

Programming for Problem Solving

As you are aware of that a computer can understand only machine language, hence the programs written in assembly language have to be converted into machine language for execution. A program that does this translation is called an **assembler**. The assembler recognizes the assembly language mnemonics and symbolic names of memory locations and converts them into appropriate binary code.

Q.4 What do you mean by wordlength of a processor. [R.T.U. 2012]

Ans. Wordlength is the number of bits that a CPU can process at once. A processor with a 32-bit wordlength has the capacity to be twice as fast as a processor with a 16 bit wordlength. Today's personal computers typically use wordlengths of 32 or 64 bits. Intel Pentium 4 processors have a 32 bit wordlength; AMD Athlon 64 processors have a word length of 64 bits. The benefits of 64 bit wordlength are only experienced if the operating system and software are designed to take advantage of the processor's 64-bit capabilities. While Intel processors and both Windows and Mac OS X have evolved to 64 bit architectures, software developers have been slower to migrate their applications from 32-bit to 64-bit. That is likely to change as 64-bit processors become the norm.

II. LONG ANSWER TYPE QUESTIONS

Q.1 Write short notes on :

- (a) System software ✓
- (b) Compiler ✓
- (c) Freeware software

[R.T.U. 2018]

Ans.(a) System Software : System software is a type of computer program that is designed to run a computer's hardware and application programs. If we think of the computer system as a layered model, the system software is the interface between the hardware and user applications.

The operating system (OS) is the best-known example of system software. The OS manages all the other programs in a computer. Other examples of system software and what each does:

- The BIOS (basic input/output system) gets the computer system started after you turn it on and manages the data flow between the operating system and attached devices such as the hard disk, video adapter, keyboard, mouse, and printer.

Programming for Problem Solving

- The boot program loads the operating system into the computer's main memory or Random Access Memory (RAM).
- An assembler takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations.
- A device driver controls a particular type of device that is attached to your computer, such as a keyboard or a mouse. The driver program converts the more general input/output instructions of the operating system to messages that the device type can understand.

According to some definitions, system software also includes system utilities, such as the disk defragmenter and system restore, and development tools, such as compilers and debuggers.

Ans.(b) Compiler : A compiler is a kind of translator that translates a program written in a high level language into machine code. The compiler replaces single high level statement into a series of machine language instructions. During the translation process, the compiler reads the source program and checks the syntax (grammatical) errors. If there is any error, the compiler generates an error message which is displayed on the screen. In case of errors, the compiler doesn't create the object code until all the errors are rectified.

Once the source code has been compiled successfully, the compiler saves the resulting machine code in a file separately. This makes recompiling necessary only if the source code has been modified.

The working of a compiler can be depicted as :

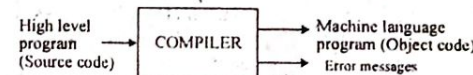


Fig. : Working of a Compiler

Ans.(c) Freeware Software : Freeware is computer software that is made available free of charge, but which is copyrighted by its developer, who retains the rights to control its distribution, modify it and sell it in the future. It is typically distributed without its source code, thus preventing modification by its users.

This can be freely used with only one restriction: any redistributed version of the software must be distributed with the original terms of free use, modification, and distribution (known as copyleft). Freeware is usually

PPS.3

distributed with a license that permits its redistribution to some extent, for example allowing users to give copies to friends. Some licenses permit the software to be freely copied but not sold.

Q.2 Write short notes on following :

- (a) Types of Primary Memory ✓
- (b) High level v/s Assembly level programming ✓
- (c) Notations of Flow chart ✓
- (d) Primary memory v/s secondary memory ✓

[R.T.U. 2016]

Ans.(a) Primary Memory : Random Access Memory (RAM) and Read Only Memory (ROM) fall in the category of the primary memory. Every computer comes with a small amount of ROM, which contains the boot firmware (called BIOS). This holds just enough information so that the computer can check its hardware and load its operating system into RAM.

RAM is the place where the computer temporarily stores its operating system, application programs, the current data, so that the computer's processor can reach them quickly and easily. It is volatile in nature, i.e. when the power is switched off, the data in the memory is lost. Unlike RAM, ROM is non-volatile. Even when the computer is switched off, the contents of ROM remain available.

Types of RAM : RAM can be further divided into two categories :

(i) **Static Random Access Memory (SRAM)** is a type of semiconductor memory that does not need to be periodically refreshed like DRAM. SRAM is considerably faster than DRAM. It is effective because most of the programs access the same data repeatedly and keeping all this information in the fast SRAM that allows the computer to avoid accessing the slow DRAM. SRAM exhibits remembrance, but is still volatile in the conventional sense that data is eventually lost when the memory is not powered.

(ii) **Dynamic Random Access Memory (DRAM)**: It is a type of RAM that stores each bit of data in a separate capacitor within an integrated circuit. Since real capacitor leak charge, the information eventually fades unless the capacitor charge is refreshed periodically. Because of this refresh requirement, it is a dynamic memory as opposed to SRAM and other static memory. Like SRAM, it is in the class of volatile memory devices, since it loses its data when the power supply is

removed. Unlike SRAM however, data may still be recovered for a short time after power-off.

Types of ROM -

Programmable read-only memory (PROM), or sometimes programmable ROM (OTP), can be written to or programmed via a special device called a PROM programmer. Typically, this device uses high voltages to permanently destroy or create internal links (fuses or antifuses) within the chip. Consequently, a PROM can only be programmed once.

Erasable programmable read-only memory (EPROM) can be erased by exposure to strong ultraviolet light (typically for 10 minutes or longer), then rewritten with a process that again requires application of higher than usual voltage. Repeated exposure to UV light will eventually wear out an EPROM, but the endurance of most EPROM chips exceeds 1000 cycles of erasing and reprogramming. EPROM chip packages can often be identified by the prominent quartz "window" which allows UV light to enter. After programming, the window is typically covered with a label to prevent accidental erasure.

Electrically erasable programmable read-only memory (EEPROM) is based on a similar semiconductor structure to EPROM, but allows its entire contents (or selected banks) to be electrically erased, then rewritten electrically, so that they need not be removed from the computer (or camera, MP3 player, etc.). Writing or flashing an EEPROM is much slower, typically taking milliseconds per bit than reading from a ROM or writing to a RAM (nanoseconds in both cases).

Electrically alterable read-only memory (EAROM) is a type of EEPROM that can be modified one bit at a time. Writing is a very slow process and again requires higher voltage (usually around 12 V) than is used for reading access. EAROMs are intended for applications that require infrequent and only partial rewriting. EAROM may be used as non-volatile storage for critical system setup information; in many applications, EAROM has been supplanted by CMOS RAM supplied by mains power and backed-up with a lithium battery.

Flash memory (or simply flash) is a modern type of EEPROM invented in 1984. Flash memory can be erased and rewritten faster than ordinary EEPROM, and newer designs feature very high endurance (exceeding 1,000,000 cycles). Modern NAND flash makes efficient use of high as 16 GB as of 2007 [update], this feature, along with its endurance and physical durability, has allowed NAND flash to replace magnetic in some applications such as USB flash drives). Flash memory is sometimes

Programming for Problem Solving

called flash ROM or flash EEPROM when used as a replacement for older ROM types, but not in applications that take advantage of its ability to be modified quickly and frequently.

Ans.(b) High level v/s Assembly level programming :

Machine language	Assembly language	High level language
1. It is the language i.e. understood by the computer without using a translator program.	In assembly language we use mnemonic for numeric opcodes of machine language.	High level language is basically problem oriented. While writing program in the HLL, a programmer use commands generally written using English words and mathematical symbols. HLL programs are translated into machine language by means of a system program called compilers.
2. This is also called as machine code of the computer and is normally written as string of binary 0's and 1's.	Computer can translate each line of assembly language into machine language. Program into machine language by means of translating program.	Programmer can use commands generally written using English words and mathematical symbols. HLL programs are translated into machine language by means of a system program called compilers.
3. Advantages Program in machine language can be executed very fast.	Advantages Easy to understand and use easier to modify & relocatable.	Advantages Machine independent fewer errors and easy debugging.
4. Disadvantages 1. Machine dependent. 2. Difficult to program and modify. 3. Time consuming and error prone.	Disadvantages 1. Machine dependent. 2. Knowledge of h/w required.	Disadvantages Speed (HLL program takes more time and memory to run).

Ans.(c) Notations of Flowchart

There are rules and standards (notations) for drawing flowcharts, prescribed by American National Standard Institute (ANSI). Table shows some standard symbols which are frequently required in the flow charts.

Symbol	Symbol Name	Description
	Flow lines	Flow lines are used to connect symbols used in flow-chart and indicate direction of flow.
	Terminal (START/STOP)	This is used to represent start and end of the flow-chart.
	Input / Output	It represents information which the system reads as input or sends as output.
	Processing	Any process is represented by this symbol. For example arithmetic operation, data movement.
	Decision	This symbol is used to check any condition or take decision for which there are two answers. Yes (True) or No (false).
	Connector	It is used to connect or join flow lines.
	Off-page Connector	This symbol indicates the communication of flowchart on the next page.
	Document	It represents a paper document produced during the flowchart process.
	Annotation	It is used to provide additional information about another flowchart symbol which may be in the form of descriptive comments, remark or explanatory notes.
	Manual Input	It represents input to be given by a developer or programmer.
	Manual operation	This symbol indicates that the process has to be done by a developer

	Online storage	It represents online data storage such as hard disks, magnetic drums or other storage devices.
	Offline storage	It represents the offline data storage such as punched card.
	Communication Link	It represents the data transmitted from an external system.
	Magnetic Disk	It represents data input or output from and to a magnetic disk.

Ans.(d) Primary memory v/s secondary memory

A computer contains a hierarchy of memory devices for storing data. They vary in their capacity, speed and cost. Primary memory (also referred to as the main memory) is the memory that is directly accessed by the CPU to store and retrieve information. Secondary memory (also referred to as the external or auxiliary memory) is a storage device that is not accessible directly by the CPU and used as a permanent storage device that retains data even after the power is turned off.

Primary memory is the memory that is directly accessed by the CPU to store and retrieve information. Most of the time, primary memory is also referred to as the RAM (Random Access Memory). It is a volatile memory, which loses its data when the power is turned off. Primary memory is directly accessible by the CPU through the address and memory bus and it is constantly accessed by the CPU to get data and instructions. Furthermore, computers contain a ROM (Read Only Memory), which holds instructions that are executed often such as the startup program (BIOS). This is a non-volatile memory, that retains its data when the power is turned off. Since the main memory is accessed often, it needs to be faster. But they are smaller in size and also costly.

Secondary memory is a storage device that is not accessible directly by the CPU and used as a permanent storage device that retains data even after power is turned off. CPU accesses these devices through an input/output channel and data is first transferred in to the primary memory, from the secondary memory before accessing. Usually, hard disk drives and optical storage devices (CDs, DVDs) are used as secondary storage devices in modern computers. In a secondary storage device, data are organized in to files and directories according to a file system. This also allows to associate

additional information with data such as the access permissions, owner, last access time, etc. Furthermore, when the primary memory is filled up, secondary memory is used as a temporary storage for keeping least used data in the primary memory. Secondary memory devices are less costly and larger in size. But they have a large access time.

However, primary memory is the memory that is directly accessed by the CPU to store and retrieve information, whereas the secondary memory is not accessible directly by the CPU. Primary memory is accessed using address and data buses by the CPU, while secondary memory is accessed using input/output channels. Primary memory does not retain data when the power is turned off (volatile while secondary memory retains data when the power is turned off (non-volatile)). Furthermore, primary memory is very fast compared to the secondary memory and has a lower access time. But, the secondary memory devices are more costly compared to primary memory devices. Due to this reason, usually a computer comprises of a smaller primary memory and a much larger secondary memory.

Q.3 Give classification for the types of language working in computer programming environment. (18 TLV, 2019)

Ans. Computers understand only one language and that is binary language or the language of 0's and 1's. Binary language is also known as machine language. In the initial years of computer programming, all the instructions were given in binary form only. Although these programs were easily understood by the computer, it proved too difficult for a normal human being to remember all the instructions in the form of 0's and 1's. Therefore, the computer remained a mystery to a common person until other languages, such as assembly and high-level languages, were developed which were easier to learn and understand. These languages use commands that have some degree of similarity with English (such as "if else," "while"). Programming languages can be divided into three major categories:

1. **Machine Language** : It is the native language of computers. It uses only 0's and 1's to represent data and instructions.
2. **Assembly Language** : It corresponds to symbolic and executable machine codes and was created to use letters instead of 0's and 1's to run a machine.
3. **High-level Language** : These languages are written using a set of words and symbols following some rules similar to a natural language, such as English. The

programmer uses additional information with data such as the access permissions, owner, last access time, etc. Furthermore, when the primary memory is filled up, secondary memory is used as a temporary storage for keeping least used data in the primary memory. Secondary memory devices are less costly and larger in size. But they have a large access time.

Q.4

However, primary memory is the memory that is directly accessed by the CPU to store and retrieve information, whereas the secondary memory is not accessible directly by the CPU. Primary memory is accessed using address and data buses by the CPU, while secondary memory is accessed using input/output channels. Primary memory does not retain data when the power is turned off (volatile while secondary memory retains data when the power is turned off (non-volatile)). Furthermore, primary memory is very fast compared to the secondary memory and has a lower access time. But, the secondary memory devices are more costly compared to primary memory devices. Due to this reason, usually a computer comprises of a smaller primary memory and a much larger secondary memory.

Central Processing Unit (CPU) : The Control Unit and the Arithmetic Logic Unit of a computer system are jointly known as the Central Processing Unit (CPU). The CPU is the brain of a computer system. In a human body, all major decisions are taken by the brain, and the other parts of the body function are directed by the brain. In a computer system, all major calculations and comparisons are made inside the CPU, and the CPU is responsible for activating and controlling the operations of other units of the computer system.

(i) Arithmetic Logic Unit : The Arithmetic Logic Unit (ALU) of a computer system is the place, where the actual execution of the instructions takes place, during the processing operation. To be more precise, calculations are performed, and all comparisons (decision) are made in the ALU. The data and instructions, stored in the primary storage before processing, are transferred as and when needed to the ALU, where processing takes place. No processing is done in the primary storage unit. Intermediate results generated in the ALU are temporarily transferred back to the primary storage, until needed later. Hence, data may move from primary storage to ALU, back again to storage, many times, before the processing is over.

The type and number of arithmetic and logic operations, which a computer can perform, is determined by the engineering design of the ALU. However, almost all ALUs are designed to perform the four basic arithmetic operations (add, subtract, multiply, and divide), and logic operations or comparisons (such as less than, equal to, and greater than).

(ii) Control Unit : How does the input device know that it is time for it to feed data into the storage unit? How does the ALU know, what should be done with the data once they are received? Moreover, how is it that only the final results are sent to the output device, and not the intermediate results? All this is possible due to the control unit of the computer system. Although, it does not perform any actual processing on the data, the control unit acts as a central nervous system, for the other components of the computer system. It manages and coordinates the entire computer system. It obtains instructions from the program stored in main memory, interprets the instructions, and issues signals, which cause other units of the system to execute them.

Q.5 Write differences between the following :

(i) Interpreter and compiler

(ii) High level, assembly and machine language.

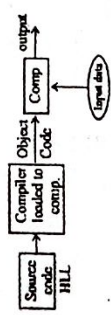
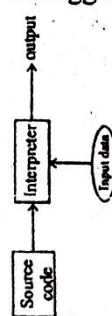
[R.T.U. 2014]

OR

Machine, Assembly and High Level Language

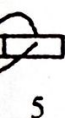
[R.T.U. 2011]

Ans.(i)

Assembler	Compiler	Interpreter
1. Assembler is a translator that translates the assembly language program into the m/c language.	1. Complex and difficult to write	1. Simple and easy to write
	2. Require large memory space to store itself	2. Doesn't require much memory space to store itself.
	3. Takes more time in translation.	3. Take less time in translation
2. It also assembles the m/c code in memory of computer and makes the program ready for execution.	4. Translates the entire program into m/c language at once.	4. Translates the program line by line into m/c language
	5. Checks the syntax errors in program when whole program become error free then it is translated into m/c code and executed	5. Check for the syntax errors line by line and executes each Line immediately if statement is error free.
	6. Takes less time to execute program	6. Takes more time to execute programs
	7. Slow for debugging	7. Fast for debugging
		

(ii) Refer to Q.2(b).

computer science, direct access) is a type of a sequence access, where a direct access. A typical example is to compare an ancient document with the data needed for a search. Random: can be accessed in any order (e.g. a page). A more efficient access is sequential—you have to get to later data to get to later data—you can jump to any data. Random Access Memory (RAM) is a semiconductor memory. (The term was used for memory in early computers.)



Sequential Access
implies the ability to access data in constant time. This, other than random access, is sometimes the only way to access data in dynamic arrays). Algorithms such as binary search, which use a structure, such as an array, to make for efficient access of data.

Random access means that data can be accessed in any order. A memory array or a list is a predetermined, sequential access. It is sometimes the only way to access data in dynamic arrays). For example, if it is on a list, you can access it in any order of choice, for example, a sequence of

many algorithms such as quicksort and binary search degenerate into bad algorithms that are even less efficient than their alternatives; these algorithms are impractical without random access. On the other hand, some algorithms, typically those which don't perform indexing, require only sequential access, such as mergesort, and so face no penalty.

-
- Q.7** (a) *What is pseudocode ? Write the advantage and limitation of pseudocode. [R.T.U. 2010]*
(b) *Write short note on volatile memory.*

[R.T.U. 2013]

Ans.(a) Pseudocode : Refer to Short Answer Q.1.

Ans.(b) Volatile Memory : It requires constant power to maintain the stored information. The fastest memory technologies of today are volatile ones. Since primary storage is required to be very fast, it predominantly uses volatile memory.

Random Access Memory (RAM) : RAM is volatile in nature, this memory consists of some integrated circuit chips either on motherboard, or on a small circuit board attached to the motherboard. A computer's motherboard is designed in a manner that its memory capacity can be easily enhanced by adding more memory chips. If you require to insert more memory, then you can easily plug-in extra memory in the empty memory slots on the motherboard. A computer containing only primary memory would not have a source to read instructions, in order to start the computer. Hence, non-volatile primary memory containing a small startup program (BIOS) is used to bootstrap the computer, that is, to read a larger program from non-volatile secondary memory to RAM and start to execute it.

-
- Q.8** *Write short note on direct access memory.*

[R.T.U. 2013]

Ans. Random Access Memory (RAM) : Refer to Q.7(b).

-
- Q.9** *Draw a flow chart which takes as input length of three sides of a triangle and check if it is a right angled triangle.*

[R.T.U. 2013]

ROM is commonly known as Programmable Read Only Memory (PROM), because a user can program it. Once the user programs are stored in a PROM chip, they can usually be executed in a fraction of the time previously required. PROMs are programmed to record information using a special device, known as PROM-programmer. However, once the chip has been programmed, the recorded information cannot be changed, i.e., the PROM becomes a ROM, and it is only possible to read the stored information. PROM is also non-volatile storage, i.e., the stored information remains intact, even if power is switched off.

Q.13 Discuss various types of memory. Explain primary and secondary memory in detail.
[R.T.U. 2010, 2008]

Ans. Memory refers to the electronic holding place for instructions and data. CPU requires memory to handle the initial input, intermediate and final results. We can classify memory into two basic categories :

- (i) Primary memory
- (ii) Secondary memory
- (i) **Primary Memory** : Refer to Long Answer Q.2(a).
- (ii) **Secondary Memory** : It is also known as auxiliary memory. Secondary memory provides backup storage for instructions (software programs) and data.

Most commonly used secondary storage devices are Hard Disk, Magnetic Disk and Magnetic Tapes. These are the least expensive among all the memories. However, they have much larger storage capacity than primary memory. Instructions and data stored on such devices are permanent in nature. It can only be removed if the user wants it or the device is destroyed the benefits of secondary storage are :

- (i) Non volatility
- (ii) Reliability
- (iii) Convenience
- (iv) Less expensive
- (v) Reusability
- (vi) Portability

Q.14 What is the function of CPU in a computer system?
[R.T.U. 2010, 2007]

Ans. The Central Processing Unit (CPU) : The CPU is the brain of a computer system. All major calculations and comparisons performed by a computer are carried out inside its CPU. The CPU is

also responsible for activating and controlling the operations of the computer system. Hence, no other single component of a computer determines its overall performance, as much as the CPU. In order to be able to quickly evaluate any computer's capabilities, it is important to know how CPUs are internally structured, how different CPUs differ from each other, and how CPU speed is evaluated. These and other related concepts about CPU are described below :

- (i) **The Control Unit** : The two basic component of a CPU are the control unit and the arithmetic logic unit. The control unit of the CPU selects and interprets program instructions, and then sees that they are executed.

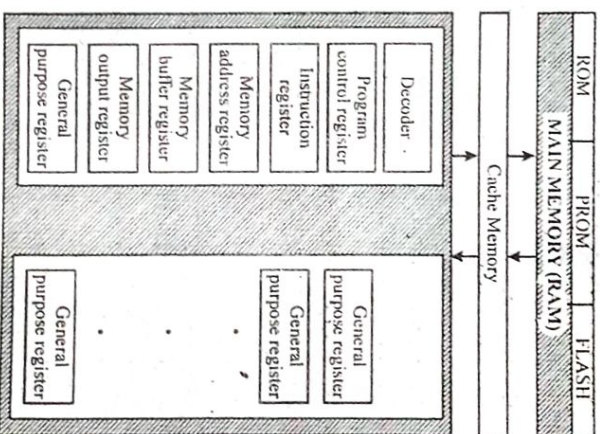


Fig. : Processor and memory architecture of a computer system

It has some special purpose registers, and a decoder to perform these activities. The special purpose registers, namely the **instruction register** and the **program control register**, respectively hold the current instruction and the next instruction to be executed, and in this way help the control unit in instruction selection. On the other hand, the decoder has the necessary circuitry to decode and interpret the meaning of every instruction supported by the CPU. Each instruction is accomplished by microcode i.e. very basic directions, which tell the CPU how to execute the instruction.

Although, the control unit does not perform any actual processing of the data, it acts as a central nervous system for the other component of the computer. It manages and coordinates the entire computer system, including the input and output units. It obtains instructions from the program stored in the main memory, interprets the instructions, and issues signals, which cause other units of the system to execute them.

- (ii) **The Arithmetic Logic Unit (ALU)** : The ALU of the CPU is the place, where the actual execution of the instructions takes place, during the data processing operation. That is, when the control unit encounters an instruction, which involves an arithmetic operation (such as, add, subtract, multiply, divide), or a logic operation (such as less than, equal to, greater than), it passes control to the ALU. The ALU has some special purpose registers and the necessary circuitry, to carry out all the arithmetic and logic operations, which are included in the instructions supported by the CPU. For example, the control unit might load two numbers into the registers in the ALU. Then, it might tell the ALU to add the two numbers (an arithmetic operation), or to check if the two numbers are equal (a logical operation).

In case of a microcomputer, the entire CPU (both the control unit and the ALU) is contained on a single tiny silicon chip, called a microprocessor.

Instruction Set : Every CPU has built-in ability to execute a set of machine instructions, called its instruction set. Most CPUs have 200 or more instructions (such as add, subtract, and compare) in their instruction set. The machine language designed for a processor (CPU), is based on the list of instructions supported by the CPU in its instruction set. Since each processor has a unique instruction set, machine language programs written for one computer will generally not run on another computer, with a different CPU.

CPUs made by different manufacturers have different instruction sets. In fact, different CPU models of the same manufacturer also often have different instruction sets. However, manufacturers tend to group their CPUs into "families", which have similar instruction sets. When a new CPU is developed, it is ensured that its instruction set included all the instructions in the instruction set of its predecessor CPU, plus some new ones. This manufacturing strategy is known as upward compatibility, and the new CPU is said to be upward compatible with its predecessor. This feature allows software written for a computer with a particular CPU, to work on computers with newer processor of the same family. In turn, it allows the users of these computer systems to easily upgrade their system, without worrying about converting all their existing software.

PPS.14

Registers : As the instructions are interpreted and executed by the CPU, there is a movement of information between the various units of the computer system. In order to handle this process satisfactorily, and to speed up the rate of information transfer, the computer uses a number of special memory units, called registers. These registers are used to hold information on a temporary basis, and are part of the CPU (not main memory).

The length of a register equals the number of bits it can store. Hence, a register that can store 8 bits is normally referred to as an 8-bit register. Most CPUs sold today, have 32-bit or 64-bit registers. The size of the registers is sometimes called the word size. The bigger the word size, the faster the computer can process a set of data. With all other parameters being same, a CPU with 32-bit registers, can process data twice as fast as one with 16-bit registers.

Although, the number of registers varies from computer to computer, there are some registers, which are common to all computers. The functions of these registers are described below.

1. Memory Address Register (MAR) : It holds the address of the active memory location. It is loaded from the program control register, when an instruction is read from memory.

2. Memory Buffer Register (MBR) : It holds the contents of the memory word read from, or written in memory. An instruction word placed in this register is transferred to the instruction register. A data word placed in this register is accessible for operation with the accumulator register, or for transfer to the I/O register. A word to be stored in a memory location must first be transferred to the MBR, from where it is written in memory.

3. Program Control Register (PCR) : It holds the address of the next instruction to be executed. Normally, the instructions of a program are stored in consecutive memory locations, and executed in sequence, unless a branch instruction is encountered. A branch instruction is an operation, which, calls for a transfer to a nonconsecutive instruction. The address part of a branch instruction is transferred to the PC register, to become the address of the next instruction.

4. Accumulator Register (AR) : It holds the data to be operated upon, the intermediate results and the results of processing. It is used during the execution of most instructions. The results of arithmetic operations are returned to the accumulator register, for transfer to main memory, through the memory buffer register. In many computers, there are more than one accumulator registers.

Programming for Problem Solving

5. Instruction Register (IR) : It holds the current instruction, which is being executed, as soon as the instruction is stored in this register, the operation part and the address part of the instruction are separated. The address part of the instruction is sent to the MAR, while its operation part is sent to the control unit, where it is decoded and interpreted, and ultimately command signals are generated to carry out the task specified by the instruction.

6. Input/Output Register (I/O) : It is used to communicate with the input/output devices. All input information, such as instructions and data, are transferred to this register by an input device. Similarly, all output information, to be transferred to an output device, are found in this register.

Table : Functions of various registers

S.No.	Name of register	Function
1.	Memory Address (MAR)	Holds the address of the active memory location.
2.	Memory Buffer (MBR)	Holds information on its way to and from memory
3.	Program Control (PC)	Holds the address of the next instruction to be executed
4.	Accumulator Register (AR)	Accumulator results and data to be operated upon
5.	Instruction Register (IR)	Holds an instruction, which it is being executed,
6.	Input/Output (I/O)	Communicate with the I/O device

The execution of an instruction by the CPU, during program execution, normally involves the following steps:

1. The control unit takes the address of the next program instruction to be executed from the program control register, and reads the instruction from the corresponding memory address, into the instruction register of the control unit.
2. The control unit then sends the operation part and the address part of the instruction, to the decoder and the memory address register, respectively.
3. The decoder interprets the instruction, and accordingly the control unit sends signals to the appropriate unit, which needs to be involved in carrying out the task specified in the instruction. For example, if it is an arithmetic or logic operation, the signal is sent to the ALU. In this case, the control unit also ensures that the data corresponding to the address part of the instruction is loaded in a suitable register in the

Programming for Problem Solving

$$\begin{array}{r}
 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \\
 + \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\
 \hline
 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0
 \end{array}$$

$$= (111011000)_2$$

$$\text{Ans. (iii)} (10101)_2 - (01110)_2$$

$$\begin{array}{r}
 1 \ 0 \ 1 \ 0 \ 1 \\
 - \ 0 \ 1 \ 1 \ 1 \ 0 \\
 \hline
 0 \ 0 \ 1 \ 1 \ 1
 \end{array}$$

$$= (00111)_2$$

$$\text{Ans. (iv)} (101)_2 \times (011)_2$$

$$\begin{array}{r}
 0 \ 1 \ 1 \\
 \times \ 1 \ 0 \ 1 \\
 \hline
 0 \ 1 \ 1 \ 0 \ 1 \\
 0 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 0 \ 1 \ 1 \ 1 \ 1
 \end{array}$$

$$\text{Here } (101)_2 \times (011)_2 = (01111)_2$$

Q2. Calculate r 's and $(r-1)$'s complement of the following:

- (i) $(111010101)_2$ (ii) $(2435)_{10}$
 (iii) $(FA2B)_{16}$ (iv) $(731.241)_8$
 (v) $(24222)_5$ (vi) $(101.001)_2$

[R.T.U. 2014]

$$\text{Ans. (i)} (111010101)_2$$

The r 's complement of this binary number $r^n - N$

$$r = 2$$

$$n = 9$$

$$N = 111010101$$

So r 's complement of this binary number is

$$= (2^9 - 111010101)$$

$$= (512 - 111010101)$$

$$[(111010101)_2 = (469)_{10}]$$

$$= 512 - 469$$

$$= 43$$

$$[(43)_{10} = (101011)_2]$$

So that value now change to the binary form

$$= (101011)_2$$

Now find the $(r-1)$'s complement of this binary number is

$$(r^n - 1) - N$$

$$= (2^9 - 1) - N$$

$$= (512 - 1) - 469$$

$$= 511 - 469$$

$$= 42$$

Now this value changes in to binary form

$$= (101010)_2$$

$$\text{(ii)} (2435)_{10}$$

Now first find the r 's complement of this decimal

number

$$r^n - N$$

$$r = 10$$

$$n = 4$$

$$N = 2435$$

So r 's complement of this decimal number is

$$(10^4 - 2435)$$

$$= 10000 - 2435$$

$$= (7565)_{10}$$

$(r-1)$'s complement of this decimal number

$$(r^n - 1) - N$$

$$= (10^4 - 1) - 2435$$

$$= 9999 - 2435$$

$$= (7564)_{10}$$

$$\text{(iii)} (FA2B)_{16}$$

The r 's complement of this hexadecimal number is

so first we convert this number into hexadecimal to decimal equivalent.

$$F \ A \ 2 \ B$$

$$\downarrow \ \downarrow \ \downarrow \ \downarrow$$

$$15 \ 10 \ \quad 11$$

So

$$15 \times 16^3 + 10 \times 16^2 + 2 \times 16^1 + 11 \times 16^0$$

$$= 61440 + 2560 + 32 + 11$$

$$= (64043)_{10}$$

So now find the r 's complement of this decimal

number

$$r^n - N$$

$$r = 10$$

$$n = 5$$

$$N = 64043$$

So

$$(10^5 - 64043)$$

$$100000 - 64043$$

$$= (35957)_{10} = (8C75)_{16}$$

So r 's complement of the given hexadecimal number

is $(8C75)_{16}$.

Now find the $(r-1)$'s complement of the given

hexadecimal number is

$$(FA2B)_{16} = (64043)_{10}$$

So $(r-1)$'s complement of the equivalent decimal

number is

$$(r^n - 1) - N$$

$$((10)^5 - 1) - 64043$$

$$= 99999 - 64043$$

$$= (35956)_{10} = (8C74)_{16}$$

So $(r-1)$'s complement of the given hexadecimal

number is $(8C74)_{16}$.

NUMBER SYSTEM 2

EXERCISES QUESTIONS

$$\text{Q.4 } (5676)_{10} = (?)_8$$

[R.T.U. 2016]

$$\text{Ans. } (5676)_{10} = (?)_8$$

$$\begin{array}{r}
 8 \overline{) 5676} \ 4 \\
 \underline{8 \ 709} \ 5 \\
 \underline{8 \ 88} \ 0 \\
 \underline{8 \ 11} \ 3 \\
 \underline{ \ 1} \ 1 \\
 \underline{ \ 0}
 \end{array}$$

$$(5676)_{10} = (13054)_8$$

Ans.

II. LONG ANSWER TYPE QUESTIONS

Q.1 Perform the following:

$$(i) (89D)_{16} + (259)_{10}$$

$$(ii) (263)_8 + (100100101)_2$$

$$(iii) (10101)_2 - (01110)_2$$

$$(iv) \text{ Multiply } (101)_2 \text{ with } (011)_2 \quad [\text{R.T.U. 2015}]$$

$$\text{Ans. (i)} (89D)_{16} + (259)_{10}$$

First we convert $(89D)_{16}$ to Decimal as:

$$(89D)_{16} = 8 \times 16^2 + 9 \times 16 + D \times 16^0$$

$$= 8 \times 256 + 144 + 13 \times 1$$

$$= (2205)_{10}$$

$$\text{Now adding } (2205)_{10} + (259)_{10}$$

$$\begin{array}{r}
 2 \ 2 \ 0 \ 5 \\
 + \ 2 \ 5 \ 9 \\
 \hline
 2 \ 4 \ 6 \ 4
 \end{array}$$

$$= (2464)_{10}$$

$$\text{Ans. (ii)} (263)_8 + (100100101)_2$$

First we convert $(263)_8$ to binary number as:

$$(263)_8 = (010110011)_2$$

$$\text{Now adding } (010110011)_2 + (100100101)_2$$

(iv) (731.241)₈
First we convert this given number into equivalent

binary form

7 3 1 . 1 4
↓ ↓ ↓ ↓ ↓
111 011 001 . 010 100
(731.24)₈ = (111 011 001 . 010 100)₂
So now find the r's complement of this given octal number change to binary equivalent number. We know that r → base of the number. So, we find the 2's complement of the given number.

So now the decimal (radix) point is removed temporarily and we will find the 2's complement.

As we see the LSB of N is 0 and Next bit is 0 and another next LSB is 1 so these three digits remain unchanged so number is

1 1 1 0 1 1 0 0 1 0 1 0 0 0
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
0 0 0 1 0 0 1 1 0 1 0 1 1 0 0

So r's complement of given number is
= (00010011010100)₂
Now change this binary equivalent number in to octal form

(00010011010100)₂ = (046.54)₈
Now find the (r-1)'s complement of this number

is
(111011001 . 010100)₂ so (r-1)'s complement is each 1's change to '0' and each '0' change to '1' then (r-1)'s complement is

(0001001100101011)₂ = (046.53)₈
So (r-1)'s complement is (046.53)₈

(v) (24222)₃
Now we find the r's complement & (r-1)'s complement of this given number first we converted this number to the decimal form

(24222)₃ = (1812)₁₀
Now, we find the r's complement of this converted decimal number is

(1812)₁₀

rⁿ - N

r = 10

n = 4

N = 1812

So r's complement is

(10)⁴ - 1812
= (8188)₁₀
= (230218)₉

Now find the (r-1)'s complement of this given

(1812)₁₀

(rⁿ - 1) - N
= (10⁴ - 1)

= 9999 - 1812

= (8187)₁₀ = (230217)₉

(vi) (101.001)₂

Again we use the concept used in question (iv) so now we find the r's of the given binary number so r = 2 so we find the 2's complement of it.

As we see the LSB of N is 1 so we keep remains unchanged of this bit and each bit changed '1' to '0' and '0' to 1 so r's complement of the given number is

r's complement (010.111)₂

Now we find the (r-1)'s complement of this given number is so each 1's change to '0' and each 0's change to 1's so

(r-1)'s complement is = (010.110)₂

Q.3 Convert the following :

(i) 3987₁₀ = ?₁₆

[R.T.U. 2013]

Ans. (i) 3987₁₀ = ?₁₆

2	3987	1
2	1993	1
2	996	0
2	498	0
2	249	1
2	124	0
2	62	0
2	31	1
2	15	1
2	7	1
2	3	1
2	1	

(3987)₁₀ = (111110010011)₂
= (F93)₁₆

Ans. (i) 101101010110₂ = ?₈

(101101010110)₂ = (7)₈
(10110101011)₂ = (26)₈

(ii) 798C₁₆ = ?₈

(798C)₁₆ = (7)₈
(798C)₁₆ = (7)₈

(798C)₁₆ = (0111100110001100)₂
= (074614)₈

Q.5(a) Write r's complement of following number where r is radix (base) of these numbers :

(i) (10111010)₂ (ii) (34056)₈

(iii) (8750)₁₀ (iv) (7F3B)₁₆

(b) Write binary codes for digits 0 to 9 in following coding systems :

(i) BCD 8421

(ii) 2421 Codes

(iii) Excess-3 Codes

(iv) Gray Codes

Ans.(a) The r's complement of a positive number N is defined as (rⁿ - N)
Where r is the base of the number system, n is the number of digits in the integer part of N.

(i) (10111010)₂
rⁿ - N
r = 2
n = 8
N = (10111010)₂
= (2⁸)₁₀ - (10111010)₂
= (256)₁₀ - (10111010)₂
= (100000000)₂ - (10111010)₂
= (1000110)₂

Thus, 2's complement of (10111010)₂ is (1000110)₂
(ii) (34056)₈
rⁿ - N
r = 8
n = 5
N = (34056)₈
= (8⁵)₁₀ - (34056)₈
= (32768)₁₀ - (34056)₈
= (100000)₈ - (34056)₈
= (43722)₈

Thus, 8's complement of (34056)₈ is (43722)₈
(iii) (8750)₁₀
rⁿ - N
r = 10
n = 4
N = (8750)₁₀
= (10⁴)₁₀ - (8750)₁₀
= 10000 - 8750
= (1250)₁₀
Thus, 10's complement of (8750)₁₀ is (1250)₁₀

(iv) (7F3B)₁₆
rⁿ - N
r = 16
n = 4
N = (7F3B)₁₆
= (16⁴)₁₆ - (7F3B)₁₆
= (65536)₁₆ - (7F3B)₁₆
= (10000)₁₆ - (7F3B)₁₆
= (A000)₁₆ - (7F3B)₁₆
= (753B)₁₆

Ans.(b) BCD : (Binary Coded Decimal) : This code is basically decimal number coded in binary. This is weighted code in which a 'weight' has been assigned to each symbol position in the code word. In this a string of binary digits represent a decimal digit. Each decimal digit 0 through 9 is represented by a string of four bits. We have number of BCD codes out of which 8421 is most popular.

(i) 8421 BCD Code : The BCD (Binary Coded Decimal) is a straight assignment of the binary equivalent. It is possible to assign weights to the binary bits according to their positions. In 8421 BCD code the weights are 8, 4, 2 and 1.
For example : The binary code 1001 can be used to represent the decimal digit 9 as
1 × 8 + 0 × 4 + 0 × 2 + 1 × 1 = 9.
We can have the following 8421 BCD codes for the ten decimal digits :

Decimal Digit	8421 BCD Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

(ii) 2421 BCD Code : In this code the weights are 2, 4, 2 and 1. The decimal numbers are represented in 4 bit form and the total four bit weight is 9(2 + 4 + 2 + 1). Thus, the 2421 BCD code represented by the decimal numbers from 0 to 9. The following table represents the 2421 BCD codes for the ten decimal digits :

Table

Decimal Digit	2421 BCD Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	1100
7	1101
8	1110
9	1111

For example :

5 9 7 → Decimal
↓ ↓ ↓ → 8421 BCD

0101 1001 0111

∴ $(597)_{10} = (0101\ 1001\ 0111)_{8421\ BCD}$

(iii) **Excess -3 Code** : Excess -3 is a non weighted code used to represent decimal numbers. This code is called excess -3 because each binary code is the corresponding excess -3 code plus 3(0011).

For Example : The 8421 BCD code for the digit 8 is 1000 and is 1011 in excess -3 code.

The excess -3 codes for the ten decimal digits can be shown as :

Decimal Digit	Excess -3 code
0	0111
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

(iv) **Gray Code** : It is named after Frank Gray. The gray code is also called **Reflective Digital Code** which has the special property that any two subsequent codes differ by only one bit. This is also called **Unit Distance Code**.

Conversion of a Binary Number to Gray Code

Step 1 : The first bit (MSB) of the gray code is the same as the first bit of the binary number.

Step 2 : The second bit of the gray code equals the exclusive-OR (XOR) of the first and second bits of the binary number.

Step 3 : The third bit of the gray code equal the exclusive-OR (XOR) of the second and third bits of the binary number.

Programming for Problem Solving

The exclusive OR (XOR) operation truth table is :

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Q.6 (a) Describe methods of representing integers and floating point numbers in computer.

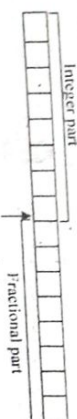
(R.T.U. 2012, 2010, 2009 R.U. 2002)

(b) Using suitable examples, explain addition and subtraction of signed integers in binary form using 2's complement representation for signed numbers.

(R.T.U. 2012)

Ans (a) **Fixed Point Representation**

In the fixed point representation, as the name suggests, the position of the decimal point is fixed. The structure of a word which has real numbers stored in it using the fixed point representation is shown below



In this, the position of the decimal point is assumed after 8 bits. Thus, the first 8 bits store the integer part of the number and the last 8 bits store the fractional part. It is not necessary that the assumed decimal point is between 8th and 9th bit in a 16 bit word. The position of the assumed decimal point can be assumed to be anywhere in the word.

Floating Point Representation

The floating point representation can be used to represent numbers in any positional number system.

Floating point representation requires that the number must be written in the format $m \times b^e$ or $m \times b^e$. Here m is the mantissa, e is the exponent and b is the base of the number being represented using floating point notation. The following table lists few floating point representations of numbers in various number systems :

Number	Mantissa	Base
Exponent		
$+2.02 \times 10^5$	+2.02	10
$+3.5 \times 10^{-3}$	+3.5	10
$+1.011 \times 2^6$	+1.011	2
-1.45×8^5	-1.45	8
$+A.F1B \times 16^4$	+A.F1B	16
-2.2×10^7	-2.2	10

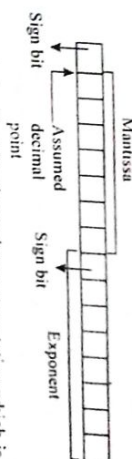
The mantissa is a signed integer which determines the actual exponent is a signed integer which determines the actual

Programming for Problem Solving

In this representation, mantissa and exponent have their own signs. In this section we will be talking about how reals are represented in computers, therefore we will be concentrating only on binary number system.

Once again we will have a problem of how to store the decimal point. Here also the decimal point is assumed and it is before the first bit in the mantissa part of the word.

The structure of a 16 bit word which will store floating point numbers can be as follows :



A standard floating point representation which is used is called **Normalized Representation**.

The **normalized representation** places the following restrictions on the mantissa of the number :

(i) The absolute value of the mantissa should be between 0 and $(10)_{10} = m < 1$, i.e., it should be a fraction.

(ii) The most significant digit after the decimal point should be non-zero for the mantissa.

To represent a real number in a word we have to follow the following steps :

Step 1 : Check if the number is in the normalized form.

Step 2 : If not, then normalize it. If it is in normalized form then perform Step 3.

Step 3 : Write the most significant bits from left to right in the mantissa portion of the word and the exponent in the exponent part.

Step 4 : Put the appropriate signs in the sign bits.

Ans. (b) One of the reasons that the 2's complement representation scheme is commonly used to stored signed integers in computers is that addition and subtraction operations for signed integers are identical to the corresponding operations for unsigned integers. This means that the CPU does not need additional circuitry for unsigned addition or subtraction operations.

Let us consider some examples :

First, 0A07 and 01D3 are added. These numbers are positive whether they are interpreted as unsigned number or as 2's complement numbers. The decimal version of the addition problem is given on the right.

0A07	2567
+ 01D3	+ 467
0BD4	3034

The answer is correct because $BDA_{16} = 3034_{10}$. Next 0206 and FFB0 are added. These are, of course, positive as unsigned number but when

interpreted as 2's complement signed numbers, 0206 is a positive number and FFB0 is a negative number. This means that there are two decimal versions of the addition problem. The signed one is given first, then the unsigned version.

0206	518
+ FFB0	+ (-80)
101B6	438

There certainly appears to be a problem because the result will not even fit in a word. In fact, because 101B6 is the hex version of 65974, there is no way to represent the correct sum of unsigned number in a word. However, if the numbers are interpreted as signed and you ignore the extra 1 on the left, then the word 01B6 is the 2's complement representation of the decimal number 438.

Now FFE7 and FFE6 are added, both negative numbers in a signed interpretation. Again, both signed and unsigned decimal interpretations are shown.

FFE7	(-25)
+ FFE6	+ (-10)
1FFDD	-35

Again the sum in hex is too large to fit in two bytes, but if you throw away the extra 1, then FFDD is the correct word-length 2's complement representation of -35.

Each of the fast two additions have carry out of the usual high-order position into an extra digit. The remaining digits do not give the correct unsigned result; however, the remaining digits to give the correct 2's complement representation. The remaining digits do not always give the correct result, even for signed numbers. Consider the addition of the following two positive numbers.

483F	18495
+ 645A	+ 25690
AC99	44185

There was no carry out of the high-order digit but the signed interpretation is plainly incorrect because AC99 represents the negative number -21351.

Intuitively what went wrong is that the decimal sum 44185 is bigger than the maximal value 32767 that can be stored in the two bytes of a word. However, when these numbers are interpreted as unsigned, the sum is correct.

Here is another example showing a "wrong" answer, this time resulting from adding two numbers that are negative in their signed interpretation.

E9FF	(-5633)
+ 8CFO	+ (-29456)
176EF	-35089

PPS.24

This time there is a carry but the remaining four digits 76 EF cannot be the right signed answer because they represent the positive number 30447. Again, intuition tells you that something had to go wrong because - 32768 is the most negative number that can be stored in a word.

In the above example with "incorrect" signed interpretations, overflow occurred. Computer hardware can detect overflow as it performs addition and the signed sum will be correct if there is no overflow. The computer actually performs addition in binary, of course and the process is logically a right-to-left pairwise addition of bits, very similar to the procedure that humans use for decimal addition. As the computer adds a pair of bits, sometimes there is a carry (of 1) into the next column to the left. This carry is added to the sum of these two bits and so on. The column of particular interest is the leftmost one-the sign position. There may be a carry into this position and/or a carry out of this position into the "extra" bit. This "carry out" (into the extra bit) is what was called just "carry" above and was seen as the extra hex 1. Figure 1 identifies when overflow does or does not occur. The table can be summarized by saying that overflow occurs when the number of carries into the sign position is different from the number of carries out of the sign position.

Each of the above addition examples is now shown again, this time in binary carries are written above the two numbers.

Carry into Sign Bit?	Carry Out of Sign Bit?	Overflow?
No	No	No
No	Yes	Yes
Yes	No	Yes
Yes	Yes	No

Fig. : Overflow in addition.

$$\begin{array}{r} 0000\ 1010\ 0000\ 0111 \\ + 0000\ 0001\ 1101\ 0011 \\ \hline 0000\ 1011\ 1101\ 1010 \end{array} \quad \begin{array}{l} 0A07 \\ + 01D3 \\ \hline 0BDA \end{array}$$

This example has no carry into the sign position (bit 15) and no carry out, so there is no overflow. The carries into bits 1, 2, and 3 have no significance with respect to overflow.

$$\begin{array}{r} 1\ 1111\ 11 \\ 0000\ 0010\ 0000\ 0110 \\ + 1111\ 1111\ 1011\ 0000 \\ \hline 1\ 0000\ 0001\ 1011\ 0110 \end{array} \quad \begin{array}{l} 0206 \\ + FFB0 \\ \hline 101B6 \end{array}$$

This example has a carry into the sign position and a carry out, so there is no overflow.

Programming for Problem Solving

$$\begin{array}{r} 1\ 1111\ 1111\ 11\ 11 \\ 1111\ 1111\ 1110\ 0111 \\ + 1111\ 1111\ 1111\ 0110 \\ \hline 1\ 1111\ 1111\ 1101\ 1101 \end{array} \quad \begin{array}{l} FFE7 \\ + FFF6 \\ \hline 1FFDD \end{array}$$

Again, there is both a carry into the sign position and a carry out, so there is no overflow.

$$\begin{array}{r} 1\ 1111\ 11 \\ 0100\ 1000\ 0011\ 1111 \\ + 0110\ 0100\ 0101\ 1010 \\ \hline 1010\ 1100\ 1001\ 1001 \end{array} \quad \begin{array}{l} 483F \\ + 645A \\ \hline AC99 \end{array}$$

Overflow does occur in this addition because there is a carry into the sign position but no carry out.

$$\begin{array}{r} 1\ 1\ 1111\ 11 \\ 1110\ 1001\ 1111\ 1111 \\ + 1000\ 1100\ 1111\ 0000 \\ \hline 1\ 0111\ 0110\ 1110\ 1111 \end{array} \quad \begin{array}{l} E9FF \\ + 8CF0 \\ \hline 176EF \end{array}$$

There is also overflow in this addition because there is a carry out of the sign bit but no carry in.

In a computer, subtraction $a - b$ of number a and b is usually performed by taking the 2's complement of b and adding the result to a . This corresponds to adding the negative of b . For example, for the decimal subtraction $195 - 618 = -423$.

00C3

-026A

is changed to addition of FD96, the 2's complement of 026A

00C3

+ FD96

FE59

The hex digits FE59 do represent -423. Looking at the above addition in binary, you have

$$\begin{array}{r} 11\ 11 \\ 0000\ 0000\ 1100\ 0011 \\ + 1111\ 1101\ 1001\ 0110 \\ \hline 1111\ 1110\ 0101\ 1001 \end{array}$$

Notice that there was no carry in the addition; however, this subtraction did involve borrow. A borrow occurs in the subtraction $a - b$ when b is larger than a as unsigned numbers. Computer hardware can detect a borrow in subtraction by looking at whether a carry occurred in the corresponding addition. If there is no carry in the addition, then there is a borrow in the subtraction. If there is a carry in the addition, then there is no borrow in the subtraction. (Remember that "carry" by itself means "carry out.")

Programming for Problem Solving

Here is one more example, Doing the decimal subtraction $985 - 411 = 574$ using word-length 2's complement representation.

03D9

-019B

is changed to addition of FE65, the 2's complement of 019B

$$\begin{array}{r} 1\ 1111\ 1111\ 1\ 1 \\ 03D9\ 0000\ 0011\ 1101\ 1001 \\ + FE65\ + 1111\ 1110\ 0110\ 0101 \\ \hline 1023E\ 1\ 0000\ 0010\ 0011\ 1110 \end{array}$$

Discarding the extra 1, the hex digits 023E do represent 574. This addition has a carry, so there is no borrow in the corresponding subtraction.

Overflow is also defined for subtraction. When you are thinking like a person, you can detect it by the wrong answer that you will expect when you know that the difference is going to be outside of the range that can be represented in the chosen length for the representation. A computer detects overflow in subtraction by determining whether overflow occurs in the corresponding addition problem. If overflow occurs in the addition problem, then it occurs in the original subtraction problem; if it does not occur in the addition, then it does not occur in the original subtraction. There was no overflow in either of the above subtraction examples. Overflow does occur if you use word-length 2's complement representation to attempt the subtraction $-29123 - 15447$. As a human, you know that the correct answer -44570 is outside the range -32,768 to +32,767. In the computer hardware

8E3D

-3C57

is changed to addition of C3A9, the 2's complement of 3C57

$$\begin{array}{r} 1\ 1\ 11\ 111\ 1 \\ 8E3D\ 1000\ 1110\ 0011\ 1101 \\ + C3A9\ + 1100\ 0011\ 1010\ 1001 \\ \hline 151E6\ 1\ 0101\ 0001\ 1110\ 0110 \end{array}$$

There is a carry out of the sign position but no carry in, so overflow occurs.

Although examples in this section have used word-size 2's complement representation, the same techniques apply when performing addition or subtraction with byte-size, double word-size, or other-size 2's complement numbers.

- Q.7(a) Calculate r 's and $(r - 1)$ complement of $(42)_{10}$
(b) Convert $(1000)_2$ to Gray Code.

Programming for Problem Solving

Q.8 Write short notes on ASCII and EBCDIC codes.

[R.T.U. 2008, Raj Univ. 2001]

OR

State the difference between ASCII and EBCDIC.

[R.T.U. 2010, Raj Univ. 2002]

OR

Explain EBCDIC.

[Raj Univ. 2004]

Ans. ASCII : This code is popular in the data communication, is used almost exclusively to represent in micro computers. ASCII is of two types : ASCII-7 and ASCII-8.

ASCII-7 is seven bit code. That allows $2^7 = 128$ different characters.

ASCII-8 is a standard version of ASCII-7. It is an 8 bit code. That allows $2^8 = 256$ different characters. ASCII also uses hexadecimal as its four-to-one short cut notation for memory dump.

EBCDIC : This character code was developed by IBM. This is alphanumeric code which stands for Extended Binary Coded Decimal Interchange Code. This is broadly similar to ASCII. With the help of this code (256) different characters can be represented since this is 8 bit code. This is extended version of BCD. BCD is a 6 bit code while this is 8 bit out of which added two bits are used as additional zone bits, expanding the zone to 4 bit. It can be easily divided in 4-bit groups, represented by one hexadecimal digit.

This code is used by large computers for transferring alphanumeric data. In this code we do not use a straight binary sequence for representing characters as in ASCII code. This makes the hardware requirement very simple and is related to punched card codes.

This also allows a large variety of printable characters and several nonprintable control characters.

Table

Character	EBCDIC Zone	Code Digit	Hexadecimal Equivalent
A	1100	0001	C1
B	1100	0010	C2
C	1100	0011	C3
D	1100	0100	C4
E	1100	0101	C5
F	1100	0110	C6
G	1100	0111	C7
H	1100	1000	C8
I	1100	1001	C9
J	1101	0001	D1
K	1101	0010	D2
L	1101	0011	D3
M	1101	0100	D4
N	1101	0101	D5

Programming for Problem Solving

O	1101	0110	D6
P	1101	0111	D7
Q	1101	1000	D8
R	1101	1001	D9
S	1110	0010	E2
T	1110	0011	E3
U	1110	0100	E4
V	1110	0101	E5
W	1110	0110	E6
X	1110	0111	E7
Y	1110	1000	E8
Z	1110	1001	E9
0	1111	0000	F0
1	1111	0001	F1
2	1111	0010	F2
3	1111	0011	F3
4	1111	0100	F4
5	1111	0101	F5
6	1111	0110	F6
7	1111	0111	F7
8	1111	1000	F8
9	1111	1001	F9

EBCDIC also uses hexadecimal number system is used as shortcut notation for memory dump. This results in a one-to-four reduction in the volume of memory dump.

Q.9 Convert the following Numbers :

(i) $(D3E0)_{16} = (?)_{10}$

(ii) $(7312)_8 = (?)_{16}$

(iii) $(736.5)_8 = (?)_{10}$

(iv) $(26.875)_{10} = (?)_2$ [R.T.U. 2010, Raj. Univ. 2004]

Ans.(i) $(D3E0)_{16} = (?)_{10}$
 $13 \times 16^3 + 3 \times 16^2 + 14 \times 16^1 + 0 \times 16^0$
 $= 53248 + 768 + 224 + 0$

$\Rightarrow (54240)_{10}$

$\therefore (D3E0)_{16} = (54240)_{10}$

Ans.

(ii) $(7312)_8 = (?)_{16}$
 $\therefore (7312)_8 = 8^3 \times 7 + 8^2 \times 3 + 8^1 \times 1 + 8^0 \times 2$

$\Rightarrow 3584 + 192 + 8 + 2$

$\Rightarrow (3786)_{10}$

16	3786	10
16	236	12
16	14	12
	0	

$\Rightarrow (0121210)_{16}$

$\therefore (7312)_8 = (CCA)_{16}$

Ans.

C PROGRAMMING 3

PREVIOUS YEARS QUESTIONS

I. SHORT ANSWER TYPE QUESTIONS

Q.1 Why do we include `<stdio.h>` in our programs? [R.T.U. 2018]

Ans. `#include<stdio.h>`

This line includes a header file which is "stdio.h" to our program. "stdio.h" means Standard Input Output header file. This file contains prototypes of some predefined functions like `printf()` and `scanf()`.

Q.2 Define a function. [R.T.U. 2018]

Ans. A function is a self-contained block of statements that performs a particular task. Once a function has been defined and packed it can be treated as a block box that takes some data from the main program and returns a value.

C functions can be defined into two categories

- Standard Library Functions
- User defined Functions

Q.3 How is an array represented in memory? [R.T.U. 2018]

Ans. Array Elements in Memory

As we have already seen that elements of array are stored continuously in memory one after one. Consider the following array declaration :

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]	arr[8]	arr[9]
12	24	-40	11	2	0	90	77		
2000	2002	2004	2006	2008	2010	2012	2014	2016	2018

in arr[8] Values of arr[9] Address of elements

Here, in the above array declaration total 16 bytes of memory is allocated, because int data type occupies 2 bytes in memory. All the elements of array contain garbage value because we have not initialized the array elements. Index of array is used to retrieve and store any value at a particular position of the array. Array index always starts from 0 and ends with (size - 1).

Q.4 Differentiate between a character and a string. [R.T.U. 2018]

Ans. Difference between Character and String

A character is just a single character enclosed in single quotes. For example:

`char initial = 'A';` /* initial declared to be a character */
And a character string is a sequence of 0 or more characters enclosed in double quotes. Each string is terminated by a NULL byte.

`char str[] = "hello, where are you these days?";`

For example, for a string containing maximum 10 characters, we need to declare character array of size 11 bytes, as:

`char msg[11] = "hello dear";` /* 'msg' is 10 char string */

Here, 'msg' is a 10 characters string while it's declared and initialized to contain 11 characters. Last character is NULL byte to terminate the string 'msg'.

Q.5 Differentiate between `ptr++` and `*ptr++`. [R.T.U. 2018]

Ans. The difference between the `ptr++` and `*ptr++` can be explained by an example-

*`ptr` - Holds the value.

`ptr` - Holds the address.

Programming for Problem Solving

So, if we write `ptr++`, then this will increment the address. But if we write `*ptr++` then it will increment the value of *`ptr`, because ++ operator is right associative.

For example,

`int *ptr = 5;`

// then if we print *`ptr`, then we will get the value 5

// but now if we write: `*ptr++`, since it is right associative, therefore first address will increment and then it will print the result.

Q.6 Differentiate between a structure and an array. [R.T.U. 2018]

Ans.

Basis for Comparison	Array	Structure
Basic	An array is a collection of variables of same data type.	A structure is a collection of variables or different data type.
Syntax	data type array_name[size];	Struct struct_name{ type element1; type element2; } variable1, variable2, ...;
Memory	Array elements are stored in contiguous memory location.	Structure elements may not be stored in a contiguous memory location.
Access	Array elements are accessed by their index number.	Structure elements are accessed by their names.
Operator	Array declaration and element accessing operator is "[]"	Structure element accessing operator is "." (Dot operator).

Q.7 What do you understand by EOF? [R.T.U. 2018]

Ans. EOF : In computing, end-of-file (commonly abbreviated EOF) is a condition in a computer operating system where no more data can be read from a data source. The data source is usually called a file or stream. In the C Standard Library, the character reading functions such as `getchar` return a value equal to the symbolic value

(macro) EOF to indicate that an end-of-file condition has occurred. The actual value of EOF is implementation-dependent and is distinct from all valid character codes. Block-reading functions return the number of bytes read, and if this is fewer than asked for, then the end of file was reached or an error occurred.

Q.8 "Structure declaration reserves memory for the structure" Comment on this statement with valid justification. [R.T.U. 2018]

Ans. The statement "structure declaration reserves memory for the structure" can be explained in following manner. In C programming the memory is allocated only when we declare the variables of the structure. In other words, the memory is allocated only when we instantiate the structure. In the absence of any variable, structure definition is just a template that will be used to reserve memory when a variable of type struct is declared. Once the variables of a structure are defined, we can perform a few operations on them. For example, we can use the assignment operator (=) to assign the values of one variable to another. Another important point is that members inside the structure definition are attached to the structure variable. They don't have any existence without structure variable. Member names inside a structure must be different from one another, but the member names of two different structures can be same.

II. LONG ANSWER TYPE QUESTIONS

Q.1 How are strings read from the standard input device? Explain the different functions used to perform string input operation. [R.T.U. 2018]

Ans. Using %s format specifier when we read a string, C compiler automatically assigns all the characters in separate memory blocks and add a null character ('\0') at the end of the string. At the time of printing %s format specifier is used in `printf()` function and it prints the character until it reaches '\0'.

Example :

```

main()
{
    char name[25];
    printf("Enter Your Name:");
    scanf("%s", &name);
    printf("\nHello %s!", name);
}
    
```

Here is a simple run of the program...

Enter Your Name :- SANJAY
Hello SANJAY!

Some other String Input/Output Functions

gets() : `gets()` function is declared in "stdio.h". Using this function we can read more than one word (including spaces) because it gets terminate when new line character (`\n`) is encountered.

puts() : `puts()` function is also declared in "stdio.h". This function can print a string on console, and has inbuilt new line character (`\n`).

getch() : `getch()` function is declared in "conio.h". This function reads a character from the console but does not echo it to the screen.

getche() : `getche()` function is also declared in "conio.h". This function reads a character from the console and echo it on the screen.

getchar() : `getchar()` reads a character from the console and reading is terminated when new line character (`\n`) is entered. The character which we enter is also displayed on the screen.

putchar() : `putchar()` prints the character argument on the standard output.

If we are using `getch()`, `getche()`, `getchar()` or `putchar()` for reading or displaying a string we require a loop as these functions/macros are used to read or display a single character.

A program to find occurrence of any particular character in a string.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char c,str[10];
    int i,count=0;
    clrscr();
    printf("\nEnter a string: ");
    gets(str);
    printf("\nEnter a character: ");
    scanf("%c",&c);
    for(i=0;str[i]!='\0';i++)
    {
        if(str[i]==c)
            count++;
    }
}
```

Programming for Problem Solving

`printf("\nThis character comes in string %d times",count);`

Output :

Enter a string: - INDIA IS GREAT
Enter a character: - A
This character comes in string 2 times

Q.2 Write a program in C that replaces a given character with another character in the string. [R.T.U. 2018]

Ans. #include <stdio.h>
#include <string.h>
int main()

```
{
    char str[100],ch,Newch;
    int i;
    printf("\n Please Enter any String : ");
    gets(str);
    printf("\n Please Enter the Character that you want to Search for : ");
    scanf("%c",&ch);
    getche();
    printf("\n Please Enter the New Character : ");
    scanf("%c",&Newch);
    for(i=0;i<=strlen(str);i++)
    {
        if(str[i]==ch)
        {
            str[i]=Newch;
        }
    }
    printf("\n The Final String after Replacing All Occurrences of '%c' with '%c' = %s",ch,Newch, str);
    return 0;
}
```

Output:

Please Enter any String : Neelekanth
Please Enter the Character that you want to Search for : g
Please Enter the New Character : h
The Final String after Replacing All Occurrences of 'g' with 'h' = Neelekanth

Programming for Problem Solving

Q.3 Write a program to create a structure with the information given below. Then read and print the data.

- (a) Emp_Id
- (b) Name
- (i) First Name
- (ii) Middle Name
- (iii) Last Name
- (c) Address
- (i) Area
- (ii) City
- (iii) State
- (d) Age
- (e) Salary
- (f) Designation

[R.T.U. 2018]

Ans. #include <stdio.h>
/*structure declaration*/
struct employee{
 int empId;

```
    char firstName[30];
    char middleName[30];
    char lastName[30];
    char area[50];
    char city[30];
    char state[30];
    float age;
    float salary;
    char designation[30];
};
```

```
int main()
{
    /*declare structure variable*/
    struct employee emp;
```

```
    /*read employee details*/
    printf("\nEnter details :\n");
    printf("Emp_Id : ");
    scanf("%d",&emp.empId);
    printf("First Name : ");
    gets(emp.firstName);
    printf("Middle Name : ");
    gets(emp.middleName);
    printf("Last Name : ");
    gets(emp.lastName);
}
```

```
printf("Address :\n");
printf("Area :");
gets(emp.area);
printf("City :");
gets(emp.city);
printf("State :");
gets(emp.state);
```

```
printf("Age :");
scanf("%d",&emp.age);
printf("Salary :");
scanf("%d",&emp.salary);
printf("Designation :");
gets(emp.designation);
```

```
/*print employee details*/
printf("\nEnter detail is:\n");
printf("Emp_Id :%d\n", emp.empId);
```

```
printf("Name:\n");
printf("First Name: %s\n", emp.firstName);
printf("Middle Name: %s\n", emp.middleName);
printf("Last Name: %s\n", emp.lastName);

printf("Address:\n");
printf("Area: %s\n", emp.area);
printf("City: %s\n", emp.city);
printf("State: %s\n", emp.state);

printf("Age: %f\n", emp.age);
printf("Salary: %f\n", emp.salary);
printf("Designation: %s", emp.designation);
return 0;
}
```

Q.4 What is the advantage of using structures? Differentiate between a structure and a union. [R.T.U. 2018]

Ans. Advantages of Structure :

1. Structure is collection of different types of variables referenced under one name. By this we need not define same category variable types in different variables.
2. Memory is not allocated at the time of declaration of structure. After creation of structure variables memory is allocated.
3. Nested structure can also be defined.

4. Continuous memory allocation for structure members.
 5. Structure is a user defined data type, so that it facilitates user to create variable according to the requirements.
 6. Working is same as predefined data types.
- Structures :** Keyword - struct
- The structures are the heterogeneous user defined data types where a structure content number of data types grouped together. These data types may or may not be same. E.g. to keep record of students we may use following structure

```
struct student
{
    char name [30];
    int age;
    char father_name [30];
    char class;
    int totalmarks;
}
```

Thus in this way we can group all common attributes of an object record and can access all of them via same name. E.g. we have declared variable of above structure student.

```
/* declaration */
struct student ram;
now if we need age of Ram we can store and retrieve it from
```

```
ram.age;
```

thus to access data struct we use . operator.

Union : Unions are used to save memory i.e. when we declare the union then the space allocated for use of that union is equal to memory required for the biggest variable.

Keyword union

Union is also heterogeneous data type. So in it we can use int, char, and float.

```
Union data
{
    int a;
    float p;
    char c;
    } d7;
```

In union only one member can be used at a time or only one member contains valid values.

Q5. What is dynamic memory allocation? What are the various functions in C that are used for dynamic memory allocation? Explain.

[R.T.U. 2018]

Ans. Sometimes it is desirable to create objects at run time. That is we do not inform the compiler that an object of a specified type and name needs to be created. Instead we create an object of the appropriate type ourselves at run time. This is done by allocating memory for the object that we need to create. Since the memory is allocated for objects "dynamically" (i.e., at run time), creation of such objects is known as **dynamic memory allocation**.

The C standard library provides the function malloc() to create object dynamically. The function takes an argument the size in bytes of the object to be created. It either returns the pointer to the created object if it was able to allocate memory or returns NULL if not.

Function	Use of Function
malloc()	Allocates requested size of bytes and returns a pointer
calloc()	Allocates space for an array elements, initializes to zero and then returns a pointer to memory
free()	deallocate the previously allocated space
realloc()	Change the size of previously allocated space

Syntax of malloc()

```
ptr = (cast-type*) malloc(byte-size);
```

Here, *ptr* is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

Example

```
ptr = (int*) malloc(100 * sizeof(int));
```

This statement will allocate either 200 or 400 according to size of int 2 or 4 bytes respectively and the pointer points to the address of first byte of memory.

Syntax of free()

```
free(ptr);
```

This statement frees the space allocated in the memory pointed by *ptr*.

calloc()

The calloc() stands for contiguous allocation. This function allocates memory at runtime as malloc() does. The difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

Syntax

```
ptr = (cast-type*)calloc(n, element-size);
```

Here, *ptr* is pointer of cast-type. The calloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

Example

```
ptr = (float*) calloc(25, sizeof(float));
```

This statement allocates contiguous space in memory for an array of 25 elements each of size of float, i.e., 4 bytes.

Program :

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int num, i, *ptr, sum = 0;
    printf("Enter number of elements: ");
    scanf("%d", &num);
    ptr = (int*) calloc(num, sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i = 0; i < num; ++i)
    {
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }
    printf("Sum = %d", sum);
    free(ptr);
    return 0;
}
```

realloc()

Sometimes previously allocated memory becomes insufficient or more memory is required as compared to already allocated memory. So, realloc() function change the previously allocated memory (allocated using malloc() and calloc()) and provides new memory blocks as required.

Syntax

```
ptr = realloc(ptr, newsize);
```

Here, *ptr* is reallocated with size of newsize.

Q6. Write a short note on the following functions. For each function, give a program code that demonstrates its usage-

- (i) fopen()
- (ii) fclose()
- (iii) fgetc()
- (iv) fseek()

[R.T.U. 2018]

Ans. (i) fopen() Function : The first operation needed to be done before processing data in a file is to open the file. This is done by the fopen() function.

The prototype of fopen() is given below :

```
FILE * fopen (char * filename, char * mode);
```

Where filename specifies the name of the file on the secondary storage and mode specifies whether the file should be opened for reading or writing or for both.

The function fopen() returns a pointer to a FILE object if it could successfully open the file. It returns NULL if it could not. It is always good to test the value returned by fopen() function.

(ii) fclose() Function : All files opened, must be closed at the end of the program. This is done by the library function fclose()

The prototype of fclose() is given by

```
int fclose(fp);
```

Where, *fp* is a pointer to a FILE object of an open file. When we close a file using fclose() following operations are performed :

- The characters in the buffer would be written to the file on the disk.
- At the end of file a character with ASCII value 26 would get written.
- The buffer would be removed from memory.

Though most operating systems automatically close files when the application terminates, explicitly closing all open files is a good programming practice.

Program to demonstrate fopen() & fclose() function:

```
#include <stdio.h>
#include <process.h>
void main()
{
    FILE *fp1, *fp2;
    char a;
    clrscr();
    fp1 = fopen("A.txt", "w");
    if(fp1 == NULL)
```

```

    puts("cannot open this file");
    exit(1);
}
fp2=fopen("B.txt", "w");
if (fp2==NULL)
{
    puts("Not able to open this file");
    fclose(fp1);
    exit(1);
}
do
{
    a=fgetc(fp1);
    a=toupper(a);
    fputc(a,fp2);
}
while(a!=EOF);
fcloseall();
getch();
}

(iii) fgetc() Function : fgetc() function reads next
n-1 characters and stops when a new line character is
encountered. The newline character is included in the
array and it is terminated by '\0'.
The prototype of fgetc() is :
char * fgetc(char * s, int n, FILE * fp);
It reads n-1 characters in s. On success it returns
on end of file and on error it returns NULL.

#include <stdio.h>
#define MAX 15
int main()
{
    char buf[MAX];
    fgetc(buf, MAX, stdin);
    printf("string is: %s\n", buf);
    return 0;
}

(iv) fseek() Function : fseek() function can be
used to directly manipulate the file pointer so that it points
to the desired byte offset. The prototype of fseek() is :
int fseek(FILE *fp, long offset, int origin);
where fp is a pointer to a FILE object representing
a file.

#include <stdio.h>
int main () {

```

Q7 Write a program to read data from the keyboard and write it to a file. Read the contents stored in the file and display it on the screen. [R.T.U. 2018]

```

Ans. #include <stdio.h>
int main()
{
    FILE *fp; /* file pointer*/
    char fName[20];
    printf("\nEnter file name to create :");
    scanf("%s", fName);
    /*creating (open) a file*/
    fp=fopen(fName, "w");
    /*check file created or not*/
    if (fp==NULL)
    {
        printf("File does not created!!!");
        exit(0); /*exit from program*/
    }
    printf("File created successfully.");
    /*writing into file*/
    putc('A', fp);
    putc('B', fp);
    putc('C', fp);
    printf("\nData written successfully.");
    fclose(fp);
    /*again open file to read data*/
    fp=fopen(fName, "r");
    if (fp==NULL)
    {
        printf("\nCan't open file!!!");
    }
}

```

```

    exit(0);
}
printf("Contents of file is :\n");
printf("%c", getc(fp));
printf("%c", getc(fp));
printf("%c", getc(fp));
printf("%c", getc(fp));
fclose(fp);
return 0;
}

```

Q8 Write a program using pointers to insert a value in an array. [R.T.U. 2018]

```

Ans. #include <stdio.h>
int main()
{
    int data[5], i;
    printf("Enter elements:");
    for(i = 0; i < 5; i++)
        scanf("%d", data + i);
    printf("You entered: \n");
    for(i = 0; i < 5; i++)
        printf("%d\n", *(data + i));
    return 0;
}

```

Q9 Explain the difference between a null pointer and a void pointer. What do you mean by pointer arithmetic? [R.T.U. 2018]

Discuss void pointer and its utility in brief. [R.T.U. 2016]

Ans. Null Pointer : Null pointer is a pointer which is pointing to nothing. In case, if we don't have address to be assigned to a pointer, then we can simply use NULL.

```

#include <stdio.h>
int main()
{
    // Null Pointer
    int *ptr = NULL;
    printf("The value of ptr is %u", ptr);
    return 0;
}

```

Void Pointer : We know that pointers are used to point to different data types. A float pointer points to float variables, int type pointer points to integer variables, a char type pointer points to character variables. In C, there is a general purpose pointer that can point to any data types and is known as void pointer. The void pointer is a generic pointer that can represent any pointer type. The syntax of its declaration is given below :

```
void *vptr; /*pointer to void */
```

In C, pointers to void cannot be directly dereferenced like other pointer variables by using *, the indirection operator. A suitable typecast is must prior to dereferencing a pointer to void as given below :

```

    * (type *)vptr
    ↑
    dereferencing operator
    ↑
    type cast

```

Here, type refers to any valid C data type. Null pointer is a special reserved value of a pointer. A pointer of any type has such a reserved value. Formally, such specific pointer type (int *, char * etc.) has its own dedicated null-pointer value. Conceptually, when a pointer has that null value it is not pointing anywhere.

Void pointer is a specific pointer type - void * - a pointer that points to some data location in storage, which doesn't have any specific type. So, once again, null pointer is a value, while void pointer is a type. These concepts are totally different and non-comparable.

Pointer Arithmetic:

Pointer arithmetic is slightly different from arithmetic we normally use in our day to day life. There are four

arithmetic operators that can be used on pointers: ++, --, +, and - operators.

To understand pointer arithmetic, let us consider that ptr is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer -

ptr++

After the above operation, the ptr will point to the location 1004 because each time ptr is incremented, it will point to the next integer location which is 4 bytes next to the current location. This operation will move the pointer to the next memory location without impacting the actual value at the memory location. If ptr points to a character whose address is 1000, then the above operation will point to the location 1001 because the next character will be available at 1001.

Pointer Expression	How it is evaluated ?
ptr = ptr + 1	ptr => ptr + 1 => 1000 + 1*4 => 1004
ptr++ or ++ptr	ptr++ => ptr + 1 => 1004 + 1*4 => 1008
ptr = ptr + 5	ptr => ptr + 5 => 1008 + 5*4 => 1028
ptr = ptr - 2	ptr => ptr - 2 => 1028 - 2*4 => 1020
ptr-- or --ptr	ptr => ptr + 2 => 1020 + 2*4 => 1028

For example:

```
int main()
{
    const char *p = "Hello";
    while(*p++)
        printf("%c", *p);
    return 0;
}
```

Output:
ello

Q.10 Write a program in 'C' to read characters one by one and display it back. [R.T.U. 2016]

Ans.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char inputString[100], c;
    int index = 0;
    printf("Enter a string\n");
    while((c = getchar()) != '\n') /* Read string from
        user using getchar inside while loop */
```

```
{
    inputString[index] = c;
    index++;
}
inputString[index] = '\0';

index = 0;
while(inputString[index] != '\0')
{
    putchar(inputString[index]); /* Print string stored in
        inputString using putchar */
    index++;
}
getch();
return 0;
}
```

Output -

Enter a string

Neelkanth

Neelkanth

Q.11 (a) Discuss Enumerated data type with the help of suitable code of 'C' language.

(b) Arrange following operators in order of precedence and associativity rules of 'C'.
+, %, <, /, *, =, !, ++, () [R.T.U. 2016]

Ans.(a) Enumerated data type : An enumeration data type is a user defined type, with values ranging over a finite set of identifiers called enumeration constant. For example

```
enum color {red, blue, green};
```

This defines color to be of a new data type which can assume the value, red, blue, or green. Each of these is an enumeration constant. In the program, color can be used as a new type. A variable of type color can have any one of the three values: red, blue or green. For example, the statement

```
color c;
```

defines c to be of type color. Internally, the C++ compiler treats an enum type (such as color) as an integer itself. The above identifiers red, blue, and green represent the integer values of 0, 1 and 2 respectively. So, the statements

```
c = blue;
```

```
count << "As an int, c has the value" << c;
```

will print As an int, c has the value 1

Constant values can be explicitly specified for the identifiers. When the value for one identifier is specified in this manner, the value of the next element is incremented by one (next higher integer). For example, if the definition of color is

Enter elements a24: 5
 Enter elements a31: 3
 Enter elements a32: 0
 Enter elements a33: 4
 Enter elements a34: 5

Enter elements of matrix 2:

Enter elements b11: 2
 Enter elements b12: 3
 Enter elements b21: -9
 Enter elements b22: 0
 Enter elements b31: 0
 Enter elements b32: 4
 Enter elements b41: 3
 Enter elements b42: -9

Output Matrix:

39 -16

21 -20

21 -20

Q.14 (a) What is scope of variable in 'C' meant? Explain type casting in detail also. [R.T.U. 2015]

OR

What is meant by scope of variables in C? What are the different types of variables on the basis of scope? List and explain with example? [Raj. Univ. 2005]

(b) Explain the difference between automatic and static variable. [R.T.U. 2015]

Ans.(a) In C the scope of variables means that in functions the value of the variable would be available.

C provides a variety of storage class specifiers that can be used to declare explicitly, scope and life time of variables. The storage class decides the portions of the program within which the variables are recognized. There are basically two types of variables on the basis of scope.

- (1) Global Variable
- (2) Local Variables

Example: int m;
 main ()

```
{
    int i;
    float balance;
    _____
    _____
    function1 ();
```

```
}
function1 ()
{
    int i;
    float sum;
    _____
    _____
}
```

1. **Global Variables** : The variable *m* which has been declared before the main is called global variable. It can be used in all the functions in the program. It need not be declare in other functions. A global variable is also known as an external variable.

2. **Local Variables** : The variables *i*, *balance* and *sum* are called local variables because they are declared inside a function. Local variables are visible and meaningful only inside the function, in which they are declared. They are not known to other function. The storage class has another qualities (like long or unsigned) that can be added to a variable

```
auto int count;
register char ch;
static int x;
extern long total;
```

Storage

Meaning

Auto

Local variable which known only to the function in which it is declared. Default is auto.

Static

Local variable which exist and retain it value even after the control transferred to the calling function.

Extern

Global variable known to all functions in the file.

Register

Local variable which is stored in the register.

Type Casting or Conversion

Type casting means of conversion of one data type to another data type. For example, a float value can be converted into an integer value. There are two types of type casting that are discussed below :

- (a) Implicit type casting
- (b) Explicit type casting

(a) **Implicit Type Casting** : In implicit type casting, the conversion is done automatically by the compiler. In this type of conversion, a smaller data type is converted into a larger data type. For example, a float value can be converted into a double value because the size of the float value is 4 byte and the size of the double value is 8 bytes. So, it is possible for a double variable to take a float value. No operator is required in an implicit

conversion. This is done without any intervention of the programmer.

(b) Explicit Type Casting : In explicit type casting, the conversion is carried out between two incompatible types.

For example, a float value can be converted into an integer value. This is done with the help of the (type) operator. The syntax is as follows :

data_type var1 = (type) var2

For example

int i = (int) 10.23

In the above example, the value (10.23) is converted into an integer value (10) and after that the resultant value (10) is assigned to the variable i. Here int is used as a (type) operator to convert the float value into an integer value.

Ans.(b) Difference between Automatic and Static Variable

S. No.	Static Variable	Automatic Variable
1.	We have to specify the storage class to make a variable static.	It is the default storage class.
2.	If it is not assigned any value then it will give 0 as output.	If it is not assigned any value then it will give garbage value as output.
3.	It is visible to the block in which it is declared and also in the function where it will be passed.	It is visible to the block in which the variable is declared.
4.	It retains its value between different function calls. It holds its last value.	It retains its value till the control remains in the block in which the variable is declared.
5.	Static variable should be compile by compiler first.	Auto variable will compile by the compiler after the static variable.

Q.15 Write a C language program to SWAP two numbers using pointers and function.

[R.T.U. 2015]

OR

Write a program in C language for swapping two numbers using pointer. [R.T.U. 2011, 2008]

Ans. #include <stdio.h>
#include <conio.h>
void swap(int*, int*);
void main()

```
{
    int a, b;
    clrscr();
    printf("\n enter two numbers:");
    scanf("%d%d", &a, &b);
    swap(&a, &b);
    printf("\n values after swapping are a = %d
and b = %d", a, b);
}

void swap (int* x, int* y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
    getch();
}
```

Output : Enter two numbers :

a = 20

b = 10

Values after swapping are

a = 10

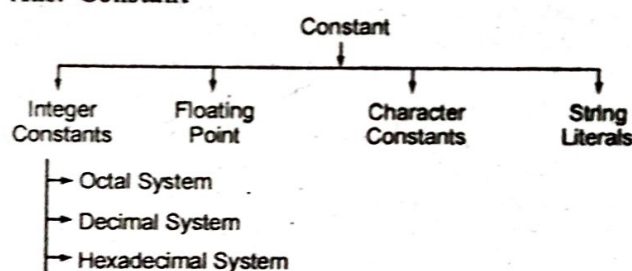
b = 20

Q.16 What is the difference between Constant and Variable? What are the data types in 'C'? Explain with examples. [R.T.U. 2014, Raj. Univ. 2006, 2001]

OR

What is the difference between Constant and Variable? [R.T.U. 2011]

Ans. Constant



A constant does not change its value during the entire execution of the program. They can be classified as integer, floating point, character and enumeration constants.

(i) Integer Constants : C++ allows to represent the integer constants in three forms. They are octal, decimal and hexadecimal.

(a) Octal System (Base 8) : Octal numbers are specified with a leading zero, rest of the digits being

Primary data types discussed above have also been explained in the following table with their Range, Memory Space and Format Specifiers.

Table : Data Type

Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
signed int	-32768 to + 32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to + 2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to + 3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to + 1.7e4932	10	%Lf

B. User Defined Data Type : Type definition is a feature in C allows users to define an identifier that would represent an existing data type. The user defined data type can later be used to declare variables.

It's general form is typedef type identifier where type refers to an existing data type and identifier refers to the "new name given to the data type"

e.g. typedef int marks
typedef float percentage

Here units symbolize int and marks symbolize float. They can be later used to declare variables as follows

units batch1, batch2 and marks name1 [50], name2 [50]

where batch1 and batch2 are declared as int variables and name1 [50] and name2 [50] are declared as 50 element floating point array variables.

The main advantage of the data type (typedef) is, we can create meaningful data type names for increasing the readability of the programme

These are defined data types are : arrays, structures, union, enum etc.

Structures : Refer to Long Answer Q.4.

Union : Refer to Long Answer Q.4.

Pointers : Pointers are the special variables used to point other storage spaces of same kind as the name indicate. Empty data set is used in functions.

Enumeration : Refer to Long Answer Q.11(a).

Q.17 Write a program of add two numbers and explain this program step by step with the help of flow chart. [R.T.U. 2014, 2011, Raj. Univ. 2006, 2001]

Ans. A program to read two numbers and display the total is as follows :

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

```
main()
{
    int a,b,c;
    printf("Enter the values of a and b");
    scanf("%d %d", &a, &b);
    c = a + b;
    printf("Sum= %d", c);
}
```

Explanation of the Program

#include<stdio.h> : Refer to Short Answer Q.1.

```
main()
{
```

Here, main() is a function. Every 'C' program contains at least one or more than one functions. One function which is compulsory that is main(). These {} braces enclose the definition of the function.

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

This line declares three variables a, b and c of integer data type. Two bytes are allocated to all these variables. With this line an address is also allocated to these variables. Data types have been explained in detail in next chapter.

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

This line or printf() function prints the text or value which is enclosed in double quotes.

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

This scanf() function scans that type of values and number of values specified in double quotes. Here double quotes contain two "%d%d", which indicates that here scanf() function scans two integer type values which are given to variables a and b. Address of these variable is accessed by address operator(&).

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

Assignment operator(=) solves the right hand side expression and assign it to left hand side variable which is c here.

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

Here printf() function doing the same job which is defined above. Here it also prints the number of values which are enclosed in double quotes. Here double quotes contain one "%d" so it will print one integer type value which is of c.

Output

```
Enter the values of a and b
10
20
Sum = 30
```

Flowchart

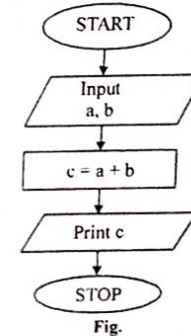


Fig.

To read an input we use READ statement and to print some output we use WRITE statement. As these statements are input/output statements, these are enclosed within Input/Output box.

Q.18 Write a program to check whether the input number is prime or not.

[R.T.U. 2014, 2011, Raj. Univ. 2004]

```
Ans. #include<stdio.h>
#include<conio.h>
void main()
```

```
{
    int f = 0, n, i;
    clrscr();
    printf("\nEnter a number");
    scanf("%d", &n);
    for(i=2; i<n; i++)
    {
        if(n%i==0)
        {
            f = 1;
            break;
        }
    }
    if(f==0)
    printf("\nNumber is Prime");
    else
    printf("\nNumber is Not Prime");
    getch();
}
```

Output :

```
Enter a number 5
Number is Prime
```

Q.19 Explain structure of program in C language briefly. [R.T.U. 2014]

Ans. Header Files : These are the files that are included at the top of any program. If we use any function inside a program, then the header file containing declaration or definition of that function, has to be included. Like `printf()` is defined in `stdio.h`. So, we must include it (by `#include` in order to use `printf()`).

Library Files : These are the files which the compiler uses in order to define the functions which have been used in the program and had been declared inside the header file. Like, `printf()` has its complete definition, i.e., how it will work etc. in an I/O library. So, the compiler uses that library to get the machine code for `printf()`.

Differences

- Header files are TEXT files while library files are BINARY. This means, we can read and modify the header file but not the library file.
- Header file is in C language while the library file is in machine language.
- Header file has to be included by the programmer while the compiler automatically relates the library file(s) with the program.

Q.22 (a) List arithmetic and logical operators available in C.

(b) Write a program to find sum of following n terms of a series. $1^2, 2^2, 3^2, 4^2, \dots$

[R.T.U. 2012]

Ans. (a) Arithmetic Operators

Arithmetic operators are used to perform numeric calculations. These can operate on any built in data type allowed in C. These operators are listed below in the table 1:

Table 1 : Arithmetic Operators

Operator Symbol	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division (applicable only on int data type)

Logical Operators

Logical operators are used to combine two expressions. These operators check logical relation between the two or more expressions. Depending upon the true and false nature of the expression they assigned value 1 and 0. Logical operators are listed below in the table 2.

Table 2 : Logical Operators

Logical Operators	Meaning
&&	AND
	OR
!	NOT

The logical operators `&&` and `||` are used when we want to test more than one condition and make decisions.

Example : `z > y && x == 10`

An expression of this kind, which combines two or more relational expression is termed as a logical expression or a compound relational expression.

1. Logical AND (&&) Operator

Logical AND works with two or more operands and gives the result true if all the operands in the condition have true value, otherwise it provides the result false.

Table 3 : Logical AND (&&)

Operand 1	Operand 2	Result
False	False	False
False	True	False
True	False	False
True	True	True

Example : Assume we want to compare age and salary of an employee

`age > 25 && salary > 10000`

Here, from the above example if age of the employee is greater than 25 and salary is greater than 10000 then the result will be True (1) only, otherwise result will be False (0).

2. Logical OR (||) Operator

Logical OR works with two or more operands and gives the result true if one of the operands in the condition have true value, otherwise it provides the result false.

Table 4 : Logical OR (||)

Operand 1	Operand 2	Result
False	False	False
False	True	True
True	False	True
True	True	True

Example : Assume that we want to compare age and salary of an employee

`age > 25 || salary > 10000`

Here, in the above example the condition will be true in all the following cases :

- If age of employee is greater than 25, salary can be anything.
- If salary of employee is greater than 10000, now age can be anything.

(iii) If both age is greater than 25 and salary greater than 10000.

(iv) Only, if age is not greater than 25 and salary is not greater than 10000 then the condition will result false.

3. Logical Not (!) Operator

This is a unary operator. This operator converts a true value into false and a false value into true.

Table 5 : Logical Not (!)

Condition	Result
True	False
False	True

Example : `!(a > b)`

Here, if `a` is greater than `b` then the actual result is true, but we have used `!` operator before the expression so the result will be false. And if `a` is not greater than `b` then the result will be false, but `!` will convert it into true.

Ans. (b) # include <stdio.h>

include <conio.h>

void main ()

{

int i, n;

long sum = 0;

clrscr ();

printf ("Enter the value of n = ");

scanf ("%d", &n);

for (i=1; i ≤ n; i++)

sum = sum + i * i;

printf ("sum of series = %ld", sum);

getch ();

}

Output

Enter the value of n = 5

Sum of series = 55.

Q.23 What are the advantages and disadvantages of C language? [R.T.U. 2011]

Ans. Advantages of C Language

- C is a building block for many other currently known languages. Take a look at Python for example a fully Object-Oriented High-Level programming language. It is written in C (perhaps C++ too). That tells you if you ever want to know what is going on under the hood in other languages; understanding C and how it works is essential.
- C is a compiled language versus an interpreted language. Explained simply, this means that the code is compacted into executable instruction (in