days");
}
else
{
printf("This month has 28 days");
}
}
else
{
printf("This month has 31 days");
}
}
```

==========

==

Homework:

Program-3:

An insurance company decides the monthly premium from following table.age lives-in heath-status gender max_policy

(C/V) (E/P) amount

25-35 C E M 2,00,000

25-39 C E F 1,50,000

25-35 V E M 1,00,000

Write a program to accept age, living region, health-status and gender of applicant and decide whether he/she can have the policy or not. The applicant is able to have policy if he matches above criteria. In all other criteria, the applicant is not allowed to have policy.

```
void main()
{
```

```
if( ... )
{
printf("You are eligible for Policy. Max policy amount can be 2,00,000");
}
else if( ... )
{
printf("You are eligible for Policy. Max policy amount can be 1,50,000");
}
else if( ... )
{
printf("You are eligible for Policy. Max policy amount can be 1,00,000");
}
else
{
printf("You are not eligible for policy");}
}
```

==========

==

*** NOTE ***

If you found "in all other cases/criteria" then that program must have that last else-block to print "Not applicable for policy"

==========

==

23-6-21

1. if statement:

---

* The general syntax is:

```
if(condition)
{
statements;
statements;
```

}

* Logical possibilities of if-statements:

- simple if statement

- multiple if statement

- nested if statement

- nested multiple if statemnets

- multiple nested if statements==========

==========

2. if else statement:

---

* The general syntax is:

if(condition)

{

statements;

statements;

}

else

{

statements;

statements;

}

* Logical possibilities of if-else statement are:

- simple if else

- multiple if else

- nested if else [3 possibilities in nested if-else]

==========

==========

3. The else-if block

---

* The general syntax is:

```
if(condition-1)
{statements;
statements;
}
else if(condition-2)
{
statements;
statements;
}
else if(condition-n)
{
statements;
statements;
}
else
{
statements;
statements;
}
```

* The last else-block is optional.

* It has 2 advantages:

- when an if-condition appears TRUE, then remaining else-blocks gets skipped.
  This minimizes execution time in checking other conditions.
- We can prepare an else-block that executes if all above if-conditions appears
  FALSE.

==========

==

==========

==

Homework:----------

Program-1:

An insurance company decides the monthly premium from following table.

age lives-in heath-status gender max_policy premium per

(C/V) (E/P) amount 1000 Rs.

25-35 C E M 2,00,000 8.20

25-39 C E F 1,50,000 7.85

25-35 V E M 1,00,000 7.40

PART-I:

Write a program to accept age, living region, health-status and gender of applicant and decide whether he/she can have the policy or not. The applicant is able to have policy if he matches above criteria. In all other criteria, the applicant is not allowed to have policy.

PART-II:

If that person is eligible for policy, then ask for policy amount he/she wants to apply.

If the requested policy amount is within range/quota, then print the premium amount.

```
void main()
{
long pamount;
if( ... ) <--- criteria 1
{printf("You are eligible for policy\n");
printf("Enter Policy amount, you want to apply for : ");
scanf("%ld", &pamount);
if(pamount>200000)
{
printf("Policy limit exceeds\n");
}
else
{
prem = (pamount/1000) * 8.20;
```

```
printf("Premium will be Rs.%f", prem);
}
}
else if( ... )
{
}
else if( ... )
{
}
else
{
printf("You are not eligible for policy");
}
}
```

==========

==

==========

==---- Conditional Operator ---- OR ---- Ternary Operator ----

========== ==========

* The conditional operator is shortcut of if-else statement.
* The general syntax of using conditional operator is:

condition ? statement-1 : statement-2 ;

* If specified condition is TRUE, then statement-1 executes.
* If specified condition is FALSE, then statement-2 executes.

For example:

Write a program to check whether entered number is even or odd.

```
void main()
{
int num;
printf("Enter a number : ");
```

```
scanf("%d", &num);
if(num%2 == 0)
{
printf("is EVEN");
}
else
{
printf("is ODD");
}
}===== OR =====
void main()
{
int num;
printf("Enter a number : ");
scanf("%d", &num);
num%2 == 0 ? printf("is EVEN") : printf("is ODD") ;
}
```

==========

**** IMP ****

- Remember that, we apply semi-colon at the end of statement. [indicates end of statement]
- When we use conditional operator as a whole single statement.
- In above example, the last simi-colon indicates end of conditional-operator's statement. It is not as the end of printf() statement.

==========

*** MOST IMP ***

- We can write only one statement as TRUE part and only one statement as FALSE part.
- This means, we can use conditional operator only when there is single statement in if-else blocks.

==========

Program-2:

Write a program to accept a number from the user and print whether it is greater than 10 or not.

[use conditional operator]void main()

{

int num;

printf("Enter a number : ");

scanf("%d", &num);

num>10 ? printf("This number is greater than 10") : printf("This number is less

than 10") ;

}

==========

Program-3:

Write a program to accept two numbers from the user and print largest of them. [use conditional operator]

void main()

{

int n1, n2;

printf("Enter two numbers : ");

scanf("%d%d", &n1, &n2);

n1>n2 ? printf("%d is largest", n1) : printf("%d is largest", n2) ;

}

====== OR ======void main()

{

int n1, n2, c;

printf("Enter two numbers : ");

scanf("%d%d", &n1, &n2);

n1>n2 ? c=n1 : c=n2;

printf("%d is largest", c);

}

==========

==========

Program-3:

Write a program to accept 3 sides of a triangle, and check whether that triangle is equileteral triangle or not.

```
void main()
{
int s1, s2, s3;
printf("Enter 3 sides of triangle\n");
scanf("%d%d%d", &s1, &22, &3);
s1==s2 && s2==s3 && s3==s1 ? printf("Triangle is Equileteral") :
printf("Triangle is not Equileteral") ;
}
```

==========

==

24-6-21

---- Conditional Operator ----

==========

* The conditional operator is also called as "ternary operator".

* The general syntax is:

condition ? statement-1 : statement-2 ;

* If specified condition is TRUE, then statement-1 executes.

* If specified condition is FALSE, then statement-2 executes.

*** Only one statement is allowed as TRUE-part and only one statement is allowed as FALSE part.

==========

==========

* We know that, the conditional operator is alternate of if-else statement.

* Similar to if-else, we can also use nesting in conditional operator.

* Same 3 possibilities like if-else.

==========

==========

What will be the output of following statements?

```
int a=10, b=5, c=0, d=-5;6)
```

a>b && c>d ? c==0? printf("Hello 1") : printf("Hello 2")

:

d>1 ? printf("Hello 3") : printf("Hello 4) ;

Hello 1

==========

5)

a>b && a>c ? printf("Hello 1") : printf("Hello 2") ;

Hello 1

==========

4)

a<b ? a>c?printf("Hello 1") : printf("Hello 2") : printf("Hello 3") ;

Hello 3

==========

3)

a>b ? a>c?printf("Hello 1") : printf("Hello 2") : printf("Hello 3") ;

Hello 1

==========2)

a>b ? printf("Hello 1") : printf("Hello 2") ;

Hello 1

==========

1)

a<b ? printf("Hello 1") : printf("Hello 2") ;

Hello 2

==========

=

Homework:

Conver above 6 conditional operator statements into if-else blocks.

==========

=

25-6-21

* The general syntax of conditional operator is:

condition ? statement-1 : statement-2 ;

* If specified condition is TRUE, then statement-1 executes.

* If specified condition is FALSE, then statement-2 executes.

* Nested conditional operator is also valid. ==========

*** IMP ***

- We can use conditional operator in assignment statement also.

For example:

---

* Refer following statement --->n1>n2 ? c=n1 : c=n2;

* If condition (n1>n2) is TRUE then value of 'n1' assigns to 'c'; and if condition (n1>n2) is FALSE then value of 'n2' assigns to 'c'.

* The same conditional operator statement can be written as:

c = n1>n2 ? n1 : n2 ;

==========

* What will be the output of following programs ?

1)

```
void main()
{
int i=-4, j;
j = (i<0 ? 0 : (i*i) );
printf("%d", j);
}
```

output:

j=0

==========2)

```
void main()
{
int i=4, j;
j = (i<0 ? 0 : (i*i) );
printf("%d", j);
```

```
}
```

output:

j=16

==========

3)

```
void main()
{
int k, num=30;
k = (num>5 ? (num<=10 ? 100:200) : 500 );
printf("%d", k);
}
```

output:

k=200

==========

4)

```
void main()
{
int k, num=10;
k = (num>5 ? (num<=10 ? 100:200) : 500 );printf("%d", k);
}
```

output:

k=100

==========

5)

```
void main()
{
int j=4;
!j != 1 ? printf("Chapter ends") : printf("Still pending") ;
}
```

==========

===

26-6-21

---- LOOP ----

==========

OR

---- Iterative Statements ----

==========

OR

---- Repeatitive Statements ----==========

**** The loops are the statements which can iterate/repeat/rotate multiple times on same block of statements.

**** Previously, the if-blocks and else-blocks executes only once (based on condition); Whereas, the loop iterates/repeats/rotates numbers of times on same block of statements.

**** The 'C' language has 3 loops:

- for loop
- while loop
- do while loop

* The purpose/working/application of these three loops is SAME. i.e. we can use them as logical alterante of each other.

* Means, the program implemented by using one loop; can be implemented by using another two loops.

* But the logical situation of these loops are different.

---> Prefer for-loop when numbers of iterations are already known.

---> Prefer while-loop when numbers of iterations are dependent on an internal situation or an internal value.

---> Prefer do-while loop when at-least one iteration is required. Particularly, for user-driven programs.

==========

28-6-21

* We know that, the loops are the statements that can iterate/repeat the block of statements.* There are 3 loops:

- for loop

- while loop

- do while loop

* We prefer for-loop when numbers of iterations are already known.

* We perfer while-loop when numbers of iterations are dependent on an internal situation or internal value.

* We perfer do-while loop when we want to iterate at-least once. Also perferred for user-driven programs.

==========

* A loop is block of statements that can iterate.

==========

---- for loop ----

==========

* We prefer for-loop when numbers of iterations are already known.

* The general syntax to use for-loop is:

```
for(initialization; condition; incr/decr)
{
statements;
statements;
.....
}
```

Initialization ---> happens first and only once

Condition ---> gets checked before every iteration

Statements ---> executes if condition is TRUE

Incr/Decr ---> happens after execution of statements

* The loop will iterate unless condition remains TRUE.

For example:

Write a program to print "Hello World" message 5 times.

```
void main()
{
```

```
int i;
for(i=1; i<=5; i++)
{
printf("Hello World\n");
}
}
```

==========

```
for(i=5; i>=1; i--)
{
printf("Hello World\n");
}
```

==========

```
for(i=185; i<=189; i++){
printf("Hello World\n");
}
```

==========

```
for(i=1; i<=10; i=i+2)
{
printf("Hello World\n");
}
```

==========

** Homework:

Write a program to print "Hello World" message 10 times.

[Do the same program with 5 different ways to use for-loop]

[Try adjusting different ranges and changes/steps]

==========

=

29-6-21

Solved Homework:

Write a program to print "Hello World" message 10 times.

[Do the same program with 5 different ways to use for-loop]

[Try adjusting different ranges and changes/steps]

```
void main(){
int i;
for(i=1; i<=10; i++)
{
printf("Hello World\n");
}
}
```

==========

Program-2:

Write a program to print your name 15 times. All in new line.

```
void main()
{
int i;
for(i=1; i<=15; i++)
{
printf("Rohan Yerme\n");
}
}
```

==========

Program-3:

Write a program to print all the numbers from 1 to 50. [use for-loop]

```
void main()
{int i;
for(i=1; i<=50; i++)
{
printf("%d\n", m);
```

```
}
}
```

===== OR =====

```
void main()
{
int i, m;
for(i=51; i<=100; i++)
{
m = i - 50;
printf("%d\n", m);
}
}
```

Program-4:

Write a program to print all the numbers from 50 to 1. [use for-loop]

```
void main()
{
int i;
for(i=50; i>=1; i--)
{
printf("%d\n", i);}
}
```

Program-5:

Write a program to print all the even numbers from 1 to 100. [use for-loop]

```
void main()
{
int i;
for(i=2; i<=100; i=i+2)
```

```
{
printf("%d\n", i);
}
}
```

===== OR =====

```
void main()
{
int i;
for(i=1; i<=100; i++)
{
if(i%2 == 0)
{
printf("%d\n", i);
}
}
```

}===== OR =====

```
void main()
{
int i;
for(i=2; i<=100; i++)
{
printf("%d\n", i);
i++;
}
}
```

===== OR =====

```
void main()
{
int i, m;
for(i=1; i<=50; i++)
{
```

```
m = i * 2;
printf("%d\n", i);
}
}
```

==========

Program-6:

Write a program to print the square and cube of all the numbers from 1 to 10. The output must be in tabular form as:1 1 1

2 4 8

3 9 27

....

....

....

10 100 1000

```
void main()
{
int i, s, c;
for(i=1; i<=10; i++)
{
s = i * i;
c = i * i * i;
printf("%d\t%d\t%d\n", i, s, c);
}
}
```

==========

Program-7:

Write a program to print all the numbers from 1 to n. Where 'n' is entered by the user.

```
void main()
{
```

```
int i, n;printf("Enter value of n : ");
scanf("%d", &n);
for(i=1; i<=n; i++)
{
printf("%d\n", i);
}
}
```

==========

==

30-6-21

**CHALLANGE

Program-1:

Write a program to print all the numbers from 'm' to 'n'. But make sure that the result must be in ASCENDING order. [from smaller to larger]

```
#include<stdio.h>
void main()
{
int m, n, i, sm, lar;
printf("Enter value of m : ");
scanf("%d", &m);
printf("Enter value of n : ");
scanf("%d", &n);
sm = m<n ? m : n;lar = m>n ? m : n;
for(i=sm; i<=lar; i++)
{
printf("%d\n", i);
}
}
```

==========

Program-2:

Write a program to print all the UPPERCASE letters with it's ASCII value. The output must be as:

A - 65

B - 66

C - 67

....

....

Z - 90

```
void main()
{
int i;
for(i=65; i<=90; i++)
{
printf("%c - %d\n", i, i);
}
}
```

====== OR ======

```
void main()
{
 char i;
 for(i='A'; i<='Z'; i++)
 {
 printf("%c - %d\n", i, i);
 }
}
```

Program-3:

Write a program to accept a number from the user and print it's multiplication table.

```
void main()
{
```

```
int i, num, m;
printf("Enter a number : ");
scanf("%d", &num);
for(i=1; i<=10; i++)
{
m = num * i;
printf("%d\n", m);
}
}
```

====== OR ======

```
void main()
{
int i, num, m;
printf("Enter a number : ");
scanf("%d", &num);
for(i=1; i<=10; i++)
{
m = num * i;
printf("%d x %d = %d\n", num, i , m);
}
}
```

Program-4:

Write a program to accept 5 numbers and print the same. Use scanf() and printf() in loop.

So that it accepts and prints in each iteration.

```
void main()
{
int i, num;
for(i=1; i<=5; i++)
{
printf("\n\nEnter number : ");
```

```
scanf("%d", &num);printf("You entered : %d", num);
}
}
```

==========

** IMP **

Program-5:

Write a program to print sum of all numbers from 1 to 10.

```
void main()
{
int i, sum=0;
for(i=1; i<=10; i++)
{
sum = sum + i;
}
printf("Sum is : %d", sum);
}
```

==========

Program-6:

Write program to print sum of all numbers from 1 to n; where, 'n' is entered by the user.

```
void main()
{int i, sum=0, n;
printf("Enter value of n : ");
scanf("%d", &n);
for(i=1; i<=n; i++)
{
sum = sum + i;
}
printf("Sum is : %d", sum);
}
```

==========

Program-7:

Write a program to print the product of all the numbers from 1 to n; where 'n' is entered by the user.

OR

Write a program to accept a number and print it's factorial.

```
void main()
{
int i, prod=1, n;
printf("Enter value of n : ");
scanf("%d", &n);for(i=1; i<=n; i++)
{
prod = prod * i;
}
printf("Product is : %d", prod);
}
```

==========

=

1-7-21

* The general syntax of using for-loop is:

```
for(initialization; condition; incr/decr)
{
statements;
.....
.....
}
```

* We can perform any logical operation in body of for-loop.

* We can write I/O statements, if-statements, conditional operator, expression, etc...

```
for(i=1; i<=5; i++)
{
```

```
printf("\nEnter a number : ");
scanf("%d", &num);
printf("You entered : %d"\n", num);}
====================
sum = 0;
for(i=1; i<=n; i++)
{
sum = sum + i;
}
printf("Sum of all numbers is : %d", sum);
======================
prod = 1;
for(i=1; i<=n; i++)
{
prod = prod * i;
}
printf("Product of all numbers is : %d", prod);
======================
for(i=1; i<=10; i++)
{
m = num * i;
printf("%d\n", m);
}==========================================================================
=
===
```

Program-1:

Write a program to accept 20 numbers from the user. And for each number, check whether it is even or odd.

```
void main()
{
```

```
int num, i;
printf("Enter 20 numbers\n");
for(i=1; i<=20; i++)
{
scanf("%d", &num);
if(num%2 == 0)
{
printf("This number is EVEN\n");
}
else
{
printf("This number is ODD\n");
}
}
}
```

==========

Program-2:----------

Write a program to accept 20 numbers from the user and print the total count of even and odd numbers entered by the user.

```
void main()
{
int i, num;
int ecount=0, ocount=0;
printf("Enter 20 numbers\n");
for(i=1; i<=20; i++)
{
scanf("%d", &num);
if(num%2 == 0)
{
ecount++;
```

```
}
else
{
ocount++;
}
}
printf("\n\nTotal count of EVEN numbers : %d", ecount);
printf("\nTotal count of ODD numbers : %d", ocount);
}
```

==========

Program-3:----------

Write a program to accept 20 numbers from the user and print total count of positive, negative and zeroes entered by the user.

==========

Program-4:

Write a program to accept 20 numbers from the user and print the individual sum of even and odd numbers entered by the user.

```
void main()
{
int i, num;
int esum=0, osum=0;
printf("Enter 20 numbers\n");
for(i=1; i<=20; i++)
{
scanf("%d", &num);
if(num%2 == 0)
{
esum = esum + num;
```

```
}
else
{
osum = osum + num;
}
}printf("\n\nSum of all EVEN numbers is : %d", esum);
printf("\nSum of all ODD numbers is : %d", osum);
}
```

==========

===

Program-5:

Write a program to accept 20 numbers from the user and print the sum of numbers in group of 5.

```
void main()
{
int i, num, sum=0;
printf("Enter 20 numbers\n");
for(i=1; i<=20; i++)
{
scanf("%d", &num);
sum = sum + num;
if(i%5 == 0)
{
printf("Sum of last 5 numbers is : %d\n", sum);
sum = 0;
}
}
}
```

===== OR =====

```
void main()
{
int i, num, sum=0;
```

```
printf("Enter 20 numbers\n");
for(i=1; i<=20; i++)
{
scanf("%d", &num);
sum = sum + num;
if(i==5)
{
printf("Sum of last 5 numbers is : %d\n", sum);
sum = 0;
}
else if(i==10)
{
printf("Sum of last 5 numbers is : %d\n", sum);
sum = 0;
}
else if(i==15)
{
printf("Sum of last 5 numbers is : %d\n", sum);
sum = 0;
}
else if(i==20)
{
printf("Sum of last 5 numbers is : %d\n", sum);
sum = 0;}
}
}
```

==========

2-7-21

* The general syntax is:

```
for(initialization; condition; incr/decr)
{
```

statements;

.....

.....

}

* We know that, we can write any logical statements in body of for-loop; like, I/O statements, if statements, conditional operator, expression, etc...

* We know that, if there is only one statement in body of if-block or else-block then curly braces { }

are optional.

* Similarly, if there is only one statement in body of any loop then curly braces { } are optional.

* We can declare variables inside body/block of if or else or loop.

* If we declare any variable inside body/block of if or else or loop, such variables are called as "block

variables". The scope/availability/accessibility of such block variable is within that block.

* For example:

void main()

{

int a,b,c,i; <--- local variables of main() method; OR ..... function variables; accessible within that entire function

.....

if(condition)

{

int x; <--- x is block variable; accessible within this if-block only.

.....

.....

}

else

{

int y; <--- y is block variable; accessible within this else-block only.

.....

.....

}

.....

.....

```
for(i=1; i<=10; i++)
{
int z; <--- z is block variable; accessible within this for-block only.
.....
.....
}
}
```

==========

==

==========

==

* "Indentation" is to improve readability of a program.
* Indentation is facility of IDE.

```
void main()
{
int a,b,c;
if(a>b)
{
printf("a is largest");
}
else
{
printf("b is largest");
}
for(i=1; i<=10; i++)
{
c = i * i;
printf("Square is : %d\n", c);
}
}
```

==========

==

==========

==

---- keyword "break" ----

==========

* We know that, the loop iterates unless condition becomes FALSE.

* In many logical situations we can experience that the result or conclusion or decision appears even before loop terminates. [in early itertations]

* In such situations, we can immediately stop/terminate the loop's iterations by using keyword "break".* In other words, keyword "break" is to immediately terminate a loop.

* Generally, we use condition based "break" statement.

```
void main()
{
int i;
for(i=1; i<=5; i++)
{
printf("Start %d", i);
if(i==3)
{
break;
}
printf("End %d", i);
}
}
```

The output will be:

Start 1

End 1

Start 2

End 2

Start 3

==========

===

2-7-21(02)---- keyword "break" ----

==========

* In many logical situations, we find that the result/conclusion is found even before loop's iterations ends. In such situation we can terminate the loop by using keyword "break".

* The use of "break" should be condition based.

```
void main()
{
int i;
for(i=1; i<=5; i++)
{
printf("\nstart : %d", i);
if(i==3)
{
break;
}
printf("\nend : %d", i);
}
}
```

The output will be:

```
start : 1
end : 1
start : 2
end : 2
start : 3
```

==========

==========

=

=

----- keyword "continue" -----

==========

* The keyword "continue" takes the controls back to condition.

* Ultimately, the statements after "continue" will get skipped for that iteration.

* The use of "continue" should be condition based.

```
void main()
{
int i;
for(i=1; i<=5; i++)
{
printf("\nstart : %d", i);
if(i==3)
{
continue;
}
printf("\nend : %d", i);
}
}
```

The output will be:

```
--------------------
start : 1
end : 1
start : 2
end : 2
start : 3
start : 4end : 4
start : 5
end : 5
```

==========

===

==========

===

* What will be the output of following programs?

```
void main()
{
int i,j=5;
for(i=1; i<=5; i++)
{
printf("start : %d - %d\n", i, j);
j--;
if(i==j)
{
break;
}
printf("end : %d - %d\n", i, j);
j--;
}
}
```

The output will be:

```
----------
start : 1 - 5
end : 1 - 4
start : 2 - 3================================================================
================================================================
```

```
void main()
{
int i,j=5;
for(i=1; i<=5; i++)
{
printf("start : %d - %d\n", i, j);
j--;
if(i==j)
{
continue;
```

```
}
printf("end : %d - %d\n", i, j);
j--;
}
}
```

The output will be:

```
----------
start : 1 - 5
end : 1 - 4
start : 2 - 3
start : 3 - 2
end : 3 - 1
start : 4 - 0
end : 4 - -1
start : 5 - -2
end : 5 - -3==================================================
==================================================
void main()
{
int i,j=72;
for(i=52; i<=80; i=i+2)
{
printf("%d - %d\n", i, j);
if(i==j)
{
break;
}
j = j - 3;
}
}
```

The output will be:

---

52 - 72

54 - 69

56 - 66

58 - 63

60 - 60

==========

==========

```
void main(){
int i,j=72;
for(i=52; i<=62; i=i+2)
{
printf("%d - %d\n", i, j);
if(i==j)
{
continue;
}
j = j - 3;
}
}
```

The output will be:

---

52 - 72

54 - 69

56 - 66

58 - 63

60 - 60

62 - 60

==========

===

3-7-21

---- keyword "break" ----

==========* In many logical situations, we find that the result/conclusion is found even before loop's iterations

ends. In such situation we can terminate the loop by using keyword "break".

* The use of "break" should be condition based.

```
void main()
{
int i;
for(i=1; i<=5; i++)
{
printf("\nstart : %d", i);
if(i==3)
{
break;
}
printf("\nend : %d", i);
}
}
```

The output will be:

```
---------- 
start : 1
end : 1
start : 2
end : 2
start : 3
```

==========

=

==========

=----- keyword "continue" -----

==========

* The keyword "continue" takes the controls back to condition.

* Ultimately, the statements after "continue" will get skipped for that iteration.

* The use of "continue" should be condition based.

```
void main()
{
int i;
for(i=1; i<=5; i++)
{
printf("\nstart : %d", i);
if(i==3)
{
continue;
}
printf("\nend : %d", i);
}
}
```

The output will be:

```
----------
start : 1
end : 1
start : 2
end : 2
start : 3
start : 4
end : 4
start : 5
end :
5==========
=
===
==========
===
```

* What will be the output of following programs?

```
void main()
{
int i,j=5;
for(i=1; i<=5; i++)
{
printf("start : %d - %d\n", i, j);
j--;
if(i==j)
{
break;
}
printf("end : %d - %d\n", i, j);
j--;
}
}
```

The output will be:

```
---------------------
start : 1 - 5
end : 1 - 4
start : 2 - 3
================================================================
================================================================
```

```
void main()
{
int i,j=5;
for(i=1; i<=5; i++)
{
printf("start : %d - %d\n", i, j);
j--;
if(i==j)
{
continue;
```

```
}
printf("end : %d - %d\n", i, j);
j--;
}
}
```

The output will be:

```
----------
start : 1 - 5
end : 1 - 4
start : 2 - 3
start : 3 - 2
end : 3 - 1
start : 4 - 0
end : 4 - -1
start : 5 - -2
end : 5 - -3
```

==========

==========

```
void main()
{
int i,j=72;
for(i=52; i<=80; i=i+2)
{
printf("%d - %d\n", i, j);
if(i==j)
{
break;
}
j = j - 3;
}
}
```

The output will be:

---

52 - 72

54 - 69

56 - 66

58 - 63

60 - 60

==========

==========

```
void main()
{
int i,j=72;for(i=52; i<=62; i=i+2)
{
printf("%d - %d\n", i, j);
if(i==j)
{
continue;
}
j = j - 3;
}
}
```

The output will be:

---

52 - 72

54 - 69

56 - 66

58 - 63

60 - 60

62 - 60

==========

===

4-7-21

Solved Homework:

Program-1:

----------Write a program to accept a 3 digit number from the user and check whether it is an ARMSTRONG

number or not. [no need to use loop] [A number is ARMSTRONG number if sum of cubes of each

digit is equal to original number.]

For example:

---

input ---> 234

(2*2*2) + (3*3*3) + (4*4*4) == 8+27+64 == 99

This is not armstrong number

input ---> 153

(1*1*1) + (5*5*5) + (3*3*3) == 1+125+27 = 153

This is an armstrong number

```
void main()
{
int num, a, b, sum;
int temp;
printf("Enter a 3 digit number : ");
scanf("%d", &num);
temp = num;
a = num % 10; <---- third digit
num = num / 10;
b = num %10; <---- second digit
num = num / 10; <---- first digit
sum = (a*a*a) + (b*b*b) + (num*num*num);if(sum == temp)
{
printf("This number is an ARMSTRONG number");
}
else
```

```
{
printf("This number is not an ARMSTRONG number");
}
}
```

==========

==

Program-2:

Write a program to print all the armstrong numbers from 100 to 999. [have to use loop]

```
#include<stdio.h>
void main()
{
int i, temp, a,b,sum;
for(i=100; i<=999; i++)
{
temp = i;
a = temp % 10;
temp = temp / 10;
b = temp % 10;temp = temp / 10;
sum = (a*a*a) + (b*b*b) + (temp*temp*temp);
if(sum == i)
{
printf("%d\n\n", i);
}
}
}
```

==========

==

----- while loop -----

==========

* We know that, we prefer while-loop when numbers of iteration are not known. In other words,

when numbers of iterations are dependent on an internal situation or internal value.

* The general syntax is:

```
initialization;
while(condition)
{
statements;
statements;
.....
incr/decr;
}
```

* Similar to for-loop, here also the loop iterates unless the condition becomes FALSE.

==========

Program-1:

Write a program to print "Hello World" message 10 times.

Using for-loop:

```
void main()
{
int i;
for(i=1; i<=10; i++)
{
printf("Hello World\n");
}
}
```

Using while-loop:

```
void main()
{
int i;
```

```
i=1;
while(i<=10)
{
printf("Hello World\n");
i++;
}
}==========================================================================
=
==
```

Program-2:

Write a program to print all the numbers from 1 to 10. [Use while-loop]

```
void main()
{
int i;
i = 1;
while(i<=10)
{
printf("%d\n", i);
i++;
}
}
```

==========

===

Program-3:

Write a program to print all the numbers from 10 to 1. [Use while-loop]

==========

===

Program-4:

----------Write a program to print all the even numbers from 1 to 100. [Use while-loop]

```
void main()
{
int i=2;
while(i<=100)
{
printf("%d\n", i);
i = i + 2;
}
}
```

====== OR ======

```
void main()
{
int i=1;
while(i<=100)
{
if(i%2 == 0)
{
printf("%d\n", i);
}
i++;
}
}
```

==========

===

5-7-21(01)* The general syntax of for-loop is:

```
for(initialization; condition; incr/decr)
{
statements;
statements;
.....
}
```

* The general syntax of while-loop is:

initialization;

while(condition)

{

statements;

statements;

.....

incr/decr;

}

* The result of for-loop and while-loop is same. Both loop iterates.

* The programs prepared by using for-loop, can be prepared by using while-loop also.

* We can also use keywords "break" and "continue" in while-loop.

* We can also use other decision making statements within while-loop.

==========

==========

* What will be the output of following programs?void main()

{

int i=52, j=72;

while(i!=j)

{

printf("%d - %d\n", i, j);

i = i + 2;

j = j - 3;

}

}

The output will be:

---

52 - 72

54 - 69

56 - 66

58 - 63

==========

==========

Program-1:

Write a program to print multiplication table of entered number. [use while-loop]

==========

Program-2:----------

Write a program to print factorial of entered number. [Use while-loop]

==========

Program-3:

Write a program to accept 20 numbers from the user and print sum of all numbers entered by the user. [use while-loop]

==========

Program-4:

Write a program to accept a number and calculate sum of all it's digits.

```
void main()
{
int r, sum=0;
long num;
printf("Enter any number : ");
scanf("%d", &num);
while(num>0)
{r = num % 10;
sum = sum + r;
num = num / 10;
}
printf("Resulting value is : %d", sum);
}
```

**** IMP ****

- This logic will find the sum of all digits; irrespective of numbers of digits.

==========

=

Program-5:

Write a program to accept a number and reverse that number. [Use while-loop]

```
void main()
{
int r, sum=0;
long num;
printf("Enter any number : ");
scanf("%ld", &num);
while(num>0)
{
r = num % 10;
sum = (sum*10) + r;
num = num / 10;}
printf("Resulting value is : %d", sum);
}
```

==========

=

5-7-21(02)

Program-1:

Write a program to print all the numbers from 'm' to 'n'. Make sure that it prints in ASCENDING order.

```
void main()
{
int m,n,i;
printf("Enter value of m : ");
scanf("%d", &m);
```

```
printf("Enter value of n : ");
scanf("%d", &n);
if(m<n)
{
for(i=m; i<=n; i++)
{
printf("%d\n", i);
}
}if(n<m)
{
for(i=n; i<=m; i++)
{
printf("%d\n", i);
}
}
}
```

==========

==

==========

==

* The general syntax of for-loop:

```
for(initialization; condition; incr/decr)
{
statements;
statements;
.....
}
```

* The general syntax of while-loop:

```
initialization;
while(condition)
{
```

statements;

.....

.....

incr/decr;

}==========
=

=

==========
=

Program-2:

Write a program to find the result of following expression using loop:

1 2 3 8

------ + ----- + ----- + ... + -----

(1*1) (2*2) (3*3) (8*8)

```
void main()
{
int i;
float sum = 0.0, temp;
for(i=1; i<=8; i++)
{
temp = i / (float)(i*i);
sum = sum + temp;
}
printf("Sum of series is : %f", sum);
}
```

===== OR =====

```
void main(){
float sum = 0.0, temp, i;
for(i=1.0; i<=8.0; i++)
{
```

```
temp = i / (i*i);
sum = sum + temp;
}
printf("Sum of series is : %f", sum);
}
```

==========

==

==========

==

Program-3:

Write a program to find the result of following expression using loop:

```
 1 2 3 8
------ + ----- + ----- + ..... + -----
(8*8) (7*7) (6*6) (1*1)
void main()
{
float i, j, temp, sum=0.0;
j=8.0;
for(i=1.0; i<=8.0; i++){
temp = i / (j*j);
sum = sum + temp;
j--;
}
printf("Result of series is : %f", sum);
}
```

====== OR =====

```
void main()
{
float i, j, temp, sum=0.0;
for(i=1.0, j=8.0; i<=8.0; i++, j--)
```

```
{
temp = i / (j*j);
sum = sum + temp;
}
printf("Result of series is : %f", sum);
}
```

==========

==

==========

==

Homework:

CHALLANGE:----------

Program-1:

Write a program to accept 20 numbers from the user and print the largest number entered by the user. [use any loop]

```
void main()
{
int i, num, lar;
i=1;
printf("Enter 20 numbers\n");
while(i<=20)
{
scanf("%d", &num);
.....
.....
i++;
}
printf("Largest number is : %d", lar);
}
```

**CHALLANGE:

Program-2:

Write a program to accept one number (of any digits) from the user and print the largest digit of it.

[Prefer while-loop]input ---> 7293

output ---> largest digit is 9

input ---> 215

output ---> largest digit is 5

==========

===

==========

===

6-7-21(01)

Program-1:

Write a program to calculate overtime pay for 10 employees. Overtime is paid at the rate of Rs. 12 per hour for every hour worked above 40 hours.

```
#include<stdio.h>
void main()
{
int i, wh, pay, extra;
i = 1;
while(i<=10)
{
printf("\n\nEnter worked hours of employee %d\n", i);
scanf("%d", &wh);if(wh>40)
{
extra = wh - 40;
pay = extra * 12;
printf("This Employee has worked extra %d hours", extra);
```

```
printf("\nOvertime payment is Rs.%d", pay);
}
else
{
printf("No extra hours. No overtime pay\n");
}
i++;
}
}
```

==========

===

Program-2:

Write a program to accept values of BASE and POWER and calculate the result of BASE raised to POWER.

```
void main()
{
int b, p, res=1, i;
printf("Enter value of BASE : ");
scanf("%d", &b);
printf("Enter value of POWER : ");scanf("%d", &p);
for(i=1; i<=p; i++)
{
res = res * b;
}
printf("Result is : %d", res);
}
```

* BASE raised to POWER ---> BASE ka multiplication POWER times.

==========

==

Program-3:

Write a program to calculate the result of following expression. [Use while-loop]

(x-1) (x-2) (x-3) (x-8)

------- + ------- + ------- + ... + -------

(8-1) (7-2) (6-3) (1-8)

Accept the value of 'x' from the user.

```
void main()
{
float x, i, j, sum=0, temp
printf("Enter value of x : ");
scanf("%f", &x);i=1;
j=8;
while(i<=8)
{
temp = (x-i) / (j-i);
sum = sum + temp;
j--;
i++;
}
printf("Result of expression is : %f", sum);
}
```

==========

==

Program-4:

Write a program to find the result of following expresion.

1 2 3 n

----- + ----- + ----- + ... + -----

1! 2! 3! n!

Accept the value of 'n' from the user.

==========

=Program-5:

Write a program to accept 10 alphabets from the user and print the swap-case of each entered alphabet, immediately. [use while-loop and multiple if statements] [Don't use any additional variable]

```
void main()
{
int i=1;
char var;
while(i<=10)
{
fflush(stdin);
printf("\n\nEnter alphabet : ");
scanf("%c", &var);
.......
.......
i++;
}
}
```

==========

6-7-21(02)

* Similar to nested if-statements and nested conditional operator, we can also prepare nested loops.

* In case of nested loops, we write logical/executable statements in inner-loop; whereas, the outer loop is to conduct the iterations of inner-loop.* For example:

```
void main()
{
int i, j;
for(i=1; i<=3; i++)
{
```

```
for(j=1; j<=3; j++)
{
printf("%d - %d", i, j);
}
printf("ABCD");
}
}
```

The output will be:

```
----------
1 - 1
1 - 2
1 - 3
ABCD
2 - 1
2 - 2
2 - 3
ABCD
3 - 1
3 - 2
3 - 3
ABCD
```

==========

* What will be the output of following program?

```
void main()
{
int i, j;
for(i=1; i<=3; i++)
{
printf("# # #");
for(j=1; j<=2; j++)
```

```
{
printf("%d - %d", i, j);
}
printf(" * * * ");
}
}
```

The output will be:

```
--------------------
# # #
1 - 1
1 - 2
* * *
# # #
2 - 1
2 - 2
* * *
# # #
3 - 1
3 - 2* * *
```

==========

==========

* What will be the output of following program?

```
void main()
{
int i,j,k;
for(i=1; i<=3; i++)
{
printf("# # #");
for(j=1; j<=2; j++)
{
printf("* * *");
```

```
for(k=1; k<=1; k++)
{
printf("%d - %d - %d", i, j, k);
}
printf("$ $ $");
}
printf("& & &");
}
}
```

The output will be:

```
--------------------
# # #
* * *
1 - 1 - 1$ $ $
* * *
1 - 2 - 1
$ $ $
& & &
# # #
* * *
2 - 1 - 1
$ $ $
* * *
2 - 2 - 1
$ $ $
& & &
# # #
* * *
3 - 1 - 1
$ $ $
* * *
```

3 - 2 - 1

$ $ $

& & &

==========

==========

7-7-21(01)

* What will be the output of following program?

```
void main()
{int i,j,k;
for(i=1; i<=3; i++)
{
printf("# # #\n");
for(j=1; j<=2; j++)
{
printf(" * ");
}
printf("\n");
printf("$ $ $\n");
for(k=1; k<=i; k++)
{
printf(" @ ");
}
printf("\n");
printf("# # #\n");
}
}
```

The output will be:

# # #

* *

$ $ $

@

# # #

# # #

* *

$ $ $

@ @# # #

# # #

* *

$ $ $

@ @ @

# # #

==========

* What will be the output of following program?

```
void main()
{
int i,j;
for(i=1; i<=5; i++)
{
for(j=1; j<=5; j++)
{
printf(" * ");
}
printf("\n");
}
}
```

The output will be:

---

* * * * *

* * * * *

* * * * *

* * * * *

* * * * *==========

* What will be the output of following program?

```
void main()
{
int i,j;
for(i=1; i<=5; i++)
{
for(j=1; j<=i; j++)
{
printf(" * ");
}
printf("\n");
}
}
```

The output will be:

```
*
* *
* * *
* * * *
* * * * *
```

==========

Homework:

Write a program to obtain following output? [use nested loop]* * * * *

```
* * * *
* * *
* *
*
```

==========

====

7-7-21(02)

Solved Homework:

Program-1:

Write a program to obtain following output? [use nested loop]

* * * * *

* * * *

* * *

* *

*

```
#include<stdio.h>
void main()
{
int i,j;for(i=1; i<=5; i++)
{
for(j=5; j>=i; j--)
{
printf(" * ");
}
printf("\n");
}
}
```

===== OR =====

```
#include<stdio.h>
void main()
{
int i,j;
for(i=5; i>=1; i--)
{
for(j=1; j<=i; j++)
```

```
{
printf(" * ");
}
printf("\n");
}
}
```

===== OR =====

```
void main()
{printf("* * * * *\n");
printf("* * * *\n");
printf("* * *\n");
printf("* *\n");
printf("*\n");
}
```

==========

* Convert the following nested for-loop into nested while-loops.

```
#include<stdio.h>
void main()
{
int i,j;
for(i=1; i<=5; i++)
{
for(j=5; j>=i; j--)
{
printf(" * ");
}
printf("\n");
}
}
```

CONVERSION:

```
void main()
{
int i,j;
i=1;while(i<=5)
{
j=5;
while(j>=i)
{
printf(" * ");
j--;
}
printf("\n");
i++;
}
}
```

==========

=

Program-2:

Write a program to find the result of following expresion.

1 2 3 n

----- + ----- + ----- + ... + -----

1! 2! 3! n!

Accept the value of 'n' from the user.

```
void main()
{
float i, f, j, sum=0, temp;
int n;printf("Enter value of n : ");
scanf("%d", &n);
i=1;
while(i<=n)
```

```
{
// calculating factorial of value of 'i'
f=1;
for(j=1; j<=i; j++)
{
f = f * j;
}
temp = i / f;
sum = sum + temp;
i++;
}
printf("Sum of expression is : %f", sum);
}
```

====== OR =======

```
void main()
{
float i, f=1, j, sum=0, temp;
int n;
printf("Enter value of n : ");
scanf("%d", &n);
i=1;while(i<=n)
{
f = f * i;
temp = i / f;
sum = sum + temp;
i++;
}
printf("Sum of expression is : %f", sum);
}
```

==========

=

SOLVED CHALLANGE:

Program-3:

Write a program to accept 20 numbers from the user and print the largest number entered by the user. [use any loop]

```
void main()
{
int i, num, lar;
i=1;
printf("Enter 20 numbers\n");
scanf("%d", &num);
lar = num;
while(i<=19)
{
scanf("%d", &num);if(num>lar)
{
lar = num;
}
i++;
}
printf("Largest number is : %d", lar);
}
```

==========

=

**SOLVED CHALLANGE:

Program-4:

Write a program to accept one number (of any digits) from the user and print the largest digit of it. [Prefer while-loop]

input ---> 7293

output ---> largest digit is 9

input ---> 215

output ---> largest digit is 5

```
void main()
{
int num, r, lar_digit=0;
printf("Enter any number : ");
scanf("%d", &num);while(num>0)
{
r = num % 10;
if(r > lar_digit)
{
lar_digit = r;
}
num = num / 10;
}
printf("Largest digit is : %d", lar_digit);
}
```

==========

===

==========

===

CHALLANGE:

Conver the following program into nested while-loop.

```
void main()
{
int i,j,k;
for(i=1; i<=3; i++)
{
```

```
printf("# # #");
for(j=1; j<=2; j++)
{
printf("* * *");for(k=1; k<=1; k++)
{
printf("%d - %d - %d", i, j, k);
}
printf("$ $ $");
}
printf("& & &");
}
}
```

==========

==

Program-5:

Write a program to obtain following output.

```
* * * # # #
* * * # # #
* * * # # #
* * * # # #
* * * # # #
```

==========

=

8-7-21(01)

SOLVED CHALLANGE:----------

Conver the following program into nested while-loop.

```
void main()
{
int i,j,k;
for(i=1; i<=3; i++)
```

```
{
printf("# # #");
for(j=1; j<=2; j++)
{
printf("* * *");
for(k=1; k<=1; k++)
{
printf("%d - %d - %d", i, j, k);
}
printf("$ $ $");
}
printf("& & &");
}
}
```

===== USING WHILE LOOP =====

```
void main()
{
int i,j,k;
i=1;
while(i<=3)
{
printf("# # #");j=1;
while(j<=2)
{
printf("* * *");
k=1;
while(k<=1)
{
printf("%d - %d - %d", i, j, k);
k++;
}
```

```
j++;
}
printf("& & &");
i++;
}
}
```

==========

==

Program-1:

Write a program to obtain following output.

```
* * * # # #
* * * # # #
* * * # # #
* * * # # #
* * * # # #
void main(){
int i,j;
for(i=1; i<=5; i++)
{
for(j=1; j<=6; j++)
{
if(j<=3)
{
printf(" * ");
}
else
{
printf(" # ");
}
}
```

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

====== OR ======

```
void main()
{
int i,j,k;
for(i=1; i<=5; i++)
{
for(j=1; j<=3; j++)
{
printf(" * ");
}for(k=1; k<=3; k++)
{
printf(" # ");
}
printf("\n");
}
}
```

==========

=

* What will be the output of following program?

```
#include<stdio.h>
void main()
{
 int i = 1 , j = 1;
 for( ; ; )
 {
 if(i>5)
{
 break;
```

```
}
 else
{
 j+=i; // j = j + i;
}
 printf("%d\n", j);
 i+=j; // i = i + j;
 }
}==========================================================================
=
=
```

Program-2:

Write a program to obtain following output.

```
* # * # * #
* # * # * #
* # * # * #
* # * # * #
* # * # * #
void main()
{
int i,j;
for(i=1; i<=5; i++)
{
for(j=1; j<=6; j++)
{
if(j%2 == 0)
{
printf(" # ");
}
else
```

```
{
printf(" * ");
}
}
printf("\n");
}}
```

===== OR =====

```
void main()
{
int i,j;
for(i=1; i<=5; i++)
{
for(j=1; j<=3; j++)
{
printf(" * #");
}
printf("\n");
}
}
```

==========

==

Program-3:

Write a program to obtain following output?

A B C D E

A B C D E

A B C D E

A B C D E

A B C D E

==========

CHALLANGE

Program-4:

Write a program to obtain following output?

A B C D E

E D C B A

A B C D E

E D C B A

A B C D E

E D C B A

[ASCII VALUES ---> A to E [65 to 69]

==========

==

** CHALLANGE

Program-5:

Write a program to obtain following output?

A B C D E

F G H I J

A B C D E

F G H I J

A B C D E

F G H I J

[ASCII VALUES ---> A to E [65 to 69] and F to J [70 to 74]

==========

==8-7-21(02)

* Whenever we are given a logical situation, then we have to:

1. prepare an algorithm

2. prepare a flowchart

3. prepare a complete program

* An algorithm is "step by step re-presentation of any logical situation"

* A flowchart is "diagramatic representation of steps of algorithm"

* By referring the algorithm and flowchart, we are able to prepare/understand a program.

==========

* We can write an algorithm in general english like language.

* To prepare a flowchart, we have fixed symbols. So that every programmer can understand it and prepare an application/program in his/her preferred programming-language.

==========

9-7-21(01)

CHALLANGE

Program-4:

Write a program to obtain following output?

A B C D E

E D C B AA B C D E

E D C B A

A B C D E

E D C B A

[ASCII VALUES ---> A to E [65 to 69]

```
void main()
{
int i;
char j;
for(i=1; i<=6; i++)
{
if(i%2==0)
{
for(j='E'; j>='A'; j--)
{
printf("%c ", j);
}
}
else
```

```
{
for(j='A'; j>='E'; j++)
{
printf("%c ", j);
}
}
printf("\n");
}
```

}** Whenever we perform an operation on character values, it will be ultimately performed on it's ASCII value.

** In above loops also, the variable 'j' will be working with ASCII values of assigned character.

==========

==

** CHALLANGE

Program-5:

Write a program to obtain following output?

A B C D E

F G H I J

A B C D E

F G H I J

A B C D E

F G H I J

[ASCII VALUES ---> A to E [65 to 69] and F to J [70 to 74]

```
void main()
{
int i;
char j;
for(i=1; i<=6; i++)
{
if(i%2 == 0)
```

```
{
for(j='F'; j<='J'; j++)
{printf("%c ", j);
}
}
else
{
for(j='A'; j<='E'; j++)
{
printf("%c ",j);
}
}
printf("\n");
}
}
```

====== OR ======

```
void main()
{
int i;
char j, st, en;
for(i=1; i<=6; i++)
{
if(i%2 == 0)
{
st = 'F;
en = 'J';
}
else
{
st = 'A';en = 'E';
}
```

```
for(j=st; j<=en; j++)
{
printf("%c ", j);
}
printf("\n");
}
}
```

==========

==

Program-1:

Write a program to print the nxn square pattern of *. Accept value of 'n' from the user.

For example:

---

input ---> n=5

output

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

input ---> n=7output

```
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
```

```
void main()
{
```

}

==========

=

Program-2:

Write a program to print the nxn square pattern of *. But the diagonal will be having '#'. Accept value of 'n' from the user.

For example:

---

input ---> n=5

output

* * * *

* # * * *

* * # * *

* * * # * * * #

input ---> n=7

output

* * * * * *

* # * * * * *

* * # * * * *

* * * # * * *

* * * * # * *

* * * * * # *

* * * * * * #

void main()

{

int n, i , j;

printf("Enter value of n : ");

scanf("%d", &n);

for(i=1; i<=n; i++)

{

```
for(j=1; j<=n; j++)
{
if(i==j)
{
printf(" #");
}
else
{printf(" *");
}
}
printf("\n");
}
}
```

==========

=

* What will be the output of following program?

```
void main()
{
int i, j, k
for(i=1; i<=5; i++)
{
for(j=5; j>i; j--)
{
printf("__"); <--- is space
}
for(k=1; k<=i; k++)
{
printf("* ");
}
printf("\n");
}
```

}The output will be:

---

*

* *

* * *

* * * *

* * * * *

==========

Program-3:

Write a program to obtain following output.

* * * * *

* * * *

* * *

* *

*

```
void main()
{
int i,j,k;
for(i ....)
{
for(j....)
{
printf("__");
}for(k...)
{
printf("* ");
}
printf("\n");
}
}
```

==========

=

9-7-21(02)

Prepare a flowchart from following program:

```
void main()
{
int i=52; j=72, k;
while(i<=70)
{
printf("%d - %d", i, j)
i = i + 2;
j = j - 3;
}
k = i+j;
printf("Final addition is : %d", k);
}
```

==========

=

====

==========

====

```
void main()
{
int i=1, j;
while(i<=5)
{
printf("Start : %d", i);
if(i==3)
{
break;
}
```

```
printf("End : %d", i);
i++;
}
j = i*i;
printf("Final result is : %d", j);
}
```

==========

==========

Function is ----> a reusable block of statements.

It is only and only to reduce length of program.

Function, array, structure, pointer, union ----> all are like facility to programmerto reduce length of program and reduce

extra programming efforts.

decision making statements, loops, ----> all are for logical reasons. We can prepare

break, continue, flowchart for these entities.

==========

==========

10-7-21(01)

Program-1:

Write a program to obtain following output.

```
* * * * *
 * * * *
 * * *
 * *
 *
```

```
void main()
```

```
{
int i,j,k;
for(i=5; i>=1; i--)
{
for(j=5; j>i; j--)
{printf("__");
}
for(k=1; k<=i; k++)
{
printf("* ");
}
printf("\n");
}
}
```

==========

=

* What will be the output of following program?

```
void main()
{
int i,j;
for(i=1; i<=5; i++)
{
for(j=1; j<=i; j++)
{
printf("%d ", j);
}
printf("\n");
}
}
```

The output will be:

----------1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

==========

=

==========

=

What will be the output of following program?

```
void main()
{
int i,j;
for(i=1; i<=5; i++)
{
for(j=1; j<=i; j++)
{
printf("%d ", i);
}
printf("\n");
}
}
```

The output will be:

1

2 23 3 3

4 4 4 4

5 5 5 5 5

==========

==

==========

==

Program-2:

Write a program to obtain following output.

```
 *
 * * *
 * * * * *
 * * * * * * *
* * * * * * * * *
```

```
void main()
{
int i, j, k, temp=1;
for(i=1; i<=5; i++)
{
for(j=5; j>i; j--)
{
printf(" ");
}
for(k=1; k<=temp; k++)
{printf("* ");
}
temp = temp + 2;
printf("\n");
}
}
```

===== OR =====

```
void main()
{
int i,j,k,l;
for(i=1; i<=5; i++)
{
for(j=5; j>i; j--)
```

```
{
printf(" ");
}
for(k=1; k<=i; k++)
{
printf("* ");
}
for(l=1; l<i;l++)
{
printf("* ");
}
printf("\n");
}}
```

==========

==

CHALLANGE:

Program-3:

Write a program to obtain following output.

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

[hint: Use another 'temp' variable and print it's value.]

```
void main()
{
int i,j, temp=1;
for(i=1; i<=5; i++)
{
```

```
for(j=1; j<=i; j++)
{
printf("%d\t", temp);
temp++;
}
printf("\n");}
}
```

==========

Homework:

Write a program to obtain following output.

```
*
# *
# * #
* # * #
* # * # *
```

==========

===

10-7-21(02)

Solved Homework:

Write a program to obtain following output.

```
*
# *
# * #
* # * #
* # * # *void main()
{
int i,j, temp=1;
for(i=1; i<=5; i++)
{
```

```
for(j=1; j<=i; j++)
{
if(temp%2 == 0)
{
printf("# ");
}
else
{
printf("* ");
}
temp++;
}
printf("\n");
}
}
```

==========

===

** CHALLANGE^CHALLANGE:

Program-2:

Write a program to print pyramid pattern for 'n' rows. Accept 'n' from the user.For example:

---

input ---> n=4

output

```
 *
 * * *
 * * * * *
* * * * * * *
```

input ---> n=7

output

```
 *
 * * *
 * * * * *
 * * * * * * *
 * * * * * * * * *
 * * * * * * * * * * *
* * * * * * * * * * * * *
```

input ---> n=9

```
 *
 * * *
 * * * * *
 * * * * * * *
 * * * * * * * * * * * * * * * * * * * *
 * * * * * * * * * * * * *
 * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * *
```

==========

=

==========

=

Program-3: (@common)

Write a program to obtain following output.

A B C D E F G H I

A B C D F G H I

A B C G H I

A B H I

A I

==========

=

==========

=

Program-4: [TCS]

Write a program to create xmas tree pattern for n-layers. Accept 'n' from the user.

input ---> n=4

output --->

```
 * * * *
 * * * * *
* * * * * * *
 *
 * * *
 * * * * *
 *
 * * *
 *
```

==== OR ====

```
 * < 1 layer
 *
 * * * <---- 2 layer
 *
 * * *
 * * * * * <---- 3 layer
 *
 * * *
 * * * * *
* * * * * * * <--- 4 layer
```

input ---> n=7

output --->

1 layer

2 layers

3 layers4 layers

5 layers

6 layers

7 layers

==========

===

==========

===

11-7-21(01)

---- do while loop ----

==========

* We are familiar with for-loop and while-loop.

* And we know that, every loop has same purpose and application.

* And we also know that, different loop is preferred in different logical situations.

* We prefer for loop ---> when numbers of iterations are already known.

* The general syntax of for-loop:

```
for(initialization; condition; incr/decr)
{
statements;
statements;
.....
.....
}
```

* We prefer while loop ---> when numbers of iterations are dependent on any internal situation or internal value.

* The general syntax of while-loop:initialization;

```
while(condition)
{
statements;
statements;
.....
```

.....

incr/decr;

}

* We prefer do-while loop ---> when at-least one iteration is required.

* The general syntax of do-while loop:

initialization;

do

{

statements;

statements;

.....

incr/decr;

}while(condition);

* The execution of do-while loop is similar to other two loops.

* The do-while loop also iterates unless condition becomes false.

For example: ---> A program to print all the numbers from 1 to 10.

```
void main(){
int i=1;
do
{
printf("%d\n", i);
i++;
}while(i<=10);
}
```

==========

* Note that, the do-while loop at-least iterates once; even if the initial condition is false.

* Following are some of the important differences between while-loop and do-while loop:

while-loop	do-while loop
1. is also called ENTRY CONTROLLED loop.	1. is also called as EXIT CONTROLLED loop

2. the condition is being checked to decide 2. the condition is being checked to
whether current iteration is possible or decide whether next iteration is
not. possible or not.
3. if initial condition is FALSE, then 3. if initial condition is FALSE, then
while-loop will not iterate. also, do-while loop iterates at-least
once.
4. There is no semi-colon at the end of 4. There is semi-colon at the end of
while-condition. while-condition.
5. We prefer while-loop when numbers of 5. We prefer do-while loop when at-least
iterations depend on any internal value one iteration is must.
or internal situation.
6. In while-loop, first condition is being 6. In do-while loop, first statements
checked and then statements get executed. gets executed and then condition is being checked.
7. The general syntax: 6. The general syntax:
initialization; initialization;
while(condition) do
{ {
statements; statements;
statements; statements;
……. ……
incr/decr; incr/decr;
} }while(condition);
==========
====
12-7-21(01)
* In a loop, actually everything is optional, but everything is logically required.
* In loops, we do:
initialization ---> so that the loop counter starts the iteration from expected
value only. If not initialized/assigned then it contains
garbage value and may iterate unexpectedly.
incr/decr ----> so that the value of loop-counter changes and it reaches to

condition. If incr/decr is not applied then the value of loop counter will not update and may iterate infinite.

condition ---> so that the loop terminates.

==========

==========

==

==

* The general syntax of for-loop:

for(initiailization; condition; incr/decr)

{

statements;

statements;

.....

}

* The general syntax of while-loop:

initialization;

while(condition)

{

statements;

statements;

.....

incr/decr;

}

* The general syntax of do-while loop:

initialization;

do{

statements;

statements;

.....

incr/decr;

}while(condition);

==========

==

==========

==

Program-1:

Write a program to print all the numbers from 1 to 10 using do-while loop.

```
#include<stdio.h>
void main()
{
int i=1;
do
{
printf("%d\n", i);
i++;
}while(i<=10);
}
```

==========

Program-2:

----------Write a program to print all the even numbers from 1 to 100. [use do-while loop]

```
void main()
{
int i=1;
do
{
if(i%2 == 0)
{
printf("%d\n", i);
}
i++;
}while(i<=100);
}
```

==========

Program-3:

Write a program to accept a number from the user and print it's factorial. [use do-while]

[multiplication from 1 to num]

```
void main()
{
int i=1, num;
long f=1;
printf("Enter a number : ");
scanf("%d", &num);
do{
f = f * i;
i++;
}while(i<=num);
printf("Factorial is : %ld", f);
}
```

==========

Program-4:

Write a program to accept a number and print it's multiplication table. [use do-while]

```
void main()
{
int i=1, num, m;
printf("Enter a number : ");
scanf("%d", &num);
do
{
m = num * i;
printf("%d x %d = %d\n", num, i, m);
i++;
```

```
}while(i<=10);
}
```

==========Program-5:

Write a program to accept the numbers till user wants; and for every entered number check whether it is even or odd. [this is a user-driven program] [make first iteration as FREE iteration, and then ask end-user for next iteration]

```
#include<stdio.h>
void main()
{
int num;
char u_choice;
do
{
printf("\n\nEnter a number : ");
scanf("%d", &num);
if(num%2 == 0)
{
printf("is EVEN");
}
else
{
printf("is ODD");
}
fflush(stdin);
printf("\nDo you want to check another number (y/n) : ");
scanf("%c", &u_choice);
}while(u_choice=='y' || u_choice=='Y');
}
```

==========

Program-6:

Write a program to accept a number till user wants and at the end print the sum of all entered numbers.

```
void main()
{
int num, sum=0;
char u_choice;
do
{
printf("\n\nEnter a number : ");
scanf("%d", &num);
sum = sum + num;
fflush(stdin);
printf("Do you .... ");
scanf("%c", &u_choice);
}while(u_choice=='y' || u_choice=='Y' || u_choice=='1');
printf("\n\nSum of all entered numbers is : %d", sum);
}
```

==========

**** General skeleton of a user-driven program ****

```
----------do
{
// perform any operation here using if, if-else, loop, switch-case, etc
.....
.....
fflush(stdin);
printf("Do you want to test another number (y/n) : ");
scanf("%c", &u_choice);
}while(u_choice=='y' || u_choice=='Y');
```

==========

12-7-21(02)

---- Programmatic skeleton of a user-driven program ----

==========

```
do
{
        .... any logic here
        if, if-else, loop, switch-case, I/O, expression, etc...



        fflush(stdin);
        printf("Do you ..... ");
        scanf("%c", &u_choice);
}while(u_choice=='y' || u_choice=='Y');
```

==========

==========

Program-1:

Write a program to accept the numbers till user wants and at the end print the total count of even numbers and odd numbers entered by the user.

```
#include<stdio.h>
void main()
{
        int num;
        int even_count=0, odd_count=0, total_count=0, sum=0;
        char u_choice;
```

```
	do
	{
		printf("\n\nEnter number : ");
		scanf("%d", &num);
		total_count++;
		sum = sum + num;

		if(num%2 == 0)
		{
			even_count++;
		}
		else
		{
			odd_count++;
		}

		fflush(stdin);
		printf("Do you want to enter another number (y/n) : ");
		scanf("%c", &u_choice);
	}while(u_choice=='y' || u_choice=='Y');

	printf("\n\nTotal numbers entered : %d", total_count);
	printf("\nTotal count of even numbers : %d", even_count);
	printf("\nTotal count of odd numbers : %d", odd_count);
	printf("\nSum of all entered numbers : %d", sum);

}
```

===== OR ===== [same program by using while-loop]

```
void main()
```

```
{
	int num, even_count=0, odd_count=0, total_count=0, sum=0;
	char u_choice='y';

	while(u_choice=='y' || u_choice=='Y')  <--- is forcefully TRUE
	{
		printf("\n\nEnter number : ");
		scanf("%d", &num);
		total_count++;
		sum = sum + num;

		if(num%2 == 0)
		{
			even_count++;
		}
		else
		{
			odd_count++;
		}

		fflush(stdin);
		printf("Do you .... ");
		scanf("%c", &u_choice);
	}

	printf("\n\nTotal numbers entered : %d", total_count);
	printf("\nTotal count of even numbers : %d", even_count);
	printf("\nTotal count of odd numbers : %d", odd_count);
	printf("\nSum of all entered numbers : %d", sum);
}
```

===== OR ===== [same program by using for-loop]

```
char u_choice;
for(u_choice='y'; u_choice=='y' || u_choice=='Y' ; )
{
        .....
        .....

        fflush(stdin);
        printf("Do you .... ");
        scanf("%c", &u_choice);
}
```

==========

==========

Program-2:

Write a program to accept the numbers till user wants and print factorial of each number.

```
void main()
{
        int num, i;
        long f;
        char  u_choice;

        do
        {
                printf("Enter a number : ");
```

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

                f=1;
                for(i=1; i<=num; i++)
                {
                        f = f *i;
                }
                printf("Factorial is : %ld", f);

                fflush(stdin);
                printf("Do you ..... : ");
                scanf("%c", &u_choice);
        }while(u_choice=='y' || u_choice=='Y');
}
```

==========

==========

13-7-21(1)

* The general skeleton of using do-while loop for user-driven programs is:

```
do
{
        .....   // use any logic here
        .....

        fflush(stdin);
        printf("Do you .... ");
        scanf("%c", &u_choice);
}while(u_choice=='y' || u_choice=='Y');
```

==========

==========

Program-1:

Write a program to solve following expression. [use any loop]

$$\frac{(x-1)}{(1^1)} + \frac{(x-2)}{(2^2)} + \frac{(x-3)}{(3^3)} + \ldots + \frac{(x-n)}{(n^n)}$$

* Accept the value of 'X' and 'n' from the user.

```
void main()
{
	float x, n, i, j;
	float temp, b, p, res, sum=0;

	printf("Enter value of n : ");
	scanf("%f", &n);

	printf("Enter value of x : ");
	scanf("%f", &x);

	i=1;
	while(i<=n)
	{
		b=i;
		p=i;
		res=1;
```

```
                // calculating B^P
                for(j=1; j<=p; j++)
                {
                        res = res*b;
                }

                temp = (x-i) / res;
                sum = sum + temp;
                i++;
        }

        printf("Result of expression is : %f", sum);

}
```

==========

==========

Program-2:

Write a user-driven program to perform arithmetic operation on two interger values based on user's choice. [similar to a menu-driven program] [using multiple if-statements]

```
#include<stdio.h>

void main()
{
        char u_choice;
        int n1,n2, c, o_choice;
```

```
float d;

do
{
	printf("\n\nEnter 2 numbers\n");
	scanf("%d%d", &n1, &n2);

	printf("Which arith. operation you want to perform on these numbers\n");
	printf("1 - ADDITION\n");
	printf("2 - SUBTRACTION\n");
	printf("3 - MULTIPLICATION\n");
	printf("4 - DIVISION\n");
	printf("Provide your choice : ");
	scanf("%d", &o_choice);

	if(o_choice==1)
	{
		c = n1 + n2;
		printf("Addition is : %d", c);
	}
	else if(o_choice==2)
	{
		c = n1 - n2;
		printf("Subtraction is : %d", c);
	}
	else if(o_choice==3)
	{
		c = n1*n2;
		printf("Multiplication is : %d", c);
	}
	else if(o_choice==4)
```

```
                {
                        d = n1 / (float)n2;
                        printf("Division is : %f", d);
                }

                fflush(stdin);
                printf("\nDo you want to perform another operation (y/n) ? : ");
                scanf("%c", &u_choice);
        }while(u_choice=='y' || u_choice=='Y');

}
```

==========

==========

Program-3:

Write a program to accept the numbers till user wants and at the end print the average value of entered numbers. [average = sum_of_all_numbers / total_entered_numbers]

```
void main()
{
        int num, sum=0, total_nums=0;
        float avg;
        char u_choice;

        do
        {
                printf("\n\nEnter a number : ");
                scanf("%d", &num);
```

```
                sum = sum + num;
                total_nums++;

                fflush(stdin);
                printf("Do you want to enter another number (y/n) ? : );
                scanf("%c", &u_choice);
        }while(....);

        avg = sum / (float)total_nums;
        printf("Average value of entered numbers is : %f", avg);
}
```

==========

***** CHALLANGE *****

Write a program to obtain following output.

```
     *
    * * *
   * * * * *
  * * * * * * *
 * * * * * * * * *
  * * * * * * *
   * * * * *
    * * *
     *
```

==========

==========
=

13-7-21(2)

***** CHALLANGE *****

Write a program to obtain following output.

```
     *
    * *
   *   *
  *     *
 *       *
*         *
 *       *
  *     *
   * *
    *
```

==========
==

==========
==

* What will be the output of following programs?

a)

```
void main( )
{
        int j ;
```

```
        while ( j <= 10 )
        {
                printf ( "\n%d", j ) ;
                j = j + 1 ;
        }
}
```

Output:

based on garbage value of 'j'

==========

b)

```
void main( )
{
        int i = 1 ;
        while ( i <= 10 ) ;
        {
                printf ( "\n%d", i );
                i++;
        }
}
```

Explanation:

If we put semi-colon at the end of while-loop, then it considers that semi-colon (;) as part of loop's body. And the written statement's of loop's body will not be considered as part of loop's body.

Therefore above program will perform infinite iterations. Because condition is remaining TRUE always.

==========

c)

```
void main( )
{
        int j ;
        while (j <= 10)
        {
                printf ( "\n%d", j ) ;
                j = j + 1 ;
        }
}
```

Output:

Based on garbage value of 'j'

==========

d)

```
void main( )
{
        int x = 1 ;
        while (x == 1)
        {
                x = x - 1;
                printf ("\n%d", x);
        }
}
```

Output:

0

==========

e)

```
void main( )
{
        int x = 1;
        while(x==1)
        x = x - 1;
        printf( "\n%d", x);
}
```

Output:

0

==========

f)

```
void main( )
{
        char x;
        while(x=0; x<=255; x++)
        printf ("\nASCII value %d Character %c", x, x);
}
```

Output:

This program will show CT error. This is not general syntax of while-loop.

==========

g)

```
void main( )
{
```

```
        int x = 4, y, z ;
        y = --x;
        z = x--;
        printf ("%d %d %d", x, y, z);
}
```

Output:

x = 2

y = 3

z = 3

==========

h)

```
void main()
{
        int x = 4, y = 3, z ;
        z = x-- -y ;
        printf ( "\n%d %d %d", x, y, z ) ;
}
```

Output:

x = 3

y = 3

z = 1

==========

i)

```
void main()
{
```

```
        while ('a' < 'b')
        printf ( "\nmalyalam is a palindrome" ) ;
}
```

Output:

infinite iterations

==========

j)

```
void main( )
{
        int i=10;
        while(i=20)
        printf ("\nA computer buff!");
}
```

Output:

infinite iterations

Explanation:

If you find an assignment statement in if or loop's condition then:

- that value gets assigned to that variable
- that value of variable is considered as result of condition (zero/non-zero)

* Here also, 20 gets assigned to 'i'. And that value of 'i' is considered as result of condition. That value is 20. Therefore result of condition is non-zero. Therefore result of condition is always TRUE.

==========

k)

```
void main()
{
	int a=-3,b=6,c;
	while(c = a+b)
	{
		printf("%d", c);
		a++;
		b = b - 2;
	}
}
```

Output:

3

2

1

==========

14-7-21(1)

* What will be the output of following programs?

k)

```
void main( )
{
	int i ;
	while (i = 10)
	{
		printf("\n%d", i);
		i = i + 1;
	}
```

```
}
```

Output:

infinite times 10

==========

l)

```
void main( )
{
        float x = 1.1;
        while(x == 1.1)
        {
                printf ("\n%f", x);
                x = x – 0.1;
        }
}
```

Output:

1.1

==========

m)

```
void main( )
{
        while ('1' < '2')
        printf ("\nIn while loop");
}
```

Output:

infinite times : "In while loop"

==========

n)

```
void main( )
{
        char x;
        for(x = 0; x <= 255; x++)
        printf ("\nASCII value %d Character %c", x, x);
}
```

Output:

Will print ASCII value and ASCII character from 0 to 255

If we use %d (in printf) for character variable, then it will print the ASCII value (int form) of that character value.

==========

o)

```
void main( )
{
        int x = 4, y = 0, z;
        while (x >= 0)
        {
                x--;
                y++;
                if(x == y)
                        continue;
                else
```

```
                printf ("\n %d  %d", x, y);
        }
}
```

Output:

3 1

1 3

0 4

-1 5

==========

p)

```
void main( )
{
        int x = 4, y = 0, z;
        while (x >= 0)
        {
                x--;
                y++;
                if(x == y)
                break;
                else
                printf ("\n %d  %d", x, y);
        }
}
```

Output:

3 1

==========

q)

```
void main( )
{
        int x = 4, y = 0, z;
        while(x >= 0)
        {
                if (x == y)
                        break;
                else
                        printf("\n%d %d", x, y );
                x--;
                y++;
        }
}
```

Output:

4 0

3 1

==========

==========

Homework:

Write a program for a matchstick game being played between the computer and a user. Your program should ensure that the computer always wins. Rules for the game are as follows:

- There are 21 matchsticks.
- The computer asks the player to pick 1, 2, 3, or 4 matchsticks.
- After the person picks, the computer does its picking.
- Whoever is forced to pick up the last matchstick loses the game.

Program-1:

Write a program to accept a number from the user and print it's binary equivalent.

```
void main()
{
	int num, r;

	printf("Enter a number : ");
	scanf("%d", &num);

	while(num>0)
	{
		r = num % 2;
		printf("%d ", r);
		num = num / 2;
	}
}
```

Program-1:

Write a program to accept a number from the user and print it's HexaDecimal equivalent.

```
void main()
{
	int num, r;

	printf("Enter a number : ");
	scanf("%d", &num);

	while(num>0)
```

```
{
        r = num % 16;
        if(r<=9)
        {
                printf("%d", r);
        }
        else if(r==10)
        {
                printf("A");
        }
        else if(r==11)
        {
                printf("B");
        }
        else if(r==12)
        {
                printf("C");
        }
        else if(r==13)
        {
                printf("D");
        }
        else if(r==14)
        {
                printf("E");
        }
        else if(r==15)
        {
                printf("F");
        }
        num = num / 16;
```

```
        }
}
```

===== OR =====

```
void main()
{
        int num, r;
        char var;

        printf("Enter a number : ");
        scanf("%d", &num);

        while(num>0)
        {
                r = num % 16;
                if(r<=9)
                {
                        printf("%d", r);
                }
                else
                {
                        var = 55 + r;
                        printf("%c", var);
                }
                num = num / 16;
        }
}
```

==========

==========

14-7-21(2)

* What will be the output of following programs?

a)

```
void main( )
{
        int i = 0 ;
        for ( ; i; )
        printf ( "\nHere is some mail for you" ) ;
}
```

Output:

No iteration. Blank output.

==========

b)

```
void main( )
{
        int i;
        for (i = 1; i <= 5; printf("%d", i))
        {
                i++;
        }
}
```

Output:

2

3

4

5

6

Explanation:

In for-loop the IMPORTANT is PLACE. Not statement.

==========

c)

```
void main( )
{
        int i = 1, j = 1 ;
        for ( ; ; )
        {
                if(i > 5)
                break;
                else
                j += i;
                printf("\n%d", j);
                i += j;
        }
}
```

Output:

2

5

==========

d)

```
void main( )
{
        int i;
        for(i = 1; i <= 5; printf("\n%c", 65 ))
        i++;
}
```

Output:

A

A

A

A

A

---

e)

```
void main()
{
        int i;
        for(i=1; i<=5; i++);
        {
                printf("Hello - %d\n", i);
        }
}
```

Output:

Hello - 6

Remember that, if we apply semi-colon:

after if-condition ---> considers as statment (always true)

block of statement executes even if condition is FALSE

after else ----> considers as statement

block of statement executes even if condition is TRUE

after while-condition ----> considers a blank body of while-loop

the statments of while-loop are considered as

open/general statements of main().

after for-loop ---> considers a blank body of for-loop

the statements of for-loop are considered as open/general

statements of main()

==========

15-7-21(1)

----- switch case ----- OR ---- Menu Driven Statement ----

========== ==========

* We prefer to use switch-case for menu-driven programs.

* Menu-Driven program, means we provide a list/menu of available/possible operations. And the end-user has to select/choose the expected operation to perform.

* The general syntax of using switch-case is:

```
switch(switch_variable)
{
```

```
        case 1: statements;
                .....
                break;
        case 2: statements;
                break;
        .
        .
        .
        case n: statements;
                break;
        default: statements;
                 break;
}
```

* In above general syntax there are 4 keywords used: switch, case, break, default.

* The 'case-number' which matches with the 'switch-variable' will be executed. And if no case number matches with the 'switch-variable' then 'default' case executes.

* Similar to loop, we use 'break' to come-out of switch-case.

* If 'break' is not provided then next written case automatically executes. This will continue unless a break appears or switch-case ends. This process is called as "fall down".

* The use of "continue" is not allowed in switch-case. If we use "continue" in switch-case then it will show CT error ---> "misplaced continue"

Refer following program.

```
#include<stdio.h>

void main()
{
```

```
int o_choice;
int a,b,c;
float d;

printf("main starts");

printf("\n\nEnter 2 values\n");
scanf("%d%d", &a, &b);

printf("\nSelect an operation to perform : \n");
printf("1 - ADDITION\n");
printf("2 - SUBTRACTION\n");
printf("3 - MULTIPLICATION\n");
printf("4 - DIVISION\n");
printf("Provide your choice : ");
scanf("%d", &o_choice);

switch(o_choice)
{
        case 1: c = a + b;
        printf("Addition is : %d", c);
         break;
        case 2: c = a - b;
    printf("Subtraction is : %d", c);
         break;
        case 3: c = a * b;
    printf("Multiplication is : %d", c);
         break;
        case 4: d = a / (float)b;
    printf("Division is : %f", d);
         break;
```

```
                default: printf("Invalid Operation Choice");
            break;
        }

        printf("\n\nmain ends\n\n\n");
}
```

* From the output, note that the case-number with which the value of 'o_choice' matches will execute.

* Because of 'break' the controls come-out of switch-case.

* Officially, only one case is expected to execute in switch-case.

==========

15-7-21(2)

* We use switch-case for menu-driven programs. Where, we provide a list/menu of operations that are available/possible; and the end-user selects the operation to perform.

* The general syntax is:

```
        switch(switch-variable)
        {
                case 1: statements;
                        ....
                        break;
                case 2: statements;
                        ....
                        break;
                .
```

```
        .
        .
        case n: statements;
                .....
                break;
        default: statements;
                 .....
                 break;
}
```

* The case with which the value of 'swith-variable' matches will execute. If no case number matches with 'switch-variable' then the default case executes.

* We use 'break' to terminate a case and come out of switch-case. If 'break' is not used then next written case automatically executes. This will continue unless 'break' is found or switch-case ends. This mechanism is called as "fall down".

* Use of "continue" in swith-case is not valid. Shows CT error.

==========

* We can write any logical statement in a switch's case.

Program-1:

Write a menu-driven program for following choices.

- check whether even or odd
- check whether positive or negative or zero
- print square and cube
- print factorial

```
#include<stdio.h>

void main()
{
	int num, o_choice;
	int sq, cu, i;
	long f;


	printf("main starts\n\n");

	// step 1: accept the number
	printf("Enter a number : ");
	scanf("%d", &num);

	// step 2: show the menu and accept choice
	printf("\nWhich operation you want to perform? : \n");
	printf("1 - To check even or odd\n");
	printf("2 - To check positive or negative or zero\n");
	printf("3 - To print square and cube\n");
	printf("4 - To print factorial\n");
	printf("Provide your choice : ");
	scanf("%d", &o_choice);

	switch(o_choice)
	{
		case 1: if(num%2 == 0)
			{
				printf("is EVEN");
			}
			else
```

```
        {
                printf("is ODD");
        }
        break;

case 2: if(num>0)
        {

        }
        else if(num<0)
        {

        }
        else if(num==0)
        {

        }
        break;

case 3: sq = num * num;
        cu = num * num * num;
        printf("Square is : %d\n" ,sq);
        printf("Cube is : %d\n", cu);
        break;

case 4: f=1;
        for(i=1; i<=num; i++)
        {
                f = f * i;
        }
        printf("Factorial is : %ld", f);
```

```
                        break;

                default: printf("Invalid Choice\n");
        }

        printf("\n\nmain ends\n\n\n");

}
```

==========

Program-3:

Write a menu-driven program for following choices.

- To reverse the number
- To print last digit of number
- To print sum of all digits
- To check prime or not
- To check armstrong or not
- To check palindrome or not

==========

==========

16-7-21(1)

Solved HomeWork:

Program-3:

Write a menu-driven program for following choices.

- To reverse the number
- To print last digit of number
- To print sum of all digits
- To check prime or not
- To check armstrong or not
- To check palindrome or not

==========

==========

---- Important Points ----

==========

1. Execution of switch-case is faster than multiple if statements and else-if statements.

* because the interpreter has to check/test the condition in if-statements. And this may waste execution time of interpreter.

* But in switch-case there is nothing like checking condition and taking decision. There are direct case numbers. The interpreter has to only check the console case-numbers.

2. The case numbers can be an interger value and character values. i.e. case numbers cannot be any floating point (REAL) values and string and objects.

* The character cases must be within single inverted comma ('A').

* The character cases are case-sensitive.

For example:

```
void main()
{
```

```
	char o_choice;
	int a,b,c;
	float d;

	printf("Enter 2 values\n");
	scanf("%d%d", &a, &b);

	printf("Which operation you want to perform?\n");
	printf("A - Addition\n");
	printf("S - Subtraction\n");
	printf("M - Multiplication\n");
	printf("D - Division\n");
	printf("Provide your choice : ");
	scanf("%c", &o_choice);

	switch(o_choice)
	{
		case 'A': c = a + b;
			printf("Addition is : %d", c);
			break;
		case 'S': c = a - b;
			printf("Subtraction is : %d", c);
			break;
		.....
		.....
		default: printf("Invalid Choice");
	}
}
```

3. variables are not allowed as case numbers. [variable cases are not allowed] [The case numbers must be a console value]

For example:

```
void main()
{
	int x,y,z;

	printf("Enter value of x : ");
	scanf("%d", &x);

	printf("Enter value of y : ");
	scanf("%d", &y);

	printf("Enter value of z : ");
	scanf("%d", &z);
	.......
	.......
	switch(o_choice)
	{
		case x: operation here	<--- variable case is not allowed
			........
			break;
		case y: operation here	<--- variable case is not allowed
			........
			break;
		case z: operation here	<--- variable case is not allowed
			........
			break;
	}
}
```

4. The case numbers must be unique. [must not repeat]

* It will be CT error if any case number is getting duplicated/repeated.

5. Expressions are allowed as case number. But only console expressions are allowed. The variable expression is not allowed.

For example:

```
void main()
{
	int o_choice = ___;

	switch(o_choice)
	{
		case 2+3: statements;
			.....
			break;
		case 5-2: statements;
			.....
			break;
		case 40<<3: statements;
			 ......
			 break;
		case 10%2: statements;
			 ......
			 break;
		case a+b: statements;    <--- variable expression is not allowed
			.....
			break;
```

```
                case 5: statements;   <--- is getting duplicated with (case 2+3)
                        ......
                        break;
        }
}
```

6. The logical operators (AND, OR, NOT) are not allowed in case numbers.

For example:

```
switch(o_choice)
{
        case 'A' || 'a': statements;    <--- OR operator is not allowed in case number
                        ......
                        break;

        case 'S' && 's': statements;    <--- NOT operator is not allowed in case number
                        ......
                        break;

        case !3: statements;    <--- NOT operator is not allowed in case number
                ......
                break;

        case !(a+b): statements;    <--- NOT operator is not allowed in case number
                  ......
                  break;
}
```

7. The case numbers can be in any random order. It is not necessary to have case numbers as 1,2,3,....

* We can use any random positive or negative or zero as case number.

```
void main()
{
	.......
	.......

	printf("Which operation you want to perform?\n");
	printf("436 - Addition\n");
	printf("2678 - Subtraction\n");
	printf("15678 - Multiplication\n");
	printf("364 - Division\n");
	printf("Provide your choice : ");
	scanf("%c", &o_choice);

	switch(o_choice)
	{
		case 436: c = a + b;
			printf("Addition is : %d", c);
			break;
		case 2678: c = a - b;
			 printf("Subtraction is : %d", c);
			 break;
		case 15678:

	}
```

* This type of complex inputs and case numbers will be confusing for end-user to input. So it will be good if we use single digit case-numbers.

==========

16-7-21(2)

8. Use of 'default' case is optional.

* The 'default' case executes when no case-number matches with value of 'switch-variable'. But if we don't use 'default' case then the switch-case executes nothing. No case matches.

9. Every case should end with 'break' statement. If 'break' is not used then next written case executes automatically. This will continue unless a 'break' appears or switch-case terminates.

```
int ch=3;
switch(ch)
{
        case 1: ......
                ......

        case 2: ......
                ......

        case 3: ......    <--- controls jumps here and executes all following
                ......

        case 4: ......
                ......

        case 5: ......
                ......

        default: .......
```

```
                .....
}
```

* In above switch-case, the controls directly jumps at case-3. And because of absence of 'break' next written case executes. Even last 'default' case also executes.

10. We can write case numbers in any random order. It's not mendatory to write 'default' case as last case. Ultimately the case number with which the value of 'switch-variable' matches will execute.

```
void main()
{
        int ch=1;

        switch(ch)
        {
                case 4: statements;
                        break;

                default: statements;
                         break;

                case 2: statements;
                        break;

                case 6: statements;
                        break;
        }
}
```

11. Use of keyword "continue" is not allowed in switch-case. It will show CT error.

For example:

```
void main()
{
	char o_choice;

	printf("A - Addition\n");
	printf("S - Subtraction\n");
	printf("M - Multiplication\n");
	printF("D - Division\n");
	scanf("%c", &o_choice);

	switch(o_choice)
	{
		case 'A':
		case 'a':
		case '+': c = a + b;
			printf("Addition is : %d", c);
			break;

		case 'S':
		case 's':
		case '-': c = a - b;
			printf("Addition is : %d", c);
			break;

		case 'M':
		case 'm':
		case '*': c = a * b;
```

```
                printf("Addition is : %d", c);
                break;

        case 'D':
        case 'd':
        case '/': c = a / b;
                printf("Addition is : %d", c);
                break;
    }
}
```

==========

17-7-21(1)

* If logically required, then we can also write the entire switch-case inside if-block, else-block, loop's block.

* And we can use any logical blocks inside a case of swith-case.

* Refer following program. It is a combination of user-driven program with a menu-driven program.

```
#include<stdio.h>
void main()
{
    int o_choice;
    int a,b,c;
    float d;
    char u_choice;

    printf("main starts");

    do
```

```
{
  printf("\n\nEnter 2 values\n");
  scanf("%d%d", &a, &b);

  printf("\nSelect an operation to perform : \n");
  printf("1 - ADDITION\n");
  printf("2 - SUBTRACTION\n");
  printf("3 - MULTIPLICATION\n");
  printf("4 - DIVISION\n");
  printf("Provide your choice : ");
  scanf("%d", &o_choice);

  switch(o_choice)
  {
    case 1: c = a + b;
            printf("Addition is : %d", c);
            break;
    case 2: c = a - b;
            printf("Subtraction is : %d", c);
            break;
    case 3: c = a * b;
            printf("Multiplication is : %d", c);
            break;
    case 4: d = a / (float)b;
            printf("Division is : %f", d);
            break;
    default: printf("Invalid Operation Choice");
             break;
  }

  fflush(stdin);
```

```
    printf("\nDo You want to perform another operation (y/n) ? : ");
    scanf("%c", &u_choice);
  }while(u_choice=='Y' || u_choice=='y');

  printf("\n\nmain ends\n\n\n");
}
```

==========

* Nested switch-case is also possible. We can use another/nested switch-case within a case of outer switch-case.

```
void main()
{
        int ch1, ch2;
        int a,b,c;
        float d;

        printf("Select an opertaion to perform\n");
        printf("1 - Arithmetic Operations\n");   <--- requires nested switch-case
        printf("2 - Numeric Operations\n");   <--- requires nested switch-case
        printf("3 - Printing Patterns\n");   <--- requires nested switch-case
        printf("4 - To check Palindrome or not\n"); <--- don't require nested switch-case
        printf("Provide your choice : ");
        scanf("%d", &ch1);

        switch(ch1)
        {
                case 1: printf("Enter 2 values\n");
                        scanf("%d%d", &a, &b);
```

```
        printf("Which operation you want to perform : ?\n");
        printf("1 - Addition\n");
        printf("2 - Subtraction\n");
        printf("3 - Multiplication\n");
        printf("4 - Division\n");
        printf("Provide your choice : ");
        scanf("%d", &ch2);

        switch(ch2)
        {
            case 1: c = a + b;
                    printf("Addition is : %d", c);
                    break;   <--- to come-out of inner switch-case
            case 2: c = a - b;
                    printf("Subtraction is : %d", c);
                    break;   <--- to come-out of inner switch-case
            .....
            .....
            default:printf("Invalid Choice\n");
        }
        break;  <--- to come-out of outer switch-case

case 2: printf("Enter a number : ");
        scanf("%d", &num);

        printf("Which operation you want to perform : ");
        printf("1 - To reverse a number\n");
        printf("2 - To print factorial\n");
        printf("3 - To check EVEN or ODD\n");
        printf("Provide your choice : ");
        scanf("%d", &ch2);
```

```
                    switch(ch2)
                    {
                        ........
                        ........
                    }
                    break;   <--- to come-out of outer switch-case

            case 3:  ......
                        ......
                        ......

            default: printf("Invalid Choice");
        }
}
```

==========

==========

* What will be the output of following programs?

a)

```
void main( )
{
    char suite = 3;
    switch(suite)
    {
        case 1: printf("\nDiamond");
        case 2: printf("\nSpade");
        default: printf("\nHeart");
```

```
        }
        printf ("\nI thought one wears a suite");
}
```

Output:

Heart

I thought one wears a suite

==========

b)

```
void main()
{
        int c = 3;
        switch(c)
        {
                case 'v' : printf( "I am in case v \n");
                           break;
                case 3 : printf( "I am in case 3 \n" );
                         break;
                case 12 : printf( "I am in case 12 \n" );
                          break;
                default : printf ( "I am in default \n");
        }
}
```

Output:

I am in case 3

==========

c)

```
void main( )
{
        int k, j=2;
        switch (k = j + 1)
        {
                case 0 :
                        printf("\nTailor");
                case 1 :
                        printf("\nTutor");
                case 2 :
                        printf("\nTramp");
                default :
                        printf("\nPure Simple Egghead!");
        }
}
```

Output:

Pure Simple Egghead!

==========

d)

```
void main()
{
        int i = 0;
        switch(i)
        {
                case 0:
                        printf("\nCustomers are dicey");
                case 1:
```

```
                printf("\nMarkets are pricey");
        case 2:
                printf("\nInvestors are moody");
        case 3:
                printf("\nAt least employees are good");
    }
}
```

Output:

Customers are dicey

Markets are pricey

Investors are moody

At least employees are good

==========

e)

```
void main
{
    int k;
    float j = 2.0;
    switch(k = j + 1)
    {
        case 3:
            printf("\nTrapped");
            break;
        default:
            printf("\nCaught!");
    }
}
```

Output:

Trapped

==========

f)

```
void main()
{
	int ch = 'a' + 'b';
	switch (ch)
	{
		case 'a':
		case 'b':
			printf( "\nYou entered b");
		case 'A':
			printf( "\na as in ashar");
		case 'b' + 'a':
			printf( "\nYou entered a and b");
	}
}
```

Output:

You entered a and b

==========

==========

19-7-21(1)

* What will be the output of following programs?

```
void main()
{
	int i = 1;
	switch(i - 2)
	{
		case -1:
			printf("\nFeeding fish");
		case 0:
			printf("\nWeeding grass");
		case 1 :
			printf("\nmending roof");
		default :
			printf("\nJust to survive");
	}
}
```

Output:

Feeding fish

Weeding grass

mending roof

Just to survive

Explanation:

* Because of expression (i-2) the switch becomes -1. Hence, the interpreter will search for case (-1) and starts with it.

* Because of absence of "break", fall-down will happen and hence every case executes.

==========

* Find out errors, if any, in following progras.

a)

```
void main( )
{
        int suite = 1;
        switch(suite);
        {
                case 0;
                printf("\nClub");
                case 1;
                printf("\nDiamond");
        }
}
```

* Above program shows CT errors:

1. There must not be a semi-colon with the case-numbers.
2. There must not be a semi-colon with switch-variable. (at starting). If we use semi-colon with switch-variable then it considers empty body of switch-case. Hence, all mentioned 'cases' and 'break' are considered as open statements. And the use of 'case' and 'break' is not allowed in open area. It must be part of switch-case.

==========

b)

```
void main()
{
        int temp;
        scanf("%d", &temp);
        switch(temp)
```

```
	{
		case(temp <= 20):
			printf ("\nOooooooohhhh! Damn cool!");
		case(temp > 20 && temp <= 30):
			printf ("\nRain rain here again!" );
		case(temp > 30 && temp <= 40):
			printf ("\nWish I am on Everest");
		default:
			printf ("\nGood old nagpur weather" ) ;
	}
}
```

Error:

- We cannot use logical operators (&&) in case-numbers.

==========

c)

```
void main( )
{
	float a = 3.5;
	switch (a)
	{
		case 0.5:
			printf( "\nThe art of C");
			break ;
		case 1.5:
			printf( "\nThe spirit of C");
			break;
		case 2.5:
			printf( "\nSee through C");
```

```
                        break;
                case 3.5:
                        printf("\nSimply c");
        }
}
```

Error:

Floating point values (REAL values) are not allowed as case-numbers.

==========

d)

```
void main( )
{
        int a = 3, b = 4, c ;
        c = b – a ;
        switch ( c )
        {
                case 1 || 2:
                        printf("God give me an opportunity to change things");
                        break;
                case a || b:
                        printf("God give me an opportunity to run my show");
                        break;
        }
}
```

Error

- We cannot use logical operator (||) in case number.

==========

Homework:

Write a program which to find the grace marks for a student using switch. The user should enter the class obtained by the student and the number of subjects he has failed in.

− If the student gets first class and the number of subjects he failed in is greater than 3, then he does not get any grace. If the number of subjects he failed in is less than or equal to 3 then the grace is of 5 marks per subject.

− If the student gets second class and the number of subjects he failed in is greater

than 2, then he does not get any grace. If the number of subjects he failed in is less than or equal to 2 then the grace is of 4 marks per subject.

− If the student gets third class and the number of subjects he failed in is greater than 1, then he does not get any grace. If the number of subjects he failed in is

equal to 1 then the grace is of 5 marks per subject

==========

---- The exit() function -----

==========

* We know that, keyword "break" is to terminate any loop or come-out of switch-case.

* Similarly, exit() is a library-function and it terminates the program execution.

* We also know that, we provide parameter to printf() and scanf() functions. Similarly, we have to provide a confirmation parameter to exit() function. That confirmation parameter must be ZERO (0) to terminate.

* The library function exit() is in <stdlib.h> header file. So include it.

* Refer and compare following two programs.

Program-1:

```
#include<stdio.h>
void main()
{
   int i;
   printf("main starts\n\n\n");
   for(i=1; i<=5; i++)
   {
      printf("Start : %d\n", i);

      if(i==3)
      {
         break;  <--- terminates only current loop. Program execution continues...
      }

      printf("End : %d\n", i);
   }

   printf("\n\nProgram execution continues....");
   printf("\n\nmain ends\n\n\n");
}
```

Output will be:

---

main starts

Start : 1

End : 1

Start : 2

End : 2

Start : 3

Program execution continues....

main ends

==========

==========

Program-2:

```
#include<stdio.h>
#include<stdlib.h>  <--- include it to use exit()

void main()
{
   int i;
   printf("main starts\n\n\n");
   for(i=1; i<=5; i++)
   {
      printf("Start : %d\n", i);

      if(i==3)
      {
         exit(0);   <--- terminates program execution.
      }

      printf("End : %d\n", i);
   }
   printf("\n\nProgram execution continues....");
   printf("\n\nmain ends\n\n\n");
```

```
}
```

Output will be:

---

main starts

Start : 1

End : 1

Start : 2

End : 2

Start : 3

==========

==========

* If logically required, then we can use it in switch-case instead of a user-driven choice.

```
void main()
{
        int a,b,c
        int o_choice;

        while(1)    <--- iterates infinite times
        {
                printf("Enter two numbers\n");
                scanf("%d%d", &a, &b);

                printf("Which operation you want to perform : \n");
```

```
                printf("1 - ADDITION\n");
                printf("2 - SUBTRACTION\n");
                printf("3 - MULTIPLICATION\n");
                printf("4 - DIVISION\n");
                printf("5 - EXIT\n");
                scanf("%d", &o_choice);

                switch(o_choice)
                {
                        case 1: ....
                                break;

                        case 2: ......
                                break;

                        case 3: .....
                                break;

                        case 4: .....
                                break;

                        case 5: exit(0);   <--- will terminate program execution
                                break;  <--- is optional

                        default: printf("Invalid Choice");
                                 break;
                }
        }
}
```

* Note that, we used exit() to terminate program execution. We also made it user-dependent. Therefore we put it within switch-case.

==========

19-7-21(2)

* We know that, switch-case executes faster than multiple-if statements.

* If logically required, then we can also put the entire switch-case inside a loop.

* We can either use a user-driven do-while loop or we can use while-loop with 1 (always true) along with EXIT option.

* The exit() is a library function in <stdlib.h> header file.

* The exit() function needs ZERO (0) as parameter.

==========

---- The "goto" statement ----

==========

* The "goto" statement is also called as "jump statement".

* The purpose of "goto" statement is to take controls at LABEL.

* To use "goto" statement, we have to prepare a LABEL and then jump to that label by using keyword "goto".

* The general syntax is:

```
goto  LABEL_NAME;
```

* Similar to break, continue and exit(); we should use "goto" based on a condition.

For example:

```
void main()
{
	int a=5;

	printf("Hello 1");
	printf("Hello 2");

	if(a<10)  <--- condition is TRUE
	{
		goto XYZ;  <--- 'goto' executes
	}

	printf("Hello 3");
	printf("Hello 4");
	XYZ:
	printf("Hello 5");
	printf("Hello 6");
}
```

The output will be:

```
Hello 1
Hello 2
Hello 5
Hello 6
```

* In above program we prepared a label 'XYZ'.

* Based on correctness of condition, the 'goto XYZ' statement will execute and controls will directly jump at label 'XYZ'

* From the output, don't think that the label 'XYZ' will execute if the 'goto XYZ' executes. The label executes even if the condition was FALSE. Because it was on the way.

Example-2:

```
#include<stdio.h>
void main()
{
	int a=5;

	printf("Hello 1\n");
	printf("Hello 2\n");

	if(a>10)  <--- condition is FALSE
	{
		goto XYZ;  <--- 'goto' does not executes, then also label executes.
	}

	printf("Hello 3\n");
	printf("Hello 4\n");
	XYZ:
	printf("Hello 5\n");
	printf("Hello 6\n");
}
```

Hello 1

Hello 2

Hello 3

Hello 4

Hello 5

Hello 6

* From the output note that, the label 'XYZ' executes even if the 'goto' statement does not execute.

==========

* If logically required, then we can also do "upward jump". i.e. we can write the label above the 'goto' statement. And controls will jump upward (will work similar to LOOP)

```
#include<stdio.h>
void main()
{
	int i=1;
	printf("main starts\n\n");

	XYZ:
	printf("Hello - %d\n", i);
	i++;
	if(i<=10)
	{
		goto XYZ;
	}

	printf("\nmain ends");
}
```

The output will be:

---

main starts

Hello - 1

Hello - 2

Hello - 3

Hello - 4

Hello - 5

Hello - 6

Hello - 7

Hello - 8

Hello - 9

Hello - 10

main ends

* From above output, note that, we can also use "goto" statement like a loop's iterations.

* Above program is equivalent to do-while loop's iterations.

==========

==========

Example-3:

```
void main()
{
        int i;

        printf("main starts\n");
```

```
	for(i=1; i<=5; i++)
	{
		printf("start : %d\n", i);

		if(i==3)
		{
			goto XYZ;  <--- controls will jump to label 'XYZ'
		}

		printf("end : %d\n", i);
	}

	printf("hello all");
	XYZ:
	printf("main ends");
}
```

The output will be:

```
main starts
start : 1
end : 1
start : 2
end : 2
start : 3
main ends
```

==========

==========

*** We can jump in/out any logical-block.... anywhere in the same function....

==========

==========

* Those 5 rules of giving variable name ----> are applicable for LABEL.

* The variable's name and LABEL name must not be same..

==========

==========

20-7-21

```
void main()
{
	int a,b,c,d,e,f,g,h;

	a = 10;
	b = 20;

	c = a + b;
	printf("I got the result");
	printf("It is correct");
	printf("Goto next step");
	printf("Good day");

	d = a - b * 3 + c;
```

```
        printf("I got the result");
        printf("It is correct");
        printf("Goto next step");
        printf("Good day");

        e = a + b - c * 3 + d + a;
        f = b - 3 * c;
        printf("I got the result");
        printf("It is correct");
        printf("Goto next step");
        printf("Good day");

        g = a + b - e * 15;
        printf("I got the result");
        printf("It is correct");
        printf("Goto next step");
        printf("Good day");

        h = b - c * 2 + a - e * f;
        printf("I got the result");
        printf("It is correct");
        printf("Goto next step");
        printf("Good day");
}
```

* Note that above program is totally correct. No error. It will execute properly.

* But also note that, there are 4 printf() statements that are repeated 5 times in main() method.

* It increases program length. And this will waste compile time.

* As a solution to this, we are allowed to prepare a separate block of statement and we can use that block's name as replacement of 4-statements.

==========

==========

```
void main()
{
	int i, num, r, s, sum=0, flag=1, temp;

	// checking prime or not


	// number reverse


	// largest digit finding


	// check even or odd


	// palindrome or not checking

}
```

* Above program is correct and valid with all respective operations.

* But the only thing is, if we are willing to do multiple operations in main() method then it is possible that we accidently over-write value of any variable; or there is a bug that also affects further operation; or the error-finding becomes complex, etc...

* The solution of such situations is, "to prepare individual block for every operation". And use the block's name wherever required.

```
check_prime_or_not
{
        int num, r, flag=1;

        // logic of prime or not

}

check_even_or_odd
{
        int num;

        // logic to check even or odd

}

check_palindrome_or_not
{
        int num;

        // logic to check palindrome or not

}

reverse_the_number
{
```

```
        // logic to reverse the number

}



find_largest_digit

{

        // logic to print largest digit

}



void main()

{

        check_prime_or_not;

        reverse_the_number;

        find_largest_digit;

        check_even_or_odd;

        check_palindrome_or_not

}
```

* Note that, we are again performing same 5 operations. But this time, instead of writing all 5 operations in main() method; we divide it into individual blocks. And use them whenever required.

* This time we did proper operational distribution. And this will also help the programmer to use proper variables, no threat of accidently over-writing any variable, bug-finding easy. No effect of any bug to another operation.

==========

==========

1. If we have set of statements that are appearing multiple times in main(), then we can prepare a separate block of those statements; and use the block's name multiple times at multiple locations.

2. If we are writing multiple complex operations in main() method, then these all operations are making the program complex and may become buggy. In this case also we can prepare separate block for every single operation and reduce the chances of bugs and variable over-writing.

==========

==========

21-7-21(1)

* Now we know that, we should prepare functions :

- If there is a set of statements which is appearing multiple times at different locations.
- If there are multiple operations in main() method.

* We can say that, "a function is set of statements which can be re-used".

==========

==========

* There are three parts of a function:

- function's body/definition
- function's calling
- function's declaration

1. To define a function ['define' ---> preparing body]

```
returntype  function_name(parameter_list)
{
        statements;
        ........
        ........
}
```

2. To call a function

```
function_name(parameter_list);
```

3. To declare a function

```
returntype  function_name(parameter_list);
```

** parameters are optional.

** parameters are also called as "arguments".

** the same 5 naming rules (identifier) are applicable to function-name.

** We can give any name to function. But always prefer to give related names. So that others can guess the application/role of that function.

==========
==========

==========
==========

For example:

```
#include<stdio.h>

_______ check_even_or_odd()
{
	int num;
	printf("Enter a number : ");
	scanf("%d", &num);

	if(num%2 == 0)
	{
		printf("is EVEN");
	}
	else
	{
		printf("is ODD");
	}
}



_______ reverse_number()
{
	int num, r, sum=0;

	printf("Enter a number : ");
```

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


	while(num>0)
	{
		r = num % 10;
		sum = (sum*10) + r;
		num = num / 10;
	}


	printf("Reverse is : %d", sum);
}
```

*** Similarly, we can prepare any numbers of functions in a program.

* After defining a function, we can use it 'n' numbers of times.

* We can write any statement in a function.

* But the only rule is "nested function is not allowed".

* i.e. we cannot define a function inside body of another function.

* We can define it even before main() and after main() method. But always PREFER to define function(s) before main() method.

* Till date, we were defininig main() method. It was also a function.

* If we define functions, then that function will not execute automatically. We must call that function to use it.

* i.e. a function executes if and only if we call it.

* Only main() method executes automatically. And every other function needs to be called.

For example:

```
void main()
{
	.....
	.....
	check_even_or_odd();   <--- calling to  check_even_or_odd()  function
```

```
        .......
        .......
        reverse_number();  <--- calling to  reverse_number()  function first time
        .......
        .......
        reverse_number();  <--- calling to  reverse_number()  function second time
        .......
        .......
        reverse_number();  <--- calling to  reverse_number()  function third time
        ......
}
```

==========

==========

* Now we can say:

1. There are total 3 functions defined in above program.

        check_even_or_odd()

        reverse_number()

        main()

2. But the execution of above program starts from main() only.
3. From main() method we have called to both other functions (as needed).

**** There is no other word or logic which can replace main() method and can begin the program execution.

********* Program execution starts from main() only only only.

*** The order of defining function does not matter.

==========

==========

21-7-21(2)

* If there are set of statements which are appearing multiple times in our program, then we should prepare a separater function for it. And call it whenever and wherever required.

* If there are multiple complex operations happening in our program, then we should prepare separate functions for every operation; and call them whenever and wherever required.

* A function is set of statements which can be reused.

* A function has 3 parts:

    function's body / definition

    function's calling

    function's declaration

* To define a function:

```
returntype  function_name(paramter_list)
{
    statements;
    statements;
    .......
    .......
}
```

* To call a function:

function_name(parameter_list);

* To declare a function:

returntype function_name(paramter_list);

* Those same 5 rules of giving an identifier, are applicable for function's name also.

==========

For example:

```
#include<stdio.h>
_______ check_even_odd()
{
        int num;

        .........
        .........
}

_______ check_prime_or_not()
{
        int num, s, r;

        .........
        .........
}

void main()
```

```
{
        .........
        .........
        check_even_odd();   <--- calling to check_even_odd() function
        .........
        .........
        check_prime_or_not();  <---- calling to check_prime_or_not() function
        .........
        .........
        check_prime_or_not();  <---- calling to check_prime_or_not() function
        .........
        .........
        check_prime_or_not();  <---- calling to check_prime_or_not() function
        .........
}
```

==========

==========

*** IMP ***

1. In a program, we can define any numbers of functions.

2. The program execution starts from main() only. There is not other word or no other logic that can begin the execution of program.

3. Nested function is not allowed. We cannot define a function inside body of another function.

4. function's name must be unique.

5. A function will not execute automatically. We must call a function to use it.

6. When a function calls to another function, the calling function transfers it's own controls to the body of called function; and itself gets paused.

7. When the execution of called function ends, then the controls returns back to the statement of calling function from where it was called.

8. The function which calls to another function -----> calling function

9. The function which is called by anther function ----> called function

10. This means, only one function executes at a time.

==========

22-7-21(1)

* A function cannot execute automatically. We must call a function to execute it.

* When a function calls to another function, the calling function transfers it's own controls to the body of called function; and itself gets paused.

* When execution of called function ends, then controls returns back to calling statement of calling function. And calling function resumes.

* This means, only one function executes at a time.

* Nested functions are not allowed.

* We can define any numbers of functions in any order. But execution starts from main() only. There is no other word or no other logic that can start the execution from somewhere else.

* The program execution starts from { of main() method and runs upto } of main() method.

* Function's name must be unique. Follow those 5 naming rules.

* Any function can call to any other function any number of times. [This means, don't think that only main() method can call to other functions.]

* We cannot define one function inside body of another function. [Nested functions are not allowed]. We can only call functions within each other.

* When we call a function, exeuction of that function starts from it's first statement.

```
void main()
{
        int num;
        printf("
        scanf("
        _____ factorial()   <--- cannot define function within body of another function
        {



        }



        _____ square_cube()  <--- cannot define function within body of another function
        {



        }
        ........
        ........
}
```

==========
==

* What will be the output of following programs?

a)

```
#include<stdio.h>
_____ chair()
{
        printf("White");
        printf("Flower");
}

_____ table()
{
        printf("Cartoon");
        printf("Vehicle");
}

void main()
{
        printf("Super");
        table();
        printf("Bridge");
        chair();
        printf("Customer");
}
```

Output will be:

Super

Cartoon

Vehicle

Bridge

White

Flower

Customer

==========

b)

```
#include<stdio.h>

_______ chair()
{
        printf("Flower");
        printf("Mobile");
}

________ table()
{
        printf("Template");
        fan();
        printf("Moon");
        chair();
        printf("Controls");
}

________ fan()
{
        printf("USA");
        chair();
        printf("Default");
```

```
}

void main()
{
        printf("Keywords");
        fan();
        printf("Environment");
        chair();
        printf("River");
        table();
        printf("Variable");
}
```

==========

22-7-21(2)

* When a function calls to another function, the calling function transfers it's own controls to the body of called function. And itself gets paused.

* When execution of called function ends, then the controls will be returned back to the statement from where it was called.

* In a program, we can define any numbers of functions in any order. But execution starts from main() only.

* Any function can call to any other function any number of times.

* A function will not execute automatically. We must call a function to execute it.

* Prefer to define new functios before main() method.

Program-1:

Write a program from following instructions:

- Define a function that accepts a number from the user and prints whether that number is even or odd.
- Define another function that accepts two numbers from the user and print their addition.
- Define main() method and call both above functions; one by one.

```
#include<stdio.h>
_______ check_even_odd()
{
	int num;
	printf("Enter a number : ");
	scanf("%d", &num);

	if(num%2 == 0)
	{
		printf("is EVEN");
	}
	else
	{
		printf("is ODD");
	}
}


_______ perform_addition()
{
	int a,b,c;
```

```
	printf("Enter 2 values\n");
	scanf("%d%d", &a, &b);

	c = a + b;
	printf("Addition is : %d", c);
}

void main()
{
	printf("main starts here\n");

	printf("Checking EVEN or ODD\n");
	check_even_odd();

	printf("Performing Addition\n");
	perform_addition();

	printf("main ends here");
}
```

** Note that, we can write any logical statement in functions.

** Till date we were defining main() method. It is also a function. We were preparing it's body/definition.

** Practically and logically there is no difference between main() method and other normal functions. The only thing is main() method can begin the exeuction; whereas, other functions needs to be called.

==========
==

Program-2:

Write a program from following instructions:

- Define a function that receives two numbers from the user and print the largest of them.
- Define a function that receives a number from user and print it's last digit.
- Define a function that prints all the numbers from 1 to 'n'. Accept 'n' from the user.
- Define main() method and call these 3 functions.

** Always prefer to give small and related name to function.

** Every function has body.

** The body of library functions are in specific header-file.

** We can call a function from a loop, if-else, switch-case, etc....

==========
==

____ addition()

{

}

_____ subtraction()

{

}

```
_____ multiplication()
{

}

_____ division()
{

}

void main()
{
        ..........
        ..........
        do
        {
                ..........
                ..........

                switch(ch)
                {
                        case 1: addition();
                                break;

                        case 2: subtraction();
                                break;

                        case 3: multiplication();
                                break;
```

```
                        case 4: division();
                                break;

                        case 5: exit(0);
                                break;

                        default: printf("Invalid operation choice");
                }
        }while(1);

}
```

==========

23-7-21(1)

* We know that, we define a function (as required) with any logical statements and call them from main() method.

* Any function can call to any other function any number of times.

* The body of library functions are in specific header-file. When we include a header file, the body of it's every library functions gets defined/available in that program.

==========

---- Types of variables (based on declaration) ----

==========

* There are three types of variables, based on declaration.

    - local variables

- global variables
- block variables

local variables:

- The variables which are declared inside body of a function, are called as local variables of that function.
- The scope/availability of local variable is within that function.
- The local variables of one function are not accessible in other functions.

global variables:

- The variables which are declared outside body of a function, are called as global variables.
- The scope/availability of global variables is in all the functions of that program. [common between all] [centeralized variable] [every function can access]

block variables:

- The variables which are declared inside a logical-block, are called as block variables.
- The scope of block variable is within that logical-block.
- The remaing part of function cannot access the block variables.

For example:

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

int m,n;  <--- are global variables
```

```
_______ check_even_odd()
{
        int num;   <--- is a local variable of check_even_odd() method

        if(num%2 == 0)
        {
                float p;   <--- is a block variable
                ..........
                ..........
        }
        else
        {
                ..........
                ..........
        }
}

float z;  <---- is a global variable

_______ reverse_num()
{
        int num, r, s=0;    <--- are local variables of reverse_num() method

        while(num>0)
        {
                int j;   <--- is a block variable
                .........
                .........
        }
        .........
```

```
}

void main()
{
        int a,b;   <--- are local variables of main() method
        float x;   <--- is local variable of main() method

        if(a>b)
        {
                int i;    <--- is a block variable
                ........
                ........
        }

        .........
        .........
}
```

==========

Example-2:

```
void main()
{
        int x = 10;  <--- is local variable

        if(CONDITION)
        {
                int y = 20;    <--- is block variable
                printf("local x = %d", x); // VALID. x IS LOCAL VARIABLE OF MAIN()
```

```
		printf("block y = %d", y); // VALID. y IS BLOCK VARIABLE
	}

	if(CONDITION)
	{
		int z = 30;  <--- is block variable
		printf("local x = %d", x); // VALID. x IS LOCAL VARIABLE OF MAIN()
		printf("block y = %d", y); //CT ERROR. y IS BLOCK VARIABLE OF PREVIOUS if
		printf("block z = %d", z); // VALID. z IS BLOCK VARIABLE HERE
	}

		printf("local x = %d", x); // VALID. x IS LOCAL VARIABLE OF MAIN()
		printf("block y = %d", y); // CT ERROR. y IS BLOCK VARIABLE OF FIRST IF
		printf("block z = %d", z); // CT ERROR. z IS BLOCK VARIABLE OF SECOND IF

}
```

==========

==========

Example-3:

```
void main()
{
	int x=10;   <--- 'x' becomes local variable of main().
				is accessible within main() only

}
```

_____ addition()

{

int y=20; <--- 'y' becomes local variable of addition()

is accessible within addition() only

}

_____ subtraction()

{

float z=331.44; <--- 'z' becomes local variable of subtraction()

is accessible within subtraction() only

}

*** We can give same names for local variables of multiple functions.

*** All those variables will have different memory blocks and different scopes.

==========

==========

*** IMP ***

---

* There are many security related problems in Global variables.

* A good programmer should avoid the use of global variables....

* You should prefer to declare global constants

==========

==========

23-7-21(2)

* We know that, there are 3 types of variables/const (based on declaration)

- local variables/const
- global variables/const
- block variables/const

* We should not prefer to declare global variables. But we can declare global constants.

* The local variables of multiple functions can have same names. Their memory-blocks and address/location and values remains individual/different.

* We know that, the scope of these variables are different (based on declaration).

*** "The lifetime of every memory-block is upto the program execution."

* This means, the memory-block remains upto end of execution. [A memory-block gets deleted only when the execution of program ends.]

==========

* Refer following programs.

Example-1:

```
#include<stdio.h>
int a,b,c;   <---- are global variables

_____ perform_addition()
{
```

```
	c = a + b;
}

void main()
{
	printf("Enter 2 values\n");
	scanf("%d%d", &a, &b);
	perform_addition();
	printf("Addition is : %d", c);
}
```

==========

Example-2:

```
#include<stdio.h>
int m1,m2,m3,m4,m5;  <--- are global variables
int tot;  <--- global variable
float av;		<--- global variable

_____ accept_marks()
{
	scanf("%d%d%d%d%d", &m1, &m2, &m3, &m4, &m5);
}

_____ calc_total()
{
	tot = m1 + m2 + m3 + m4 + m5;
}
```

```
_____ cal_average()
{
        av = tot / 5.0;
}

____ print_result()
{
        printf("Total Marks : %d", tot);
        printf("Average Marks : %f", av);
}

void main()
{
        printf("Enter marks of 5 subjects\n");
        accept_marks();

        calc_total();

        calc_average();

        print_result();
}
```

==========

24-7-21(1)

* We know that, there are 3 types of variables/constants [memory-blocks] based on declaration:
    - local variables

- global variables

- block variables

* The scope of local variable/constants is within that function. Therefore local variable/constants of one function are not available in another function.

* But we also agree that, use of global variables is not useful/secure/good-habit. We can declare gobal constants.

==========

==========

---- Parameter Passing ----

==========

* We know that, local variables/constants of one function are not accessible in another function.

* In any logical case, if you want to share any value between multiple functions then simply pass that value as parameter while calling a function.

* And receive that passed value while defining a function.

* The parameters which are passed while calling ----> ACTUAL PARAMETERS/ARGUMENTS

* The parameters which receives these passed parameters ----> FORMAL PARAMETERS/ARGUMENTS

**** IMP ****

1. A function can pass any numbers of actual-parameters of any random datatype.

2. The AP and FP can have same names.

3. The FP are ultimately local variables of that function. [Their scope is within that function].

4. The numbers of AP and FP must must must must must must be same. [It will show CT error if count mismatches.]. [Difference in datatype is OK/valid.]

5. We can also pass console values as AP.

==========

Example-1: ---> Passing and Sharing a value between two functions.

```
#include<stdio.h>
_____ square(int x)  <--- 'x' is FP that receives passed value of 'num'
{
        int sq;
        sq = x * x;
        printf("Square is : %d", sq);
}

_____ cube(int y)  <--- 'y' is FP that receives passed value of 'num'
{
        int cu;
        cu = y * y * y;
        printf("Cube is : %d", cu);
}

void main()
{
        int num;
        printf("Enter a number : ");
        scanf("%d", &num);

        square(num);   <--- calling square() and passing 'num' as AP
        cube(num);   <--- calling cube() and passing 'num' as AP
```

```
}
```

* In above example, execution begins from main() method. And from main() we are passing value of local variable 'num' as ACTUAL PARAMETER to both functions.

* The function square() recieves passed value as FORMAL PARAMATER 'x'; and function cube() receives the passed value as FORMAL PARAMETER in 'y'.

==========

Example-2: ---> Passing values multiple times to a function.

```
#include<stdio.h>
_______ square_cube(int x)   <--- 'x' is FORMAL PARAMETER
{
        int sq, cu;

        sq = x * x;
        cu = x * x * x;

        printf("Square is : %d", sq);
        printf("Cube is : %d", cu);
}

void main()
{
        int num1, num2, num3;

        printf("Enter a number : ");
```

```
        scanf("%d", &num1);
        square_cube(num1);  <--- passing value of 'num1' as ACTUAL PARAMETER

        printf("Enter another number : ");
        scnaf("%d", &num2);
        square_cube(num2);  <--- passing value of 'num2' as ACTUAL PARAMETER

        printf("Enter another number : ");
        scanf("%d", &num3);
        square_cube(num3);  <--- passing value of 'num3' as ACTUAL PARAMETER
}
```

==========

24-7-21(2)

---- Parameter Passing ----

==========

* We know that, a function can pass parameters to another function while calling it.

```
#include<stdio.h>

_______ square_cube(int n) <--- receiving value of 'num' in FP 'n'
{
        int sq, cu;
        sq = n * n;
        cu = n * n * n;
        printf("Square is : %d", sq);
        printf("Cube is : %d", cu);
```

```
}

void main()
{
        int num;

        printf("Enter a number : ");
        scanf("%d", &num);

        square_cube(num);  <--- calling to square_cube() and passing 'num' as AP
}
```

==========

*** IMP ***

- If we call a function multiple times, then each time that function creates new memory-blocks for variables. [It will not use previous memory-blocks.]

==========

* What will be the output of following program?

```
_____ temp()
{
        int p=1;
        printf("%d\n", p);
        p++;
}

void main()
```

```
{
        int i;
        for(i=1; i<=5; i++)
        {
                temp();
        }
}
```

The output will be:

1

1

1

1

1

** Because on every call, it creates new memory-block for 'p' and initilizes with 1.

==========

*** IMP ***

- We have a keyword "static". We can use this keyword while declaring local variables and block variables.

- If we declared a static variable, then it will continue with previously prepared memory-block. [static variables will not do double declaration and creation.]

For example:

```
void temp()
{
        static int p=1;  <--- 'p' is declared as 'static'
        printf("%d\n", p);
```

```
	p++;
}

void main()
{
	int i;
	for(i=1; i<=5; i++)
	{
		temp();
	}
}
```

The output will be:

```
--------------------
1
2
3
4
5
```

==========

==========

25-7-21(1)

* We can call to any function any number of times. And on each call, a function creates new memory-blocks for it's local and block variables. [i.e. old/previous memory-block will not be used]

* In any logical situation, if we want to continue/use old/previous memory-blocks, then we have to make them "static".

* We can call a function and pass any numbers of parameters/arguments to it. The parameters which are passed during calling ---> ACTUAL PARAMETERS/ARGUMENTS

* The parameters in which we receive the actual parameters are ---> FORMAL PARAMETERS/ ARGUMENTS.

printf("Enter 2 values"); <--- we are calling library function printf()
and passing string message as AP.
This means, printf() has a string-type FP.

scanf("%d", &num); <--- we are calling library function scanf()
and passing format-specifier and address as AP.
This format-specifier and address are used in body of scanf()

fflush(stdin); <--- we are calling fflush() and passing 'stdin' as AP

clrscr(); <--- The clrscr() function don't have any FP [in it's original body]
therefore we are not passing any AP to it.

getch(); <--- The getch() function don't have any FP [in it's body] therefore we
are not pssing any AP to it.

==========

==========

Program-1:

Write a program from following instructions.

- Define a function that receives marks of 5 subjects as parameters from main() method and calculates total-marks & average marks.
- Define main() method that accepts marks of 5 subjects of a student from the user and pass it to our user-defined function.

```
#include<stdio.h>
_____ calc_total_average(int a, int b, int c, int d, int e)
{
        int tot;
        float av;

        tot = a + b + c + d + e;
        av = tot/5.0;

        printf("Total Marks : %d", tot);
        printf("Average Marks : %f", av);
}

void main()
{
        int m1, m2, m3, m4, m5;

        printf("Enter marks of 5 subjects\n");
        scanf("%d%d%d%d%d", &m1, &m2, &m3, &m4, &m5);

        calc_total_average(m1, m2, m3, m4, m5);   <--- called and passed 5 AP.
}
```

** We must have to individually declare every formal-parameter with personalized datatype.

_____ calc_total_average(int a, int b, int c, int d, int e) <--- is valid

{ ... }

_____ calc_total_average(int a,b,c,d,e) <--- is not valid

{ ... }

==========

==========

Program-2:

Write a program from following instructions.

- Define a function that receives radius of a circle as parameter and calculates area and circumference of circle.
- Define main() method that receives radius of circle from the user and pass it to above defined function.

==========

==========

26-7-21(1)

#include<stdio.h>

_____ calc_area_cir(float r)

{

float a, c;

```
        a = 3.14 * r * r;
        c = 2 * 3.14 * r;

        printf("Area of circle is : %f", a);
        printf("Circumference of circle is : %f", c);
}

void main()
{
        float rad;

        printf("Enter radius of circle : ");
        scanf("%f", &rad);

        calc_area_cir(rad);
}
```

==========

==========

* We can pass any numbers of parameters to a function.
* The parameters which are passed while calling ----> ACTUAL PARAMETERS/ARGUMENTS
* The parameters which receives actual parameters ----> FORMAL PARAMETERS/ARGUMENTS

* Any function can call to any other and pass parameters.
* We need individual declaration of FP.

==========

Example-1:

```
#include<stdio.h>
void calc_average(int t)
{
	float av;
	av = t/5.0;
	printf("Average Marks : %f", av);
}

void calc_total(int a, int b, int c, int d, int e)
{
	int tot;
	tot = a + b + c + d + e;
	printf("Total Marks : %d", tot);

	calc_average(tot); <--- or even we can also pass FP (a,b,c,d,e) to calc_average()
}

void main()
{
	int m1, m2, m3, m4, m5;

	printf("Enter marks of 5 subjects\n");
	scanf("%d%d%d%d%d", &m1, &m2, &m3, &m4, &m5);

	calc_total(m1, m2, m3, m4, m5);
}
```

==========
==

==========
==

Program-2:

Write a program from following instructions.

- Define main() method and accept 3 integers from the user.
- Pass these 3 integers to function find_smallest().
- The find_smallest() function finds the smallest among passed FP
- And passes this smallest number to another function square_cube() as AP
- Calculate and print the square-cube of FP in square_cube() fuction.

```
#include<stdio.h>

void square_cube(int n)
{
	int sq, cu;

	sq = n * n;
	cu = n * n * n;

	printf("Square of smallest number is : %d", sq);
	printf("Cube of smallest number is : %d", cu);
}

void find_smallest(int a, int b, int c)
{
	int temp;
```

```
	if(a<b && a<c)
	{
		temp = a;
	}
	else if(b<a && b<c)
	{
		temp = b;
	}
	else if(c<a && c<b)
	{
		temp = c;
	}

	square_cube(temp);  <--- call to square_cube() with smallest value as AP
}

void main()
{
	int n1, n2, n3;

	printf("Enter 3 values\n");
	scanf("%d%d%d", &n1, &n2, &n3);

	find_smallest(n1,n2,n3);
}
```

==========

==========

* We know that, if we call a function then execution of that function starts from first statement.

```
void car()
{
	printf("Hello 7");
	chair()
	printf("Hello 9");
}

void chair()
{
	printf("Hello 5");
	car();
	printf("Hello 6");
	main();
}

void main()
{
	printf("Hello 1");
	table();
	printf("Hello 2");
	chair();
	printf("Hello 3");
	car();
	printf("Hello 4");
}
```

* We can call to any previously called function or calling function. But make sure that it must not become an infinite times iterating execution.

* If for any logical reason, we need such calling of each other, then we must be aware of the condition/situation or logic to terminate this infinite calling process.

==========
==

==========
==

---- returning a value ----

==========

```
#include<stdio.h>
int addition(int x, int y)
{
	int z;
	z = x + y;
	return z;  <--- returns value of 'z' back to calling statement
}

void main()
{
	int a, b, c;

	printf("Enter 2 values\n");
	scanf("%d%d", &a, &b);

	c = addition(a,b);
	// calling to addition() function with 'a' and 'b' as AP
	// whatever is returned from addition() function will be assigned/received in 'c'

	printf("Addition is : %d", c);
}
```

* The returntype of a function will be ----> the datatype of returned value/variable

* If a function is not returning anything ---> then return-type will be "void"

* A function can have any numbers of parameters; but A function can return only one value.

* A function returns value back to calling statement.

* The use of 'return' should be last statement of any function.

* Because the 'return' will also return the controls back to calling function; along with value.

==========

==========

27-7-21(1)

---- returning value from function ----

==========

* We can also return a value from function, if required.

* For that, we have keyword "return".

```
#include<stdio.h>
int get_square(int n)
{
        int sq;
        sq = n * n;
        return sq;
}
```

```
int get_cube(int n)
{
	return (n*n*n);  <--- even we can do this type of shortcuts
}

void main()
{
	int num, s, c;

	printf("Enter a number : ");
	scanf("%d", &num);

	s = get_square(num);   <--- calling to get_square() and passing 'num' as AP
	c = get_cube(num);

	printf("Square is : %d", s);
	printf("Cube is : %d", c);
}
```

*** IMP ***

1. A function can have any numbers of parameters; but it can return only one value.

2. This returning should be last statement of any function. The statements after return will not execute. [similar to break, continue, exit(0)]

3. The return statement will also return the controls (back to calling function) along with returned value.

4. The datatype of returned value/variable will be the returntype of that function.

5. If a function is not returning then returntype will be 'void'.

	'void' means ---> no return

==========

==========
===

* There is no concern of returning and parameters.

* If a function don't have any parameters, then also it can return.

* If a function has parameters, then it may not return.

* Based on this, there are 4 possibilities of a function:

- a function with parameters and with returning
- a function with parameters but without returning (void)
- a function without parameters and with returning
- a function without parameters and wihtout returning

1. a function with parameters and with returning

```
int addition(int x, int y)
{
    int z;
    z = x + y;
    return z;
}
```

2. a function with parameters but without returning (void)

```
void addition(int x, int y)
{
    int z;
    z = x + y;
    printf("Addition is : %d", z);
```

```
}
```

3. a function without parameters and with returning

```
int addition()
{
        int a,b,c;
        printf("Enter 2 values\n");
        scanf("%d%d", &a, &b);
        c = a + b;
        return c;
}
```

4. a function without parameters and wihtout returning

```
void addition()
{
        int a, b, c;
        printf("Enter 2 values\n");
        scanf("%d%d", &a, &b);
        c = a + b;
        printf("Addition is : %d", c);
}
```

==========

==========

Program-1:

Write a program from following instructions.

- Define a function that receives an integer value as FP and return factorial.
- Define main() method that receives one integer value from the user and passes it to our user-defined function.
- Print the factorial result in main() method.

==========

27-7-21(2)

* Prepare functions from following instructions.
* Don't define main() method and perform calling.
* Only define functions for practice of parameter passing and returning.

Program-1:

Define a function that receives a number as parameter and returns sum of all it's digits.

```
int find_sum_of_digits(int num)
{
	int r, s=0;
	while(num>0)
	{
		r = num % 10;
		s = s + r;
		num = num / 10;
```

```
        }
        return s;
}
```

==========

==========

Program-2:

Define four different functions that and performs all 4 arithmetic operations individually and returns result. These 4 functions has 2 formal parameters of 'int' type.

```
int addition(int x, int y)
{
        int z;
        z = x + y;
        return z;
}

int subtraction(int x, int y)
{
        int z;
        z = x - y;
        return z;
}

int multiplication(int x, int y)
{
        int z;
        z = x * y;
        return z;
```

```
}

float division(int x, int y)
{
	float z;
	z = x / (float)y;
	return z;
}
```

==========

==========

Program-3:

Define a function that receives a character as parameter and returns it's ASCII value.

```
int get_ASCII_value(char c)
{
	int x = c;
	return x;
}
```

==========

==========

Program-4:

Define two different functions; both receives radius of circle as parameter and returns area and circumference of circle individually.

```
float get_area(float rad)
```

```
{
        float a;
        a = 3.14 * rad * rad;
        return a;
}

float get_circ(float rad)
{
        float c;
        c = 2 * 3.14 * rad;
        return c;
}
```

==========

==========

What will be the output of following programs?

a)

```
void main()
{
        printf("\nOnly stupids use C?");
        display();
}

void display()
{
        printf("\nFools too use C!");
        main();
}
```

Output:

infinite times iterates...

==========

b)

```
void main()
{
        printf("\nC to it that C survives");
        main();
}
```

Output:

infinite times iterates...

==========

(c)

```
void main()
{
        int i = 45, c;
        c = check(i);
        printf("\n%d", c);
}

int check(int ch)
{
        if(ch >= 45)
          return(100);
        else
```

```
        return(10);
}
```

Output:

100

==========

(d)

```
void main( )
{
        int i = 45, c;
        c = multiply(i * 1000);
        printf("\n%d", c);
}

void check(int ch)
{
        if(ch >= 40000)
                return(ch / 10);
        else
                return(10);
}
```

Output:

4500

==========

*** CHALLANGE ***

Write a general-purpose function to convert any given year into its roman equivalent. The following table shows the roman equivalents of decimal numbers:

Decimal	Roman
1	i
5	v
10	x
50	l
100	c
500	d
1000	m

Example:

Roman equivalent of 1988 is ---> mdcccclxxxviii

Roman equivalent of 1525 is ---> mdxxv

==========

==========

*** IMP ***

1. We know that, every function has body.

2. We must call a function to exeucte it.

3. Return-type of a function depends on returned value/variable.

4. A function can return only one value.

5. If a function has FP, then we must pass AP to it.

6. If a function returns, then receiving that value is optional.

==========

```
int addition(int x, int y)
{
	int z;
	z = x + y;
	reutrn z;
}

void main()
{
	addition(45, 22);   <--- it is optional to receive returned addition result.
}
```

==========

*** There are many library functions that returns. But we never received it's returned value.

```
int x = printf("Hello all");
x ---> 9

int y = scanf("%d%d", &a, &b);
y ---> 2

char getch()  <--- returns the key you pressed at last

void clrscr()
```

==========

28-7-21(01)

* We know that, a function has 3 parts:
    - definition/body
    - calling
    - declaration

* The body of function contains the set of statements to be executed for that function.

* The calling statement is to invoke/execute/awake the function's execution.

* We are also aware of function's parameters and returning.

* We also know that, function's parameter and returning has no concern.

* If a function returns, then it is optional to receive that value.

==========

==========

* To declare a function, general syntax is:

    returntype function-name(parameter list);

For example:

#include<stdio.h>

```
int addition(int, int);  <--- declaration of function


int addition(int x, int y)   <--- body of function
{
        int z = x + y;
        return z;
}


void main()
{
        ........
        c = addition(a,b);  <--- calling of function
        ........
}
```

*** IMP ***

1. If body of called function is defined/appearing after the body of it's calling function, then declaration is compulsory.

2. If body of called function is defined/appearing above/before the body of it's calling function, then declaration is optional.

3. During this declaration, names of FP is optional. Only datatypes of FP is menatory.

==========

==========

* We have an operator '&' ---> address specifier

"address of" operator

referencing operator

* We also know that, a memory-block has 4 attributes:

- name (identifier)
- datatype
- value
- address

* If we want to print/obtain work-with address of any memory-block then we have operator '&'.

* To print the address of a memory-block ---> %u (unsinged int)

* For example, if we want to print address of a memory-block:

```
#include<stdio.h>
void main()
{
	int a;
	float b;
	char c;

	printf("Address of a is : %u", &a);  ---> &a ---> prints address of 'a'
	printf("Address of b is : %u", &b);  ---> &b ---> prints address of 'b'
	printf("Address of c is : %u", &c);  ---> &c ---> prints address of 'c'

}
```

* We can print the address of memory-block to cross-check whether those memory-blocks are same or it's new.

==========
==========

---- pointer ----

==========

* A POINTER is a memory-block which stores/holds the address of another memory-block.

* A pointer has some additional special characteristics as compared to normal memory-blocks.

* Similar to a normal memory-block, we must also declare a pointer.

* To declare a pointer, the general syntax is:

```
datatype  *pointer_name;
```

- This * indicates a pointer.
- The datatype of pointer must be same as the datatype of variable whose address it is going to store.

* A pointer is ultimately a VARIABLE. [we can overwrite addresses in it.]

* A pointer can store/hold the address of another variable with same datatype.

For example:

```
void main()
{
    int a;
    float b;
    int c;
```

```
        char d;
        int *p;   <--- pointer of type 'int'

        p = &a;  <--- VALID (same datatypes)
        p = &b;  <--- INVALID (different datatypes - CT error)
        p = &c;  <--- VALID (same datatypes)
        p = &d;  <--- INVALID (different datatypes - CT error)

        .............
        .............
}
```

==========

29-7-21(1)

- There are two important operators in case of pointers:

&	"value at address" operator
	"address of" operator
	"referencing" operator
*	"value at address" operator
	"dereferencing" operator

- We can use * operator with pointer in statement; then it behaves as value at address operator.
    declaration me * ----> indicates pointer's declaration
    statement me * ----> indicates "value at address" operator

* A pointer can only hold/store address; It cannot store normal console values.
* A normal variable can hold normal console value; It cannot hold/store address.

* We cannot perform any arithmetic or relational operations on pointer and address.

```
p3 = p1 + p2;  <--- INVALID
p3 = &a + &b;  <--- INVALID
if(&a > &b)  <--- INVALID
if(p1 > p2)  <--- INVALID
```

==========

1 to 65535 <--- address range in Borland C

1 to 4294967295 <--- address range in GCC

==========

==========

* Refer following program.

Example-1:

```
#include<stdio.h>
void main()
{
    int a, b, c;
    int *p1, *p2, *p3;

    p1 = &a,
    p2 = &b;
```

```
        p3 = &c;

        printf("Enter 2 values\n");
        scanf("%d%d", &a, &b);

        c = a + b;
        OR
        c = (*p1) + (*p2);
        OR
        *p3 = a + b;
        OR
        *p3 = (*p1) + (*p2);

        printf("Addition is : %d", c);
        OR
        printf("Addiriont is : %d", *p3);

}
```

==========

==========

* Till date we were performing every operation by using NAME of variables. But if we use pointer, then we can also perform same operation by using ADDRESS of variables.

==========

==========

Example-2:

```
#include<stdio.h>
void addition(int *x, int *y)  <--- pointers as FORMAL PARAMETER
{
        int c;
        c = (*x) + (*y);  <--- performing addition based on address of 'a' and 'b'
        printf("Addition is : %d", c);
}

void main()
{
        int a, b;

        printf("Enter 2 values\n");
        scanf("%d%d", &a, &b);

        addition(&a, &b);  <--- passed address as ACTUAL PARAMAETER
}
```

Example-3:

```
#include<stdio.h>
int addition(int *x, int *y)  <--- pointers as FORMAL PARAMETER
{
        int c;
```

```
	c = (*x) + (*y);  <--- performing addition based on address of 'a' and 'b'
	return c;  <--- returning resulting addition
}

void main()
{
	int a, b, s;

	printf("Enter 2 values\n");
	scanf("%d%d", &a, &b);

	s = addition(&a, &b);  <--- passed address as ACTUAL PARAMAETER
	printf("Addition is : %d", s);
}
```

Example-4:

```
#include<stdio.h>
void operations(int x, int y, int *p1, int *p2, int *p3, float *p4)
{
	*p1 = x + y;
	*p2 = x - y;
	*p3 = x * y;
	*p4 = x / (float)y;
}

void main()
```

```
{
        int n1, n2, a, s, m;
        float d;

        printf("Enter 2 values\n");
        scanf("%d%d", &n1, &n2);

        operations(n1, n2, &a, &s, &m , &d); <--- passed 2 values and 4 addresses as AP

        printf("Addition is : %d", a);
        printf("Subtraction is : %d", s);
        printf("Multiplication is : %d", m);
        printf("Division is : %f", d);
}
```

==========

==========

* a pointer provides external controls.

* In above program, we did not return, but still all 4 values are filled (indirectly) by using pointer and "value at address" operator.

==========

==========

29-7-21(2)

* We know that, we can pass values as AP and we can also pass address as AP.

Skeleton-1:

```
#include<stdio.h>
____ addition(int x, int y)
{
        ........
        ........
}

void main()
{
        ........
        ........
        addition(a,b); <--- passing values of 'a' and 'b' as AP
        ........
        ........
}
```

- In above skeleton, we are passing values as AP ---> call by (passing) value

==========

Skeleton-2:

```
____ addition(int *x, int *y)
{
        ........
        ........
}
```

```
void main()
{
        ........
        ........
        addition(&a, &b); <--- passing address of 'a' and 'b' as AP
        ........
        ........
}
```

- In above skeleton, we are passing address as AP ---> call by (passing) reference

==========

==========

* Difference between Call-by-Value and Call-by-Reference.

call by value	call by reference
1. We pass value of variable as AP.	1. We pass address of variable as AP.
2. We declare normal variables as FP.	2. We declare pointers as FP.
3. A function can return only one value.	3. A function can return multiple values. (indirectly).

*** If logically required, then we can also use combination of CBV and CBR for a single function.

==========

==========

Program-1:

Write a function which receives a float and an int from main(), finds the product of these two and returns the product which is printed through main().

Program-2:

Write a function that receives 5 integers and (indirectly) returns the sum, average and standard deviation of these numbers. Call this function from main() and print the results in main().

Program-3:

Write a function that receives marks received by a student in 3 subjects and returns the average and percentage of these marks. Call this function from main() and print the results in main().

==========

* What will be the output of following programs?

a)

```
void main()
{
        int i = 5, j = 2;
        junk(i, j);
        printf("%d %d", i, j);
```

```
}

void junk(int i, int j)
{
        i = i * i ;
        j = j * j ;
}
```

Output:

5 2

==========

b)

```
void main()
{
        int i = 5, j = 2;
        junk(&i, &j);
        printf("%d %d", i, j);
}
void junk(int *i, int *j)
{
        *i = *i * *i;
        *j = *j * *j;
}
```

Output:

25 4

==========

c)

```
void main( )
{
        int i = 4, j = 2;
        junk(&i, j);
        printf("%d %d", i, j);
}
void junk(int *i, int j)
{
        *i = *i * *i;
        j = j * j;
}
```

Output:

16 2

==========

d)

```
void main()
{
        float a = 13.5;
        float *b, *c;
        b = &a; <--- suppose address of a is 1006
        c = b;
        printf("%u %u %u", &a, b, c);
        printf("%f %f %f %f %f", a, *(&a), *&a, *b, *c);
}
```

Output:

1006 1006 1006

13.5 13.5 13.5 13.5

==========

30-7-21(1)

Program-1:

Write a function which receives a float and an int from main(), finds the product of these two and returns the product which is printed through main().

Program-2:

Write a function that receives 5 integers and (indirectly) returns the sum, average and standard deviation of these numbers. Call this function from main() and print the results in main().

Program-3:

Write a function that receives marks received by a student in 3 subjects and returns the average and percentage of these marks. Call this function from main() and print the results in main().

==========

* We know that, every memory-block has address. We can store this address into a pointer.

* Then by using value at address operator (*), we can refer to original memory-block's value.

* As a benefit, we can externally use and fill a memory-block's value (from another function).

* To achieve this application, we pass address of memory-block as AP. This mechanism is called as "call by reference".

==========

==========

*** IMP ***

1. A pointer also has addreess. (we can print it by using %u).

2. A pointer can store address of a normal memory-block; but a pointer cannot store address of another pointer. (will show CT error).

3. To store address of a pointer, we need "double pointer". To declare 'double pointer' we use ** during declaring.

4. A double pointer also has address. To store address of a double pointer, we need "triple pointer". To declare a 'triple pointer' we use *** while declaring.

5. In similar way, we can continue the pointer's 'rank' and capacity by increasing numbers of *.

6. The datatype must match.

7. A pointer is variable. We cannot store multiple addresses in a pointer. A POINTER CAN STORE ONLY ONE ADDRESS.

==========

==========

* Rerfer following program.

```
#include<stdio.h>
void increment(int **x, int **y)  <--- receiving in double pointers as FP
{
	(**x)++;  <--- indirectly increments 'a'
	(**y)++;  <--- indirectly increments 'b'
}

void main()
{
	int a, b;

	printf("Enter two values : ");
	scanf("%d%d", &a, &b);

	int *p1 = &a;
	int *p2 = &b;

	increment(&p1, &p2);  <--- passing address of pointers as AP

	printf("Incremented values are : a = %d and b = %d", a, b);
}
```

==========

==========

* Refer following another program.

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

```
void swap(int *p1, int *p2)
{
	int t;
	t = *p1;
	*p1 = *p2;
	*p2 = t;
}

void main()
{
	int a, b;
	printf("Enter 2 values\n");
	scanf("%d%d", &a, &b);

	printf("Before swapping, a = %d and b = %d", a, b);

	swap(&a, &b); <--- passed address of a and b, for external changing their values

	printf("After swapping, a = %d and b = %d", a, b);
}
```

==========

==========

31-7-21(1)

* In call-by-value, we pass value of variable as AP; and receive them in normal variables as FP. [can return only one value]

* In call-by-reference, we pass address of variable as AP; and receive them in pointers as FP. [can indirectly return multiple values]

==========

==========

* A program without main() method is possible. Such program will compile successfully; but such programs will not execute.

* If logically required, (For further use) we can prepare a user-defined header file also.

* Follow the following steps:

step 1: Define required user-defined functions in a file.

step 2: Don't define main() method in it.

step 3: Save this program with (.h) extension. [in "include" folder]

step 4: Whenever required, we can include our user-defined header-file in any further program and use it as normal functions.

***

- Remember that, use of global-variables should be avoided.
- Better to declare global constants.
- We can define any numbers of user-defined header-files.
- We can give any names to header-file.

==========

==========

* We can prepare a single header-file containing the commonly used header-files. And include this single (all-in-one) header file.

* But this is not a good option. This will unnecessarily waste compile-time and length of program.

==========

==========

*** IMP ***

- We know that, by using keyword "return" we can return a value.
- And based on datatype of returned value, we decide return-type of that function.
- We know that:
    - by using keyword 'break' ---> we can terminate a loop
    - by using exit() function ---> we can termiante a program's execution
    - by using keyword 'return' ---> we can terminate function. It returns controls.

Skeleton-1:

```
int abcd()
{
    int a;
    .....
    .....
    return a;  <--- returns controls back to calling f/n (along with value of 'a')
}
```

Skeleton-2:

```
void abcd()
{
    .......
```

```
        .......
        return;  <--- returns controls back to calling function
        .......
        .......
}
```

==========

==========

Example-1:

```
#include<stdio.h>
void abcd(int x)
{
   printf("abcd function starts\n");

   if(x>10)
   {
      return;  // only returns controls back to calling function
   }

   printf("abcd function ends\n");
}

void main()
{
   printf("main starts\n");
   abcd(__);  // we can also pass console values
   printf("main ends\n");
```

}

==========

==========

*** IMP ***

- We can also pass console values as AP.

- If we execute above program:

Output-1: ---> if 15 is passed as AP

---

main starts

abcd function starts

main ends

Output-2: ---> if 5 is passed as AP

---

main starts

abcd function starts

abcd function ends

main ends

==========

==========

31-7-21(2)

---- Recursion ----

==========

* We know that, recursion means a function calls to itself.

```
#include<stdio.h>
void facto(int n, int i, int f)
{
	f = f * i;
	i++;
	if(i<=n)
	{
		facto(n, i, f);
	}
	printf("\nFactorial is : %d", f);
	printf("\nFunction ends");
}

void main()
{
	facto(5, 1, 1);
}
```

==========

1-8-21(01)

* We know that, "recursion means a function calls to itself".

* We can solve/replace many loop's logic via recursion.

* There are some steps to convert a loop's logic into recursive function.

step 1: convert that loop's logic into while-loop.

step 2: Write the loop's statements as recursive function's statements.

step 3: Use loop's condition as recursive call's condition. [if-statement]

step 4: Every variable that we used in loop's body must be used as parameter.

step 5: Every initial values must be passed while making first-call to that recursive function.

** It is obvious, that a recursive function takes more memory and more execution time as compared to a loop.

** In general situations, always prefer loop.

* But the situations where you want to grab/retrieve/use previous iteration's value; and you want a logic where you want to go to previous iterations then you must prefer recursive function.

==========

Example-1:

Print 1 to 10 using while-loop.

```
void main()
{
	int i;
	i = 1;
	while(i<=10)
	{
		printf("Hello %d", i);
```

```
                i++;
        }
}
```

Print 1 to 10 using recursive function.

```
void printing(int i)
{
        printf("Hello %d", i);
        i++;
        if(i<=10)
        {
                printing(i);
        }
}

void main()
{
        printing(1);
}
```

==========

==========

*** If you want to print the result after loop (like factorial, sum of digits, reversed number, largest element, etc...); In such cases use an else-block and print the result in that else-block.

Example-2:

Printing factorial of entered number by using while-loop.

```
void main()
{
	int i, long f=1;
	int num;

	printf("Enter a number : ");
	scanf("%d", &num);

	i=1;
	while(i<=num)
	{
		f = f * i;
		i++;
	}

	printf("Factorial is %ld", f); <--- must be done in else-block (recursive call's)
}
```

Printing factorial of entered number by using recursive function.

```
void facto(int num, int i, long f)
{
	f = f * i;
	i++;
	if(i<=num)
	{
		facto(num, i, f);
	}
```

```
        else
        {
                printf("Factorial is %ld", f);
        }
}

void main()
{
        int num;

        printf("Enter a number : ");
        scanf("%d", &num);
        facto(num, 1, 1);
}
```

==========

==========

Program-3:

Prepare a recursive function to print multiplication table of entered number.

Printing multiplication table using while-loop.

```
void main()
{
        int i, num, m;

        printf("Enter a number : ");
        scanf("%d", &num);
```

```
	i=1;
	while(i<=10)
	{
		m = num * i;
		printf("%d x %d = %d\n", num, i, m);
		i++;
	}
}
```

Printing multiplication table using recursive function.

```
void print_table(int num, int i, int m)
{
	m = num * i;
	printf("%d x %d = %d\n", num, i, m);
	i++;
	if(i<=10)
	{
		print_table(num, i, m);
	}
}

void main()
{
	int num;

	printf("Enter a number : ");
	scanf("%d", &num);
```

```
        print_table(num, 1, 1);
}
```

==========

==========

Program-4:

Prepare a recursive function to print sum of all digits of entered number.

Printing sum of all digits using while-loop.

```
void main()
{
        long num;
        int r, sum=0;

        printf("Enter any number : ");
        scanf("%ld", &num);

        while(num>0)
        {
                r = num % 10;
                sum = sum + r;
                num = num / 10;
        }

        printf("Sum of all digits is : %d", sum);
}
```

Printing sum of all digits using recursive function.

```
void find_sum_of_digits(long num, int r, int sum)
{
	r = num % 10;
	sum = sum + r;
	num = num / 10;
	if(num>0)
	{
		find_sum_of_digits(num, r, sum);
	}
	else
	{
		printf("Sum of all digits is : %d", sum);
	}
}

void main()
{
	long num;

	printf("Enter any number : ");
	scanf("%ld", &num);

	find_sum_of_digits(num, 1, 0);
}
```

==========

==========

2-8-21(02)

* In some logical situations, we can also call a function from another normal executable statement.
* But the only condition is, such called functions must return something.
* Refer following examples.

Example-1:

```
#include<stdio.h>
int addition(int x, int y)
{
	int z;
	z = x + y;
	return z;
}

void main()
{
	printf("Addition is : %d", addition(45, 20) );
}
```

==========

==========

Example-2:

```
#include<stdio.h>
int multip(int x, int y)
{
	return (x*y);
```

```
}

int addition(int x, int y)
{
	return (x+y);
}

void main()
{
	printf("Addition of 45 and 20 is : %d", addition(45,20) );
	printf("Multiplication of 15 and 5 is : %d", multip(15, 5) );
}
```

==========

Example-3:

```
#include<stdio.h>
int multip(int x, int y)
{
	return (x*y);
}

int addition(int x, int y)
{
	return (x+y);
}

void main()
{
	int c;
```

```
	c = addition(45, 20) + multip(15, 3) ;
	printf("Result is : %d", c);
}
```

* Above function will first call to addition() and retruned value is placed in buffer.
* Then multip() function will be called and returned value is placed in buffer.
* Then the addition of both returned value (in buffer) will be assigned into 'C'.

==========

Example-4:

```
#include<stdio.h>
int addition(int x, int y)
{
	return (x+y);
}

int multip(int x, int y)
{
	return (x*y);
}

void main()
{
	int a,b;

	printf("Enter 2 values\n");
	scanf("%d%d", &a, &b);

	if( addition(a,b) > 100)
```

```
        {
                .....
        }
        if(mutip(a,b) != 0)
        {
                .....
        }

}
```

==========

Example-5:

```
#include<stdio.h>
int addition(int x, int y)
{
        return (x+y);
}

void main()
{
        int c = addition( addition(10,20), addition(30,40) );
        printf("%d", c);
}
```

* The statement will first call to addition(10,20) and receives returned value in buffer.
* Then calls to addition(30,40) and receives returned value in buffer.
* Then calls to outer addition() with both buffer values as AP.
* Then the final returned result is assigned into 'c'.

==========

```
#include<stdio.h>
void main()
{
        printf("%d", printf("Hello") );
}
```

Output:

Hello5

==========

2-8-21(2)

* The logical concepts like functions, arrays, structures, pointers will not be helpful in minimizing memory.

* But these are advantageous in program length and debugging and programmer's efforts.

==========

==========

---- Array ----

==========

* An array is set of similar datatypes.

* If you want to perform similar operations on multiple values; then don't declare those values/variables in individual memory-blocks. But declare them as an array.

* To declare an array, general syntax is:

datatype arr_name[SIZE];

* For example:

int a[5];

This means one set of 5 'int' and it's collective name is 'a'

float b[10];

This means one set of 10 'float' and it's collective name is 'b'

==========

*** IMP ***

1. We can declare an array of any datatype.
2. Array SIZE is also referred as "subscript"
3. The same 5 naming rules for array's name.
4. The SIZE of array, must be non-zero positive integer.
5. In a program, we can declare any numbers of arrays.
6. Variable sized array is not valid [based on compiler]

For example:

```
void main()
{
    int n;

    printf("Enter array size you want : ");
    scanf("%d", &n);

    int a[n];  <--- variable sized array  [depends on compiler]
```

.....

.....

}

==========

==========

* Even we declare an array, it will not be helpful in minimizing memory.

int a[5]; ---> 5 blocks, each of 2 bytes ---> total 10 bytes

int a,b,c,d,e; ---> 5 blocks, each of 2 bytes ---> total 10 bytes

float b[5]; ---> 5 blocks, each of 4 bytes ---> total 20 bytes

float a,b,c,d,e; ---> 5 blocks, each of 4 bytes ---> total 20 bytes

* If you are willing to input 10 values and perform following operations:

first 2 values --- addition

next 2 values --- subtr

next 2 values --- mult

next 2 values --- division

next 2 values --- modulus

then never ever never ever prefer array. Better you go with 10 different variables.

==========

==========

* An array has a special characteristic of assigning a serial-number to every-block.

* This serial number is called as "index number" of that block.

* This indexing starts from ZERO (0)

* Therefore every block is identified with this index-number.

For example:

```
void main()
{
        int a[5]; // array will have 5 'int' type blocks. Will have index from 0 to 4.
        ....        // These blocks are identified as : a[0], a[1], a[2], a[3], and a[4]
        ....
}
```

* The [] is called as ----> "array indexing" operator

==========

==========

3-8-21(1)

* First thing is, the use of array is not dependent on numbers of inputs. But it is depenedent on the operation.

* To declare an array, general syntax is:

```
datatype  name[size];
```

For example:

```
int a[50];
```

```
        // delcares an array of SIZE 50 and named as 'a'
        // total memory ---> 50 x 2bytes = 100 bytes

    float b[26];
        // declares an array of SIZE 26 and named as 'b'
        // total memory ---> 26 x 4 = 104 bytes
```

* The blocks of array will have an index-number. That starts from ZERO (0)

* Because of this index-number, we can identify each block of array.

* To identify a block in array ----> arr_name[INDEX]

* The [] is ---> "array indexing" operator

For example:

```
    int a[5];
    // will have index numbers from 0 to 4
    // and every element will be at ---> a[0], a[1], a[2], a[3] and a[4]
```

* Similar to a normal variable's name, these index-numbers are to identify each array-block for operation.

For example:

```
void main()
{
    int a[5];
    int tot;
    float av;
```

```
        printf("Enter 5 elements\n");
        scanf("%d%d%d%d%d", &a[0], &a[1], &a[2], &a[3], &a[4]);

        tot = a[0] + a[1] + a[2] + a[3] + a[4];
        av = tot/5.0;

        printf("\nSum of array elements is : %d", tot);
        printf("\nAverage of array elements is : %f", av);

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

* Even if logically required, then we can also do operations and assign values at any index.

```
        a[3] = a[1] + a[0];  <--- can store expression results

        a[2] = a[1] * 15;  <--- can store expression results

        if(a[3] > 95) <--- can compare
        { ... }

        if(a[2] > b[5]) <--- can compare
        { ... }

        addition(a[2], a[3], b[5]);   <--- can pass as AP
```

==========

==========

* We cannot perform any arithmetic and relation operation on entire array. But we can do the same operations on array-blocks [elements/index]

For example:

```
int a[10], b[10], c[10];


c = a + b;  <--- is not valid


if(a>b)  <--- is not valid
{ ... }
```

==========

==========

** From above examples and discussion, remember that it is not mendatory to use loop for every array operation.

** From above examples and discussion, it is also proved that we can also use single/individual elements of array by referring it's index-number.

** But when it comes to multiple values and their operations, we PREFER a loop to minimize programmer's efforts.

Example-1:

```
#include<stdio.h>
void main()
{
    int a[10];
```

```
	int i;

	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
	}

	printf("Array elements are\n");
	for(i=0; i<=9; i++)
	{
		printf("%d\n", a[i]);
	}
}
```

==========

==========

Example-2:

```
#include<stdio.h>
void main()
{
	int a[10];
	int i;

	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
```

```
	}

	printf("\nArray elements are\n");
	for(i=0; i<=9; i++)
	{
		printf("At index %d element is : %d\n", i, a[i]);
	}
}
```

==========

==========

*** Don't think about following situation.

```
void main()
{
	int m1, m2, m3, m4, m5, m6, m7;  <--- indivudial 7 variables.
	int i;

	printf("Enter marks of 7 subjects\n");
	for(i=1; i<=7; i++)
	{
		scanf("%d", &mi);  <--- shows CT error.
	}

	.....
	.....
}
```

* It will show CT error.

* This facility of replacing index-number in a loop, is only for ARRAY.

* In above example, you must scan them individually.

==========

==========

Example-4:

```
#include<stdio.h>
void main()
{
	int a[10];
	int i,j;

	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);  <--- taking input by referring 'i' as counter
	}

	printf("Array elements are\n");
	for(j=0; j<=9; j++)
	{
		printf("%d\n", a[j]); <--- showing values by referring 'j' as counter
	}
}
```

* Above example works correctly as expected.

* To work on array elements, index-number matters; the counter does not matter.

==========
===

3-8-21(2)

* To declare an array, general syntax is:

datatype arr_name[SIZE];

* It is not mendatory to use loop for array. But we prefer to use loop for ease of operations.

* In case of array, the loop-counter variable does not matter. Only index-number matters.

- Take input by using 'i'
- Perform operations by using 'j'
- Print the array by using 'k'

* We can do any operations on array elements; but we cannot do any operations on entiere array.

==========
====

==========
====

Program-1:

Write a program to declare an array of size 10. Accept and display elements of it.

```
#include<stdio.h>
void main()
{
	//step 1: Declaring array
	int a[10], i;
```

```
	//step 2: Accept elements
	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
	}

	//step 3: Display elements
	printf("\nArray elements are\n");
	for(i=0; i<=9; i++)
	{
		printf("%d\n", a[i]);
	}
}
```

==========

Program-2:

Write a program to accept array elements and display them in reverse order.

```
#include<stdio.h>
void main()
{
	// step 1: Declaring array
	int a[10], i;

	// step 2: Accepting elements
	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
```

```
		scanf("%d", &a[i]);
	}

	printf("\nElements in reverse order are\n");
	for(i=9; i>=0; i--)
	{
		printf("%d\n", a[i]);
	}
}
```

==========

Program-3:

Write a program to accept array elements and print the elements at odd index only.

```
#include<stdio.h>
void main()
{
	int a[20], i;

	printf("Enter 20 array elements\n");
	for(i=0; i<=19; i++)
	{
		scanf("%d", &a[i]);
	}

	printf("Elements at odd index are\n");
	for(i=0; i<=19; i++)
	{
		if(i%2 == 1)
```

```
		{
			printf("At index %d element is %d\n", i, a[i]);
		}
	}

	--- OR ---

	for(i=1; i<=19; i=i+2)
	{
		printf("At index %d element is %d\n", i, a[i]);
	}
}
```

==========

Program-4:

Write a program to accept array elements; and print only even elements of it.

```
#include<stdio.h>
void main()
{
	int a[20], i;

	printf("Enter 20 array elements\n");
	for(i=0; i<=19; i++)
	{
		scanf("%d", &a[i]);
	}

	printf("Even elements are\n");
```

```
	for(i=0; i<=19; i++)
	{
		if(a[i]%2 == 0)
		{
			printf("At index %d element is %d\n", i, a[i]);
		}
	}
}
```

==========

Program-5:

Write a program to accept array elements from the user and print the total count of even elements and odd elements entered by the user.

```
#include<stdio.h>
void main()
{
	int a[20], i;
	int e_count=0, o_count=0;

	// step 1: Accepting array elements
	for(i=0; i<=19; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: checking even-odd counts
	for(i=0; i<=19; i++)
	{
```

```
            if(a[i]%2 == 0)
            {
                  e_count++;
            }
            else
            {
                  o_count++;
            }
      }

      // step 3: display counts
      printf("Total EVEN elements : %d", e_count);
      printf("Total ODD elements : %d", o_count);
}
```

==========

Program-6:

Write a program to accept array elements and over-write them by their squares.

```
#include<stdio.h>
void main()
{
      int a[20], i, sq;

      // step 1: Accepting array elements
      printf("Enter 20 array elements\n");
      for(i=0; i<=19; i++)
      {
            scanf("%d", &a[i]);
```

```
	}

	// step 2: over-write elements by their square
	for(i=0; i<=19; i++)
	{
		a[i] = a[i] * a[i];
	}

	--- OR ---

	for(i=0; i<=19; i++)
	{
		sq = a[i] * a[i];
		a[i] = sq;
	}

	// step 3: Display updated elements
	printf("Updated array elements are\n");
	for(i=0; i<=19; i++)
	{
		printf("%d\n", a[i]);
	}
}
```

==========

Program-7:

Write a program to accept elements for one array. And copy these elements into another array. [We cannot assign/copy entire array; we have to assign/copy element-by-element]

```
#include<stdio.h>
void main()
{
	int a[20], b[20], i;

	// step 1: Accepting for first array 'a'
	printf("Enter 20 elements\n");
	for(i=0; i<=19; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: Assigning/Copying element-by-element
	for(i=0; i<=19; i++)
	{
		b[i] = a[i];
	}

	// step 3: Dispaly 'a' or/and 'b'
	.......
	.......

}
```

==========

4-8-21(1)

Program-1:

Write a program to accept elements of one array and copy them in another array; in reverse order. [The order of elements of both arrays must be opposite]

```
#include<stdio.h>
void main()
{
	int a[10], b[10];
	int i,j;

	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: main logic to copy in opposite
	j=9;
	for(i=0; i<=9; i++)
	{
		b[j] = a[i];
		j--;
	}

	--- OR ---

	for(i=0, j=9; i<=9; i++, j--)
	{
		b[j] = a[i];
	}

	// step 3: Display both arrays
```

```
	printf("Elements of both arrays are\n");
	for(i=0; i<=9; i++)
	{
		printf("At index %d : %d - %d\n", i, a[i], b[i]);
	}
}
```

======= OR =======

```
#include<stdio.h>
void main()
{
	int a[10], b[10];
	int i,j;

	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: main logic to copy in opposite
	for(i=0; i<=9; i++)
	{
		b[i] = a[9-i];
	}

	// step 3: Display both arrays
	printf("Elements of both arrays are\n");
	for(i=0; i<=9; i++)
	{
```

```
                printf("At index %d : %d - %d\n", i, a[i], b[i]);
        }
}
```

==========

Program-2:

Write a program to accept elements for 2 arrays and add them into third array at same index.

```
        c[0] = a[0] + b[0]
        c[1] = a[1] + b[1]
        c[2] = a[2] + b[2]
        .....
        .....
        c[9] = a[9] + b[9];
```

```
#include<stdio.h>
void main()
{
        int a[10], b[10], c[10];
        int i,j;

        printf("\nEnter elements for first array\n");
        for(i=0; i<=9; i++)
        {
                scanf("%d", &a[i]);
        }

        printf("\nEnter elements for second array\n");
        for(i=0; i<=9; i++)
```

```
	{
		scanf("%d", &b[i]);
	}

	for(i=0; i<=9; i++)
	{
		c[i] = a[i] + b[i];
	}

	printf("\nElements of resulting array are\n");
	for(i=0; i<=9; i++)
	{
		printf("%d\n", c[i]);
	}
}
```

==========

Program-3:

Write a program to accept elements of an array and print sum of all elements.

```
#include<stdio.h>
void main()
{
	int a[10], sum=0, i;

	printf("Enter elements of an array\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
```

```
        }

        for(i=0; i<=9; i++)
        {
                sum = sum + a[i];
        }

        printf("Sum of all elements is : %d", sum);
}
```

==========

CHALLANGE:

Program-4:

Write a program to accept 10 elements of an array. And reverse the array elements. [without using another array]

a[0] ---> a[9] a[9] ---> a[0]

a[1] ---> a[8] a[8] ---> a[1]

Program-5:

Write a program to accept a 5 digit number from the user and fill it's each digit into an array of size 5.

For example:

input ---> 71462

output ---> a[4] ---> 2

a[3] ---> 6

a[2] ---> 4

a[1] ---> 1

a[0] ---> 7

==========

==========

4-8-21(2)

Solved Homework:

Program-4:

Write a program to accept 10 elements of an array. And reverse the array elements. [without using another array]

```
#include<stdio.h>
void main()
{
	int a[10], i, temp;

	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
	}
```

```
        for(i=0; i<=4; i++)
        {
                temp = a[i];
                a[i] = a[9-i];
                a[9-i] = temp;
        }

        printf("\nReversed elements are\n");
        for(i=0; i<=9; i++)
        {
                printf("%d\n", a[i]);
        }
}
```

==========

==========

Program-5:

Write a program to accept a 5 digit number from the user and fill it's each digit into an array of size 5.

```
#include<stdio.h>
void main()
{
        long num, r, ind=4;
        int a[5];

        printf("Enter a number : ");
        scanf("%ld", &num);  ---> 19842
```

```
        while(num>0)
        {
                r = num % 10;
                a[ind] = r;
                num = num / 10;
                ind--;
        }

        printf("\nArray elements are\n");
        for(i=0; i<=4; i++)
        {
                printf("At index %d element is : %d\n", i, a[i]);
        }
}
```

==========

==========

HOMEWORK:

Program-3:

Write a program to accept 20 elements of an array and print the individual sum of even elements and odd elements of that array.

Program-4:

Write a program to accept 20 elements of an array and swap the elements of even index with the exact next element at odd index.

a[0] <---> a[1]

a[2] <---> a[3]

a[4] <---> a[5]

a[6] <---> a[7]

.....

.....

a[18] <---> a[19]

*** CHALLANGE ***

Program-5:

Write a program to accept elements of 2 arrays. And print the common elements of them.

==========

==========

5-8-21(1)

Program-1:

Write a program to accept 25 elements of an array; and also input a number to search in it. Check whether that desired element is present in that array or not.

[Search for specific element in array]

#include<stdio.h>

void main()

```
{
	int a[25], num, flag=1;

	// step 1: accept 25 elements
	for(i=0; i<=24; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: accept number to be searched in array
	printf("\nEnter element to search for : ");
	scanf("%d", &num);

	// step 3: logic to search 'num' in array 'a'
	for(i=0; i<=24; i++)
	{
		if(a[i] == num)
		{
			flag=2;
			break;
		}
	}

	// step 4: print the decision
	if(flag==2)
	{
		printf("Element found");
	}
	if(flag==1)
	{
		printf("Element not found");
```

```
	}
}
```

==========

Program-2:

Write a program to accept 25 elements of an array; and also input the number to search in that array; And print the total occurrence count of that number. [Total count: How many times that number appeared]

```
#include<stdio.h>
void main()
{
	int a[25], num, count=0;

	// step 1: accept 25 elements
	for(i=0; i<=24; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: searching for element and count
	for(i=0; i<=24; i++)
	{
		if(a[i]==num)
		{
			count++;
		}
	}

	// step 3: print count
```

```
        printf("Total occurrence of element %d is : %d times", num, count);
}
```

==========

==========

* What will be the output of following programs?

a)

```
void main()
{
        int num[26],temp;

        num[0] = 100;
        num[25] = 200;

        temp = num[25];
        num[25] = num[0];
        num[0] = temp;

        printf ("%d %d", num[0], num[25]);
}
```

Output:

200, 100

==========

b)

```
void main()
```

```
{
	int array[26], i;
	for(i=0; i<=25; i++)
	{
		array[i] = 'A' + i;
		printf("%d  %c \n", array[i], array[i]);
	}
}
```

Output:

It fills array with 65 to 90

65 A

66 B

67 C

68 D

...

...

90 Z

==========

c)

```
void main()
{
	int sub[50], i;
	for(i=0; i<=48; i++)
	{
		sub[i] = i;
		printf("%d \n", sub[i]);
	}
```

}

Output:

0

1

2

..

..

48

==========

Homework:

Program-3:

Write a program to DECLARE an array of 25. And directly print the elements to check whether it has GV or ZERO.

Program-4:

Write a program to accept 25 elements of an array; and check whether all elements are same or not.

==========

==========

5-8-211(2)

Solved Homework:

Program-1:

Write a program to accept 20 elements of an array and print the individual sum of even elements and odd elements of that array.

Program-2:

Write a program to accept 20 elements of an array and swap the elements of even index with the exact next element at odd index.

a[0] <---> a[1]

a[2] <---> a[3]

a[4] <---> a[5]

a[6] <---> a[7]

.....

.....

a[18] <---> a[19]

*** CHALLANGE ***

Program-3:

Write a program to accept elements of 2 arrays. And print the common elements of them.

Program-4:

Write a program to DECLARE an array of 25. And directly print the elements to check whether it has GV or ZERO.

*** IMP ***

By-default array appears with GARBAGE VALUE.

Program-5:

Write a program to accept 25 elements of an array; and check whether all elements are same or not.

==========

==========

** CHALLANGE **:

Program -6:

Prepare a program from following instructions.

- Declare an array of size 5; and accept 4 digits elements in it.
- Declare another array of size 10; fill it with 2 digit combinations of elements of first array.
- Print resulting second array.

Program-7:

Prepare a program from following instructions.

- Declare first array of size 5. Accept elements for it.

- Declare second array of size 5. Accept elements for it.

- Declare third array of size 10. And Merge first two arrays into third array.

In third array ---> [0] to [4] ---> elements of 'a'

[5] to [9] ---> elements of 'b'

==========

==========

Program 1, 2, 5 ---> tomorrow mornining

Program 6, 7 ---> tomorrow evening

==========

==========

* Python considers a function as VARIABLE/OBJECT. Therefore we can pass a function as AP.

* But in other languages, we cannot pass a function as AP. Because other languages considers a function as set/group of statements.

==========

==========

6-8-21(1)

Solved Homework:

Program-1:

Write a program to accept 20 elements of an array and print the individual sum of even elements and odd elements of that array.

```
#include<stdio.h>
void main()
{
	int a[20], i;
	int e_sum=0, o_sum=0;

	// step 1: Accepging 20 elements
	printf("Enter 20 elements\n");
	for(i=0; i<=19; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: performing sum of even-odd elements
	for(i=0; i<=19; i++)
	{
		if(a[i]%2 == 0)
		{
			e_sum = e_sum + a[i];
		}
		else
		{
			o_sum = o_sum + a[i];
```

```
            }
        }

        // step 3: Displaying sums
        printf("Sum of EVEN elements is : %d", e_sum);
        printf("Sum of ODD elements is : %d", o_sum);
}
```

==========

Program-2:

Write a program to accept 20 elements of an array and swap the elements of even index with the exact next element at odd index.

a[0] <---> a[1]

a[2] <---> a[3]

a[4] <---> a[5]

a[6] <---> a[7]

.....

.....

a[18] <---> a[19]

```
#include<stdio.h>
void main()
{
        int a[20], i;
        int temp;

        // step 1: Accepging 20 elements
        printf("Enter 20 elements\n");
        for(i=0; i<=19; i++)
```

```
	{
		scanf("%d", &a[i]);
	}

	// step 2: Performing swapping
	for(i=0; i<=19; i=i+2)
	{
		temp = a[i];
		a[i] = a[i+1];
		a[i+1] = temp;
	}

	// step 3: Displaying updated array elements
	for(i=0; i<=19; i++)
	{
		printf("At index %d element is %d\n", i, a[i]);
	}
}
```

==========

Program-3:

Write a program to accept 25 elements of an array; and check whether all elements are same or not.

```
#include<stdio.h>
void main()
{
	int a[25], i;
	int flag=1;
```

```
	// step 1: Accepging 25 elements
	printf("Enter 25 elements\n");
	for(i=0; i<=24; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: checking whether all elements are same or not
	for(i=1; i<=24; i++)
	{
		if(a[0] != a[i])
		{
			flag=2;
			break;
		}
	}

	// step 3: Display decidsion
	if(flag==1)
	{
		printf("All elements are EQUAL");
	}
	if(flag==2)
	{
		printf("All elements are NOT EQUAL");
	}
}
```

==========

---- Array as Parameter ----

==========

* Similar to a normal variable, we can also pass entire array as parameter [while calling a function].

* There are two important points to remember:

1. While passing entire array as AP, don't use [] (use only name of array)

2. While receiving an entire array as FP, must use []

(specifying SIZE is dependent on compiler)

3. ??????

Example-1:

```
#include<stdio.h>
void disp_array_ele(int b[])  <--- declaring array 'b' as FORMAL PARAMETER
{
	int i;
	printf("Array elements are\n");
	for(i=0; i<=9; i++)
	{
		printf("At index %d element is %d\n", i, b[i]);
	}
}

void main()
{
	int a[10], i;

	printf("main starts here\n");

	printf("\nEnter 10 elements\n");
	for(i=0; i<=9; i++)
```

```
	{
		scanf("%d", &a[i]);
	}

	disp_array_ele(a);  <--- passing entire array 'a' as ACTUAL PARAMETER

	printf("\n\nmain ends here\n");
}
```

==========

==========

** In any logical case, if you have to declare such functions (that has array as FP) then we can declare it as follows:

```
void disp_array_ele(int b[]);  <--- declaration of function
```

OR

```
void disp_array_ele(int[]);  <--- declaration of function
```

==========

==========

* If you notice in Example-1, the function disp_arr_ele() is to print the elements of array. This means, I can't use this for diffent sized arrays.

* The solution is, to pass the size of array as second parameter.

Example-2:

```
#include<stdio.h>
void disp_array_ele(int b[], int sz)
{
	int i;
	printf("Array elements are\n");
	for(i=0; i<sz; i++)
	{
		printf("At index %d element is %d\n", i, b[i]);
	}
}

void main()
{
	int a[10], b[20], c[8], d[5], e[25];

	.....
	.....

	disp_array_ele(a, 10); <--- passed entire array 'a' with size 10
	disp_array_ele(b, 20); <--- passed entire array 'b' with size 20
	disp_array_ele(c, 8); <--- passed entire array 'c' with size 8
	disp_array_ele(d, 5); <--- passed entire array 'd' with size 5
	disp_array_ele(e, 25); <--- passed entire array 'e' with size 25

}
```

==========

* If you need to declare above function, then do it as:

```
        void disp_array_ele(int[], int);
        OR
        void disp_array_ele(int b[], int sz);
```

==========

==========

6-8-21(2)

* We know that, we can pass an entire array as ACTUAL PARAMETER.
* For this, there are two things to remember:
    1. Don't use [] while passing an array as ACTUAL PARAMETER. (use only name)
    2. Must use [] while declaring array as FORMAL PARAMETER
                                                (specifying size is compiler dependent)

For example:

```
#include<stdio.h>
void disp_arr_ele(int temp[])   <--- array 'temp' is declared as FP
{
        int i;
        printf("Array elements are\n");
        for(i=0; i<=9; i++)
        {
                printf("%d\n", a[i]);
        }
}

void main()
{
```

```
	int a[10], i;

	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
	}

	disp_arr_ele(a);  <--- passing entire array 'a' as AP

}
```

*** To make a function suitable for multiple array sizes, always prefer to pass array-size as second parameter.

==========

==========

```
void disp_arr_ele(int temp[], int sz)
{
	int i;
	for(i=0; i<sz; i++)
	{
		.......
		.......
	}
}

void main()
{
```

```
	int a[10], b[20], c[5];

	disp_arr_ele(a, 10);
	disp_arr_ele(b, 20);
	disp_arr_ele(c, 5);

}
```

==========

==========

* What will be the output of following programs?

** CHALLANGE **

a)

```
#include<stdio.h>
void temp(int b)
{
	b = 20;
}

void main()
{
	int a=10;
	printf("\n\nInitially : %d", a);

	temp(a);

	printf("\n\nNew value : %d", a);
```

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

Output:

Initially : 10

New value : 10

*** In above example, the 'a' and 'b' are different memory blocks. And value of 'b' is not returned. Hence, the change is made in 'b'; where as 'a' remains unchanged.

==========

==========

** CHALLANGE **

b)

```
#include<stdio.h>
void temp(int b[])  <--- declared array 'b' as FP
{
	b[0] = 333;
	b[2] = 555;
}

void main()
{
	int a[4];
	a[0] = 10;  a[1] = 20;  a[2] = 30; a[3] = 40;
```

```
        printf("\n\nInitially, Array elements are\n");
        for(i=0; i<=3; i++)
        {
                printf("%d\n", a[i]);
        }

        temp(a);   <--- passing entire array as AP

        printf("\n\nUpdated, Array elements are\n");
        for(i=0; i<=3; i++)
        {
                printf("%d\n", a[i]);
        }
}
```

Output:

Initially, Array elements are

10

20

30

40

Updated, Array elements are

333

20

555

40

==========

==========

**** In above example remember that, BY-DEFAULT AN ARRAY PERFORMS CALL-BY-REFERENCE.

* In normal variables, AP and FP makes different memory-blocks. Therefore if we change in FP then it will not affect AP. (because both are different)

* In case of array, AP and FP both are same memory-blocks. Therefore if we update/change in FP array, then it indirectly affects/updates/changes in AP array.

==========

==========

*** IMP ***

- In C/C++, a function cannot return ARRAY. [We can return single index/element of an array, but cannot return entire array]

For example:

```
int temp()
{
        int a[10];
        .....
        .....
        return a[2];  <--- returning single index/element : IS VALID
}

void main()
```

```
{
        int c;
        .......
        .......
        c = temp();  <--- receiving it in single 'int' variable
        .......
        .......
}
```

==========

==========

Example-2:

```
int temp()
{
        int a[10];
        .....
        .....
        return a;  <--- returning entire array : IS INVALID
}
```

* A function cannot return array.

==========

==========

7-8-21(1)

* We know that, we can pass an array as parameter.

- Don't use [] while passing

- Must use [] while receiving

* An array, by-default performs call-by-reference. i.e. If we change/update in FP array then it will ultimately change in AP array.

* In C/C++, a function cannot return array. We can return single element/index of an array.

==========

==========

Example-1:

```
#include<stdio.h>
void accept_arr_ele(int temp[], int sz)
{
	int i;
	printf("Enter %d elements\n", sz);
	for(i=0; i<sz; i++)
	{
		scanf("%d", &temp[i]);
	}
}

void show_arr_ele(int temp[], int sz)
{
	int i;
	for(i=0; i<sz; i++)
	{
```

```
                printf("%d\n", temp[i]);
        }
}

void main()
{
        int a[10], b[25], c[8], d[12], e[50];

        printf("\nAccepting elements for first array\n");
        accept_arr_ele(a, 10);

        printf("\nAccepting elements for second array\n");
        accept_arr_ele(b, 25);

        printf("\nAccepting elements for third array\n");
        accept_arr_ele(c, 8);

        printf("\nAccepting elements for fourth array\n");
        accept_arr_ele(d, 12);

        printf("\nAccepting elements for fifth array\n");
        accept_arr_ele(e, 50);


        printf("\nElements of first array are\n");
        show_arr_ele(a, 10);

        printf("\nElements of second array are\n");
        show_arr_ele(b, 25);

        printf("\nElements of third array are\n");
```

```
	show_arr_ele(c, 8);

	printf("\nElements of fourth array are\n");
	show_arr_ele(d, 12);

	printf("\nElements of fifth array are\n");
	show_arr_ele(e, 50);

}
```

==========

==========

Program-2:

```
void update_array(int temp[])
{
	int ind, ele;

	printf("Enter index-number where you want to update new value : ");
	scanf("%d", &ind);

	printf("Enter new value to update at index %d : ", ind);
	scanf("%d", &ele);

	temp[ind] = ele;  <--- assigning 'ele' at temp[ind]
}

void main()
```

```
{
        int a[50], i;

        for(i=0; i<=49; i++)
        {
                scanf("%d", &a[i]);
        }

        XYZ:
        fflush(stdin);
        printf("All values are correct..? Should we proceed..? : ");
        scanf("%c", &u_choice);

        if(u_choice=='n')
        {
                update_array(a);
                goto XYZ;
        }

        ..... continue with your main logic.....
        .....
        .....

}
```

==========

==========

* In POPL, We are working on single/individual variables.

* In OOPL, we work on a bunch of variables (of different datatypes). That bunch is called as "OBJECT" / "ENTITY"

==========

==========

HOMEWORK

*** CHALLANGE ***

Write a program from following instructions.

- Declare an array of size 25; and accept 24 elements in it.
- Accept an index-number and one-element.
- Insert that new element at specified index by shifting all elements one step backward.

==========

==========

7-8-21(2)

* We know that, we declare array that has elements of similar datatype.

* To declare an array, general syntax is:

```
datatype  name[SIZE];
```

* Each block of array, has index-number to identify that block.

```
a[0], a[1], a[2], ....
```

* We also know that, it is not mendatory use loop to work with array. But we PREFER to use loop for effective logic and minimizing efforts.

==========

==========

*** IMP ***

- In case of array, it is assurred that array elements/blocks will have serial address.

- If we declare 10 individual variables, then it is not assurred that they will have serial addresses.

- Refer following programs.

Program-1: ---> (GCC compiler) 'int' type array

```
#include<stdio.h>
void main()
{
   int a[5], i;

   printf("\n\nAddress are \n");
   for(i=0; i<=4; i++)
   {
      printf("Address of index %d is : %u\n",i, &a[i]);  <--- prints address
   }

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

Address are

Address of index 0 is : 6422280

Address of index 1 is : 6422284

Address of index 2 is : 6422288

Address of index 3 is : 6422292

Address of index 4 is : 6422296

==========

==========

Program-2: ---> (GCC compiler) 'float' type array

```
#include<stdio.h>
void main()
{
   float a[5];
   int i;

   printf("\n\nAddress are \n");
   for(i=0; i<=4; i++)
   {
      printf("Address of index %d is : %u\n",i, &a[i]);  <--- prints address
   }

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

Address are

Address of index 0 is : 6422280

Address of index 1 is : 6422284

Address of index 2 is : 6422288

Address of index 3 is : 6422292

Address of index 4 is : 6422296

==========

==========

Program-3: ---> (GCC compiler) 'char' type array

```
#include<stdio.h>
void main()
{
   char a[5];
   int i;

   printf("\n\nAddress are \n");
   for(i=0; i<=4; i++)
   {
      printf("Address of index %d is : %u\n",i, &a[i]);
   }

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

Address are

Address of index 0 is : 6422295

Address of index 1 is : 6422296

Address of index 2 is : 6422297

Address of index 3 is : 6422298

Address of index 4 is : 6422299

==========

==========

*** IMP ***

* From above outputs, note that address of array elements are in serial.

* Address are assigned per byte. This means:

    - int occupies 4 bytes ---> 'int' block will have 4 addresses

    - float occupies 4 bytes ---> 'float' block will have 4 addresses

    - char occupies 1 byte ---> 'char' block will have 1 address

* If we declare 5 individual variables, then their addresses may/may-not be in serial.

* If we print address of any variable, then it will print first-address (base address)

* And if we print base-address of entire array, then it will print first-address of 0th index.

==========

==========

*** IMP ***

- If we increment in normal variable, then it increases the value by 1.

- If we increment in a pointer, then it increases by it's SIZE (datatype).

- If we decrement in normal variable, then it decreases the value by 1.

- If we decrement in a pointer, then it decresases by it's SIZE (datatype).

```
#include<stdio.h>
void main()
{
   int a=10;
```

```
    int *p;

    p = &a;

    printf("\n\nValue in pointer p is : %u", p);
    p++;
    printf("\n\nNew value in pointer p is : %u", p);
}
```

The output will be:

Value in pointer p is : 6422296

New value in pointer p is : 6422300

==========

==========

* Refer and try to understand following program.

```
#include<stdio.h>
void main()
{
        int a[5], i, *ptr;

        printf("Enter 5 array elements\n");
        for(i=0; i<=4; i++)
        {
                scanf("%d", &a[i]);
        }
```

```
        printf("\nArray elements are\n");
        ptr = &a[0];
        for(i=11; i<=15; i++)
        {
                printf("%d\n", *ptr); <--- prints element at address 'ptr'
                ptr++;  <--- refers to next index/element
        }
}
```

* Above program proves that we can iterate on an array by using pointer and increment operator.

==========

==========

9-8-21(1)

* In array, every memory-block gets address in serial. But it is not assurred in individual memory-blocks.

* Address is assigned per byte.

* If we print, we always gets BASE address of that memory-block.

* If we increment in pointer, then it will increament based on size of it's datatype.

* Therefore we can also iterate on array by using pointer with incre/decre operator.

* BASE address of an array is BASE address of it's 0th index.

For example:

```
void main()
{
	int a[10], *ptr, i;

	printf("Enter 10 elements\n");
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);  <--- accepting by using index-numbers
	}

	printf("Array elements are\n");
	ptr = &a[0];  <--- assigning address of 0th index in pointer 'ptr'
	for(i=11; i<=19; i++)
	{
		printf("%d\n", *ptr);  <--- prints element at 'ptr'
		ptr++; <--- indirectly refers to next index/element
	}
}
```

==========

* We can also accept/input the elements by using pointer.

```
void main()
{
	int a[10], *ptr, i;

	printf("Enter 10 elements\n");
	ptr = &a[0];
	for(i=11; i<=19; i++)
```

```
	{
		scanf("%d", ptr);  <--- accepting by using pointer
		ptr++;  <--- indirectly pointer refers to next index/element
	}

	printf("Array elements are\n");
	ptr = &a[0];  <--- assigning address of 0th index in pointer 'ptr'
	for(i=11; i<=19; i++)
	{
		printf("%d\n", *ptr);  <--- prints element at 'ptr'
		ptr++; <--- indirectly refers to next index/element
	}
}
```

==========

==========

* Remember that, the scanf() function don't need '&' mendaorily. It needs the address of memory-block where you want that input value to assign.

==========

Program-1:

Write a program to declare an array of size 10. Accept the elements and print sum of all elements by using pointer.

```
#include<stdio.h>

void main()
{
```

```
	int a[10], *ptr, i, sum=0;

	// step 1: accepting elements
	printf("Enter 10 elements\n");
	ptr = &a[0];
	for(i=101; i<110; i++)
	{
		scanf("%d", ptr);
		ptr++;
	}

	// step 2: calculating sum of elements
	ptr = &a[0];
	for(i=0; i<=9; i++)
	{
		sum = sum + (*ptr);
		ptr++;
	}

	// step 3: display sum
	printf("Sum of all elements is : %d", sum);
}
```

==========

Program-2:

Write a program to declare an array of size 10. Accept the elements [0 to 9] and print the elements in reverse order [9 - 0]

==========

Program-3:

Write a program from following instructions.

- Declare two arrays of size 10 each.
- Accept elements for first array.
- Copy them into another array [at same index] by using pointers.

```
#include<stdio.h>
void main()
{
	int a[10], b[10];
	int *ptra, *ptrb, i;

	// step 1: accepting elements for first array
	for(i=0; i<=9; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: copying elements into second arrays
	ptra = &a[0];
	ptrb = &b[0];

	for(i=1; i<=10; i++)
	{
		*ptrb = *ptra;
		ptra++;
		ptrb++;
```

```
        }

        // step 3: displaying both array's elements
        .......
        .......

}
```

==========

==========

9-8-21(2)

* We know that, array index has address in serial.

* If we perform incr/decre in pointer, then it depends on size of datatype of that pointer.

* We also that, we can pass entire array AP. But a function cannot reutrn an array. By-default an array peforms CBR.

==========

==========

---- Initializing Array ----

==========

* We know that, initilization means:

    declare + assign in single statement.

For example:

```
int a;   <--- declaration
a = 10;  <--- assigninig value

OR

int a = 10;   <--- intialization: declare + assign
float b = 91.65;  <--- initilization
char gen = 'M';  <--- initilization
float pi = 3.14;  <--- initialization
```

* Similar to normal variables, we can also intialize array elements. To do this:
  - Assign value within { }
  - Separate the values by comma (,)

For example:

```
void main()
{
    int a[10] = {64, 22, 56, 5, 89, 678, 889, 31, 59, 78};

    ........
    ........
}
```

* From above statement, the array 'a' will initialize with specified 10 elements.
  a[0] ---> 64
  a[1] ---> 22

a[2] ---> 56

.....

.....

a[8] ---> 59

a[9] ---> 78

Refer following program.

```
#include<stdio.h>
void main()
{
   int a[10] = {64, 22, 56, 5, 89, 678, 889, 31, 59, 78};
   int i;



   printf("\nArray elements are\n");
   for(i=0; i<=9; i++)
   {
      printf("At index %d element is %d\n", i, a[i]);
   }
}
```

The output will be:

```
--------------------
Array elements are
At index 0 element is 64
At index 1 element is 22
At index 2 element is 56
At index 3 element is 5
At index 4 element is 89
At index 5 element is 678
```

At index 6 element is 889

At index 7 element is 31

At index 8 element is 59

At index 9 element is 78

==========

==========

**** IMP ****

1. It is optional to specify SIZE of array, if we initialize it.

For example:

```
int a[] = {64, 22, 56, 5, 89, 678, 889, 31, 59, 78};
        // will automatically create 10 blocks as initliaze it.

float b[] = {15.22, 71.67, 982.70, 25.23};
        // will automatically create 4 blocks and initialize it.

char c[] = {'R', 'a', 'v', 'i' };
        // will automatically create 4 blocks and initialize it.
```

2. If we specify SIZE of array and initialize with less number of elements, then remaining elements will be initialized with ZERO.

For example:

```
#include<stdio.h>
void main()
{
   int a[10] = {64, 22, 56, 5};
   int i;
```

```
    printf("\nArray elements are\n");
    for(i=0; i<=9; i++)
    {
        printf("At index %d element is %d\n", i, a[i]);
    }
}
```

The output will be:

```
--------------------
Array elements are
At index 0 element is 64
At index 1 element is 22
At index 2 element is 56
At index 3 element is 5
At index 4 element is 0
At index 5 element is 0
At index 6 element is 0
At index 7 element is 0
At index 8 element is 0
At index 9 element is 0
```

3. If we specify SIZE of array but initialize it with excess elements, then GCC will show CT error.

For example:

```
-------------
#include<stdio.h>
void main()
{
    int a[2] = {64, 22, 56, 5};
    int i;
```

```
   printf("\nArray elements are\n");
   for(i=0; i<=9; i++)
   {
      printf("At index %d element is %d\n", i, a[i]);
   }
}
```

This will show CT error:

C:\MinGW\bin\Program40.c|5|warning: excess elements in array initializer|

==========

==========

* Please do handrun of following program.

```
#include<stdio.h>
void show_arr_ele(int t[], int sz)
{
        int i;
        for(i=0; i<sz; i++)
        {
                printf("%d\n", t[i]);
        }
}

void main()
{
        int a[5] = {51, 35, 22, 71, 44};
        int i, j, temp;
```

```
	// step 1: showing original elements
	printf("\nInitially, array elements are\n");
	show_arr_ele(a,5);

	// step 2: performing sorting logic
	for(i=0; i<=4; i++)
	{
		for(j=0; j<=3-i; j++)
		{
			if(a[j] > a[j+1])
			{
				temp = a[j];
				a[j] = a[j+1];
				a[j+1] = temp;
			}
		}
	}

	// step 3: showing sorted elements
	printf("After sorting, array elements are\n");
	show_arr_ele(a, 5);
}
```

==========

==========

10-8-21(01)

---- Two Dimensional Array ----

==========

* Till date, we worked with One-Dimensional arrays. In which, there is one column.

```
int a[10];  <--- makes 1 column and 10 rows
float b[7];  <--- makes 1 column and 7 rows
```

* If logically required, then we can also work with two-dimensional arrays; in which, we can specify numbers of rows and numbers of columns.

* To declare a Two-Dimensional array, general syntax is:

```
datatype  arr_name[ROW_SIZE][COL_SIZE];
```

For example:

```
int a[5][3];  <--- makes 5 rows and 3 columns
float b[3][3];  <--- makes 3 rows and 3 columns
```

* Similar to One-Dimensional arrays, here also every block has index-number. Here, rows and columns both will have index numbers.

* Similar to One-Dimensional array, here also we need to use loop to iterate through index-numbers. But in Two-Dimensional array, we need to use nested loop.

For example:

```
#include<stdio.h>
void main()
{
    int a[4][3];
    int i,j;

    printf("Enter elements for 4x3 array\n");
```

```
	for(i=0; i<=3; i++)
	{
		for(j=0; j<=2; j++)
		{
			scanf("%d", &a[i][j]);
		}
	}

	printf("Array elements are\n");
	for(i=0; i<=3; i++)
	{
		for(j=0; j<=2; j++)
		{
			printf("%d\t", a[i][j]);
		}
		printf("\n");
	}
}
```

==========

11-8-21(1)

* In Two-Dimensional array, we specify numbers of rows and numbers of columns (while declaring and using].

* General syntax is:

```
	datatype  arr_name[ROW_SIZE][COL_SIZE];
```

For example:

int a[5][3]; <--- makes 5 rows and 3 columns

float b[7][5]; <--- makes 7 rows and 5 columns

* To work on Two-Dimensional arrays, we use nested loop.

- Outer loop is to iterate on rows
- Inner loop is to iterate on columns

* To identify each block, we have to specify [row-index] and [column-index]

* To print Two-Dimensional array in matrix style, we use "\t" and "\n"

* Refer following program.

```
void main()
{
	int a[6][4];
	int i,j;

	printf("Enter elements for 6x4 array\n");
	for(i=0; i<=5; i++)
	{
		for(j=0; j<=3; j++)
		{
			scanf("%d", &a[i][j]);
		}
	}


	printf("Array elements are\n");
	for(i=0; i<=5; i++)
	{
```

```
            for(j=0; j<=3; j++)
            {
                printf("%d\t", a[i][j]);
            }
            printf("\n");
        }
}
```

*** Finally, to work on Two-Dimensional array we prefer to use nested loops.

==========

Program-1:

Write a program to accept elements for 5x3 array. And print the total count of even elements and odd elements of that array.

```
void main()
{
        int a[5][3];
        int i,j, e_count=0, o_count=0;

        // step 1: accept elements
        printf("Enter elements for 5x3 array\n");
        for(i=0; i<=4; i++)
        {
            for(j=0; j<=2; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }
```

```
	// step 2: find even-odd count
	for(i=0; i<=4; i++)
	{
		for(j=0; j<=2; j++)
		{
			if(a[i][j] %2 == 0)
			{
				e_count++;
			}
			else
			{
				o_count++;
			}
		}
	}

	// step 3: display even-odd count
	printf("Total count of even elements is : %d", e_count);
	printf("Total count of odd elemnets is : %d", o_count);
}
```

==========

Program-2:

Write a program to accept elements of a 5x5 array. And print the sum of all elements of that array.

```
void main()
{
```

```
	int a[5][5];
	int i, j, sum=0;

	// step 1: accept elements
	printf("Enter elements for 5x5 array\n");
	for(i=0; i<=4; i++)
	{
		for(j=0; j<=4; j++)
		{
			scanf("%d", &a[i][j]);
		}
	}

	// step 2: calculate sum of elements
	for(i=0; i<=4; i++)
	{
		for(j=0; j<=4; j++)
		{
			sum = sum + a[i][j];
		}
	}

	// step 3: display sum
	printf("Sum of all elements is : %d", sum);
}
```

==========

Program 3:

Write a program to accept elements of 5x5 matrix and print only diagonal elements of it.

```
void main()
{
	int a[5][5];
	int i,j;

	// step 1: accept elements
	printf("Enter elements of 5x5 matrix\n");
	for(i=0; i<=4; i++)
	{
		for(i=0; i<=4; i++)
		{
			scanf("%d", &a[i][j]);
		}
	}

	// step 2: print diagonal elements
	for(i=0; i<=4; i++)
	{
		for(j=0; j<=4; j++)
		{
			if(i==j)
			{
				printf("%d\n", a[i][j]);
			}
		}
	}

	==== OR ===
```

```
	for(i=0; i<=4; i++)
	{
		printf("%d\n", a[i][i]);
	}
}
```

* To work with Two-Dimensional array, we must specify two indexes [row and column]. But it is not compulsory to use two different loop-counters/variables for indexes.

==========

Program-4:

Write a program to accept elements for 5x5 matrix and print the sum of diagonal elements only.

```
void main()
{
	int a[5][5];
	int i,j, sum=0;

	// step 1: accept elements
	printf("Enter elements of 5x5 matrix\n");
	for(i=0; i<=4; i++)
	{
		for(i=0; i<=4; i++)
		{
			scanf("%d", &a[i][j]);
		}
	}
```

```
// step 2: print diagonal elements
for(i=0; i<=4; i++)
{
	for(j=0; j<=4; j++)
	{
		if(i==j)
		{
			sum = sum + a[i][j];
		}
	}
}

=== OR ===

for(i=0; i<=4; i++)
{
	sum = sum + a[i][i];
}

// step 3: print sum of diagonal elements
printf("Sum of all diagonal elements is : %d", sum);
```

==========

Homework:

Program-1:

Write a program to accept 25 elements of One-Dimensional array. And copy these 25 elements into another 5x5 matrix (row wise)

```
void main()
{
	int a[25], b[5][5];
	int i,j;

	printf("Enter 25 elements \n");
	for(i=0; i<=24; i++)
	{
		scanf("%d", &a[i]);
	}


}
```

Program-2:

Write a program from following instructions.

- Declare a One-Dimensional array of size 5.
- Accept 5 elements for it (each of 4 digit)
- Fill each digit of this array into another array of 5x4 size.

==========

12-8-21(1)

Solved Homework:

Program-1:

Write a program to accept 25 elements of One-Dimensional array. And copy these 25 elements into another 5x5 matrix (row wise)

```
void main()
{
	int a[25], b[5][5];
	int i,j,k;

	// step 1: accepting 25 elements
	printf("Enter 25 elements \n");
	for(i=0; i<=24; i++)
	{
		scanf("%d", &a[i]);
	}

	// step 2: copying into two-dim array
	k=0;
	for(i=0; i<=4; i++)
	{
		for(j=0; j<=4; j++)
		{
			b[i][j] = a[k];
			k++;
		}
	}

	// step 3: Displaying two-dim array
```

```
	for(i=0; i<=4; i++)
	{
		for(j=0; j<=4; j++)
		{
			printf("%d\t", a[i][j]);
		}
		printf("\n");
	}
}
```

==========

==========

Program-2:

Write a program from following instructions.

- Declare a One-Dimensional array of size 5.
- Accept 5 elements for it (each of 4 digit)
- Fill each digit of this array into another array of 5x4 size.

```
#include<stdio.h>
void main()
{
	int a[5], b[5][4];
	int i,j,k, temp, d;

	// step 1: accepting 5 elements
	printf("Enter 5 elements, each of 4 digit\n");
	for(i=0; i<=4; i++)
	{
```

```
            scanf("%d", &a[i]);
        }

        // step 2: distributing each digit into two-dim array
        k = 0;
        for(i=0; i<=4; i++)
        {
            temp = a[k];
            for(j=3; j>=0; j--)
            {
                d = temp % 10;
                b[i][j] = d;
                temp = temp/10;
            }
            k++;
        }

        // step 3: displaying two-dim array
        printf("Elements of two-dim array are\n");
        for(i=0; i<=4; i++)
        {
            for(j=0; j<=3; j++)
            {
                printf("%d\t", b[i][j]);
            }
            printf("\n");
        }
}
```

==========
===

Program-3:

Write a program from following instructions.

- Declare and accept elements for 4x5 two-dim array. [4 rows and 5 columns]
- Swap the elements of even-indexed row with exact next odd-indexed row elements.

0th index----> 1st index

2nd index ----> 3rd index

```
#include<stdio.h>
void disp_arr_ele(int b[][], int r, int c)
{
	int i,j;




}



void main()
{
	int a[4][5];
	int i,j, temp;

	// step 1: accepting elements for 4x5 matrix
	for(i=0; i<=3; i++)
	{
		for(j=0; j<=4; j++)
		{
```

```
                        scanf("%d", &a[i][j]);
                }
        }

        // step 2: showing original elements (before swapping)
        disp_arr_ele(a,4,5);

        // step 3: swapping row-wise
        for(i=0; i<=3; i=i+2)
        {
                for(j=0; j<=4;j++)
                {
                        temp = a[i][j];
                        a[i][j] = a[i+1][j];
                        a[i+1][j] = temp;
                }
        }

        // step 4: displaying swapped elements
        disp_arr_ele(a,4,5);
}
```

==========

**** CHALLANGE-1 ****

Write a program to accept 10 elements of one-dim array. And print YES if order of elements is already in ASCENDING order; otherwise print NO.

**** CHALLANGE-2 ****

Write a program to accept elements of 5x5 matrix. And print YES if elements of each row are in ASCENDING order; otherwise print NO.

==========

13-8-21(1)

Answer the following questions.

(a) An array is a collection of

1. different data types scattered throughout memory

2. the same data type scattered throughout memory

3. the same data type placed next to each other in memory

4. different data types placed next to each other in memory

Answer : 3

==========

(b) Are the following array declarations correct?

int a(25);

int size = 10, b[size];

int c = {0,1,2};

Answer: All are incorrect.

int a(25); ---> paranthesis is not allowed. We must use []

int size = 10, b[size]; ---> variable sized array is not allowed.

int c = {0,1,2}; ---> no [] is specified with 'c'

==========

(c) Which element of the array does this expression refers?

num[4]

Answer ---> 5th element of array [4th index means 5th element]

==========

(d) What is the difference between the 5's in these two expressions? (Select the correct answer)

int num[5];

num[5] = 11;

1. first is particular element, second is type
2. first is array size, second is particular element/index
3. first is particular element, second is array size
4. both specify array size

Answer ---> 2

int num[5]; ---> declaration of array of size 5

num[5] = 11; ---> assigning 11 at index [5]

==========

(e) State whether the following statements are True or False:

1. The array int num[26] has twenty-six elements.
2. The expression num[1] designates the first element in the array
3. It is necessary to initialize the array at the time of declaration.

4. The expression num[27] designates the twenty-eighth element in the array.

Answer :

1 ---> True

2 ---> False

3 ---> False

4 ---> True

==========

==========

What will be the output of following programs?

(a)

```
#include<stdio.h>
void main()
{
   int b[] = { 10, 20, 30, 40, 50 };
   int i;
   for(i=0; i<=4; i++)
   {
      printf("\n%d", * (b+i));
   }
}
```

==========

(b)

```
void main()
{
        int a[5] = { 5, 1, 15, 20, 25 };
```

```
	int i, j, k = 1, m;
	i = ++a[1];
	j = a[1]++;
	m = a[i++];
	printf("\n%d %d %d", i, j, m);
}
```

Output: 3 2 15

==========

==========

* When we declare an array, it is not actually an array's name. But an array's name is actually a pointer that holds the address of 0th index.

a[1] ---> this indicates to interpreter to do increment by 1 in base-address
of 'a'

a[2] ---> this indicates to interpreter to do increment by 2 in base-address
of 'a'

```
#include<stdio.h>
void main()
{
	int a[] = {10, 20, 30, 40, 50}, *b;

	b = a;  <--- is valid.

}
```

==========

==========

*** Any static entity by-default gets initialized with ZERO.

*** If normal memory-block (non-static) then GARBAGE VALUE.

static int a; ---> a=0

int a; ---> a=GV

==========

==========

14-8-21(1)

* We know that, an array is ultimately a pointer. This pointer stores address of 0th index.

* The index-numbers are ultimately the count of increment applied to that pointer.

==========

==========

---- String ----

==========

* A string is "character array".

For example:

---

char a[20]; <--- is a string

char b[40]; <--- is a string

* We can use String to store name, address, PAN card number, aadhar number, bank-account number, IFSC code, etc...

* We can store any combination of alphabet, digits and special-symbols.

*** Whenever we feel that some value is exceeding the datatype's value range, then prefer Strings.

*** We also know that, we cannot perform any operations on arrays and pointers. Similarly, we cannot perform any operation on Strings.

==========

==========

* While taking input for Strings, we can use loop. But if we use loop to take inputs, then we must always enter/input individual characters. And additionally it becomes complex for end-users to count the length and enter single-single characters.

```
void main()
{
	char sname[20];
	char saddr[150];
	int t;

	int i;

	printf("Enter length of your name : ");
	scanf("%d", &t);

	printf("Enter student's name\n");
	for(i=0; i<t; i++)
	{
		scanf("%c", &sname[i]);
```

```
        }

        printf("Enter how many letters are there in your address\n");
        scanf("%d", &t);

        printf("Enter student's address\n");
        for(i=0; i<t; i++)
        {
                scanf("%c", &saddr[i]);
        }

        .........
        .........
}
```

* Therefore we should not prefer loops to take inputs for Strings.

*** To take input for Strings, we have another special format-specifier %s. We can use this format-specifier directly like variables.

For example:

```
#include<stdio.h>
void main()
{
        char sname[25];
        char scol[20];
        int sroll;

        fflush(stdin);
        printf("Enter student name : ");
```

```
        scanf("%s", &sname);

        fflush(stdin);
        printf("Enter student's college name : ");
        scanf("%s", &scol);

        fflush(stdin);
        printf("Enter student's roll number : ");
        scanf("%d", &sroll);

        printf("\nStudent's Name : %s", sname);
        printf("\nStudent's College : %s", scol);
        printf("\nStudent's Roll Number : %s", sroll);

}
```

*** IMP ***

1. When we input a String, after last character the interpreter will put a special character '\0' [NULL CHARACTER]. This '\0' indicates end-of-string.

2. If we manually generate a String or if we modify a String, then we have to manually insert '\0' to specify end-of-string.

==========

==========

15-8-21

---- String ----

==========

* We know that, a String is character array.

* We prefer Strings to store any alpha-nemeric values, larger data/values, name, address, etc..

* We can use loops to take input for Strings. But the use of loop is not preferrable; because the end-user has to enter single-single characters.

* Therefore we have a special format-specifier %s to take input-output for Strings.

* When we input a String, the interpreter will automatically place NULL CHARACTER '\0' at the end of String. This NULL CHARACTER don't have any mask character.

* If we create a string or manipulate a String or modify a string then we have to manually insert '\0' to indicate end-of-string.

==========

==========

Program-1:

Write a program to accept name, college-name, aadhar-number and roll number of a student and display the same with proper message.

```
#include<stdio.h>
void main()
{
   char sname[25];
   char scol[20];

   fflush(stdin);
```

```
    printf("Enter Student Name : ");
    scanf("%s", &sname);

    fflush(stdin);
    printf("Enter Student College Name : ");
    scanf("%s", &scol);

    printf("\n\nStudent Name : %s", sname);
    printf("\nStudent's College : %s", scol);

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

==========

*** IMP ***

- If we use %s to take input, then it will take input only upto one word. [only upto first space].

- If we want to take input with multiple words [with multiple spaces], then we have two options:
    1. use gets() function
    2. use special format-specifier ---> used for validations and formatting

Example-1:

```
#include<stdio.h>
void main()
{
    char sname[25];
    char scol[20];

    fflush(stdin);
```

```
    printf("Enter Student Name : ");
    gets(sname);  // <--- calling gets() and passing 'sname' as AP

    fflush(stdin);
    printf("Enter Student College Name : ");
    gets(scol);  // <--- calling gets() and passing 'scol' as AP

    printf("\n\nStudent Name : %s", sname);
    printf("\nStudent's College : %s", scol);

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

** The gets() will input String with spaces.

==========

Example-2:

```
#include<stdio.h>
void main()
{
    char sname[25];
    char scol[20];

    fflush(stdin);
    printf("Enter Student Name : ");
    scanf("%[^\n]s", &sname);

    fflush(stdin);
    printf("Enter Student College Name : ");
```

```
    scanf("%[^\n]s", &scol);

    printf("\n\nStudent Name : %s", sname);
    printf("\nStudent's College : %s", scol);

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

==========

==========

*** The %s can print multiple-word string. But it cannot receive multiple-word string.

==========

==========

* We know that, a string ends with '\0'. It don't have any mask character.

* But if logically required, then we can also directly compare '\0' in out string.

* We can use a while-loop and compare '\0' at every index, to find where the string is ended.

```
i=0;
while(st[i] != '\0')
{
        .......
        .......
        i++;
```

```
}
```

OR

```
for(i=0; st[i]!= '\0'; i++)
{
        .......
        .......
}
```

==========

==========

* When we work with String:

    1. Take input by using gets()
    2. use while-loop upto '\0' to do any logical operations.

Program-2:

Write a program to accept a String from the user and print the total count of spaces in that String. [ASCII value of space is 32]

```
#include<stdio.h>
void main()
{
   char st[100];
   int i, count=0;

   fflush(stdin);
   printf("Enter any String\n");
```

```
    gets(st);

    i=0;
    while(st[i] != '\0')
    {
       if(st[i]==32)  <--- directly compare with ASCII value (32) of space
       {
          count++;
       }
       i++;
    }

    printf("The occurrence of spaces is %d times", count);
    printf("\n\n\n");
}
```

==========

Program-3:

Write a program to accept a string from the user and print total numbers of characters in that String.

```
#include<stdio.h>
void main()
{
    char st[100];
    int i, count=0;

    fflush(stdin);
    printf("Enter any String\n");
    gets(st);
```

```
   i=0;
   while(st[i] != '\0')
   {
      count++;
      i++;
   }

   printf("\nThe of String is : %d", count);
   printf("\n\n\n");
}
```

Program-4:

Write a program to accept a string and print total character except spaces.

```
#include<stdio.h>
void main()
{
   char st[100];
   int i, count=0;

   fflush(stdin);
   printf("Enter any String\n");
   gets(st);

   i=0;
   while(st[i] != '\0')
   {
      if(st[i] != 32)
      {
         count++;
      }
```

```
    i++;
  }

  printf("\nThe of String is : %d", count);
  printf("\n\n\n");
}
```

==========

Program-5:

Write a program to accept a string from the user and print total count of upper-case letters and lower-case letters in it.

```
#include<stdio.h>
void main()
{
  char st[100];
  int i, ucount=0, lcount=0;

  fflush(stdin);
  printf("Enter any String\n");
  gets(st);

  i=0;
  while(st[i] != '\0')
  {
      if(st[i]>=65 && st[i]<=90)
      {
          ucount++;
      }
```

```
        else if(st[i]>=97 && st[i]<=122)
        {
                lcount++;
        }
        i++;
   }

   printf("Total count of Uppercase letters : %d", ucount);
   printf("Total count of Lowercase letters : %d", lcount);
}
```

==========

==========

17-8-21

- We know that, we can initialize an array within { }.

For example:

```
        int a[] = {10, 54, 12, 78, 24, 90, 73};
        float b[] = {12.45, 82.61, 87.34, 56.7};
        char c[] = {'R', 'a', 'v', i', '\0'};
```

*** IMP ***

1. If we initialize a character array, then we should manually assign '\0' to indicate end-of-string. [in some compilers it will be done automatically]

2. Similar to one-dim array, we can also initialize two-dim array. We initialize two-dim array, row-wise. Each row in different pairs of { }. Size is optional.

For example:

```
int a[4][3] = {
                {45, 66, 71},
                {81, 95, 34},
                {12, 43, 56},
                {85, 45, 62}
            };
```

=== OR ===

```
int a[4][3] = { {45, 66, 71}, {81, 95, 34}, {12, 43, 56}, {85, 45, 62} };
```

==========

==========

Program-1:

Write a program to accept a string from the user and print how many times any continues characters appeared in that String.

```
if(st[i] == st[i+1])
{
        count++;
        printf("%c appearred continuesly at index %d and %d\n", st[i],i,(i+1) );
}
```

==========

* What will be the output of following programs?

a)

```
void main()
{
	int a[5], i=0, j=0;
	while(i<5)
	{
		a[i] = ++j;
		i++;
	}

	printf("Array elements are\n");
	for(i=0; i<=4; i++)
	{
		printf("%d\n", a[i]);
	}
}
```

Output:

a[0] - 1

a[1] - 2

a[2] - 3

a[3] - 4

a[4] - 5

==========

b)

```
void main()
```

```
{
        float a[] = {1.2, 45.66, 71.22, 61.32};   <--- creates a float array of size 4
        printf("%d", sizeof(a) );
}
```

output:

16   (4 elements x 4 bytes)

==========

c)

```
void main()
{
        int a[4][3] = {
                        {45, 66, 71},
                        {81, 95, 34},
                        {12, 43, 56},
                        {85, 45, 62}
                    };

        int s = a[0][1] + a[1][0] + a[2][0] + a[3][2];
        printf("%d", s);
}
```

Output:

221   ---> (66 + 81 + 12 + 62)

==========

d)

```
void main()
```

```
{
        char a[] = {'\0', '\0', '\0', '\0'};
        printf("%d", sizeof(a) );
}
```

Output:

4

The '\0' is ultimately a character. Hence, it will occupy 1 byte memory.

Therefore the array will be initialized with 4 characters.

==========

e)

```
void main()
{
        char a[] = {'\0', '\0', '\0', '\0'};
        int i;
        for(i=0; i<=3; i++)
        {
                printf("%c\n", a[i]);
        }
}
```

Output:

blank output...

Because the '\0' don't have any mask character to print. Therefore the for-loop iterates and it will print null-character (blank-character) 4 times.

==========

f)

```
void main()
{
	int a,b,c;
	int arr[5] = {1, 2, 3, 25, 7};

	a = ++arr[1];
	b = arr[2]++;
	c = ++arr[++a];

	printf("%d - %d - %d - %d", a, b, c, arr[2]);
}
```

Output:

4 - 3 - 8 - 4

==========

18-8-21

* What will be the output of following programs?

a)

```
void main()
{
	char s[15];

	gets(s);  <--- consider input is "Dipesh"
	int i=0;
	while(s[i+1] != '\0')
```

```
        {
                i++;
        }

        printf("%c", s[i]);
}
```

* Above logic is to print last-character of String.

* To obtain first-character of String, it will be at 0th index.

==========

==========

---- String library functions ----

==========

* We know that, library-functions are the pre-defined functions. [whose body is already written in specific header-file]

* We have a few commonly used library-functions to work on Strings. These String related library-functions are in <string.h>. Therefore we need to include it.

* Some of the commonly used String function are:

- strlen()
- strupr()
- strlwr()
- strcpy()
- strrev()
- strcat()

==========

1. strlen():

This function returns the length of String.

The general syntax is:

```
int VAR = strlen(str_name);
```

This function calculates length of parameter String and returns. This function will not consider '\0' in String.

Calculating length of String by using strlen():

```
#include<stdio.h>
#include<string.h>
void main()
{
    char st[50];
    int x;

    printf("Enter any String ");
    gets(st);

    x = strlen(st);
    printf("Length of String is : %d", x);
}
```

Calculating length of String without using strlen():

```
#include<stdio.h>
void main()
{
    char st[50];
```

```
	int i=0, count=0;

	printf("Enter any String ");
	gets(st);

	while(st[i] != '\0')
	{
		i++;
		count++;
	}

	printf("Length of String is : %d", count);
}
```

Calculating length of String by a user-defined function:

```
#include<stdio.h>
int find_length(char temp[])
{
	int count=0, i=0;
	while(temp[i] != '\0')
	{
		i++;
		count++;
	}
	return count;
}

void main()
{
```

```
    char st[50];
    int x;

    printf("Enter any String ");
    gets(st);

    x = find_length(st);
    printf("Length of String is : %d", x);
}
```

* In case of other arrays, we should pass size of array as second-parameter.

* But in case of char-array / Strings, we don't need to pass size of array; because the while-loop will iterate upto '\0'.

==========

==========

2. strupr():

* This function permanently converts the String into upper-case letters.

* The general syntax is:

```
strupr(str_name);
```

* This function will not return anything; because the parameter string itself gets converted into upper-case (because array by-default performs CBR) [And ultimately in C/C++ a function cannot return array).

* This function will only update lower-case letter with upper-case letters. The already upper-case letters, digits and special symbols will not be affected by this function [remains as it is].

Converting into upper-case by using strlen():

```
#include<string.h>
#include<stdio.h>
void main()
{
	char st[50];

	printf("Enter any String ");
	gets(st);

	printf("Original String is : %s", st);
	strupr(st);
	printf("Updated String is : %s", st);
}
```

Converting into upper-case without using strlen():

```
#include<stdio.h>
void main()
{
	char st[50];
	int i=0;

	printf("Enter any String ");
	gets(st);

	printf("Original String is : %s", st);

	while(st[i]!= '\0')
	{
		if(st[i]>=97 && st[i]<=122)
```

```
		{
			st[i] = st[i] - 32;
		}
	}

	printf("Updated String is : %s", st);
}
```

Converting into upper-case by preparing user-defined function:

```
#include<stdio.h>
void conv_into_uppercase(char temp[])   <--- temp[] is FP
{
	int i=0;
	while(temp[i]!= '\0')
	{
		if(temp[i]>=97 && temp[i]<=122)
		{
			temp[i] = temp[i] - 32;  <--- will ultimately update in 'st'
		}
	}
}

void main()
{
	char st[50];
	int i=0;

	printf("Enter any String ");
	gets(st);
```

```
        printf("Original String is : %s", st);
        conv_into_uppercase(st);  <--- passing 'st' as AP
        printf("Updated String is : %s", st);
}
```

==========

==========

3. strlwr():

* This function permanently converts a string into lower-case letters.

* This function does not return. It will update parameter String.

* The general syntax is:

```
        strlwr(str_name);
```

* Only upper-case letters gets converted into lower-case letters. And already lower-case letters, digits and special symbols remains as it is.

Converting String into lower-case by using strlwr():

```
#include<string.h>
#include<stdio.h>
void main()
{
        char st[50];

        printf("Enter any String ");
        gets(st);

        printf("Original String is : %s", st);
```

```
	strlwr(st);
	printf("Updated String is : %s", st);
}
```

Converting into lower-case without using strlen():

```
#include<stdio.h>
void main()
{
	char st[50];
	int i=0;

	printf("Enter any String ");
	gets(st);

	printf("Original String is : %s", st);

	while(st[i]!= '\0')
	{
		if(st[i]>=65 && st[i]<=90)
		{
			st[i] = st[i] + 32;
		}
	}

	printf("Updated String is : %s", st);
}
```

Converting into lower-case by preparing user-defined function:

```
#include<stdio.h>
void conv_into_lowercase(char temp[])   <--- temp[] is FP
{
        int i=0;
        while(temp[i]!= '\0')
        {
                if(temp[i]>=65 && temp[i]<=90)
                {
                        temp[i] = temp[i] + 32;  <--- will ultimately update in 'st'
                }
        }
}

void main()
{
        char st[50];
        int i=0;

        printf("Enter any String ");
        gets(st);

        printf("Original String is : %s", st);
        conv_into_lowercase(st);  <--- passing 'st' as AP
        printf("Updated String is : %s", st);
}
```

==========

==========

19-8-21

* To work with Strings, we have some library functions in "string.h"

* We discussed about:

strlen() - returns length of parameter String

strupr() - permanently converts parameter String into uppercase letters

strlwr() - permanently converts parameter String into lowercase letters

==========

==========

4) strcpy()

* This function is to copy contents of one string into another string.

* The general syntax to use this function is:

strcpy(target_string, source_string);

* This function will not return anything. This function will copy contents of source-string into target-string.

copying string by using strcpy():

```
#include<stdio.h>
#include<string.h>
void main()
{
	int st1[50], st2[50];

	printf("Enter a string\n");
	gets(st1);
```

```
	strcpy(st2, st1);  <--- copies 'st1' into 'st2'

	printf("\nString-1 is : %s", st1);
	printf("\nString-2 is : %s", st2);
}
```

copying string without using strcpy():

```
#include<stdio.h>
void main()
{
	int st1[50], st2[50];
	int i;

	printf("Enter a string\n");
	gets(st1);

	i=0;
	while(st1[i] != '\0')
	{
		st2[i] = st1[i];
		i++;
	}

	st2[i] = '\0';  <--- manually applying '\0' at last index

	printf("\nString-1 is : %s", st1);
	printf("\nString-2 is : %s", st2);
}
```

copying string by a user-defined function:

```
#include<stdio.h>

void copy_string(char s1[], char s2[])
{
	int i=0;
	while(s1[i] != '\0')
	{
		s2[i] = s1[i];
		i++;
	}
	s2[i] = '\0';
}

void main()
{
	int st1[50], st2[50];
	int i;

	printf("Enter a string\n");
	gets(st1);

	copy_string(st1, st2);  <--- 'st1' is source and 'st2' is target

	printf("\nString-1 is : %s", st1);
	printf("\nString-2 is : %s", st2);
}
```

==========

==========

5) strcmp()

* This function is to compare two strings. This function considers case-sensitivity.

* This function compares both parameter strings and returns ZERO if both strings are totally equal; otherwise returns any non-zero value.

* The general syntax is:

```
int VAR = strcmp(string1, string2);
```

* This function is a library-function (pre-defined) it has 2 FP.

comparing two strings by using strcmp():

```
#include<stdio.h>
#include<string.h>
void main()
{
    char st1[50], st2[50];
    int x;

    printf("Enter first string\n");
    gets(st1);

    printf("Enter second string\n");
    gets(st2);

    x = strcmp(st1, st2);
```

```
	if(x==0)  <--- is strcmp() returns ZERO
	{
		printf("Both Strings are equal");
	}
	else
	{
		printf("Both Strings are unequal");
	}
}
```

comparing two strings without using strcmp():

```
#include<stdio.h>
void main()
{
	char st1[50], st2[50];
	int flag=1, i=0;
	int l1=0, l2=0;

	printf("Enter first string\n");
	gets(st1);

	printf("Enter second string\n");
	gets(st2);

	i=0;
	while(st1[i] != '\0')
	{
		l1++;
```

```
        i++;
}

i=0;
while(st2[i] != '\0')
{
        l2++;
        i++;
}

// variable 'l1' contains length of 'st1'
// variable 'l2' contains length of 'st2'

if(l1==l2)
{
        // length of both strings are equal, now we should compare characters
        i=0;
        while(st1[i] != '\0)
        {
                if(st1[i] != st2[i])
                {
                        flag=2;
                        break;
                }
                i++;
        }

        if(flag==1)
        {
                printf("Both Strings are equal");
        }
```

```
		if(flag==2)
		{
			printf("Both Strings are unequal");
		}
	}
	else
	{
		printf("Both strings are unequal");
	}
}
```

comparing two strings by a user-defined function:

```
#include<stdio.h>
int compare_strings(char s1[], char s2[])
{
	int flag=1, i=0;
	int l1=0, l2=0;

	i=0;
	while(s1[i] != '\0')
	{
		l1++;
		i++;
	}

	i=0;
	while(s2[i] != '\0')
	{
		l2++;
```

```
		i++;
	}

	if(l1==l2)
	{
		// length of both strings are equal, now comparing characters
		i=0;
		while(s1[i] != '\0')
		{
			if(s1[i] != s2[i])
			{
				flag=2;
				break;
			}
			i++;
		}

		if(flag==1)
		{
			return 1;  <--- returning '1' if strings are equal
		}
		if(flag==2)
		{
			return 2;  <--- returning '2' if strings are unequal
		}
	}
	else
	{
		return 2;   <--- returning '2' if strings are unequal
	}
}
```

```
void main()
{
	char st1[50], st2[50];
	int x;

	printf("Enter first string\n");
	gets(st1);

	printf("Enter second string\n");
	gets(st2);

	x = compare_strings(st1, st2);

	if(x==1)
	{
		printf("Both Strings are euqal");
	}
	if(x==2)
	{
		printf("Both Strings are unequal");
	}
}
```

==========

==========

6) strrev():

* This function permanently reverse the parameter String.

* This function will not return anything. It will revers the parameter string itself.

* The general syntax is:

```
strrev(string_name);
```

Reversing a string by using strrev():

```
#include<stdio.h>
#include<string.h>
void main()
{
        char st[50];

        printf("Enter any string\n");
        gets(st);

        printf("Original string is : %s", st);
        strrev(st);   <--- permanently reverses string 'st'
        printf("Updated string is : %s", st);
}
```

Reversing a string without using strrev():

```
#include<stdio.h>
void main()
{
        char st[50], temp;
        int l, i, j;

        printf("Enter any string\n");
        gets(st);
```

```
	printf("Original String is : %s", st);

	l = strlen(st);  <--- directly getting length of 'st'
	j = l-1;
	for(i=0; i<l/2; i++)
	{
		temp = st[i];
		st[i] = st[j];
		st[j] = temp;
		j--;
	}

	printf("Updated String is : %s", st);
}
```

Reversing a String by a user-defined function:

```
#include<stdio.h>
void reverse_string(char s[])
{
	int l, i, j;

	l = strlen(s);
	j = l-1;
	for(i=0; i<l/2; i++)
	{
		temp = s[i];
		s[i] = s[j];
		s[j] = temp;
```

```
                j--;
        }
}

void main()
{
        char st[50], temp;
        int l, i, j;

        printf("Enter any string\n");
        gets(st);

        printf("Original String is : %s", st);

        reverse_string(st);

        printf("Updated String is : %s", st);
}
```

==========

==========

20-8-21

---- Structure ----

==========

* Whenever we want to work with any real-world entity, then we should prefer 'structure'.

* In otherwords, if we want to work with any attribute-set then we should prefer 'strucutre'.

```
void main()
{
        int roll1, roll2, roll3 age1, age2, age3;
        float avg1, avg2, avg3;
        char sname1[25], sname2[25], sname3[25];
        char gen1, gen2, gen3;

        ......
        ......

}
```

* Above declarations are totally valid. But the programmer has to manage these many variables.

* So it is better to pefer structure for such situations and entities and attribute-sets.

==========

```
void main()
{
        int roll[100], age[100];
        float avg[100];
        char sname[25][100];
        char gen[100];

        .......
        .......

}
```

* We can also do above delcarations, to manage data of 100 students. But it becomes complex task to use many loops and manage every index.

==========

==========

* A structure is set of attributes of different datatypes. [An array is set of similar datatypes]

* To declare a structure, general syntax is:

```
struct NAME
{
        attribute-1 declaration;
        attribute-2 declaration;
        .....
        .....
        attribute-n declaration;
};
```

* The same 5 rules to give structure-name.

* Here 'struct' is 'C' keyword.

* We can declare only and only variables/attribute in a structure. [We cannot define any function in structure]

Example-1:

```
struct student_info
{
        int sroll, sage;
        char sgen, sname[25];
        float savg;
};
```

Example-2:

```
struct mobile_phone
{
        char make[25];
        char model[25];
        long price;
        float wt;
}
```

==========

==========

*** IMP ***

---

1. We cannot write any executable statement in structure.
2. We cannot call a structure and execute it. [only function can be called and executed]
3. We cannot initialize these attributes/variables of strcuture.
4. We can declare any numbers of strcutres in a program.
5. We can declare a structure within a function or in global/open area.

Example-1: ---> declaring strcuture in global/open area.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
struct student_info
{
```

```
        int sroll, sage;
        char sgen, sname[25];
        float savg;
};

void main()
{
        .......
        .......
}

void print_values()
{
        .......
        .......
}
```

* Above structure is declared in open/global area. Therefore it's scope is in all functions of this program.

==========

Example-2: ---> declaring a structure within function

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        int a,b;
        struct student_info    <--- is locally declared in main()
```

```
        {
                int sroll, sage;
                char sgen, sname[25];
                float savg;
        };

        ..........
        ..........
}

void method1()
{
        struct mobile_phone   <--- is locally declared in method1()
        {
                char make[25];
                char model[25];
                long price;
                float wt;
        };

        ........
        ........
}

void method2()
{
        ........
        ........
}
```

* In above example, structure 'student_info' is declared locally in main() method. Therefore it's scope/availability is within main() method.

* The structure 'mobile_phone' is declared locally in method1(); therefore it's scope is within method1().

==========

Example-3: ---> declaring structure in a logical-block.

```
void main()
{
        .......
        .......

        if(condition)
        {
                struct student_info  <--- is declared in if-block
                {
                        int sroll, sage;
                        char sgen, sname[25];
                        float savg;
                };
        }
        ........
        ........

}
```

==========

==========

21-8-21

* We know that, if we want to work with any real-world entity that has attributes, then we should prefer structure.

* To declare a structure, general syntax is:

```
struct NAME
{
    attribute-1 declaration;
    attribute-2 declaration;
    ........
    ........
    attribute-n declaration;
};
```

* In a structure, we can declare any numbers of attributes.

* In a program we can declare any numbers of structures.

* We also know that, we can declare structure either globally or locally (within function).

* We can:

- only declare attributes
- no initialization to these attributes
- no function definitions in structure
- no executable statements in structure
- cannot call and execute a structure

==========

==========

* A structure does not occupy any memory, during runtime.

* Now question is, if a structure does not occupy any runtime memory then how can we store values in it's attributes?

* The answer is, "We have to declare objects of structure."

* An object is ---> "copy of structure with memory."

* To declare an object, general syntax is:

```
structure_name  obj_name;
OR
struct structure_name  obj_name;
```

* This declaration statement is similar to declaration of variables.

For example:

```
#include<stdio.h>

struct student_info
{
	int sroll, sage;
	char sgen, sname[25];
	float savg;
};

void main()
```

{

student_info s1; <--- 's1' is an object

.....

.....

}

*** IMP ***

* We can declare any numbers of objects of a structure.

* Here, structure's name is behaving like ---> datatype

* and object's name is behaving like ---> variable-name

* We have to store values in these objects. Because they have memory.

* These attribute-blocks in object are called as ---> instances

* To identify these instances ---> obj_name.attrib_name

==========

==========

Complete example on structure:

```
#include<stdio.h>
struct student_info
{
	int sroll, sage;
	char sgen, sname[25];
	float savg;
};

void main()
```

```
{
        // step 1: declaring object
        student_info s1;   <--- delcaring structure object 's1'


        // step 2: taking input for object's instances
        printf("Enter student roll number : ");
        scanf("%d", &s1.sroll);

        printf("Enter student age : ");
        scanf("%d", &s1.sage);

        fflush(stdin);
        printf("Enter student gender : ");
        scanf("%c", &s1.sgen);

        fflush(stdin);
        printf("Enter student name : ");
        gets(s1.sname);

        printf("Enter student average marks : ");
        scanf("%f", &s1.savg);


        // step 3: showing values of object's instances
        printf("\n\n");
        printf("\nStudent Name : %s", s1.sname);
        printf("\nStudent Roll Number : %d", s1.sroll);
        printf("\nStudent Age : %d", s1.sage);
        printf("\nStudent Gender : %c", s1.sgen);
        printf("\nStudent Average Marks : %f", s1.savg);
```

}

==========

==========

* From all above discussion and examples, we should prefer to declare set of attributes as a structure attributes; and declare structure objects.

* Instead of declaring following numbers of variables

```
void main()
{
	int sroll1, sroll2, sroll3;
	int sage1, sage2, sage3;
	float savg1, savg2, savg3;
	char sgen1, sgen2, sgen3;
	char sname1[25], sname2[25], sname3[25];
	.....
	.....

}
```

we better prepare a structure with attributes and it's 3 objects.

==========

==========

* Similar to array, we cannot perform any arithmetic and relational operation on entiere object.

```
if(s1>s2)  <--- is error
{

}

s3 = s1  + s2;  <--- is error
```

* But we can perform any arithemtic and relational operations on instances of object.

```
if(s1.savg > s2.savg)  <--- is valid
{
        ......
        ......
}

s3.sroll = s1.sroll + s2.sroll + s1.sage;  <--- is valid
```

==========

==========

Homework:

Program-1:

Write a program from following instrucutions:

- Declare a structure related to employee attributes
- Declare two objects of this structure
- Accept and Display information of two employees by using these two objects.

Program-2:

Write a program from following instructions:

- Declare a structure related to bank account
- Declare two objects of this structure
- Accept and display information of two bank accounts by using these two objects.

==========

==========

22-8-21

Solved Homework:

Program-1:

Write a program from following instrucutions:

- Declare a structure related to employee attributes
- Declare two objects of this structure
- Accept and Display information of two employees by using these two objects.

```
#include<stdio.h>
struct Emp_info
{
    char ename[25];
    char epost[15];
    char edept[15];
    int eexp;
```

```
        long esal;
};

void main()
{
        Emp_info e1, e2;  <--- declaring objects 'e1' and 'e2'

        // step 1: Accepting details of first employee
        printf("Accepting Details of First Employee\n");
        fflush(stdin);
        printf("Enter Employee Name : ");
        gets(e1.ename);

        fflush(stdin);
        printf("Enter Employee Post : ");
        gets(e1.epost);

        fflush(stdin);
        printf("Enter Employee Department : ");
        gets(e1.edept);

        printf("Enter Employee Experience in years : ");
        scanf("%d", &e1.eexp);

        printf("Enter Employee Salary : ");
        scanf("%ld", &e1.esal);


        // step 2: Accepting details of second employee
        printf("\nAccepting Details of Second Employee\n");
        fflush(stdin);
```

```
printf("Enter Employee Name : ");
gets(e2.ename);

fflush(stdin);
printf("Enter Employee Post : ");
gets(e2.epost);

fflush(stdin);
printf("Enter Employee Department : ");
gets(e2.edept);

printf("Enter Employee Experience in years : ");
scanf("%d", &e2.eexp);

printf("Enter Employee Salary : ");
scanf("%ld", &e2.esal);


// step 3: Showing details of first employee
printf("Employee Name : %s", e1.ename);
printf("Employee Department : %s", e1.edept);
printf("Employee Post : %s", e1.epost);
printf("Employee Experience : %d", e1.eexp);
printf("Employee Salary : %ld", e1.esal);


// step 4: Showing details of second employee
printf("Employee Name : %s", e2.ename);
printf("Employee Department : %s", e2.edept);
printf("Employee Post : %s", e2.epost);
printf("Employee Experience : %d", e2.eexp);
```

```
        printf("Employee Salary : %ld", e2.esal);

}
```

==========

==========

* Similar to a normal variable and arrays, we can also pass a structure object as parameter.

* Remember that, object's datatype is ---> strucutre's name

* We know that, an array by-default performs CBR; But structure object performs CBV.

==========

* We know that in C/C++, a function cannot return entire array ; But a function can return structure object.

* In such case, returntype of that function will be ---> structure's name

==========

* We know that an entire array cannot be assigned into another array. Therefore we were peforming element-by-element copying.

```
        for(i=0; i<=9' i++)
        {
                b[i] = a[i];
        }
```

* But an entire strucutre object can be assigned/copied into another object. Therefore, we can write:

```
Bank_Acc ac1, ac2;
// taking input in 'ac1'
.....
ac2 = ac1;  <--- directly copy instances of 'ac1' into instances of 'ac2'
```

==========

==========

Program-2:

Write a program from following instructions:

- Declare a structure related to bank account
- Declare two objects of this structure
- Accept and display information of two bank accounts by using these two objects.

```
#include<stdio.h>
struct Bank_Acc
{
    char acnum[20];
    long acbal;
    char actype;
};

Bank_Acc accept_acc_details()
{
    Bank_Acc temp;  <--- declaring an object to take input in it.
    fflush(stdin);
    printf("Enter Account Number : ");
```

```
        gets(temp.acnum);

        printf("Enter Account Balance : ");
        scanf("%ld", temp.acbal);

        fflush(stdin);
        printf("Enter Account Type : ");
        scanf("%c", &temp.actype);

        return temp;  <--- returning object 'temp' with all instance values
}

void show_acc_details(Bank_Acc ob)  <--- object 'ob' as FP
{
        printf("\nAccount Number : %s", ob.acnum);
        printf("\nAccount Balance : %ld", ob.acbal);
        printf("\nAccount Type : %c", ob.actype);
}

void main()
{
        Bank_Acc ac1, ac2;

        printf("Accepting Details of First Account\n");
        ac1 = accept_acc_details();   <--- receiving returned object 'temp' in 'ac1'

        printf("\nAccepting Details of Second Account\n");
        ac2 = accept_acc_details();   <--- receiving returned object 'temp' in 'ac2'

        printf("\n\nShowing Details of First Account");
        show_acc_details(ac1);  <--- passing object 'ac1' as AP
```

```
        printf("\nShowing Details of Second Account");
        show_acc_details(ac2);  <--- passing object 'ac2' as AP
}
```

==========

* We cannot perform any relational and arithmetic operations on entire structure object. But we can perform relational and arithemtic operations on instaces of structure object.

Homework:

Program-1:

Write a program from following instructions:

- Declare a structure related to student's information
- Declare 3 objects of this structure and accept details of 3 students in it.
- Print the name of student who has highest average marks.

[Hint: Compare instaces and print one instance]

==========

23-8-21

Solved Homework:

Program-1:

Write a program from following instructions:

- Declare a structure related to student's information

- Declare 3 objects of this structure and accept details of 3 students in it.

- Print the name of student who has highest average marks.

[Hint: Compare instaces and print one instance]

```
#include<stdio.h>
struct student_info
{
        ........
        ........
};

student_info accept_stud_details()
{
        student_info temp;
        ........
        ........
        return temp;
}

void main()
{
        student_info s1,s2,s3;

        printf("Accepting Details of first student\n");
        s1 = accept_stud_details();

        ........
        ........

        if(s1.savg > s2.savg && s1.savg>s3.savg)
        {
```

```
                printf("%s has highest average", s1.sname);
        }

        else if(s2.savg > s1.savg && s2.savg>s3.savg)
        {
                printf("%s has highest average", s2.sname);
        }

        else if(s3.savg>s1.savg && s3.savg>s2.savg)
        {
                printf("%s has highest average", s3.sname);
        }
}
```

==========

==========

* Similar to an array of primitive datatypes, we can also declare an array of structure objects.

* Refer following program.

```
#include<stdio.h>
struct student_info
{
        char sname[25];
        float savg;
        int sroll;
        char sgen;
};
```

```
void main()
{
	student_info  s[50];   <--- array of objects
	int i;

	printf("Accepting Details of students\n");
	for(i=0; i<=49; i++)
	{
		printf("\n\nAccepting Details of Student : %d", (i+1) );

		fflush(stdin);
		printf("\nEnter Student Name : ");
		gets(s[i].sname);

		printf("\nEnter Student Average Marks : ");
		scanf("%f", &s[i].savg);

		printf("\nEnter Student Roll Number : ");
		scanf("%d", &s[i].sroll);

		fflush(stdin);
		printf("\nEnter Student Gender : ");
		scanf("%c", &s[i].sgen);
	}

	printf("\n\nShowing Details of students\n");
	for(i=0; i<=49; i++)
	{
		printf("\n\nDetails of Student : %d", (i+1) );
		printf("\nStudent Name : %s", s[i].sname);
		printf("\nStudent Roll Number : %d", s[i].sroll);
```

```
                printf("\nStudent Average Marks : %f", s[i].savg);
                printf("\nStudent Gender : %c", s[i].sgen);
        }
}
```

==========

==========

Program-1:

Write a program from following instructions:

- Declare a structure related to student's attributes
- Declare an array of 50 objects of this structure
- Accept details of 50 students for this array of objects
- Print the names and roll-numbers of students whose gender is 'Female' and average is more than 60.00

```
#include<stdio.h>
// step 1: declaring structure
struct student_info
{
        char sname[25];
        float savg;
        int sroll;
        char sgen;
};

void main()
{
        // step 2: declaring array of 50 objects
```

```
	student_info  s[50];   <--- array of objects
	int i;



	// step 3: accepting details of 50 students
	printf("Accepting Details of students\n");
	for(i=0; i<=49; i++)
	{
		printf("\n\nAccepting Details of Student : %d", (i+1) );

		fflush(stdin);
		printf("\nEnter Student Name : ");
		gets(s[i].sname);

		printf("\nEnter Student Average Marks : ");
		scanf("%f", &s[i].savg);

		printf("\nEnter Student Roll Number : ");
		scanf("%d", &s[i].sroll);

		fflush(stdin);
		printf("\nEnter Student Gender : ");
		scanf("%c", &s[i].sgen);
	}



	// step 4: check and print names and roll-numbers
	.........
	.........

}
```

==========

==========

* We know that, every memory-block has address ---> normal memory-blocks, arrays, constants, structrue objects.

* We can also declare a pointer that can hold the address of structure's object. This pointer will store only base-address of that objects. [means address of first instance of that object]

For example:

```
#include<stdio.h>
struct student_info
{
        char sname[25];
        float savg;
        int sroll;
        char sgen;
};

void main()
{
        student_info  s1;
        student_info  *ptr; <--- pointer of same datatype

        ptr = &s1;   <--- pointer 'ptr' can hold the address of object 's1'

        .........
```

```
        .........
}
```

* We can print the base address of any object by using '&' and "%u".

```
        printf("Address of object s1 is : %u", &s1);
```

* If a pointer is pointing to an object, then we have two options to identify an instance.

        1. Object ke saath DOT operator (.)

        2. Pointer ke saath ARROW operator (->)

* In any logical reason, if we need to use pointer then we have to use ARROW operator.

For example:

```
#include<stdio.h>
struct student_info
{
        char sname[25];
        float savg;
        int sroll;
        char sgen;
};

void main()
{
        student_info  s1;
        student_info  *ptr; <--- pointer of same datatype

        ptr = &s1;   <--- pointer 'ptr' can hold the address of object 's1'

        // Accepting by using object's name and DOT operator
```

```
	fflush(stdin);
	printf("Enter Student Name : ");
	gets(s1.sname);

	fflush(stdin);
	printf("Enter Student Gender : ");
	scanf("%c", &s1.sgen);

	printf("Enter Student Roll Number : ");
	scanf("%d", &s1.sroll);

	printf("Enter Student Average Marks : ");
	scanf("%f", &s1.savg);



	// Displaying by using pointer's name and ARROW operator
	printf("Student Name : %s", ptr->sname);
	printf("Student Roll Number : %d", ptr->sroll);
	printf("Student Average Marks : %f", ptr->savg);
	printf("Student Gender : %c", ptr->sgen);
}
```

* From above example, note that we can refer object's instances by using object's name and also by using pointer's name.

* Object ke saath DOT operator

* Pointer ke saath ARROW operator

==========

==========

----------THE END----------